TinkerGen STEM made simple | seeed studio

TinyML with Wio Terminal

Basic Course Information

Course name: TinyML with Wio Terminal

Course introduction

Learn how to train and deploy deep neural network models on Cortex–M core microcontroller devices from Seeed studio. Course content features seven detailed step–by–step projects, that will allow students to grasp basic ideas about modern Machine Learning and how it can be used in low–power and footprint microcontrollers to create intelligent and connected systems.

Course category

Minimum age of students: 12+
Planned number of classes: 16
Duration of a single lesson: 45 mins

Curriculum requirements: Basic knowledge of Arduino IDE and C++
Media required for the course: (final delivered document content)

- Teaching content PDF: Yes
- Teaching content PPT slides: No
- Source code: Yes
- Publish on M2L: Possibly
- Other: None

Course Overview

Course Description

Learn how to train and deploy deep neural network models on Cortex–M core microcontroller devices from Seeed studio. Course content features seven detailed step–by–step projects, that will allow students to grasp basic ideas about modern Machine Learning and how it can be used in low–power and footprint microcontrollers to create intelligent and connected systems. After completing the course, the students will be able to design and implement their own Machine Learning enabled projects on Cortex–M core microcontrollers, starting from defining a problem to gathering data and training the neural network model and finally deploying it to the device to display inference results or control other hardware appliances based on inference data.

Course content is based on using Edge Impulse platform, that simplifies data collection/ model training/ conversion pipeline.

EDGE IMPULSE

Additionally, two of the course projects use Python with tf.keras to demonstrate the usage of industry standard open–source software.

TensorFlow Lite

The specific content includes:

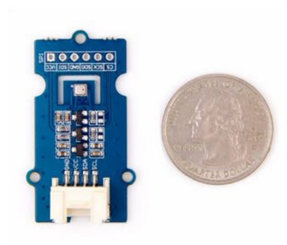
Machine Learning, TinyML overview
Recognizing gestures with light sensor
Classifying hand gestures with accelerometer
Audio scene recognition with microphone
People counting with Ultrasonic sensor
Intelligent meteostation with DHT11/pressure sensor
Student project

Course Product Requirements

Hardware requirements (boards, modules):

• Wio Terminal x1

• Grove – Ultrasonic Sensor x1


• Grove – Temp&Humi&Barometer Sensor (BME280)

• Grove cables x4

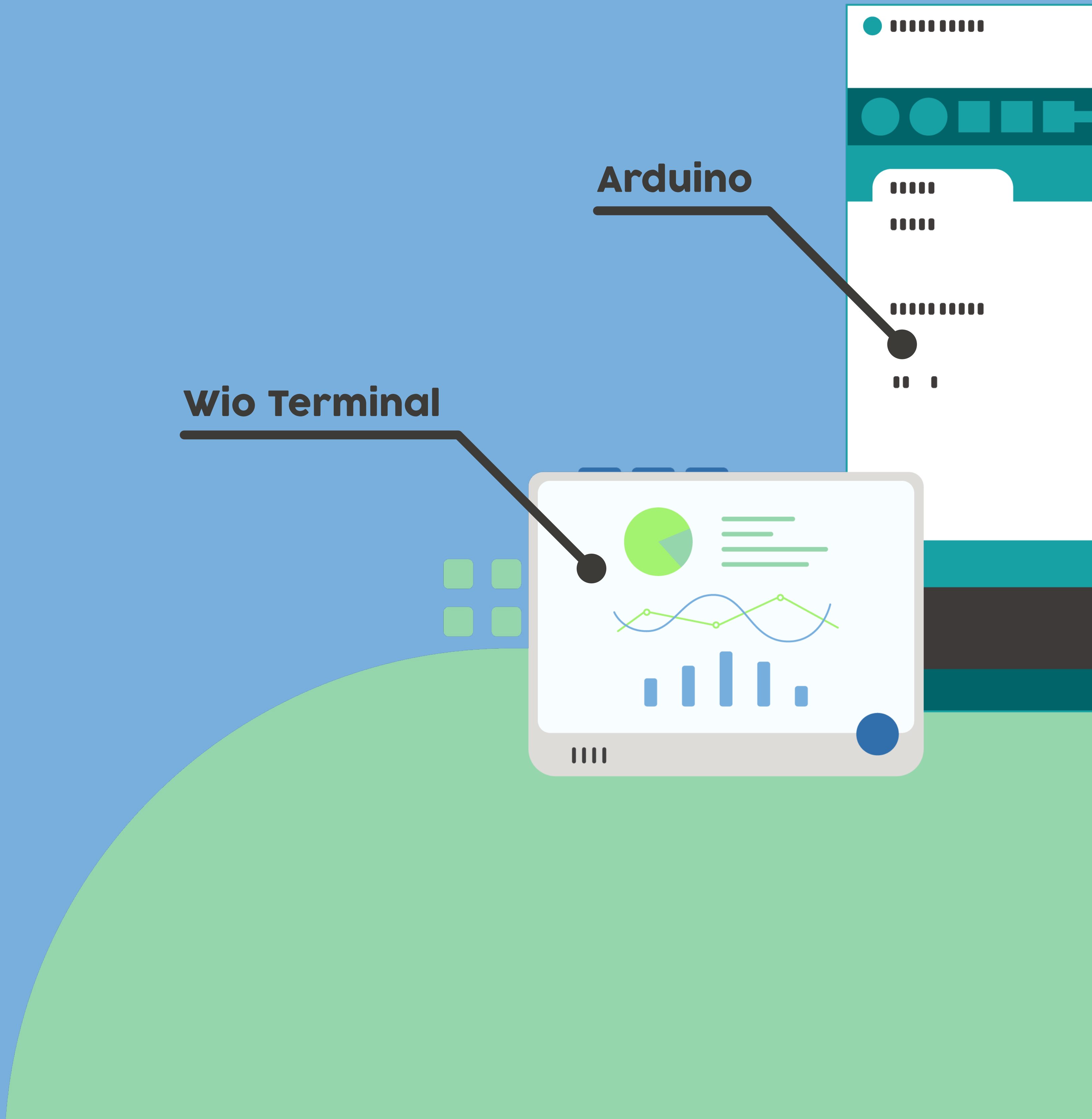
Software requirements (possible platform or language): Arduino IDE, Edge Impulse

Curriculum outline

Lesson 01 Introduction to TinyML with Wio Terminal
Lesson 02 Project I: Recognizing gestures with light sensor: theory and data collection
Lesson 03 Project I: Recognizing gestures with light sensor: model training and deployment
Lesson 04 Project II: Classifying hand gestures with accelerometer: theory and data collection
Lesson 05 Project II: Classifying hand gestures with accelerometer: model training and deployment
Lesson 06 Project III: Audio scene recognition with microphone: theory and data collection
Lesson 07 Project III: Audio scene recognition with microphone: model training and deployment
Lesson 08 Project IV: People counting with Ultrasonic sensor: theory and data collection
Lesson 09 Project IV: People counting with Ultrasonic sensor: model training and deployment
Lesson 10 Project V: Intelligent meteostation with BME280: theory and data collection
Lesson 11 Project V: Intelligent meteostation with BME280: model training and deployment (tf.keras)
Lesson 12 Student project

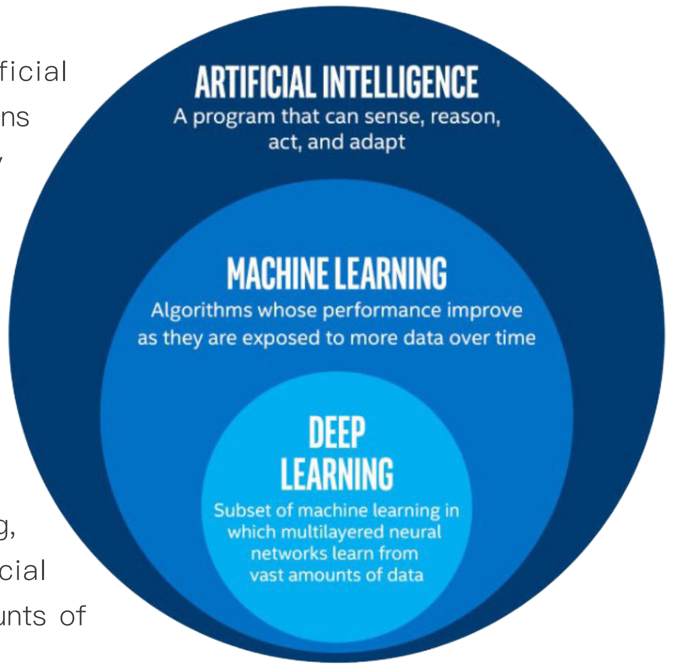
Lesson 01

Introduction to TinyML with Wio Terminal



Theory

Machine Learning and Deep Learning


ARTIFICIAL INTELLIGENCE
A program that can sense, reason, act, and adapt
MACHINE LEARNING
Algorithms whose performance improve as they are exposed to more data over time
DEEP LEARNING
Subset of machine learning in which multilayered neural networks learn from vast amounts of data

Machine learning is a branch of artificial intelligence (AI) focused on building applications that learn from data and improve their accuracy over time without being programmed to do so. The foundation of machine learning is that rather than have to be taught to do everything step by step, machines, if they can be programmed to think like us, can learn to work by observing, classifying and learning from its mistakes, just like we do. A Deep Learning is a subset of machine learning, that utilizes Deep (hence the name) artificial neural networks for learning from large amounts of data.

An artificial neural network is an attempt to simulate the network of neurons that make up a human brain. ANNs are created by programming regular computers to behave as though they are interconnected brain cells.

In order for ANNs to learn, they need to have a tremendous amount of information thrown at them called a training set. When you are trying to teach an ANN how to differentiate a cat from dog, the training set would provide thousands of images tagged as a dog so the network would begin to learn. Once it has been trained with the significant amount of data, it will try to classify future data based on what it thinks it's seeing (or hearing, depending on the data set) throughout the different units. During the training period, the machine's output is compared to the human-provided description of what should be observed. If they are the same, the machine is validated. If it's incorrect, it uses back propagation to adjust its learning—going back through the layers to tweak the mathematical equation. Known as deep learning, this is what makes a network intelligent.

Normally Deep Neural Networks require rather powerful compute resources to be trained and deployed. However recently, a branch of ML on the Edge or Embedded Machine Learning called TinyMl have appeared – it represents a technique or field of study in machine learning and embedded systems that explores which machine-learning applications (once reduced, optimized and integrated) can be run on devices as small as microcontrollers.

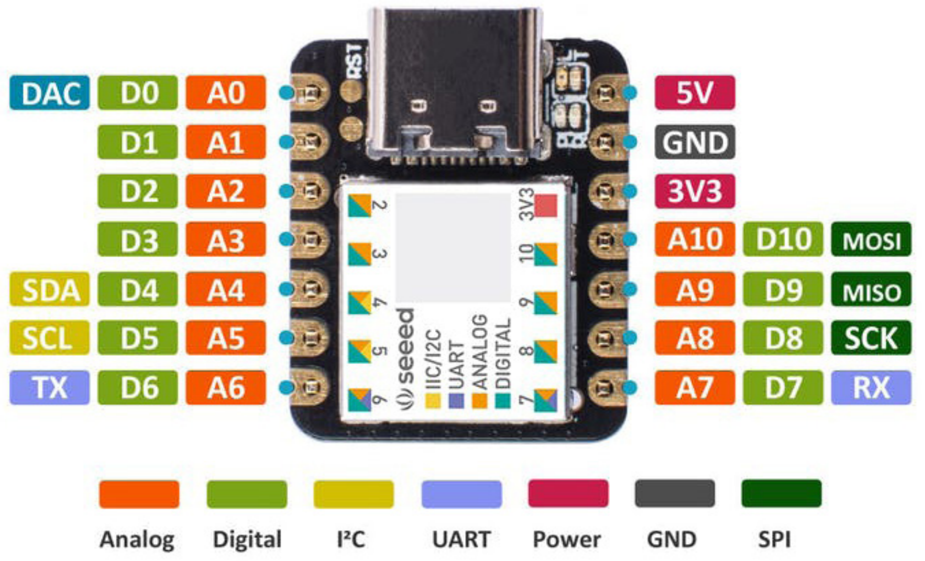
What is TinyMl and why is TinyML important?

In TinyML, ML as you might have guessed stands for Machine Learning and in most of cases (not always though) nowadays refers to Deep Learning. Tiny in TinyML means that the ML models are optimized to run on very low-power and small footprint devices, such as various MCUs.

Embedded devices come in all sorts of shapes and sizes, starting from “embedded supercomputer” Nvidia Jetson Xavier AGX to the tiniest of microcontrollers, for example ESP32 or Cortex M0.

Why embedded ML on microcontrollers is put in a special category and even given its own cool name?

Because it comes with it's own set of advantages and limitations. The attraction of TinyML is in fact that MCUs are ubiquitous, small, consume small amounts of energy and comparatively cheap. Take ARM Cortex M0+ and the little Seeeduino XIAO board which is built around it — the board is as small as a thumb(20×17.5mm), consumes only 1.33 mAh of power (which means it can work ~112 hours on a 150 mA battery, much more if put in deep sleep) and cost as little as 4.3 USD.



Thanks to recent improvements in model optimization and emergence of frameworks specifically created for running machine learning model inference on microcontrollers, it has became possible to give more intelligence to these tiny devices. We now can deploy neural networks on microcontrollers for **audio scene recognition** (for example elephant activity or sound of breaking glass), **hot-word detection**(to activate device with a specific phrase) or even for simple **image recognition** tasks. The devices with embedded microcontrollers can be used to give new life and meaning to old sensors, such as using an accelerometer installed on a mechanism for anomaly detection and predictive maintenance — or to distinguish various kinds of liqueurs as in this demo! **The possibilities of TinyML are truly huge.**

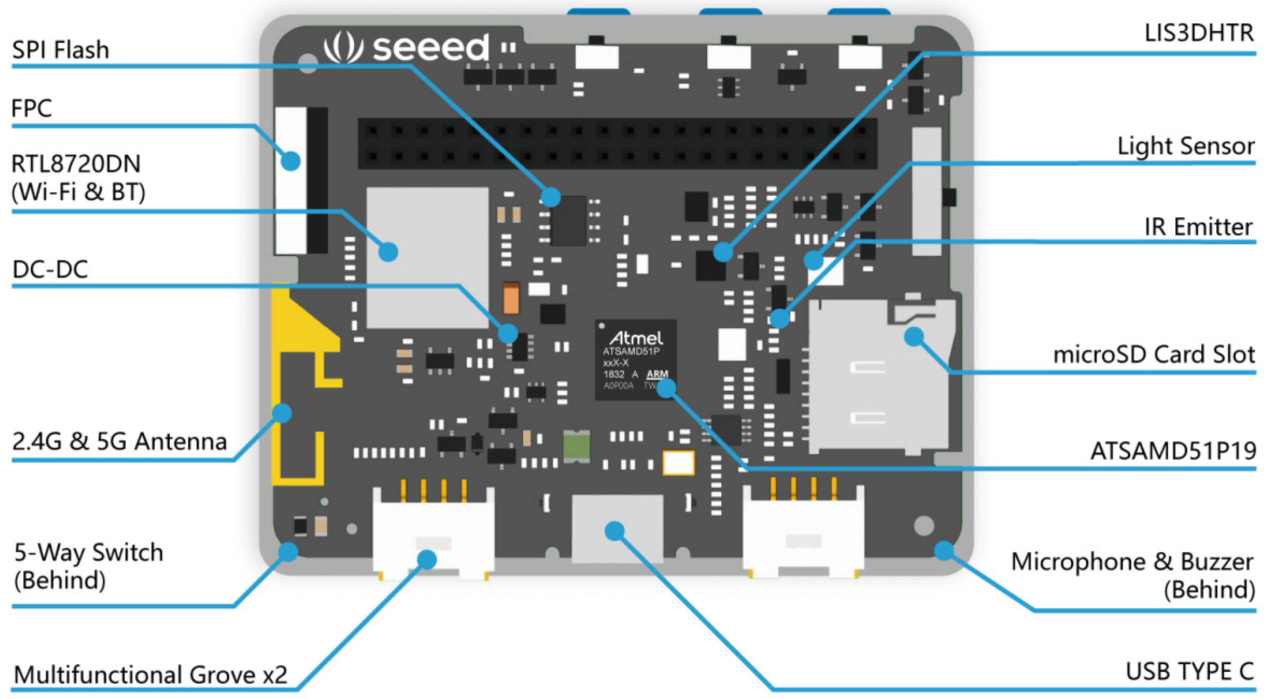
What about limitations? The main limiting factor is RAM/FLASH size of MCUs — no matter how you well optimize, you wouldn't be able to fit that YOLO9999 into a tiny microcontroller. Same goes for automatic speech recognition — while simple hot word (or voice command detection) is possible, open domain speech recognition is out of reach of MCUs. **For now.**

Preparation

In this course we'll mainly be using **ARM Cortex M4F** core inside Wio Terminal development board.

Wio Terminal is a perfect tool to get started with IoT and TinyML — it is built around ATSAMD51P19 chip with ARM Cortex-M4F core running at 120MHz, which is very well supported by various frameworks for ML inference on microcontrollers.

The board also has
- built-in light sensor
- microphone
- programmable buttons
- 2.4 inch LCD display
- accelerometer
- 2 Grove ports for easy connection of more than 300 various Grove ecosystem sensors



Software-wise we will be using Arduino IDE for programming the devices and a mix of Edge Impulse and Tensorflow Lite for Microcontrollers for model training and inference. Edge Impulse is a user-friendly development platform for machine learning on edge devices, providing beginner friendly (yet powerful) web interface and toolkit for whole TinyMl pipeline, from data collection all the way to model deployment. In later lessons of the course we will also demonstrate how you can use pure Tensorflow Lite for Microcontrollers to implement your own model training and inference pipeline.

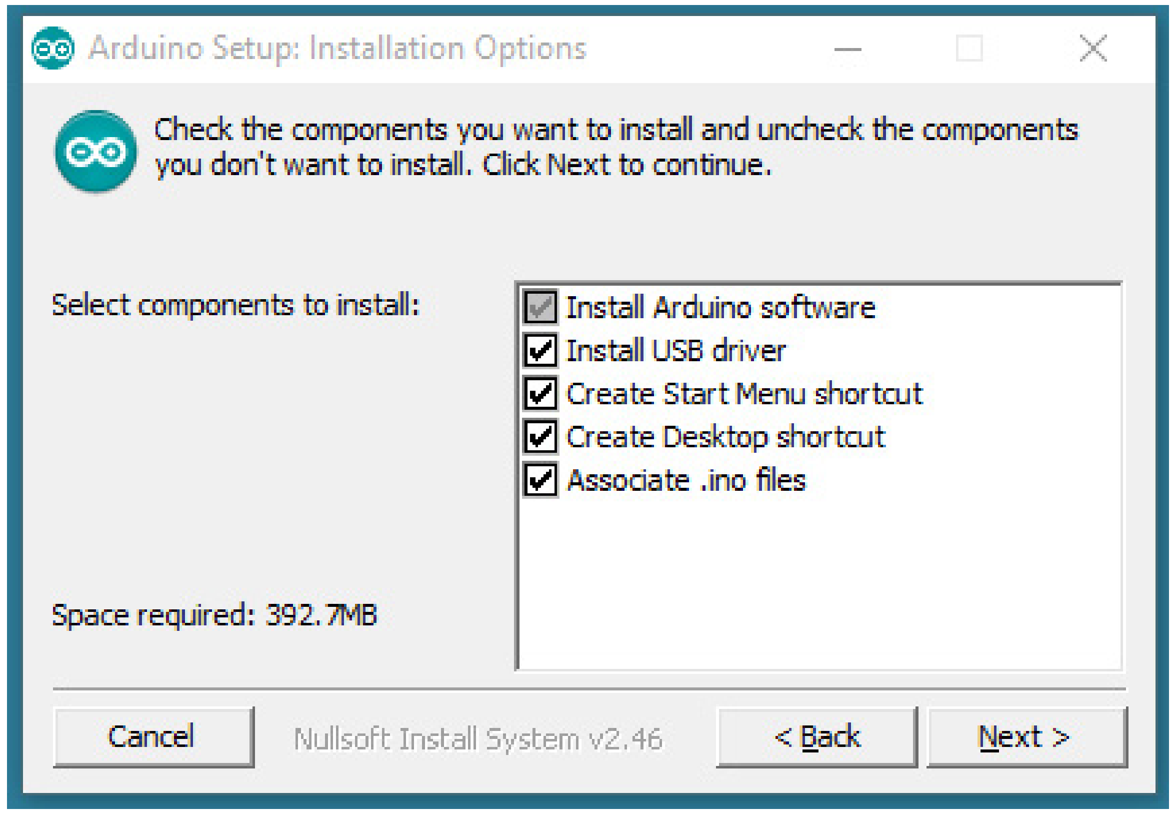
Practice

Before we set on our journey to explore the possibilities of TinyML, we will need to set up the working environment, namely Arduino IDE, Edge Impulse CLI and (optionally) Tensorflow.

Installing Arduino IDE

Get the latest version from the download page. You can choose between the Installer (.exe) and the Zip packages. We suggest you use the first one that installs directly everything you need to use the Arduino Software (IDE), including the drivers.

When the download finishes, proceed with the installation and please allow the driver installation process when you get a warning from the operating system.



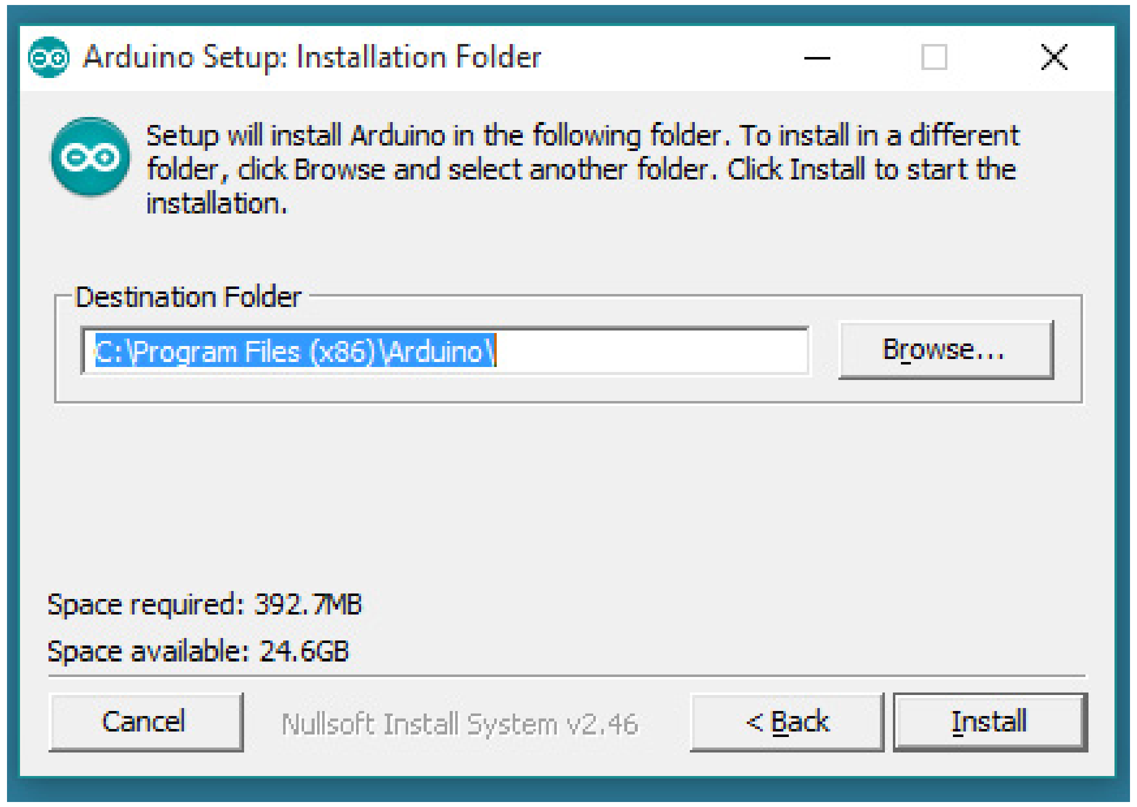
Arduino Setup: Installation Options

Check the components you want to install and uncheck the components you don't want to install. Click Next to continue.

Select components to install:
- [✓] Install Arduino software
- [✓] Install USB driver
- [✓] Create Start Menu shortcut
- [✓] Create Desktop shortcut
- [✓] Associate .ino files

Space required: 392.7MB

Choose the components to install


Arduino Setup: Installation Folder

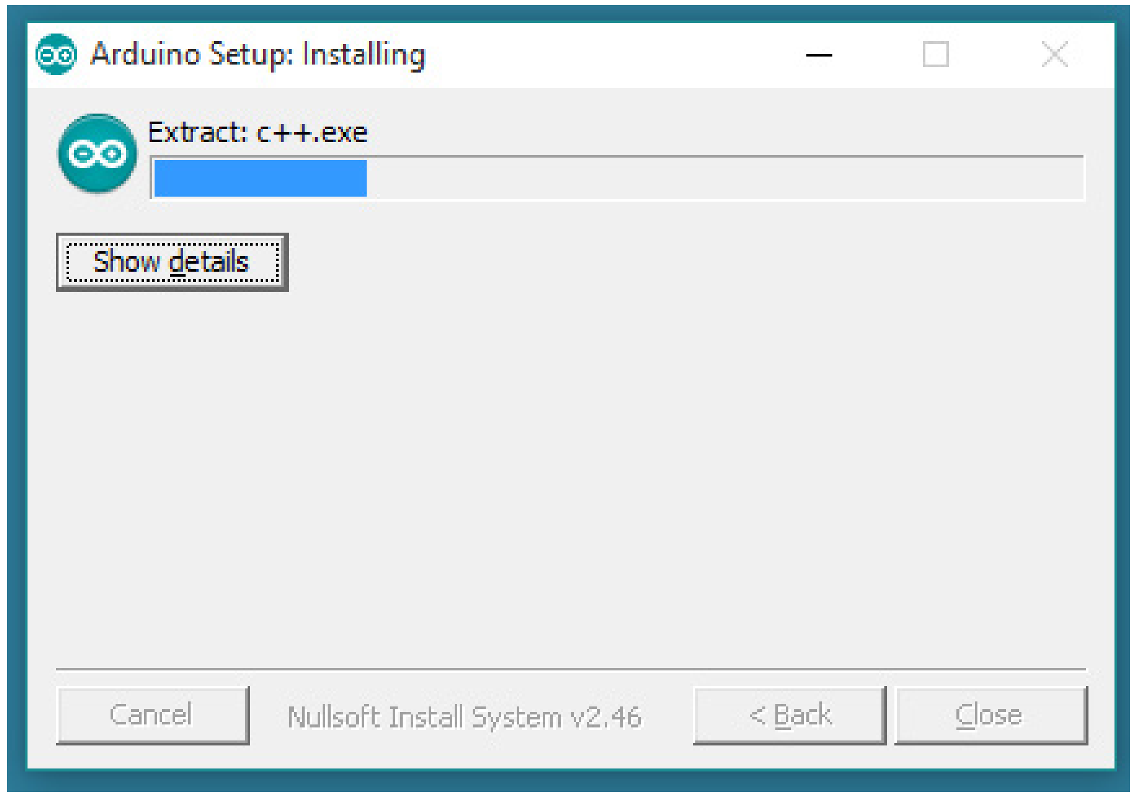
Setup will install Arduino in the following folder. To install in a different folder, click Browse and select another folder. Click Install to start the installation.

Destination Folder: C:\Program Files (x86)\Arduino\

Space required: 392.7MB

Space available: 24.6GB

Choose the installation directory (we suggest to keep the default one)


Arduino Setup: Installing

Extract: c++.exe

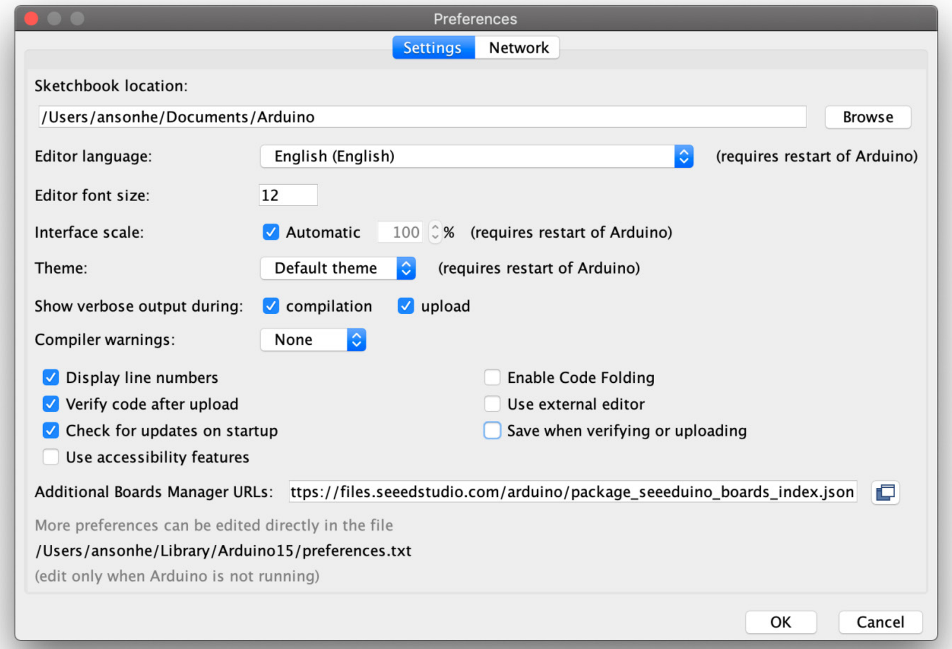
The process will extract and install all the required files to execute properly the Arduino Software (IDE).

To compile and upload code for Wio Terminal you will need to install Wio Terminal–specific tools in Arduino IDE:

Step 1. Add Additional Boards Manager URLs

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

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


Preferences

Sketchbook location: /Users/ansonhe/Documents/Arduino

Editor language: English (English) (requires restart of Arduino)

Editor font size: 12

Interface scale: [✓ Automatic] 100 % (requires restart of Arduino)

Theme: Default theme (requires restart of Arduino)

Show verbose output during: [✓ compilation] [✓ upload]

Compiler warnings: None

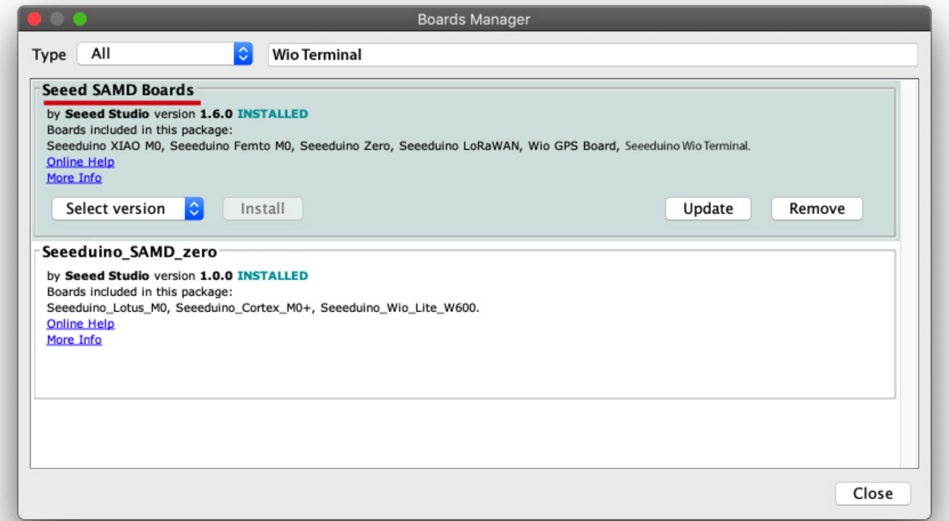
- [✓ Display line numbers]
- [✓ Verify code after upload]
- [✓ Check for updates on startup]
- Use accessibility features
- Enable Code Folding
- Use external editor
- Save when verifying or uploading

Additional Boards Manager URLs: ttps://files.seeedstudio.com/arduino/package_seeeduino_boards_index.json

More preferences can be edited directly in the file
/Users/ansonhe/Library/Arduino15/preferences.txt
(edit only when Arduino is not running)

Step 2. Install Wio Terminal Tools

Click on **Tools** > **Board** > **Board Manager** and Search **Wio Terminal** in the Boards Manager. Press on Install and wait until installation process finishes.


Boards Manager

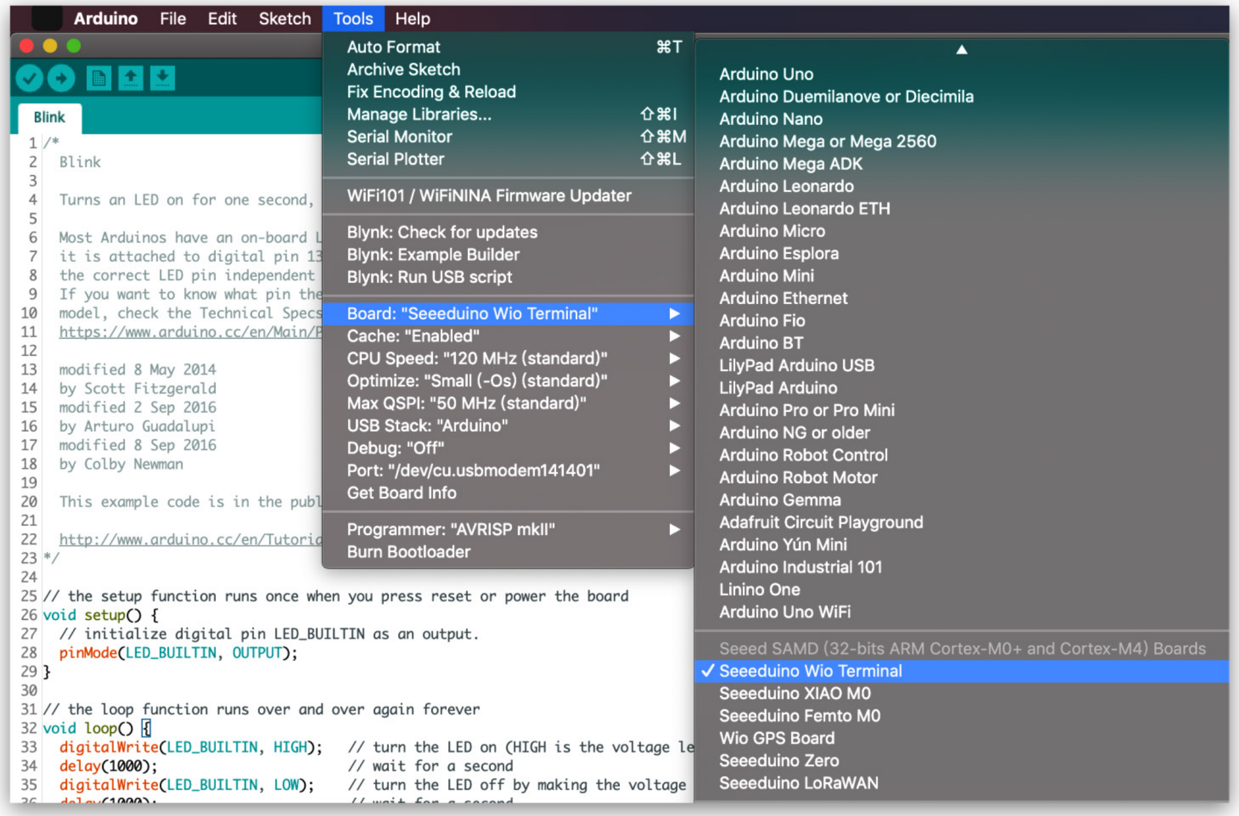
Type: All — Wio Terminal

Seeed SAMD Boards
by **Seeed Studio** version **1.6.0** **INSTALLED**
Boards included in this package:
Seeeduino XIAO M0, Seeeduino Femto M0, Seeeduino Zero, Seeeduino LoRaWAN, Wio GPS Board, Seeeduino Wio Terminal.
Online Help
More Info

Seeeduino_SAMD_zero
by **Seeed Studio** version **1.0.0** **INSTALLED**
Boards included in this package:
Seeeduino_Lotus_M0, Seeeduino_Cortex_M0+, Seeeduino_Wio_Lite_W600.
Online Help
More Info

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**.



```
/*
  Blink

  Turns an LED on for one second,

  Most Arduinos have an on-board L
  it is attached to digital pin 13
  the correct LED pin independent
  If you want to know what pin the
  model, check the Technical Specs
  https://www.arduino.cc/en/Main/P

  modified 8 May 2014
  by Scott Fitzgerald
  modified 2 Sep 2016
  by Arturo Guadalupi
  modified 8 Sep 2016
  by Colby Newman

  This example code is in the publ

  http://www.arduino.cc/en/Tutoria
*/

// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage le
  delay(1000);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage
```

Installing Edge Impulse CLI

This Edge Impulse CLI is used to control local devices, act as a proxy to synchronize data for devices that don't have an internet connection, and to upload and convert local files.

1. Install Node.js v10 or higher on your host computer.

For Windows users, install the Additional Node.js tools when prompted. You may skip this setup if you have Visual Studio 2015 or more.

2. Install the CLI tools via:

```
npm install -g edge-impulse-cli
```

Afterwards you should have the tools available in your PATH.

(Optional) Installing Tensorflow

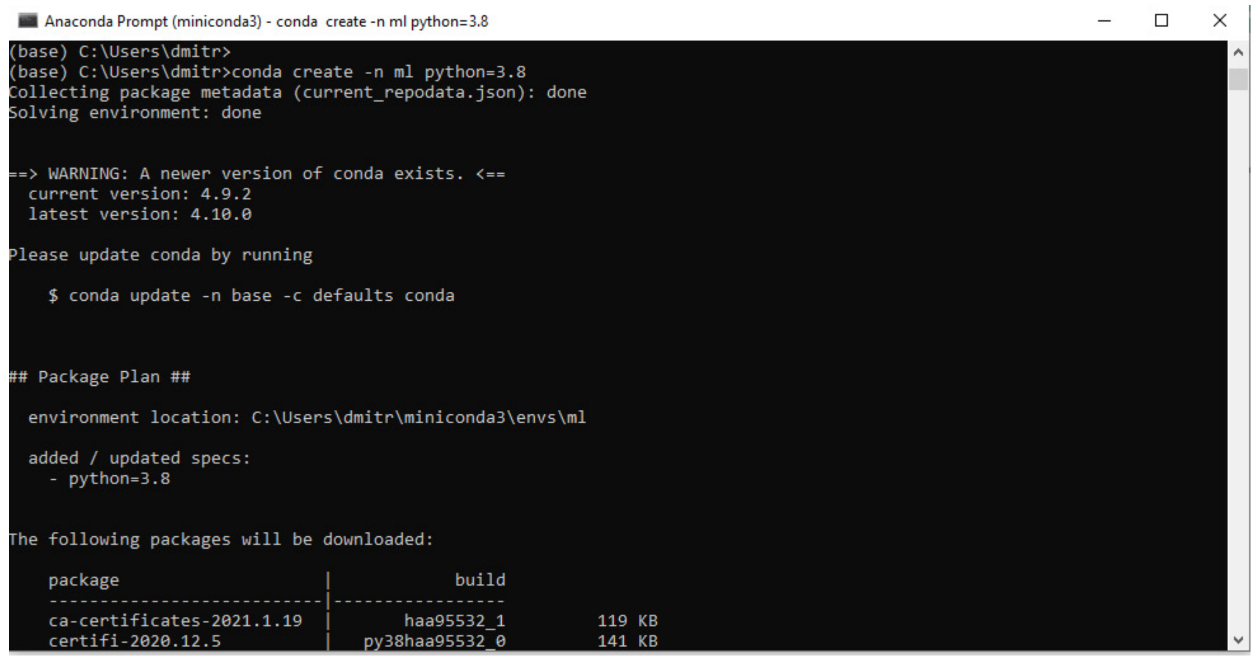
We will use Python and Tensorflow for more in-depth lessons later in the course. Local installation of Tensorflow is optional though, if you can get access to Google Colab, an online environment where you can execute Python code and train your models.

To install tensorflow on Windows:

1. Install Miniconda (Python virtual environment manager), which you can download from the official website.

2. Create a new virtual environment with

```
conda create -n ml python=3.8
```


Anaconda Prompt (miniconda3) - conda create -n ml python=3.8

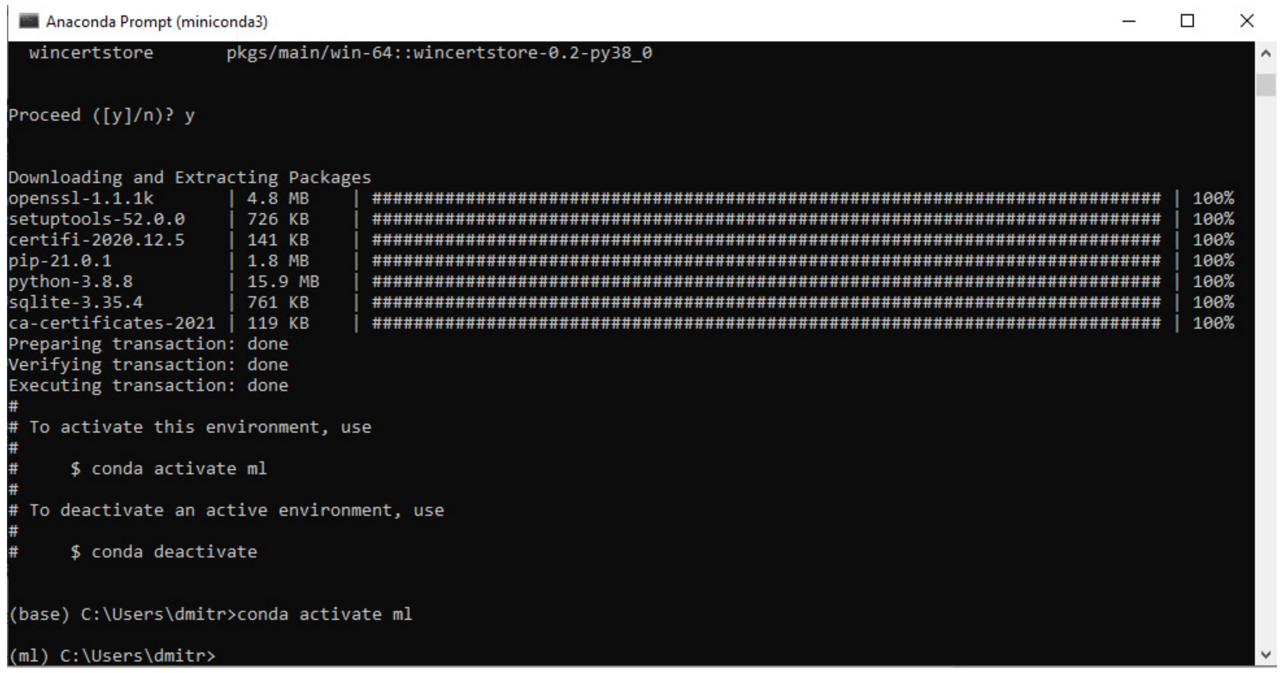
```
(base) C:\Users\dmitr>
(base) C:\Users\dmitr>conda create -n ml python=3.8
Collecting package metadata (current_repodata.json): done
Solving environment: done


==> WARNING: A newer version of conda exists. <==
  current version: 4.9.2
  latest version: 4.10.0

Please update conda by running

    $ conda update -n base -c defaults conda



## Package Plan ##

  environment location: C:\Users\dmitr\miniconda3\envs\ml

  added / updated specs:
    - python=3.8


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    ca-certificates-2021.1.19  |       haa95532_1         119 KB
    certifi-2020.12.5          |   py38haa95532_0         141 KB
```

3. Install Tensorflow in pip in that environment

```
pip install tensorflow
```

4. Next time you need to use TensorFlow, simply open Miniconda prompt and activate the virtual environment you created with

```
conda activate ml
```



```
  wincertstore       pkgs/main/win-64::wincertstore-0.2-py38_0


Proceed ([y]/n)? y


Downloading and Extracting Packages
openssl-1.1.1k       | 4.8 MB    | ############################################################################ | 100%
setuptools-52.0.0    | 726 KB    | ############################################################################ | 100%
certifi-2020.12.5    | 141 KB    | ############################################################################ | 100%
pip-21.0.1           | 1.8 MB    | ############################################################################ | 100%
python-3.8.8         | 15.9 MB   | ############################################################################ | 100%
sqlite-3.35.4        | 761 KB    | ############################################################################ | 100%
ca-certificates-2021 | 119 KB    | ############################################################################ | 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
#
# To activate this environment, use
#
#     $ conda activate ml
#
# To deactivate an active environment, use
#
#     $ conda deactivate


(base) C:\Users\dmitr>conda activate ml

(ml) C:\Users\dmitr>
```

Expansion tasks

Explore ready-to-use demos of Edge Impulse and Tensorflow by uploading sample codes

1) Magic wand with TF

2) Voice command detection with EI

Lesson 02

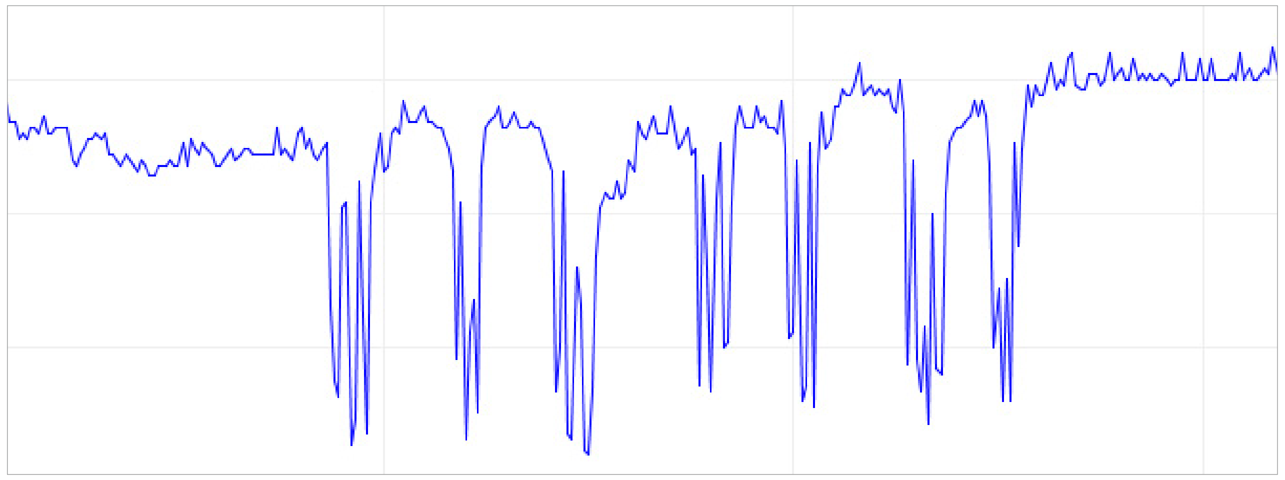
Project I: Recognizing gestures with light sensor: theory and data collection

Theory

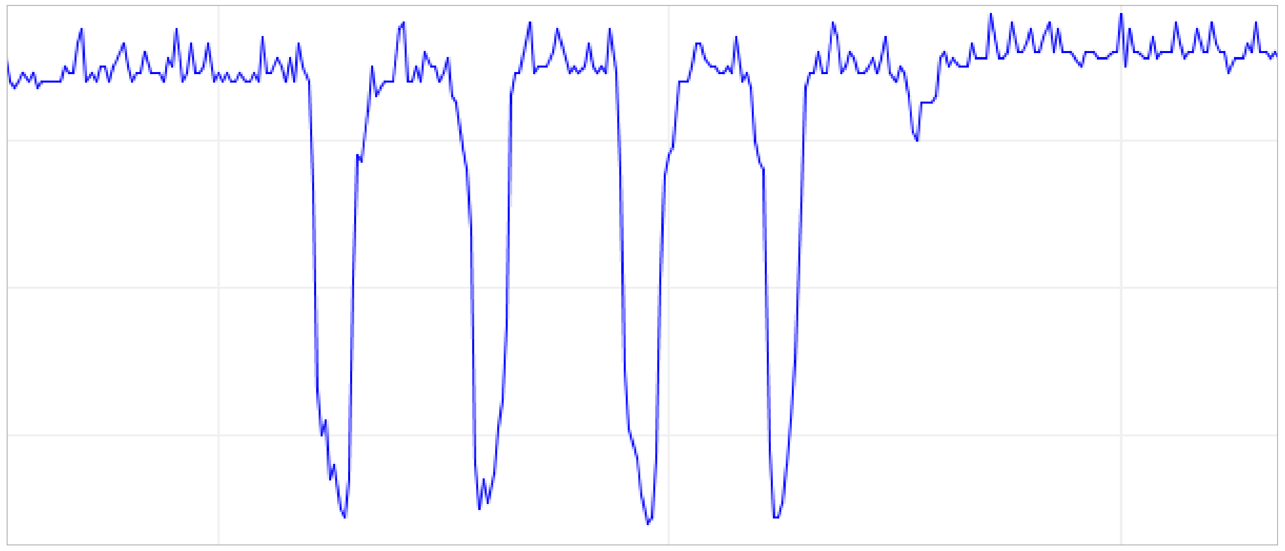
In this lesson we are going to train and deploy a simple neural network for classifying rock–paper–scissors gestures with just a **single light sensor**.

If you think about it, the working principle of this project is actually quite trivial – different gestures being moved above the light sensor will block certain amount of light for certain periods of time. For example, for rock we will have high values first (nothing above sensor), then low values for the time "rock" passes above the sensor and then high values again. For paper we will have high–low–high–low–high–low–high–low values, when each of the fingers in "paper" passes above the sensor.

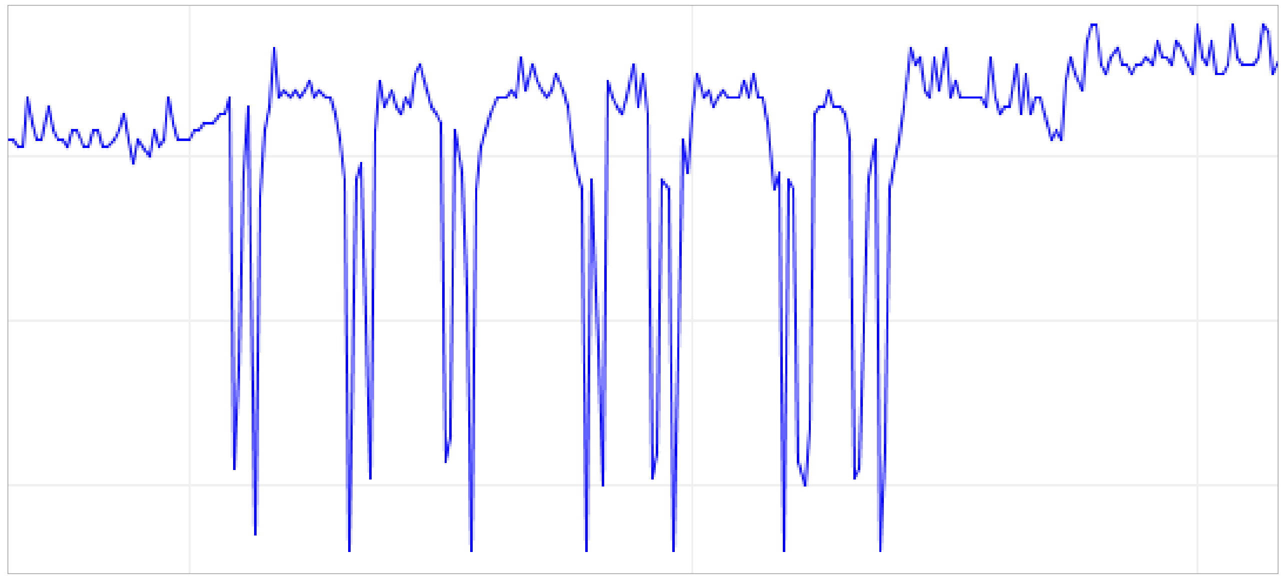
Paper



Rock



Scissors



There is high variance in speed and amplitude of the values from sensor, which makes a great case for using machine learning model and not hand–crafted algorithm for the task.

Preparation

Make sure you have Arduino IDE installed and working with Wio Terminal. Create an account on Edge Impulse website and make a new project there.

Practice

You can forward any type of sensor data to Edge Impulse platform with data forwarder — here is a pseudo code for forwarding arbitrary sensor data from device:

```
#define FREQUENCY_HZ        50
#define INTERVAL_MS         (1000 / (FREQUENCY_HZ))

void setup() {
    Serial.begin(115200);
    initialize_sensor();
    Serial.println("Started");
    }
}
void loop() {
    static unsigned long last_interval_ms = 0;
```

```
float sensor_data_1, sensor_data_2, sensordata_3;
if (millis() > last_interval_ms + INTERVAL_MS) {
    last_interval_ms = millis();
    sensor.readdata(sensor_data_1, sensor_data_2, sensor_data_);

    Serial.print(sensor_data_1);
    Serial.print('\t');
    Serial.print(sensor_data_2);
    Serial.print('\t');
    Serial.print(sensor_data_3);
    Serial.print("\n");
}
}
```

If you have multiple sensor values in one packet, each sensor value should be separated with **a comma or tab character**.

The end of the packet is denoted by a **new line character**, so you can just use `Serial.println` for last value in a packet or send the newline character separately as in above example `Serial.print("\n")`.

For example, this is data from a 3-axis accelerometer:

```
-0.12,-6.20,7.90
-0.13,-6.19,7.91
-0.14,-6.20,7.92
-0.13,-6.20,7.90
-0.14,-6.20,7.91
```

In this project we just have one sensor that sends one value in each packet.

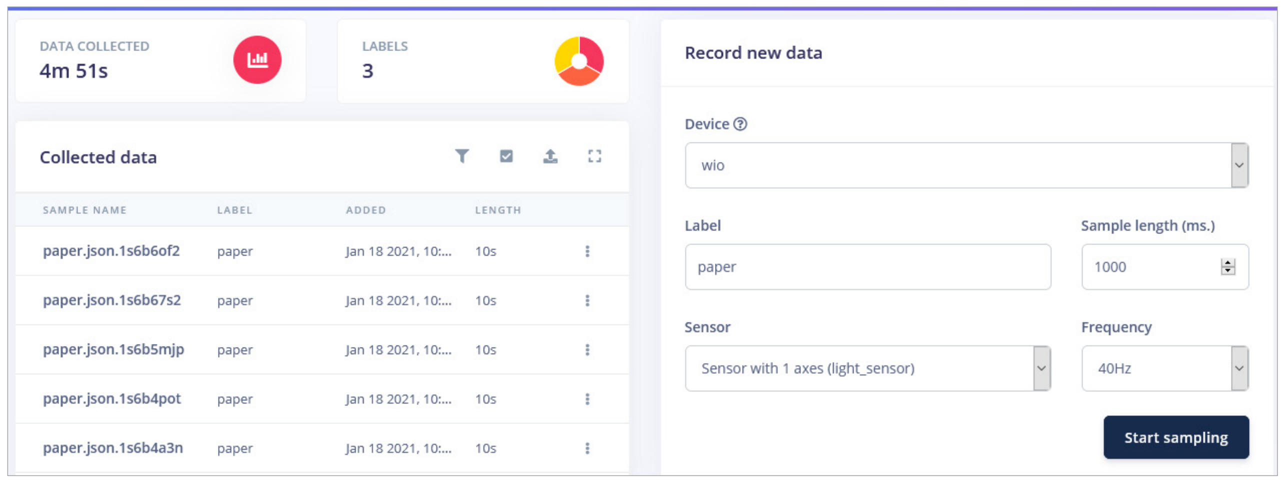
```
#define FREQUENCY_HZ        40
#define INTERVAL_MS         (1000 / (FREQUENCY_HZ))

void setup() {
    Serial.begin(115200);
    Serial.println("Started");
    }
void loop() {
    static unsigned long last_interval_ms = 0;
    float light;
    if (millis() > last_interval_ms + INTERVAL_MS) {
        last_interval_ms = millis();
        light = analogRead(WIO_LIGHT);
        Serial.println(light);
        //Serial.print('\t');
    }
}
```

Once you uploaded the code to Wio Terminal, run

`edge-impulse-data-forwarder`

in Command prompt and log in with your Edge Impulse credentials.

Now you are ready to receive data in Edge Impulse dashboard. Go to Data acquisition tab and you should see your device there.



DATA COLLECTED
4m 51s

LABELS
3

Collected data

SAMPLE NAME	LABEL	ADDED	LENGTH
paper.json.1s6b6of2	paper	Jan 18 2021, 10:...	10s
paper.json.1s6b67s2	paper	Jan 18 2021, 10:...	10s
paper.json.1s6b5mjp	paper	Jan 18 2021, 10:...	10s
paper.json.1s6b4pot	paper	Jan 18 2021, 10:...	10s
paper.json.1s6b4a3n	paper	Jan 18 2021, 10:...	10s

Record new data

Device: wio

Label: paper

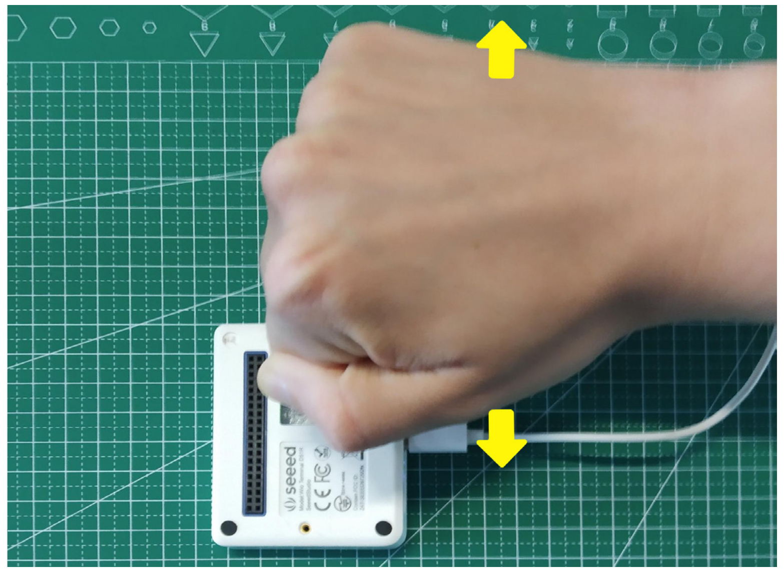
Sample length (ms.): 1000

Sensor: Sensor with 1 axes (light_sensor)

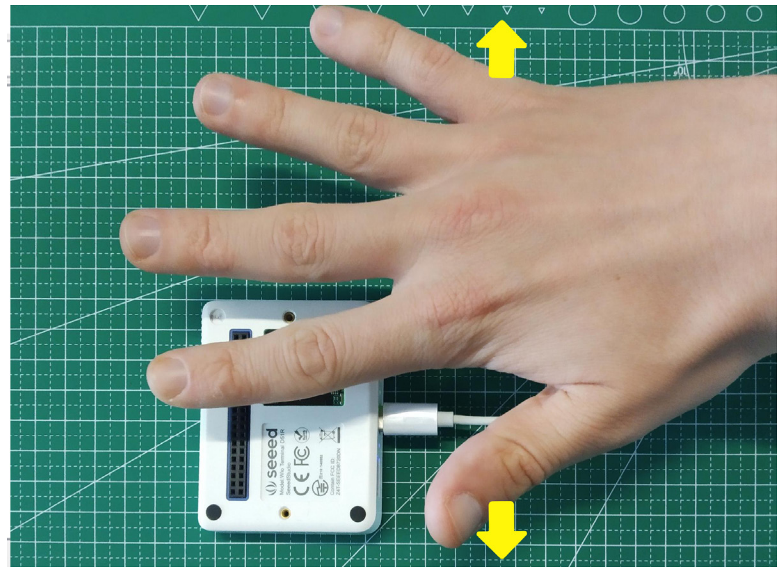
Frequency: 40Hz

Set sample length to 10000 ms or 10 seconds and create 10 samples for each gesture, waving the hand in vicinity of Wio terminal.

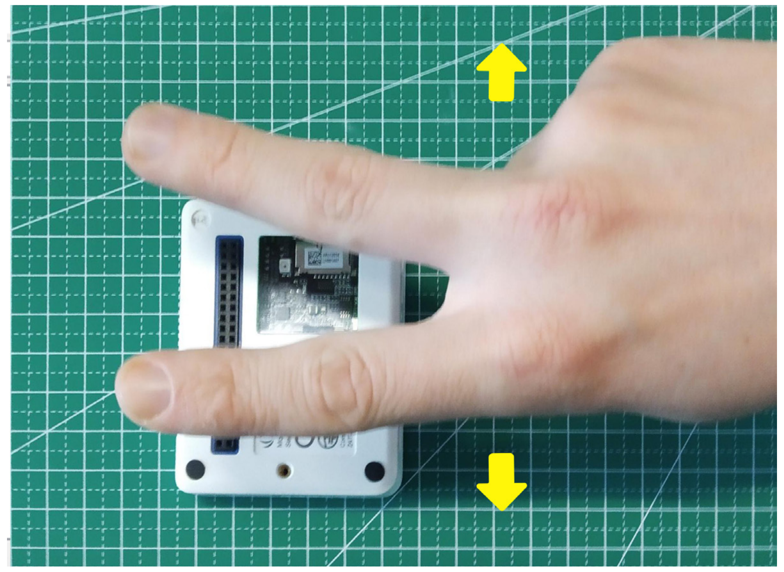
Rock



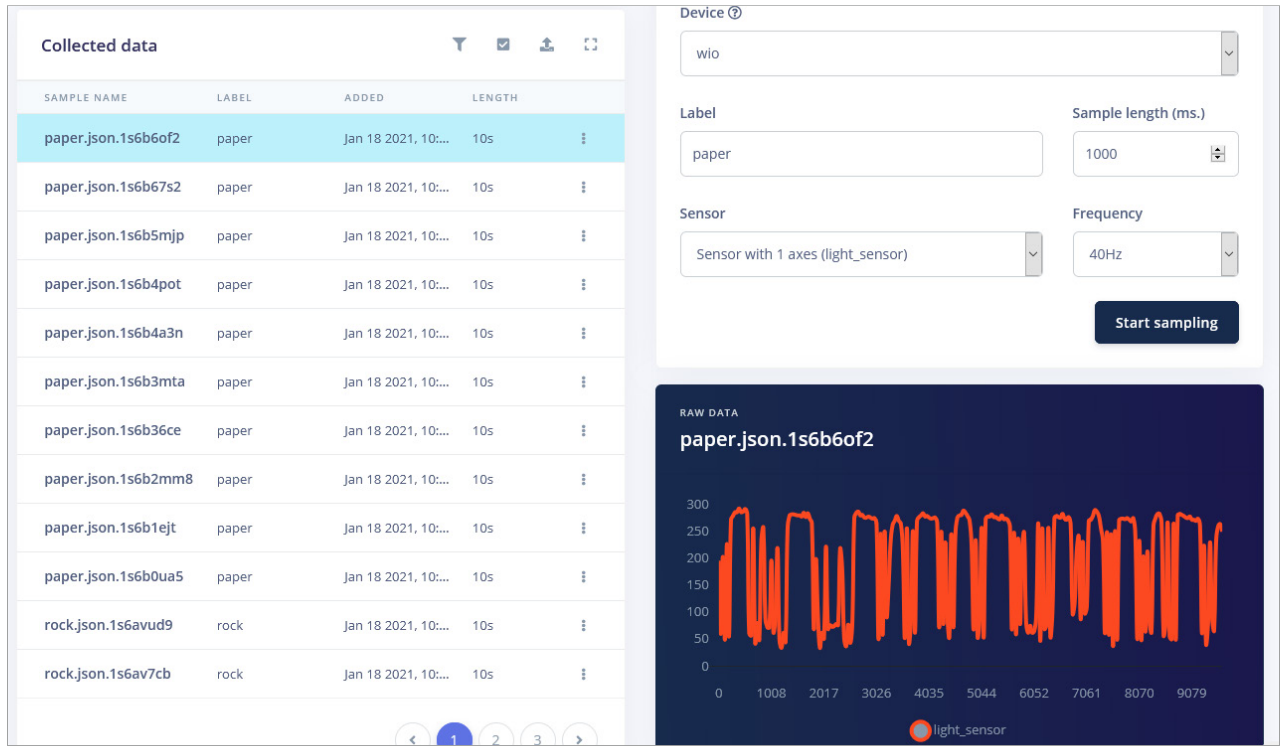
Paper



Scissors



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



Collected data

SAMPLE NAME	LABEL	ADDED	LENGTH
paper.json.1s6b6of2	paper	Jan 18 2021, 10:...	10s
paper.json.1s6b67s2	paper	Jan 18 2021, 10:...	10s
paper.json.1s6b5mjp	paper	Jan 18 2021, 10:...	10s
paper.json.1s6b4pot	paper	Jan 18 2021, 10:...	10s
paper.json.1s6b4a3n	paper	Jan 18 2021, 10:...	10s
paper.json.1s6b3mta	paper	Jan 18 2021, 10:...	10s
paper.json.1s6b36ce	paper	Jan 18 2021, 10:...	10s
paper.json.1s6b2mm8	paper	Jan 18 2021, 10:...	10s
paper.json.1s6b1ejt	paper	Jan 18 2021, 10:...	10s
paper.json.1s6b0ua5	paper	Jan 18 2021, 10:...	10s
rock.json.1s6avud9	rock	Jan 18 2021, 10:...	10s
rock.json.1s6av7cb	rock	Jan 18 2021, 10:...	10s

Device: wio
Label: paper
Sample length (ms.): 1000
Sensor: Sensor with 1 axes (light_sensor)
Frequency: 40Hz

RAW DATA
paper.json.1s6b6of2

This is a small dataset, but we also have a tiny neural network, so underfitting is more likely than overfitting in this particular case.

> **Underfitting:**
> A statistical model or a machine learning algorithm is said to have underfitting when it cannot capture the underlying trend of the data, that happens (among other cases) when model size is too small to develop a general rule for data that has large variety and amount of noise.
> **Overfitting:**
> A statistical model is said to be overfitted, when it starts learning from the noise and inaccurate data entries in our data set. That happens when you have large model and relatively small dataset – the model can just learn "by heart" all the data points without generalizing.

When collecting samples it is important to provide diversity for model to be able to generalize better, for example have samples with different direction, speed and distance from 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 trained model to be able to recognize gestures being moved right to left or up and down.

★ Expansion tasks

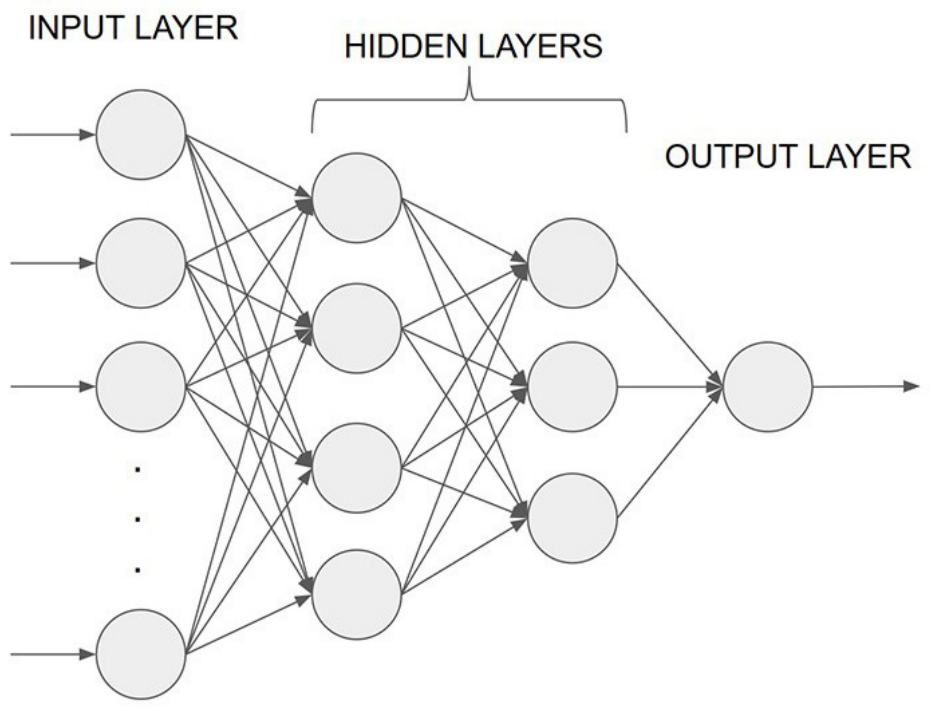
Collect more data samples in different light conditions.

Lesson 03

Project I: Recognizing gestures with light sensor: model training and deployment

Theory

In the last lesson we collected the data samples necessary to train the model. Before being fed to neural network, data needs to be preprocessed. Sometimes we just re-scale the data, for example turning values in range from 0 to 1000 into range of 0 to 1 – because neural networks work with smaller numbers better than with large ones. Neural networks used in TinyML are very small in size and number of parameters (connections between digital neurons), so oftentimes we also apply more sophisticated preprocessing techniques to extract so-called features from Raw data, which speeds up the training process.



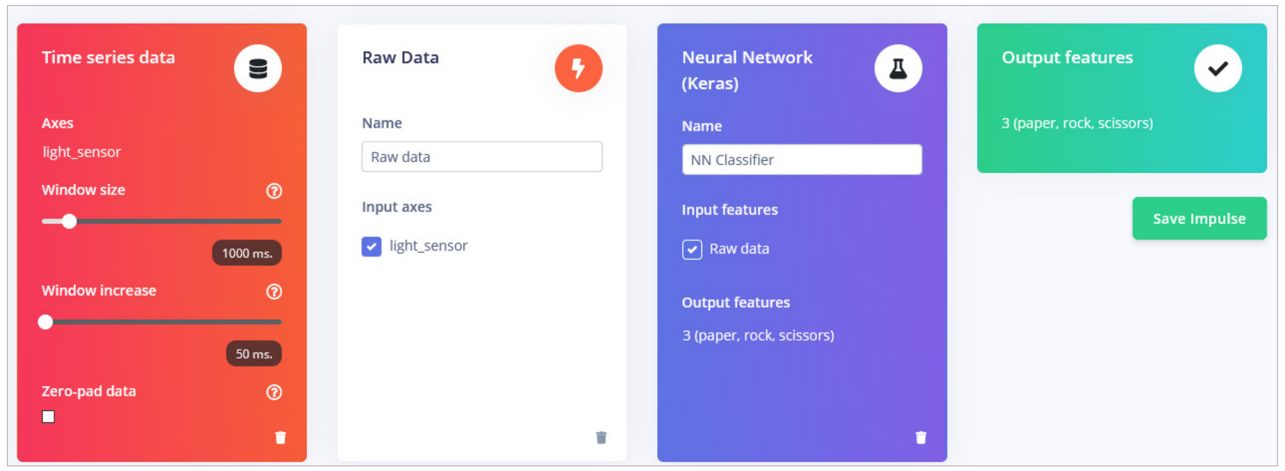
In this lesson we will learn more about preprocessing functions available in Edge Impulse and train simple Fully Connected network.

Preparation

Make sure you have Arduino IDE installed and working with Wio Terminal.

Practice

After you collected the samples it is time to design an "impulse". Impulse here is the word Edge Impulse used to denote data processing — training pipeline. Press on Create Impulse and set Window length to 1000 ms. and Window length increase to 50 ms.


Time series data
Axes
light_sensor
Window size
1000 ms.
Window increase
50 ms.
Zero-pad data

Raw Data
Name
Raw data
Input axes
✓ light_sensor

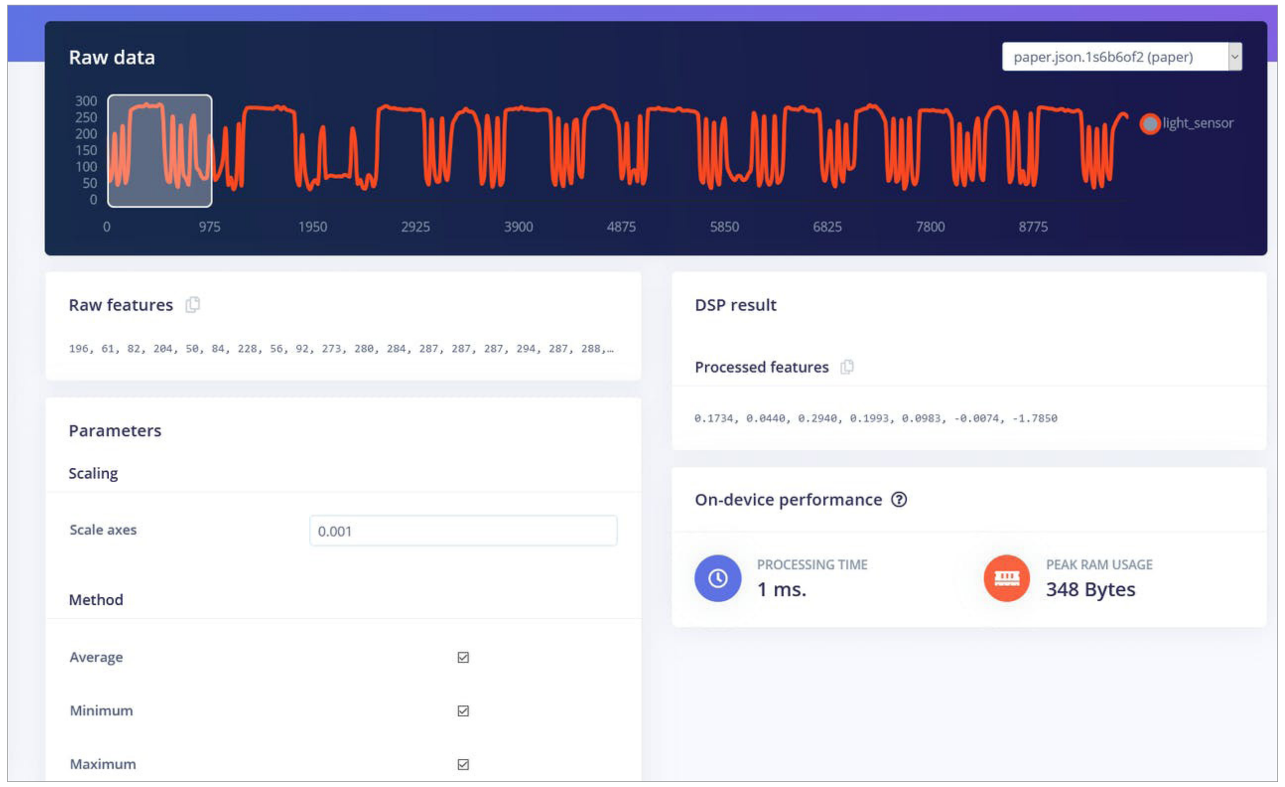
Neural Network (Keras)
Name
NN Classifier
Input features
✓ Raw data
Output features
3 (paper, rock, scissors)

Output features
3 (paper, rock, scissors)

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 you set sampling frequency to 40 Hz, or 40 times per 1 second. So, to sum it up, your device is going to gather 40 data samples within 1000 ms. time window and then take these values, preprocess them and feed them to neural network to get inference result. Of course we use the same window size during the training.

For this proof-of-concept project, we are going to try three different prepossessing blocks with default parameters(except for adding scaling) —

Flatten block, which takes computes Average, Min, Max and other functions of raw data within time window.



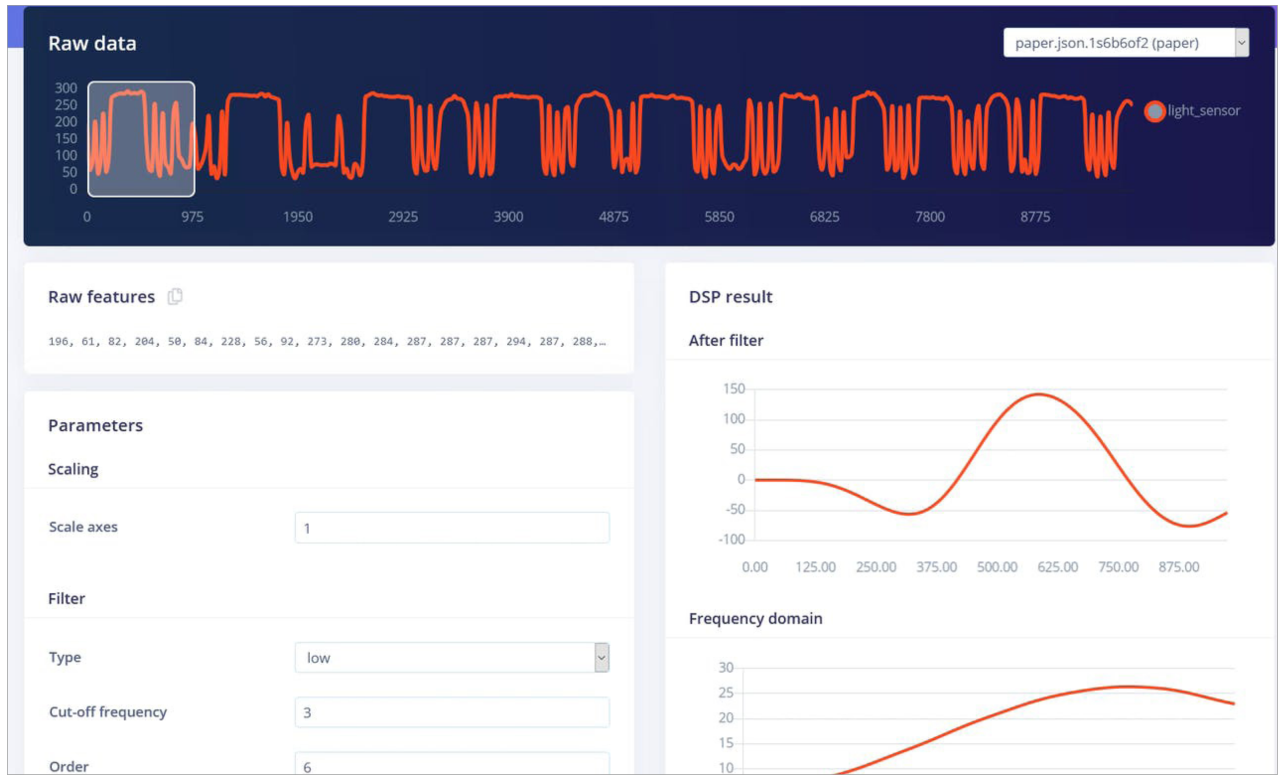
Raw features
196, 61, 82, 204, 50, 84, 228, 56, 92, 273, 280, 284, 287, 287, 287, 294, 287, 288,…

DSP result
Processed features
0.1734, 0.0440, 0.2940, 0.1993, 0.0983, -0.0074, -1.7850

Parameters
Scaling
Scale axes 0.001
Method
Average ✓
Minimum ✓
Maximum ✓

On-device performance
PROCESSING TIME 1 ms.
PEAK RAM USAGE 348 Bytes

Spectral Features block, which extracts the frequency and power characteristics of a signal over time.


Raw data

paper.json.1s6b6of2 (paper)

Raw features

196, 61, 82, 204, 50, 84, 228, 56, 92, 273, 280, 284, 287, 287, 287, 294, 287, 288,…

Parameters

Scaling

Scale axes	1

Filter

Type	low
Cut-off frequency	3
Order	6

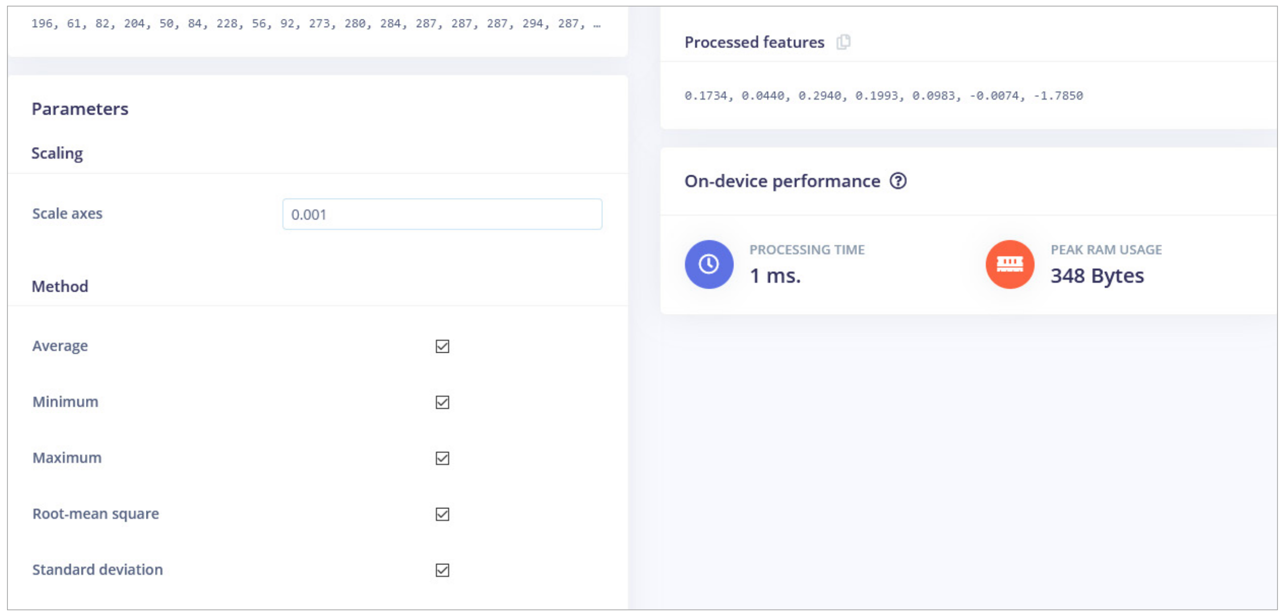
DSP result

After filter

Frequency domain

and **Raw data** block, which as you might have guessed just feeds raw data to NN learning block (optionally normalizing the data).

We'll start with Flatten block. Add this block and then add Neural Network (Keras) as learning block, check Flatten as input features and click on Save Impulse. Go to the next tab, which has a name of the processing block you have chosen – Flatten. There enter 0.001 in scaling and leave other parameters the same. Press on Save parameters and then Generate features.


196, 61, 82, 204, 50, 84, 228, 56, 92, 273, 280, 284, 287, 287, 287, 294, 287, …

Parameters

Scaling

Scale axes	0.001

Method

Average	☑
Minimum	☑
Maximum	☑
Root-mean square	☑
Standard deviation	☑

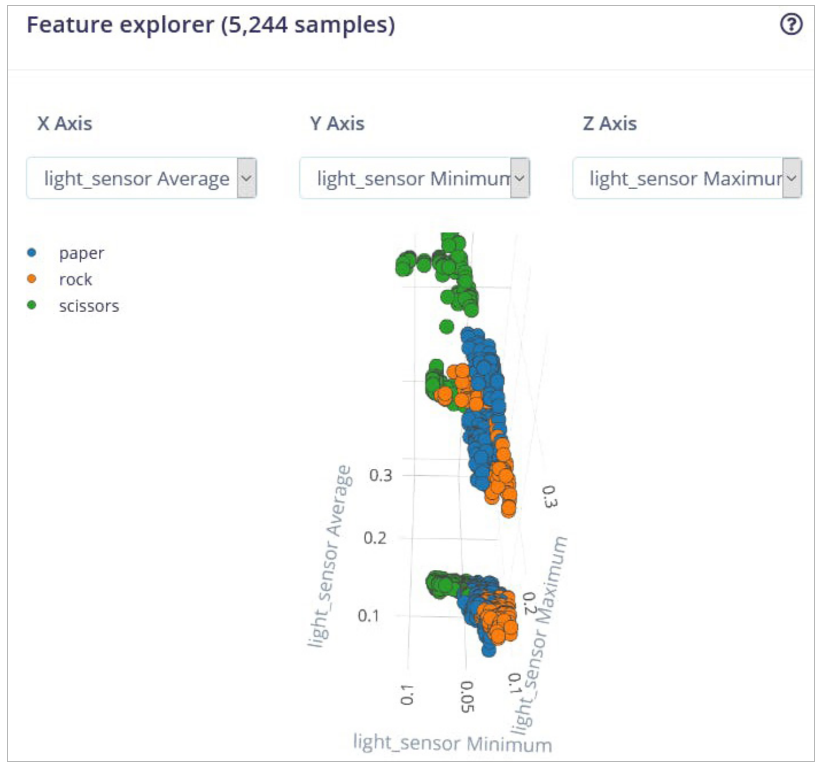
Processed features

0.1734, 0.0440, 0.2940, 0.1993, 0.0983, -0.0074, -1.7850

On-device performance

PROCESSING TIME	1 ms.
PEAK RAM USAGE	348 Bytes

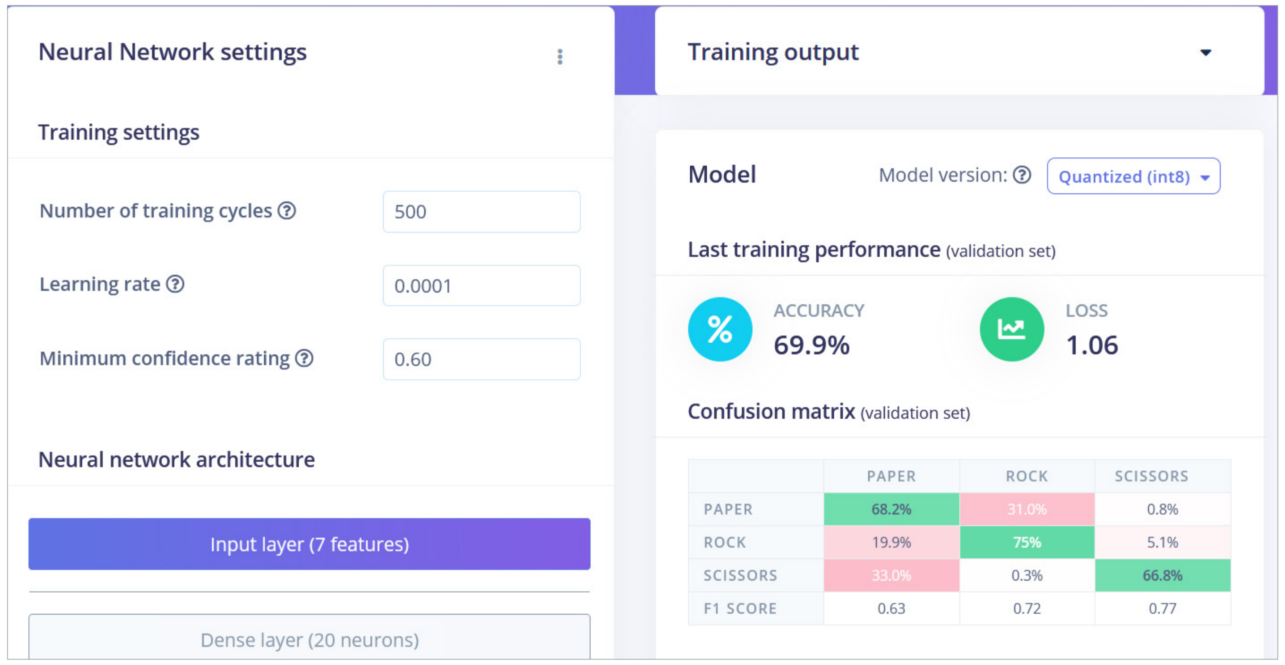
Feature visualization is particularity useful tool in Edge Impulse web interface, as it allows users to get graphical insights into how the data looks after prepossessing. For example this is data after Flatten processing block:


Feature explorer (5,244 samples)

X Axis	Y Axis	Z Axis
light_sensor Average	light_sensor Minimum	light_sensor Maximum

We can see that the data points for different classes are roughly divided, but there is a lot of overlap between rock and other classes, which will cause issues and low accuracy for these two classes. After you generated and inspected the features, go to NN CLassifier tab.

Train a simple fully–connected network with 2 hidden layers, 20 and 10 neurons in each hidden layer for 500 epochs with 1e–4 learning rate. After the training is done you're going to see test results in confusion matrix, similar to this:


Neural Network settings

Training settings

Number of training cycles	500
Learning rate	0.0001
Minimum confidence rating	0.60

Neural network architecture

Input layer (7 features)

Dense layer (20 neurons)

Training output

Model — Model version: Quantized (int8)

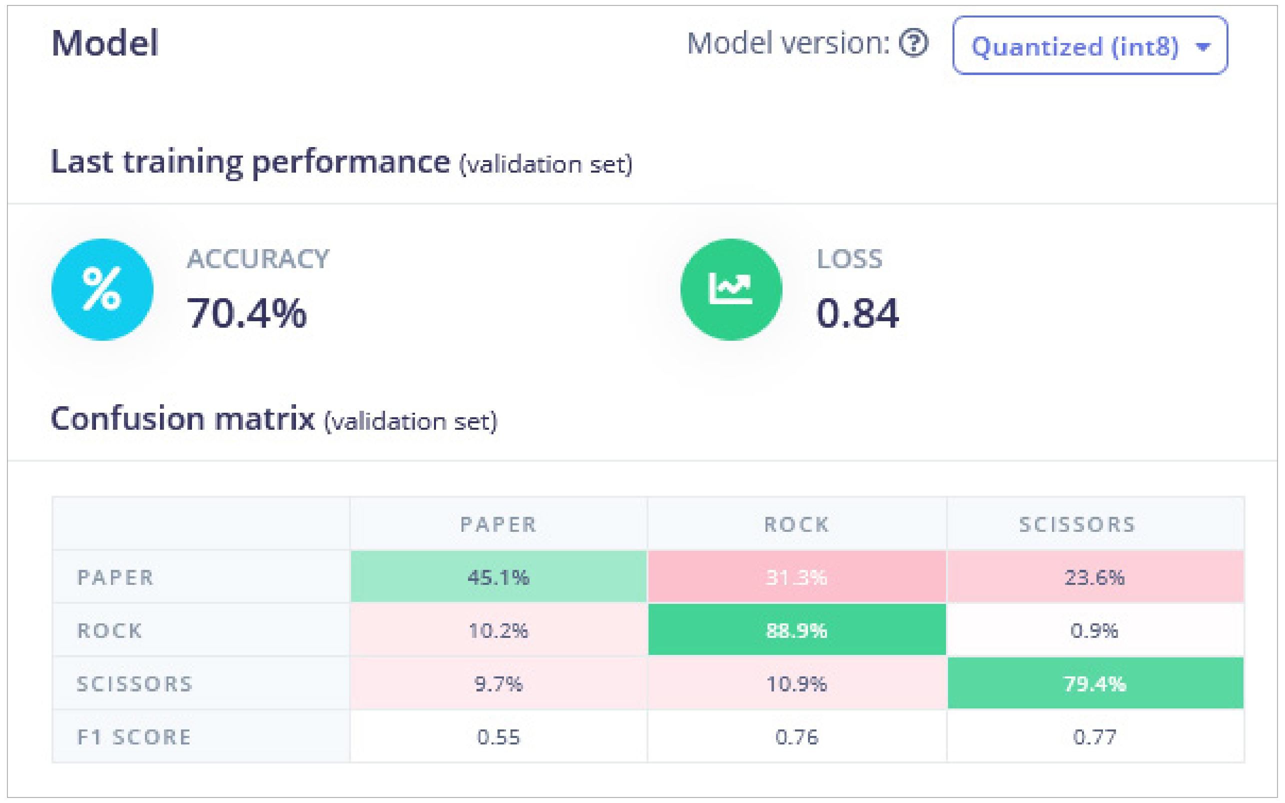
Last training performance (validation set)

ACCURACY	69.9%
LOSS	1.06

Confusion matrix (validation set)

	PAPER	ROCK	SCISSORS
PAPER	68.2%	31.0%	0.8%
ROCK	19.9%	75%	5.1%
SCISSORS	33.0%	0.3%	66.8%
F1 SCORE	0.63	0.72	0.77

Go back to Create Impulse tab, delete Flatten block and choose Spectral Features block, generate the features (remember to set scaling to 0.001!) and train Neural network on Spectral features data. You should see slight improvement.



Model   Model version: ⓘ Quantized (int8)

Last training performance (validation set)

ACCURACY 70.4%

LOSS 0.84

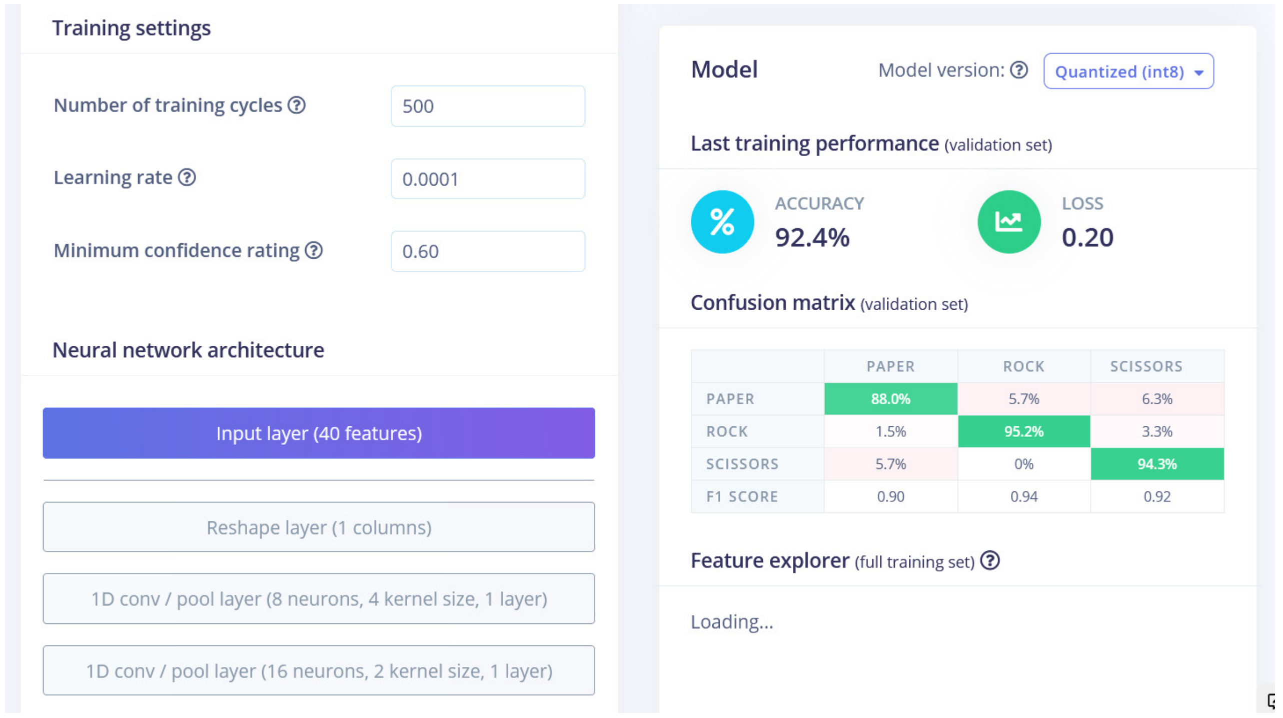
Confusion matrix (validation set)

	PAPER	ROCK	SCISSORS
PAPER	45.1%	31.3%	23.6%
ROCK	10.2%	88.9%	0.9%
SCISSORS	9.7%	10.9%	79.4%
F1 SCORE	0.55	0.76	0.77

Both Flatten and Spectral Features blocks are actually not the best processing methods for rock–paper–scissors gesture recognition task. If we think about it, for classifying rock–paper–scissors gestures we just need to count how many times and for how long the light sensor has received "lower–than–normal" values. If it is one relatively long time — then it is rock (fist passing above the sensors). If it is two times, then it is scissors. Anything more than that is paper. Sounds easy, but preserving time series data is really important for neural network to be able to learn this relationship in data points.

Both Flatten and Spectral Features processing blocks remove the time relationship within each window — Flatten block simply turns the raw values, that are initially in sequence to Average, Min, Max, etc. values calculated on all values in time window, **irrespective** of their order. Spectral Features block extracts the frequency and power characteristics and the reason it didn't work that well for this particular task is probably, that the duration of each gesture is too short.

So, the way to achieve best performance is to use Raw data block, which will preserve the time series data. Have a look at sample project where we used Raw data and Convolutional 1D network, a more specialized type of network, compared to fully–connected. We were able to achieve 92.4% accuracy on the same data!



Training settings

Number of training cycles ⓘ	500
Learning rate ⓘ	0.0001
Minimum confidence rating ⓘ	0.60

Neural network architecture

Input layer (40 features)

Reshape layer (1 columns)

1D conv / pool layer (8 neurons, 4 kernel size, 1 layer)

1D conv / pool layer (16 neurons, 2 kernel size, 1 layer)

Model   Model version: ⓘ Quantized (int8)

Last training performance (validation set)

ACCURACY 92.4%

LOSS 0.20

Confusion matrix (validation set)

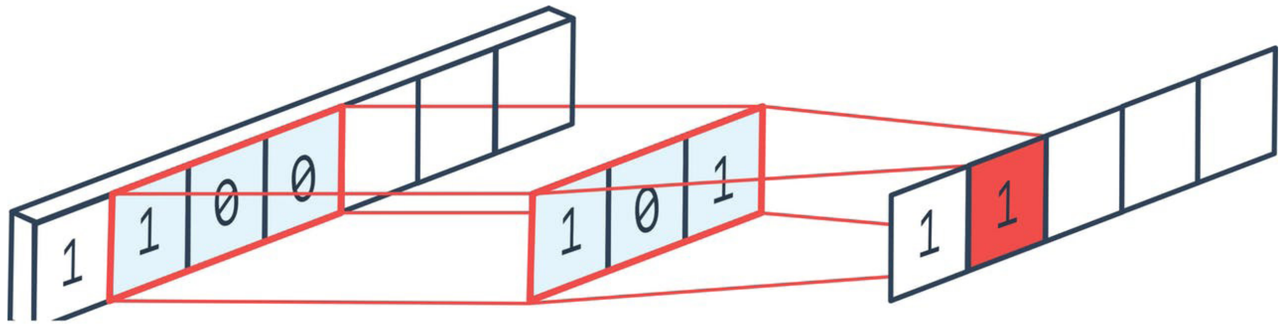
	PAPER	ROCK	SCISSORS
PAPER	88.0%	5.7%	6.3%
ROCK	1.5%	95.2%	3.3%
SCISSORS	5.7%	0%	94.3%
F1 SCORE	0.90	0.94	0.92

Feature explorer (full training set) ⓘ

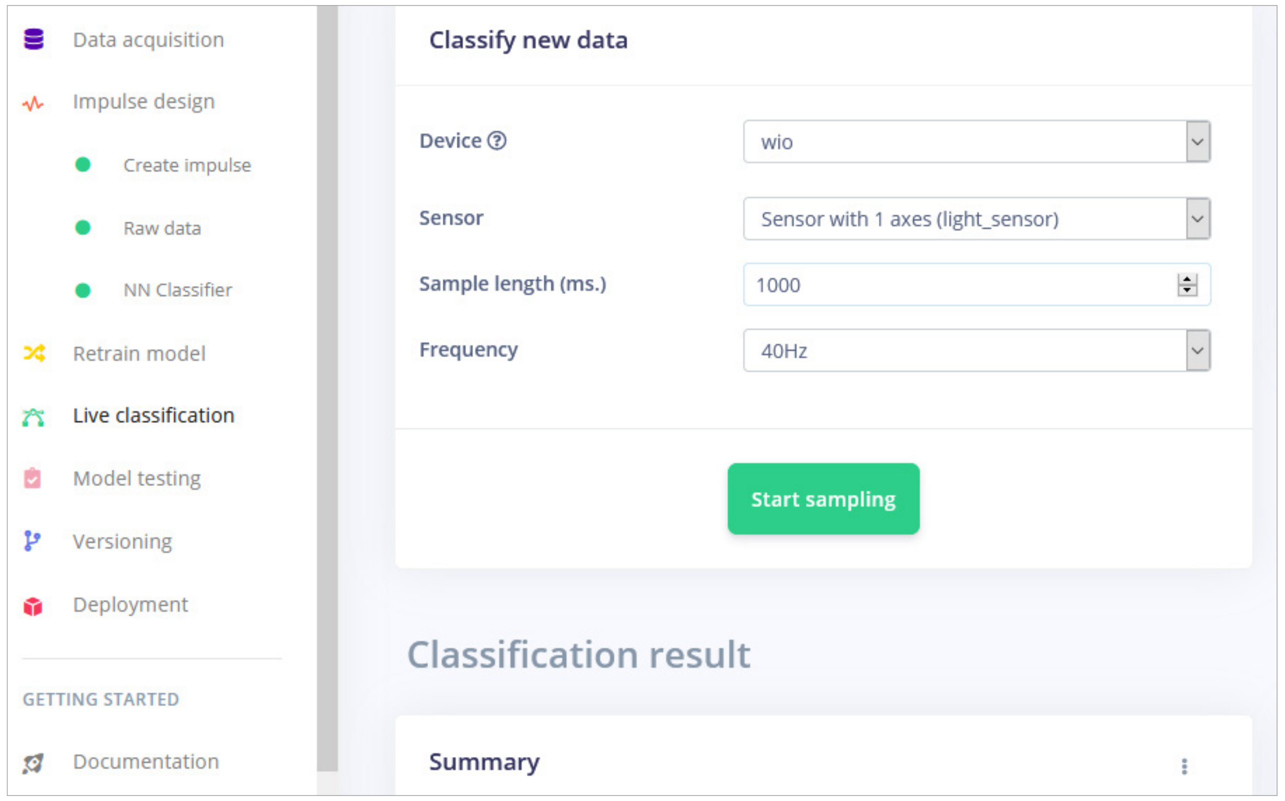
Loading...

The final results after training were
• Flatten FC 69.9 % accuracy
• Spectral Features FC 70.4 % accuracy
• Raw Data Conv1D 92.4 % accuracy

We will discuss Convolutions and how useful they are in later articles about sound processing. For now we're going to use simple fully–connected model and Spectral Features processing.



After the training you can test the model using Live classification tab, which will gather a data sample from device and classify it with model hosted on Edge Impulse. We test with three different gestures and see the accuracy is satisfactory as far as proof of concept goes.



Classify new data

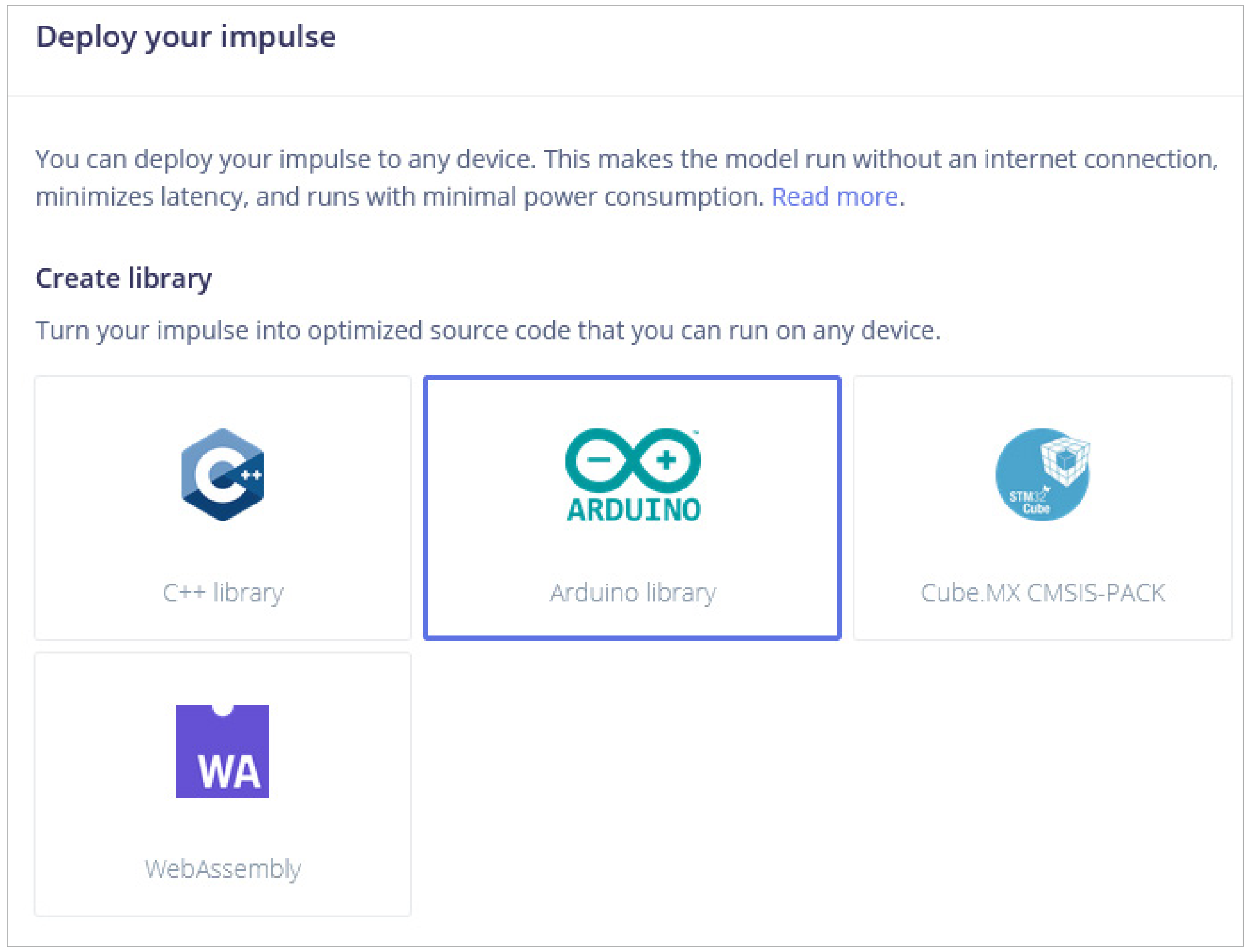
Device: wio

Sensor: Sensor with 1 axes (light_sensor)

Sample length (ms.): 1000

Frequency: 40Hz

The next step is deployment on device. After clicking on Deployment tab, choose Arduino library and download it.



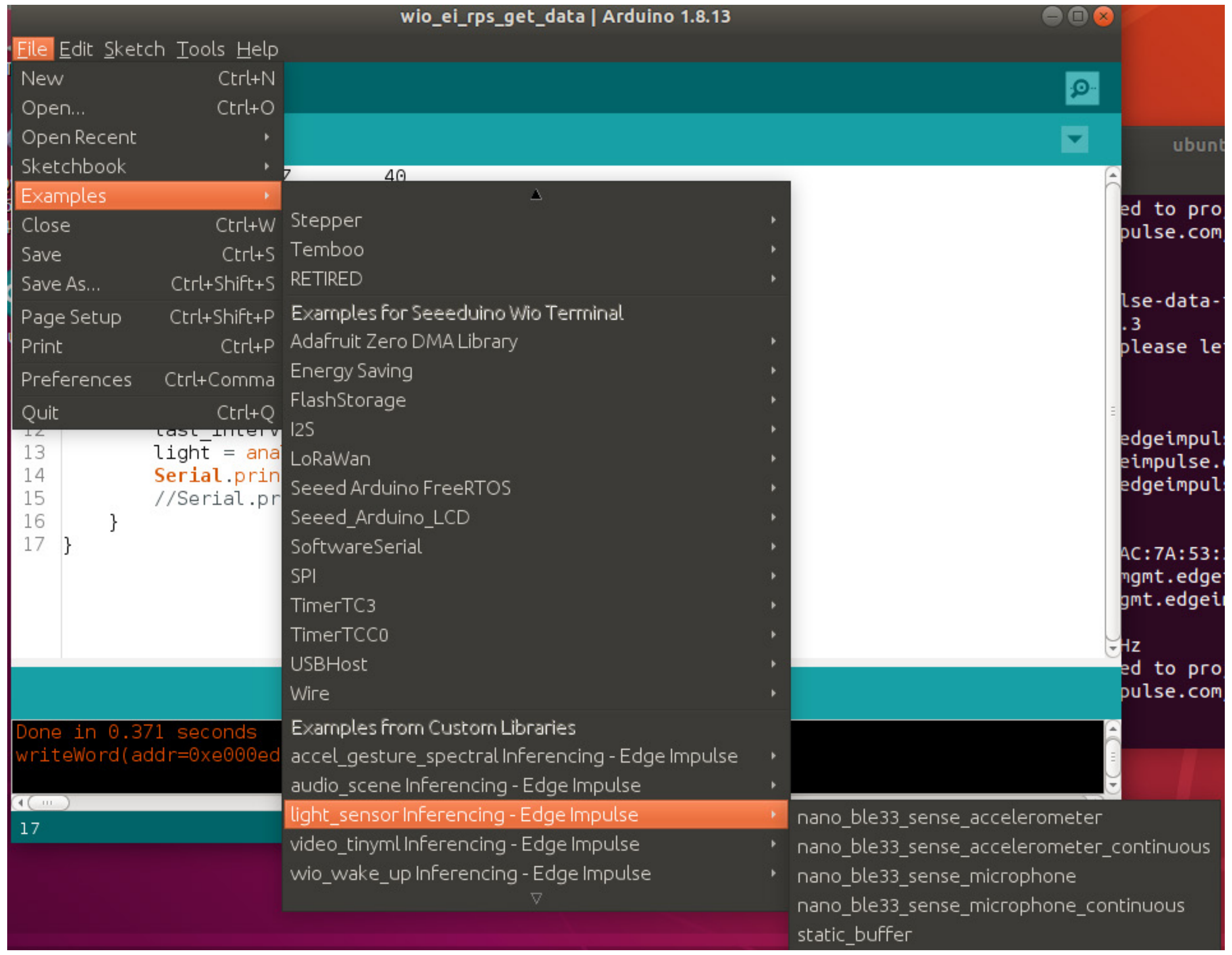
Deploy your impulse

You can deploy your impulse to any device. This makes the model run without an internet connection, minimizes latency, and runs with minimal power consumption. Read more.

Create library

Turn your impulse into optimized source code that you can run on any device.

Extract the archive and place it in your Arduino libraries folder. Open Arduino IDE and choose static buffer sketch (located in File –> Examples –> name of your project –> static_buffer) , which already has all the boilerplate code for classification with your model in place. Neat!


wio_ei_rps_get_data | Arduino 1.8.13
File Edit Sketch Tools Help
New Ctrl+N
Open... Ctrl+O
Open Recent
Sketchbook
Examples
Close Ctrl+W
Save Ctrl+S
Save As... Ctrl+Shift+S
Page Setup Ctrl+Shift+P
Print Ctrl+P
Preferences Ctrl+Comma
Quit Ctrl+Q
Stepper
Temboo
RETIRED
Examples for Seeeduino Wio Terminal
Adafruit Zero DMA Library
Energy Saving
FlashStorage
I2S
LoRaWan
Seeed Arduino FreeRTOS
Seeed_Arduino_LCD
SoftwareSerial
SPI
TimerTC3
TimerTCC0
USBHost
Wire
Examples from Custom Libraries
accel_gesture_spectral Inferencing - Edge Impulse
audio_scene Inferencing - Edge Impulse
light_sensor Inferencing - Edge Impulse
video_tinyml Inferencing - Edge Impulse
wio_wake_up Inferencing - Edge Impulse
nano_ble33_sense_accelerometer
nano_ble33_sense_accelerometer_continuous
nano_ble33_sense_microphone
nano_ble33_sense_microphone_continuous
static_buffer
Done in 0.371 seconds

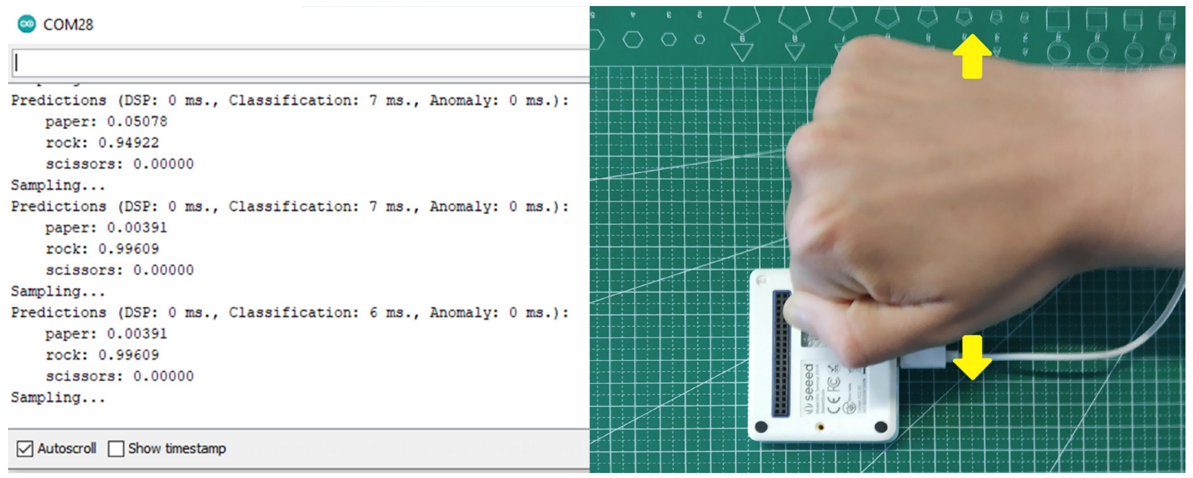
The only thing for use to fill in is the data acquisition on–device. We'll use a simple for loop with delay to account for frequency (if you remember we had 25 ms delay when gathering data for training dataset).

```
int raw_feature_get_data(size_t offset, size_t length, float *out_ptr) {
float features[40];
for (byte i = 0; i < 40; i = i + 1)
  {
  features[i]=analogRead(WIO_LIGHT);
  delay(25);
  }
  memcpy(out_ptr, features + offset, length * sizeof(float));
  return 0;
}
```

Certainly there are better ways to implement this, for example a sensor data buffer, which would allow us to perform inference more often. But we'll get to that in later lessons of the course.

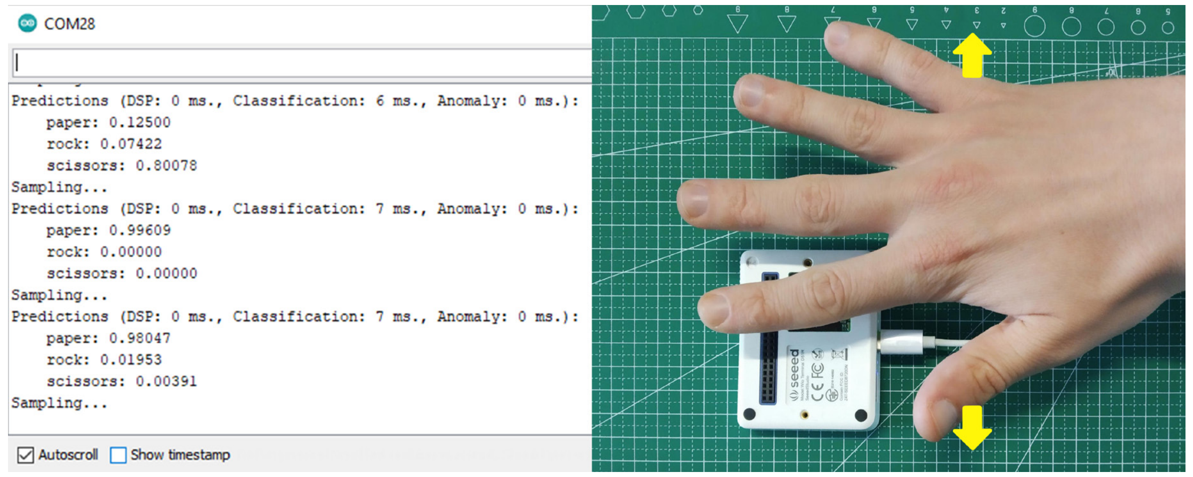
After you added sensor data acquisition part to sample code upload it to Wio Terminal and open Serial monitor. Move your hand while performing a gesture and see the probability results printed out on Serial monitor.

Rock:



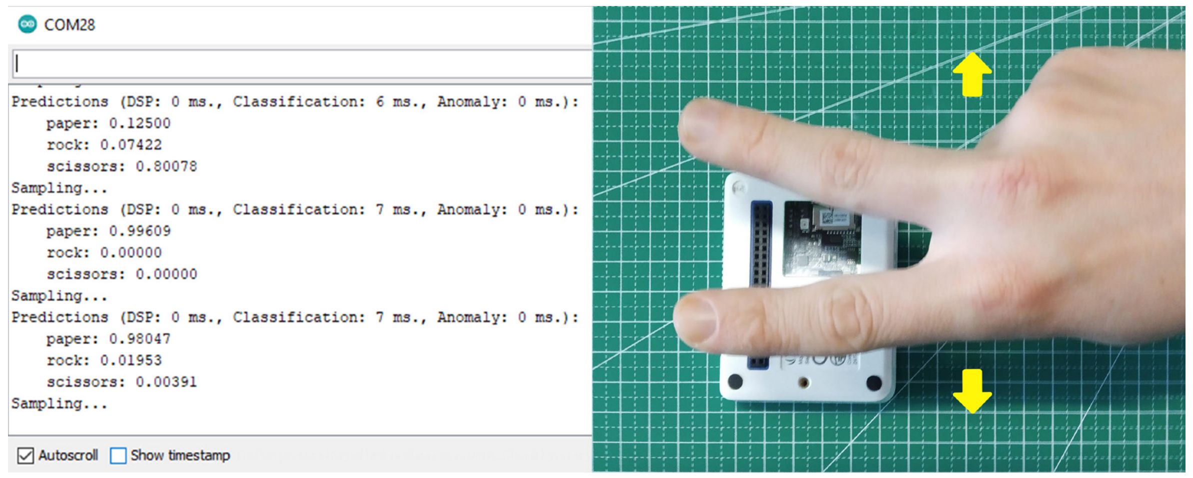
```
Predictions (DSP: 0 ms., Classification: 7 ms., Anomaly: 0 ms.):
    paper: 0.05078
    rock: 0.94922
    scissors: 0.00000
Sampling...
Predictions (DSP: 0 ms., Classification: 7 ms., Anomaly: 0 ms.):
    paper: 0.00391
    rock: 0.99609
    scissors: 0.00000
Sampling...
Predictions (DSP: 0 ms., Classification: 6 ms., Anomaly: 0 ms.):
    paper: 0.00391
    rock: 0.99609
    scissors: 0.00000
Sampling...
```

Paper:



```
Predictions (DSP: 0 ms., Classification: 6 ms., Anomaly: 0 ms.):
    paper: 0.12500
    rock: 0.07422
    scissors: 0.80078
Sampling...
Predictions (DSP: 0 ms., Classification: 7 ms., Anomaly: 0 ms.):
    paper: 0.99609
    rock: 0.00000
    scissors: 0.00000
Sampling...
Predictions (DSP: 0 ms., Classification: 7 ms., Anomaly: 0 ms.):
    paper: 0.98047
    rock: 0.01953
    scissors: 0.00391
Sampling...
```

Scissors:



```
Predictions (DSP: 0 ms., Classification: 6 ms., Anomaly: 0 ms.):
    paper: 0.12500
    rock: 0.07422
    scissors: 0.80078
Sampling...
Predictions (DSP: 0 ms., Classification: 7 ms., Anomaly: 0 ms.):
    paper: 0.99609
    rock: 0.00000
    scissors: 0.00000
Sampling...
Predictions (DSP: 0 ms., Classification: 7 ms., Anomaly: 0 ms.):
    paper: 0.98047
    rock: 0.01953
    scissors: 0.00391
Sampling...
```

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 image is down–scaled a lot. But recognizing gestures with **just 1 pixel** is entirely different level!

Expansion tasks

Try increasing/decreasing number of neurons in first and second hidden layers and see how that affects the accuracy and inference time.

Lesson 04

Project II: Classifying hand gestures with accelerometer: theory and data collection

Theory

In this lesson, we'll take on the similar task, gesture recognition, but will use a different sensor for that – 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.

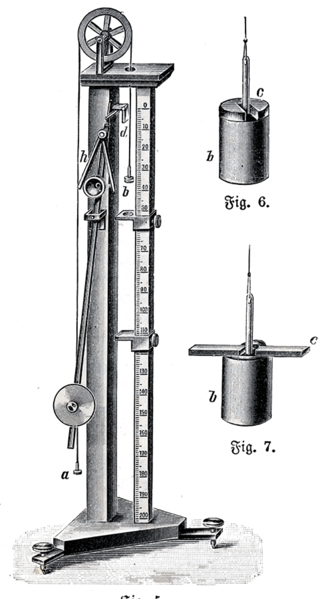
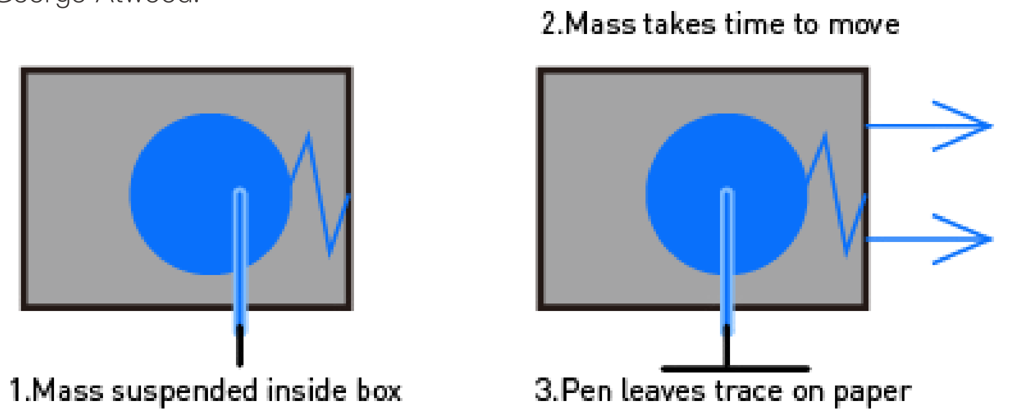


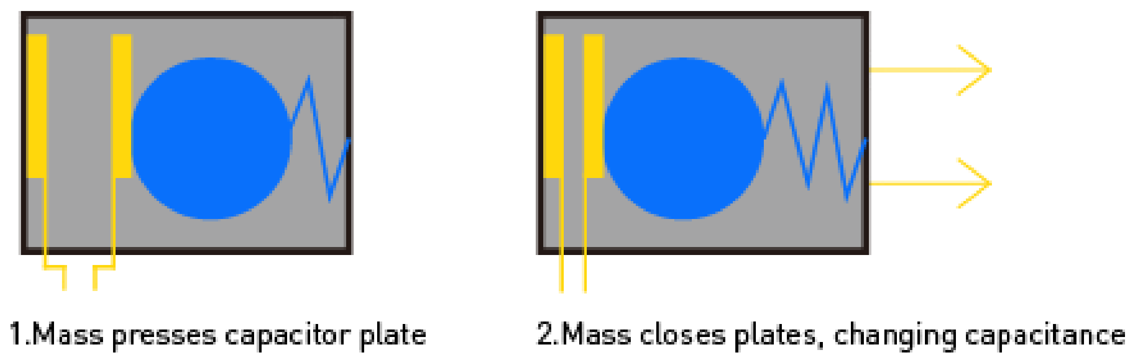
Fig. 5.
Atwoodſche Fallmaſchine

As you might guess from the name, accelerometers are devices that measure acceleration, which is the rate of change of the velocity of an object. They measure in meters per second squared (m/s2) or in G-forces (g). A single G-force for us here on planet Earth is equivalent to 9.8 m/s2. As with other sensors, there are different kinds of accelerometers and the first ones invented were mechanical ones.

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


2.Mass takes time to move
1.Mass suspended inside box
3.Pen leaves trace on paper

The accelerometer in your phone and in Wio Terminal is a MEMS (Microelectromechanical) accelerometer. The exact module used in Wio Terminal is called 3-Axis Digital Accelerometer(LIS3DHTR). Generally, accelerometers contain capacitive plates internally. Some of these are fixed, while others are attached to minuscule springs that move internally as acceleration forces act upon the sensor.

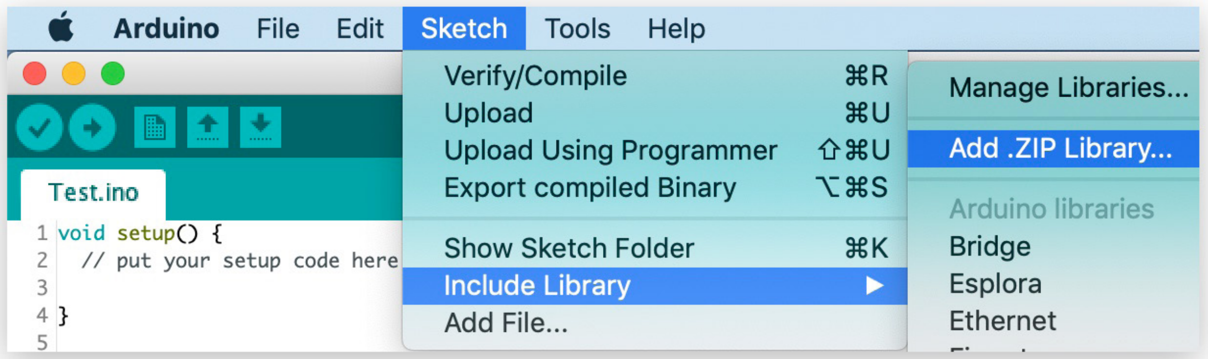

1.Mass presses capacitor plate
2.Mass closes plates, changing capacitance

Preparation

Open Arduino IDE, make sure you have accelerometer libraries installed. To install 3-Axis Digital Accelerometer(LIS3DHTR) libraries:

1. Visit the Seeed_Arduino_LIS3DHTR repositories and download the entire repository to your local drive.

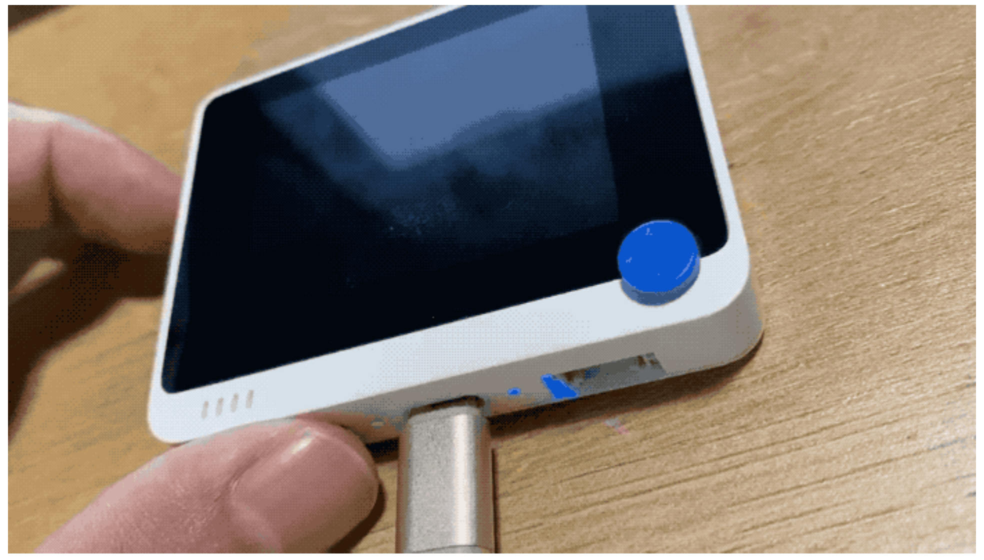
2. Now, the LIS3DHTR can be installed to the Arduino IDE. Open the Arduino IDE, and click `sketch` -> `Include Library` -> `Add .ZIP Library`, and choose the `Seeed_Arduino_LIS3DHTR` file that you've have just downloaded.



```
void setup() {
  // put your setup code here
}
```

Practice

Make a new project on Edge Impulse dashboard. Accelerometer data has three data samples in each data packet and we need to sample it faster, than we did before with light sensor, increasing the frequency to 62.5 Hz. It means that we cannot use data forwarder tool for data collection and will need to use specilized firmware for data collection. To do that, connect Wio Terminal to your computer. Entering the bootloader mode by sliding the power switch twice quickly.



An external drive named Arduino should appear in your PC. Download the firmware for data collection here :

Edge Impulse uf2 firmware files

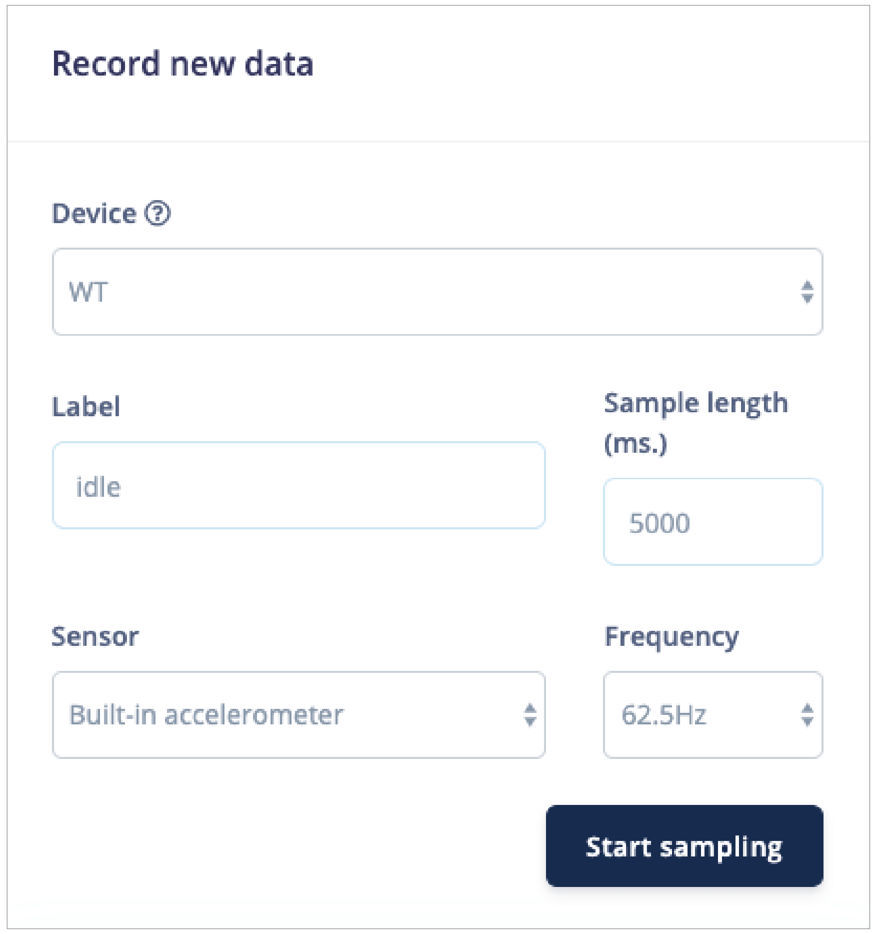
Drag the the downloaded to the Arduino drive. Now, Edge Impulse data collection firmware is loaded on Wio Terminal!

Once you uploaded the firmware to Wio Terminal, run

```
edge-impulse-daemon --clean
```

in Command prompt and log in with your Edge Impulse credentials. Clean command clean your credentials and project name. Now you are ready to receive data in Edge Impulse dashboard. Go to Data acquisition tab and you should see your device there.

Under **Record new data**, select your device, set the label to shake, the sample length to 10000, the sensor to Built-in accelerometer and the frequency to 62.5Hz. This indicates that you want to record data for 10 seconds, and label the recorded data as shake. You can later edit these labels if needed.



Record new data

Device: WT

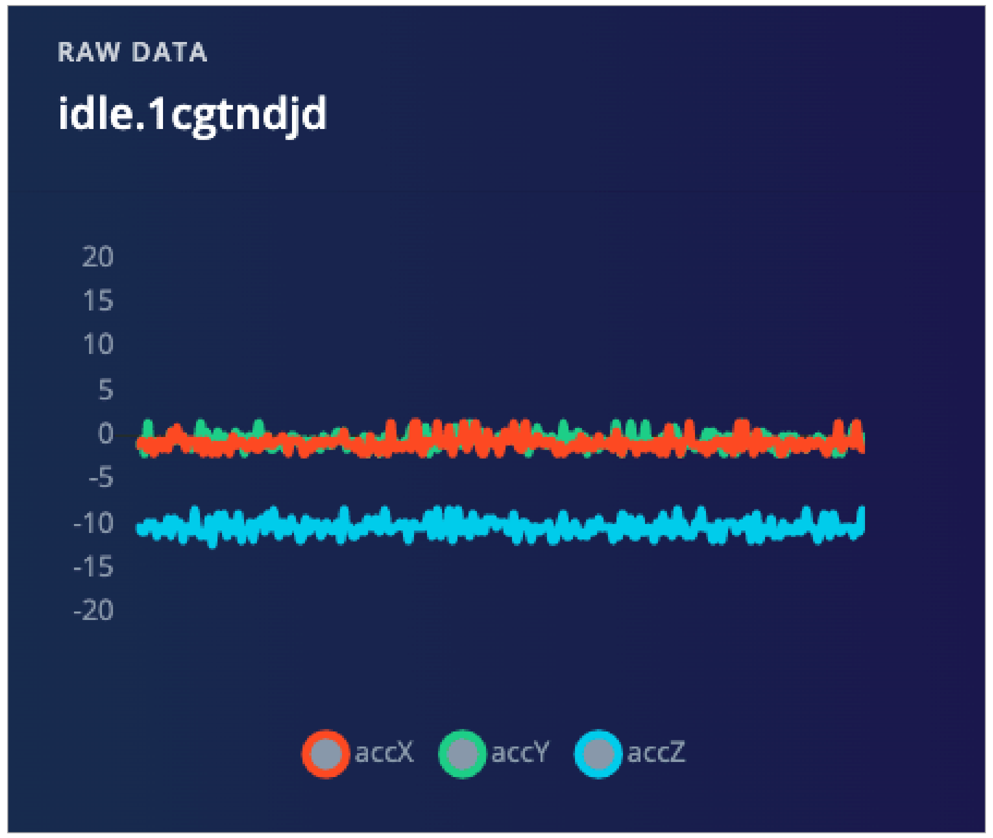
Label: idle

Sample length (ms.): 5000

Sensor: Built-in accelerometer

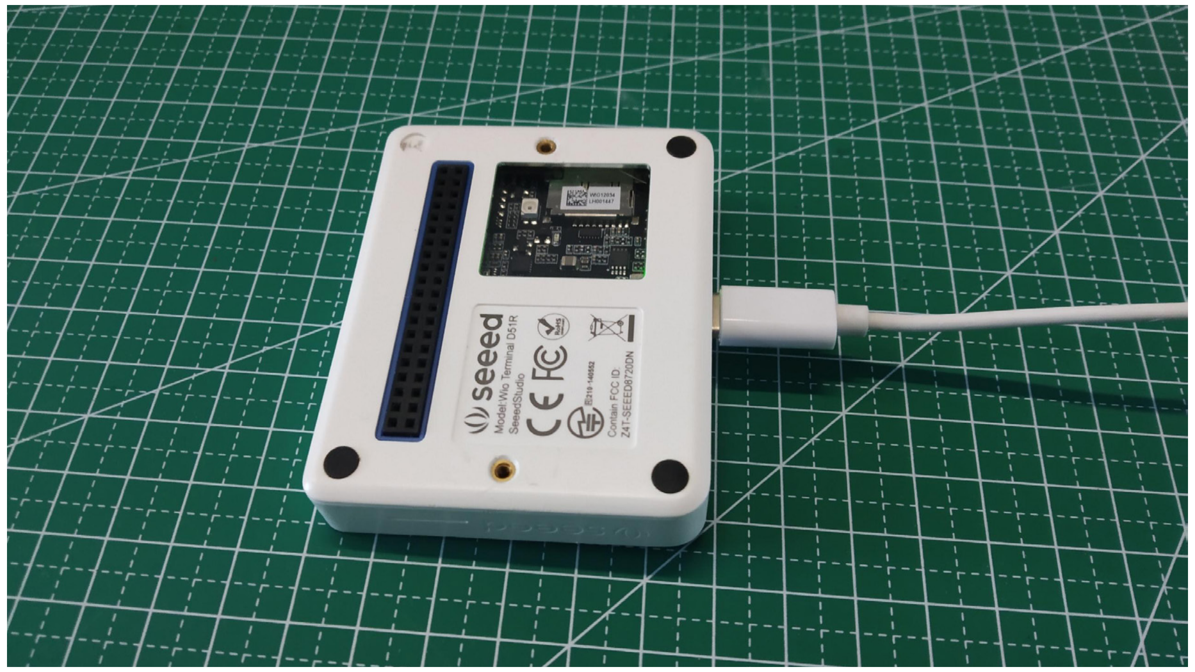
Frequency: 62.5Hz

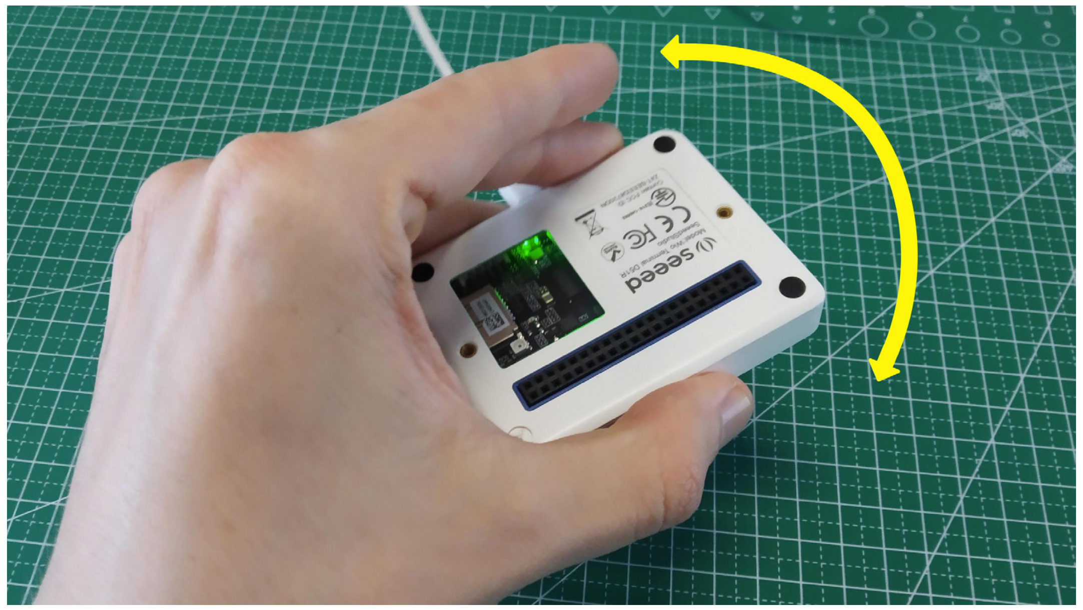
After you click **Start sampling** shake your device up and down and side to side in a continuous motion. In about twelve seconds the device should complete sampling and upload the file back to Edge Impulse. You see a new line appear under 'Collected data' in the studio. When you click it you now see 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.

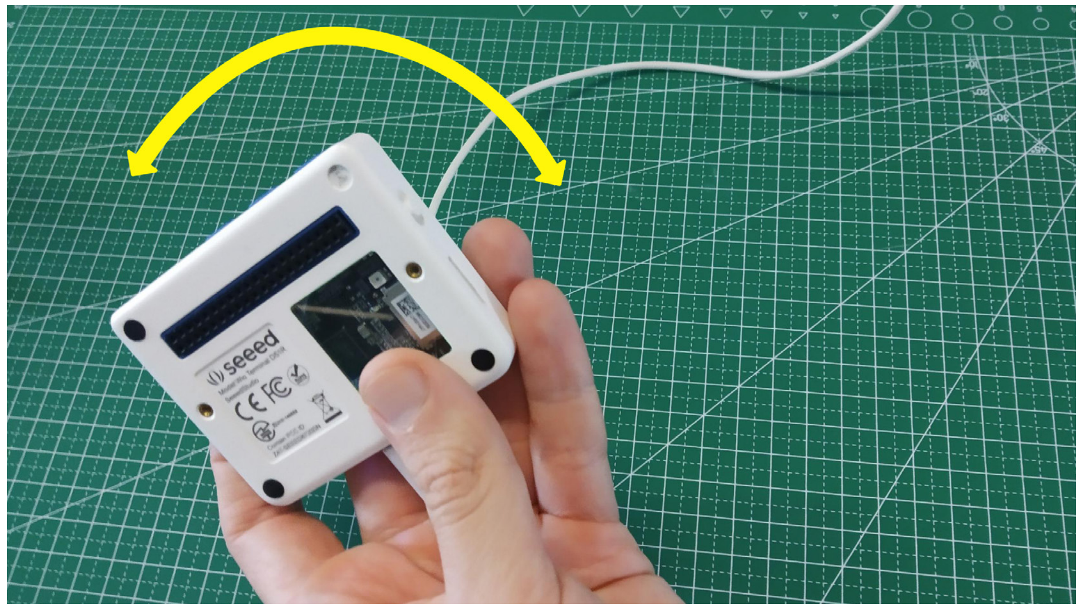


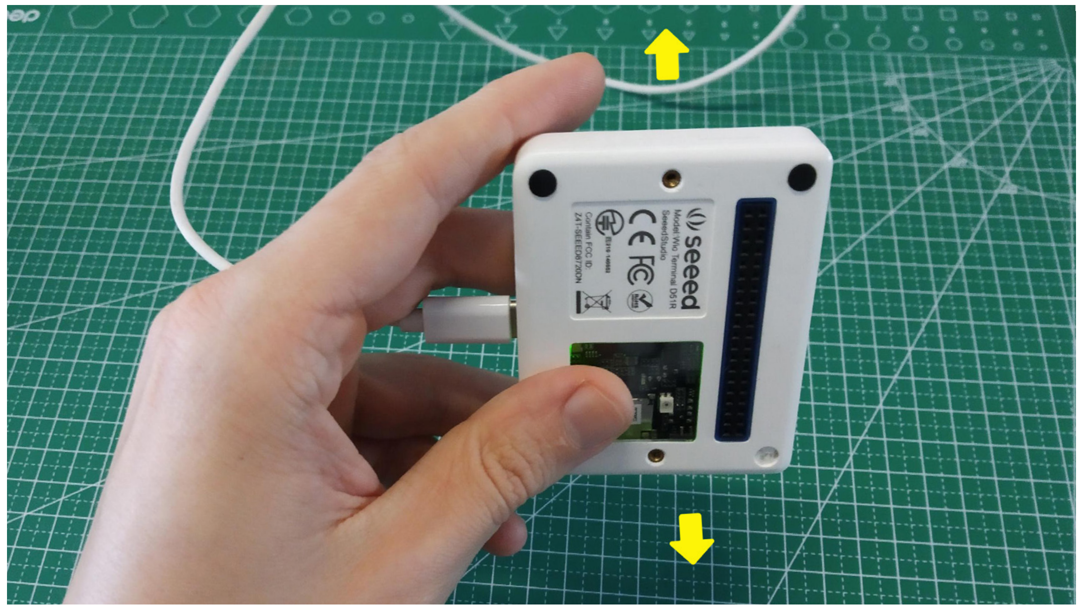
Machine learning works best with lots of data, so a single sample won't cut it. Now is the time to start building your own dataset. For example, use the following four classes, and record around 3 minutes of data per class:

- Idle – just sitting on your desk while you're working.
- Turn – turn device, similar how you would turn a valve
- Wave – waving the device from left to right.
- Shake – moving the device up and down.









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. It's best to collect samples of ~10 seconds each.

Expansion tasks

Gather data from 2 other people, except for yourself.

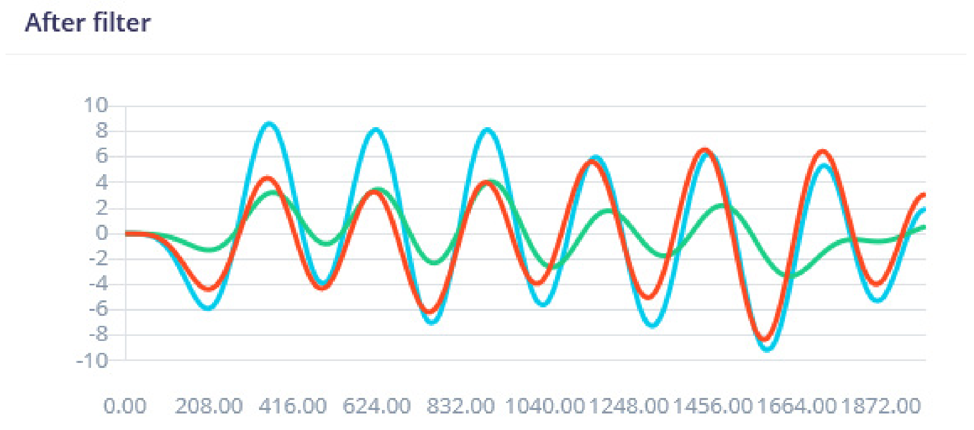
Lesson 05

Project II: Classifying hand gestures with accelerometer: model training and deployment

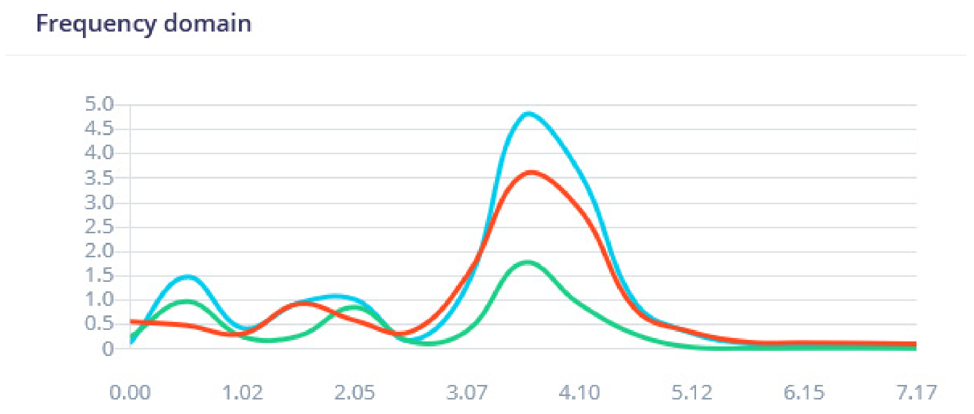
Theory

In the last lesson we have already used Spectral Analysis block for data processing briefly. Spectral analysis block slices your data into smaller windows and applies Butterworth filter, Fast Fourier Transform and calculates power edges on each window of every data axis to output the following features:

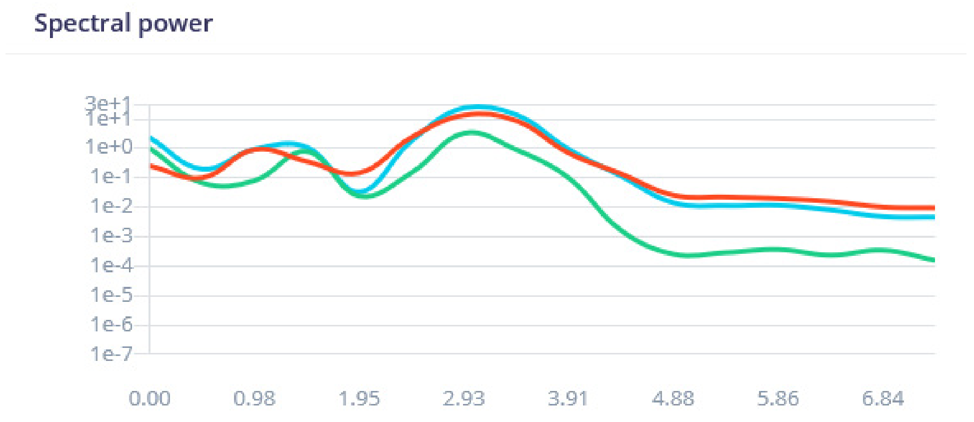
• Values after Butterworth filter – normally, depending on filter type applied (low–pass or high–pass) it smoothens the signal



• Peaks in frequency domain – you can specify the number of peaks, these are the most prominent frequencies in that window of data



• Spectral power of peaks – the amplitude of frequency at the peaks, i.e. how powerful is the signal



Spectral analysis block is the most suitable for types of data, that contain periodically repeating elements, for example accelerometer motion or (to some extent) audio signal, vibration, etc. It does not generate useful features for signal that is devoid of repeating elements, that is the reason that it didn't work well for data in the previous project.

Preparation

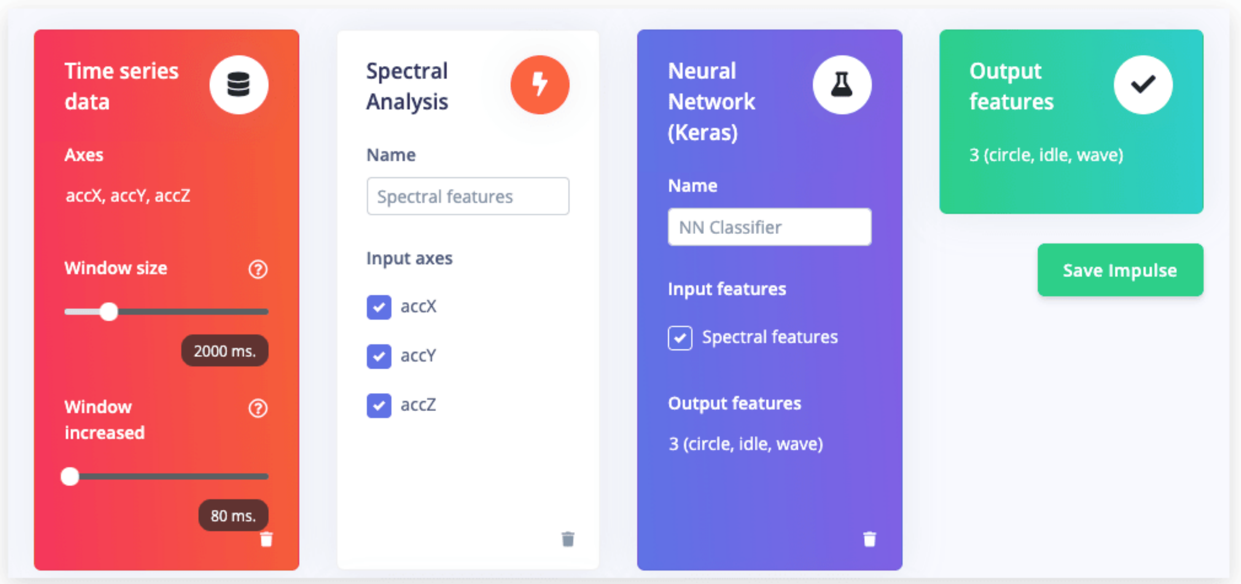
Make sure you are using Arduino IDE > 1.9.

Practice

With the training set in place you can design an impulse. 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. 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, shake, turn and wave) classes.

In the studio go to **Create impulse**, set the window size to `2000` (you can click on the `2000 ms.` text to enter an exact value), the window increase to `80`, and add the 'Spectral Analysis' and 'Neural Network (Keras)' blocks. Then click **Save impulse**.



Time series data
Axes
accX, accY, accZ
Window size
2000 ms.
Window increased
80 ms.

Spectral Analysis
Name
Spectral features
Input axes
✓ accX
✓ accY
✓ accZ

Neural Network (Keras)
Name
NN Classifier
Input features
✓ Spectral features
Output features
3 (circle, idle, wave)

Output features
3 (circle, idle, wave)

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:

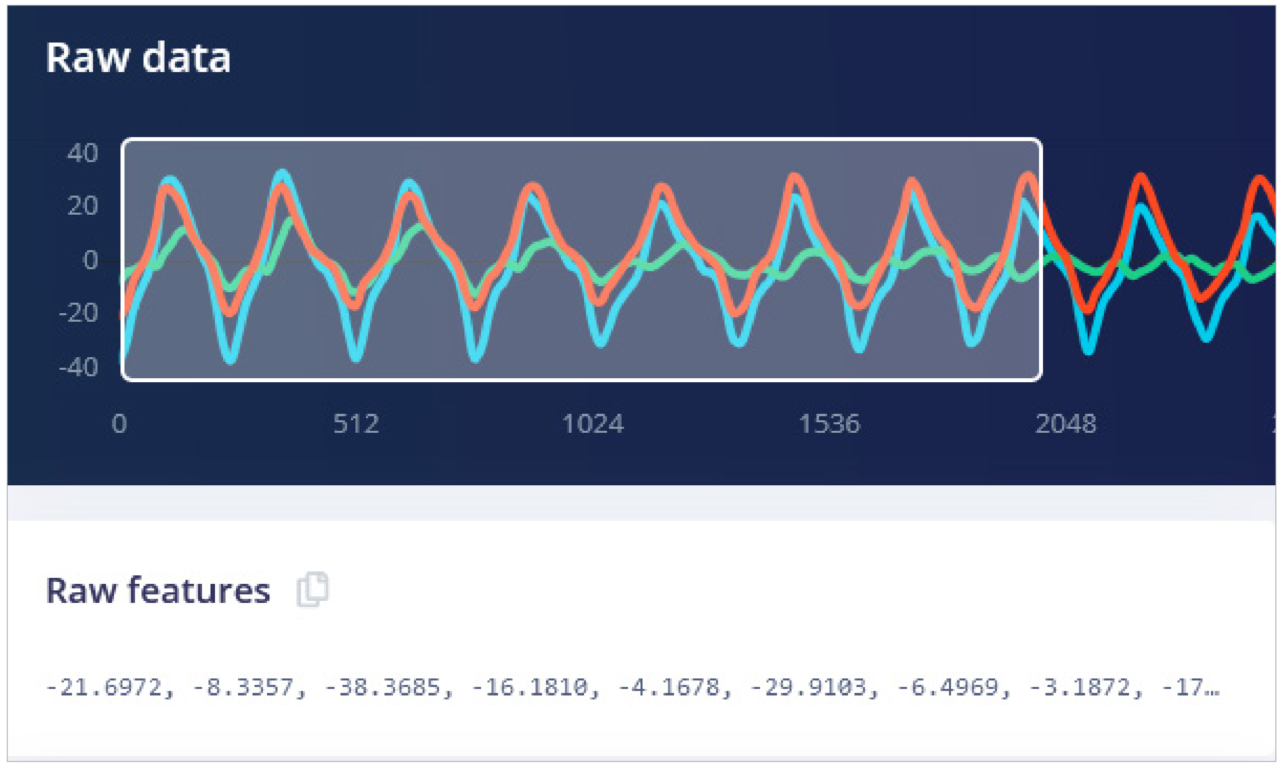
• After filter – the signal after applying a low-pass filter. This will remove noise.

• 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.

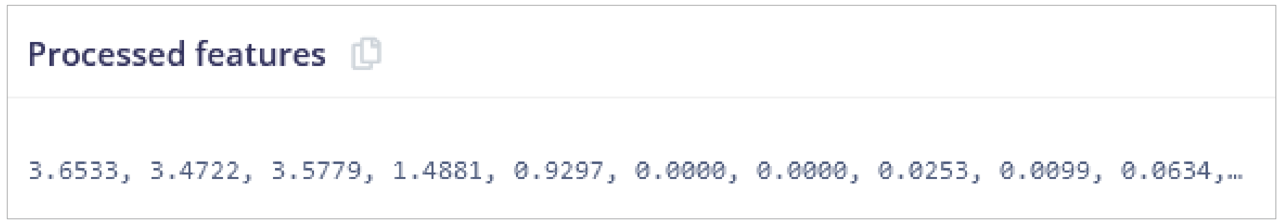
A good signal processing block will yield similar results for similar data. If you move the sliding window (on the raw data graph) around, the graphs should remain similar. Also, when you switch to another file with the same label, you should see similar graphs, even if the orientation of the device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

One window of data before processing:



```
-21.6972, -8.3357, -38.3685, -16.1810, -4.1678, -29.9103, -6.4969, -3.1872, -17…
```

One window of data after processing:



```
3.6533, 3.4722, 3.5779, 1.4881, 0.9297, 0.0000, 0.0000, 0.0253, 0.0099, 0.0634,…
```

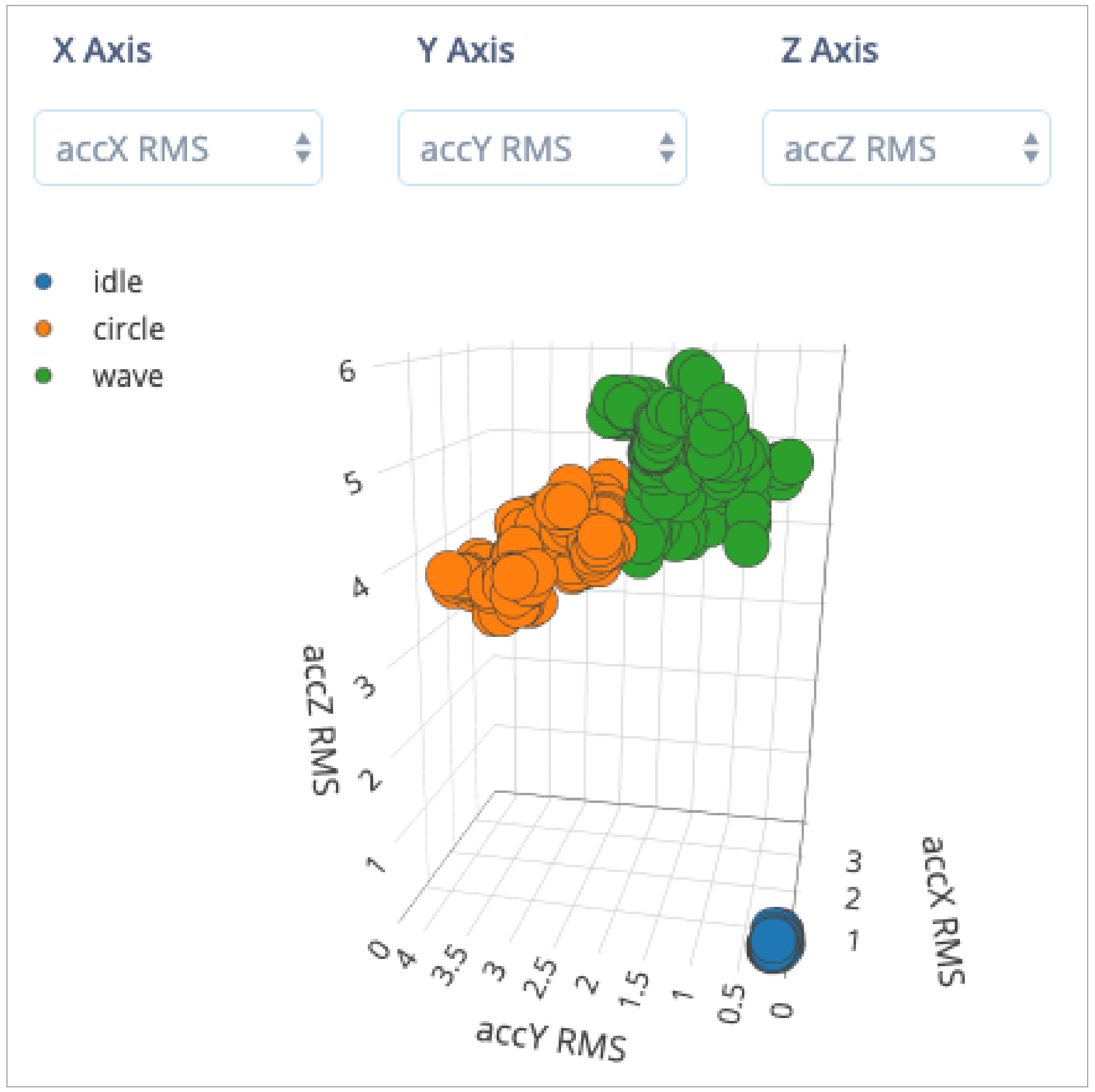
Once you're happy with the result, click **Save parameters**. This will send you to the 'Feature generation' screen.

In here you'll:

1. Split all raw data up in windows (based on the window size and the window increase).
2. Apply the spectral features block on all these windows.

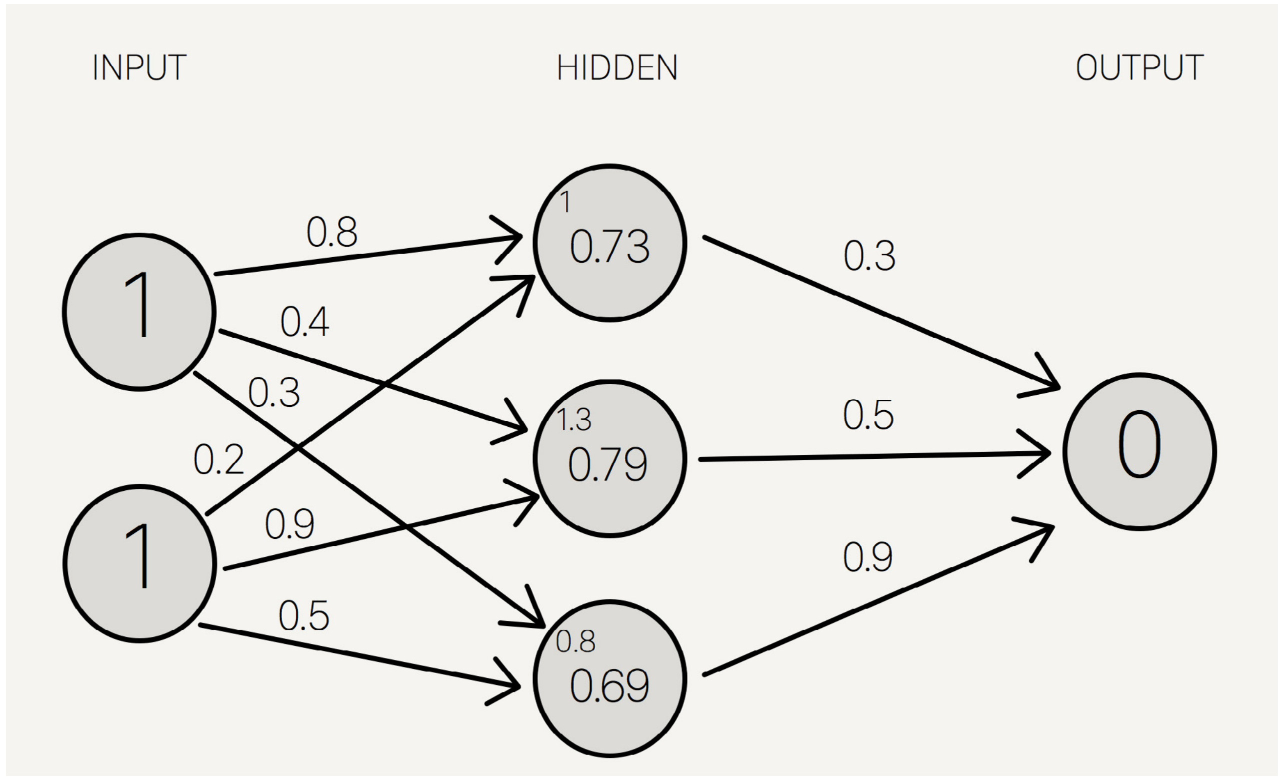
Click **Generate features** to start the process.

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. E.g. by plotting the height of the first peak on the X-axis against the spectral power between 0.5 Hz and 1 Hz on the Y-axis. 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.



With all data processed it's time to start training a neural network. Neural networks are a set of algorithms, modeled loosely after the human brain, that are 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.

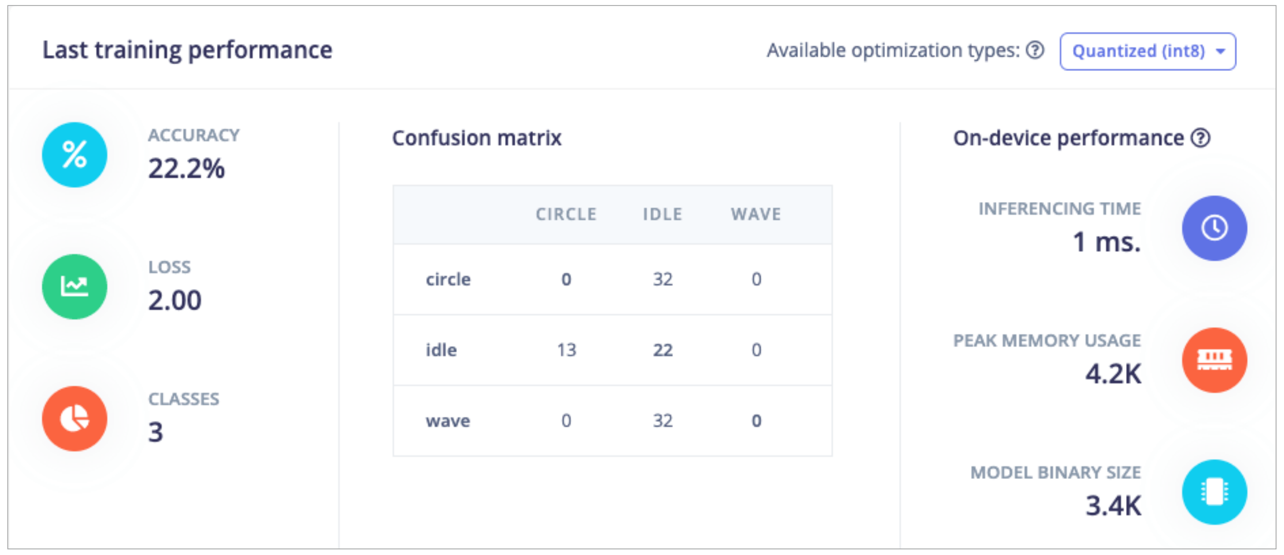
So how does a neural network know what to predict? A neural network consists of layers of neurons, all interconnected, and each connection has a weight.



One such neuron in the input layer would be the height of the first peak of the X–axis (from the signal processing block); and one such neuron in the output layer would be wave (one the classes). When defining the neural network all these connections are initialized randomly, and thus the neural network will make random predictions. During training we then take all the raw data, ask the network to make a prediction, and then make tiny alterations to the weights depending on the outcome (this is why labeling raw data is important).

This way, after a lot of iterations, the neural network learns; and will eventually become much better at predicting new data. Let's try this out by clicking on **NN Classifier** in the menu.

Set 'Number of training cycles' to 1. This will limit training to a single iteration. And then click Start training.



Last training performance

Available optimization types: Quantized (int8)

ACCURACY 22.2%

LOSS 2.00

CLASSES 3

Confusion matrix

	CIRCLE	IDLE	WAVE
circle	0	32	0
idle	13	22	0
wave	0	32	0

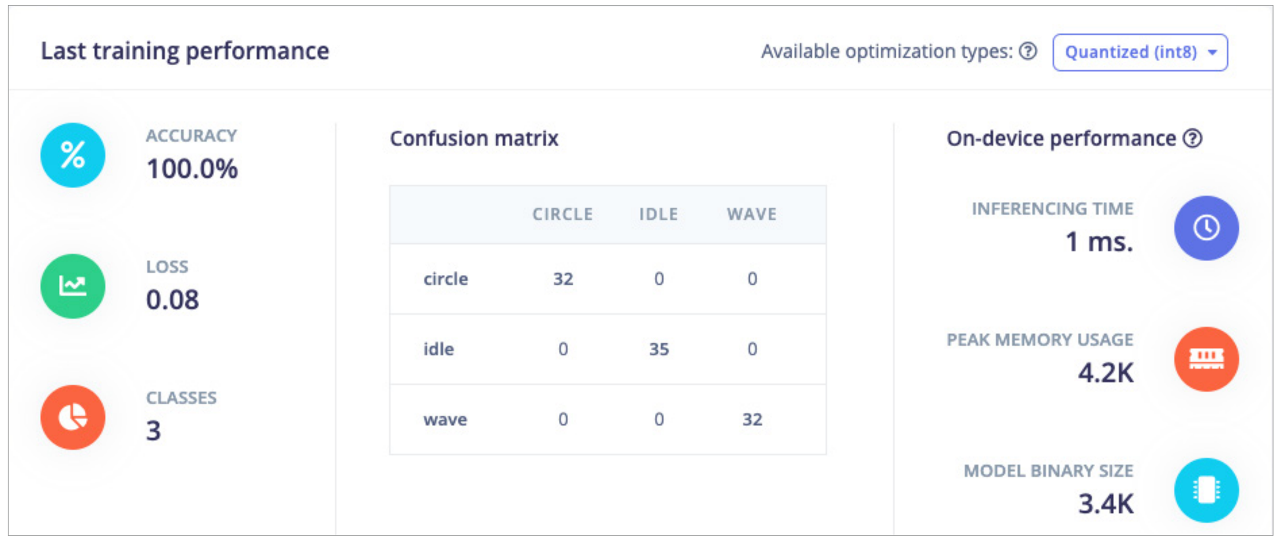
On-device performance

INFERENCING TIME 1 ms.

PEAK MEMORY USAGE 4.2K

MODEL BINARY SIZE 3.4K

Now change 'Number of training cycles' to 2 and you'll see performance go up. Finally, change 'Number of training cycles' to 100 or more and let training finish. You've just trained your first neural network!



Last training performance

Available optimization types: Quantized (int8)

ACCURACY 100.0%

LOSS 0.08

CLASSES 3

Confusion matrix

	CIRCLE	IDLE	WAVE
circle	32	0	0
idle	0	35	0
wave	0	0	32

On-device performance

INFERENCING TIME 1 ms.

PEAK MEMORY USAGE 4.2K

MODEL BINARY SIZE 3.4K

Note

You might end up with a 100% accuracy after training for 100 training cycles. This is not necessarily a good thing, as it might be a sign that the neural network is too tuned for the specific test set and might perform poorly on new data (overfitting). The best way to reduce this is by adding more data, adding Dropout block or reducing the learning rate.

With the impulse designed, trained and verified you can deploy this model back to your device. This makes the model run without an internet connection, minimizes latency, and runs with minimum power consumption.

After clicking on Deployment tab, choose Arduino library and download it. Extract the archive and place it in your Arduino libraries folder. Open Arduino IDE and choose Examples–> name of your project Inferencing Edge Impulse – > nano_ble33_sense_accelerometer sketch. Our board is similar to Arduino Nano BLE33 Sense, but uses different accelerometer (LIS3DHTR instead of LSM9DS1), so we will need to change the data acquisition section accordingly. Also, since Wio Terminal has an LCD screen, we're going to display name of the detected class if this class confidence value is above threshold.

First change the header

```
#include <Arduino_LSM9DS1.h>
```

to

```
#include"LIS3DHTR.h"
#include"TFT_eSPI.h"
```

```

LIS3DHTR<TwoWire> lis;
TFT_eSPI tft;
```

Then change initialization in setup function

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

to

```

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

    lis.begin(Wire1);

    if (!lis.available()) {
    Serial.println("Failed to initialize IMU!");
    while (1);
    }
    else {
        ei_printf("IMU initialized\r\n");
    }
    lis.setOutputDataRate(LIS3DHTR_DATARATE_100HZ); // Setting output data rage
 to 25Hz, can be set up tp 5kHz
      lis.setFullScaleRange(LIS3DHTR_RANGE_16G); // Setting scale range to 2g,
 select from 2,4,8,16g
```

We do data collection and inference within loop function, here is where we need to change data acquisition with LSM9DS1 to data acquisition function for LIS3DHTR

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

to

```
lis.getAcceleration(&buffer[ix], &buffer[ix + 1], &buffer[ix + 2]);
```

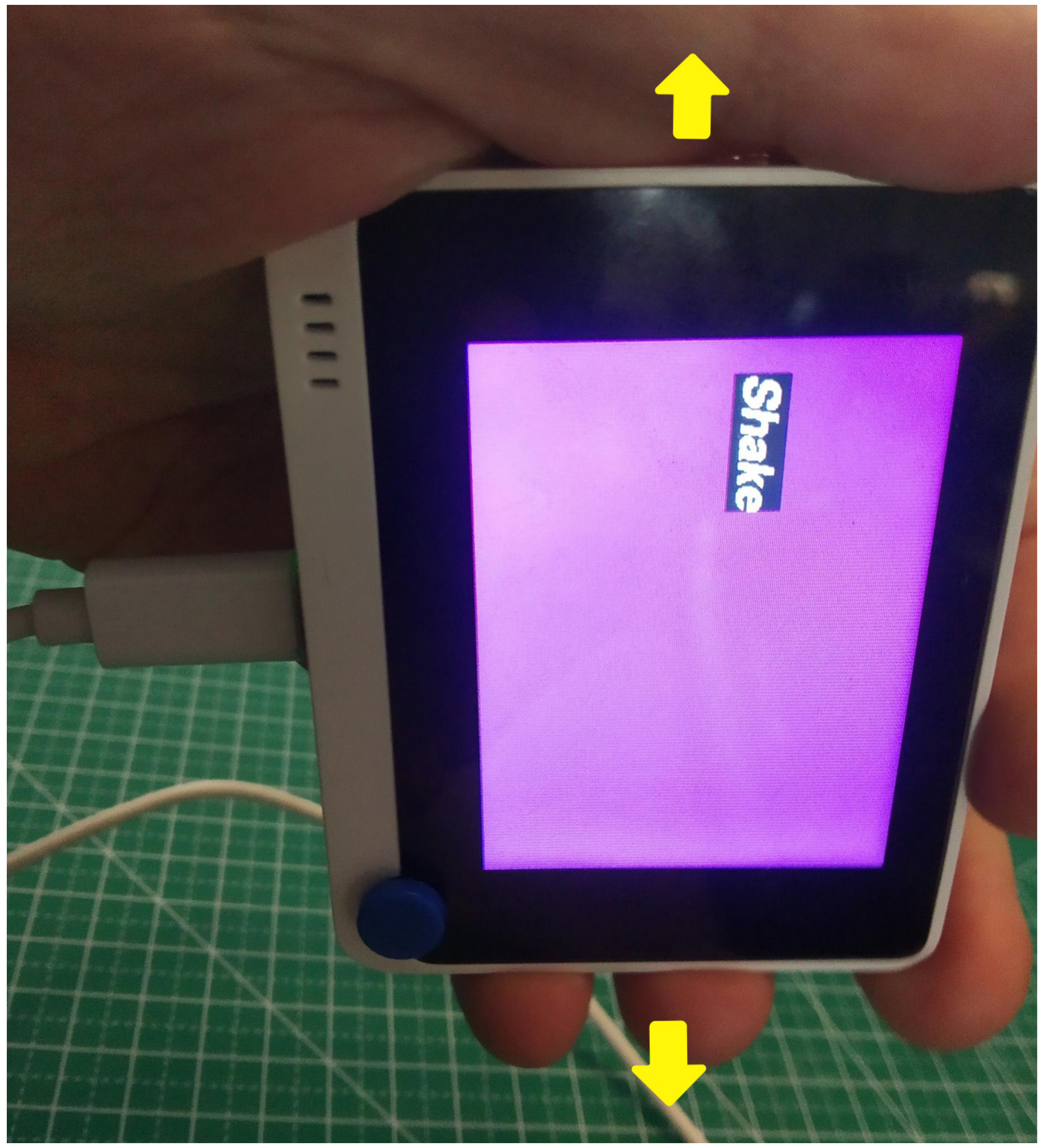
And then to display the class name on the LCD screen, after

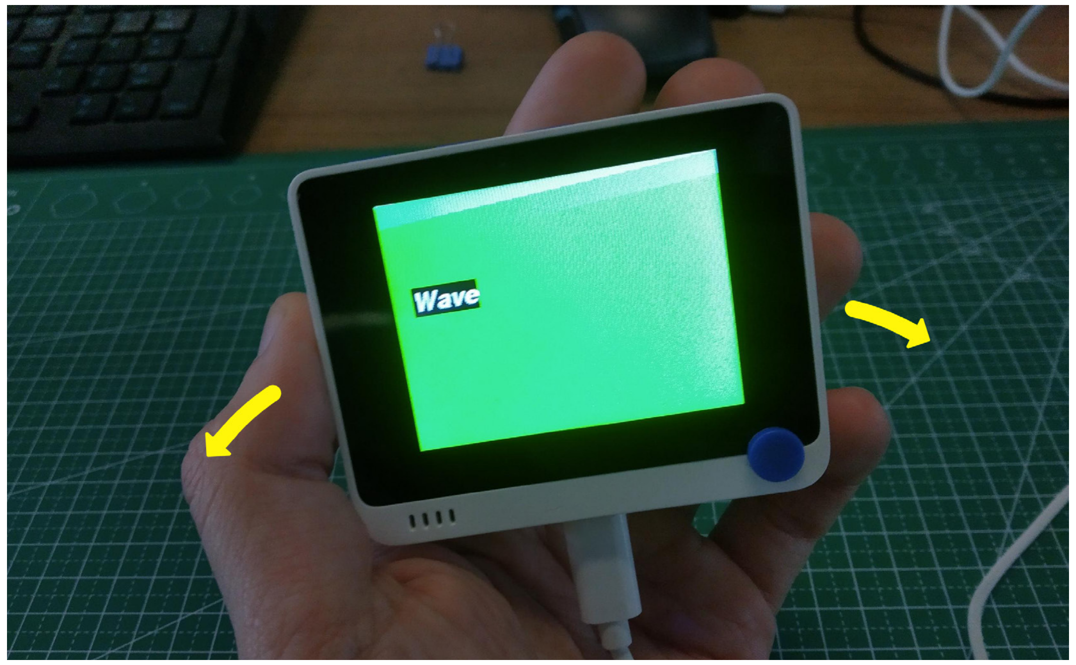
```
#if EI_CLASSIFIER_HAS_ANOMALY == 1
    ei_printf("    anomaly score: %.3f\n", result.anomaly);
#endif
```

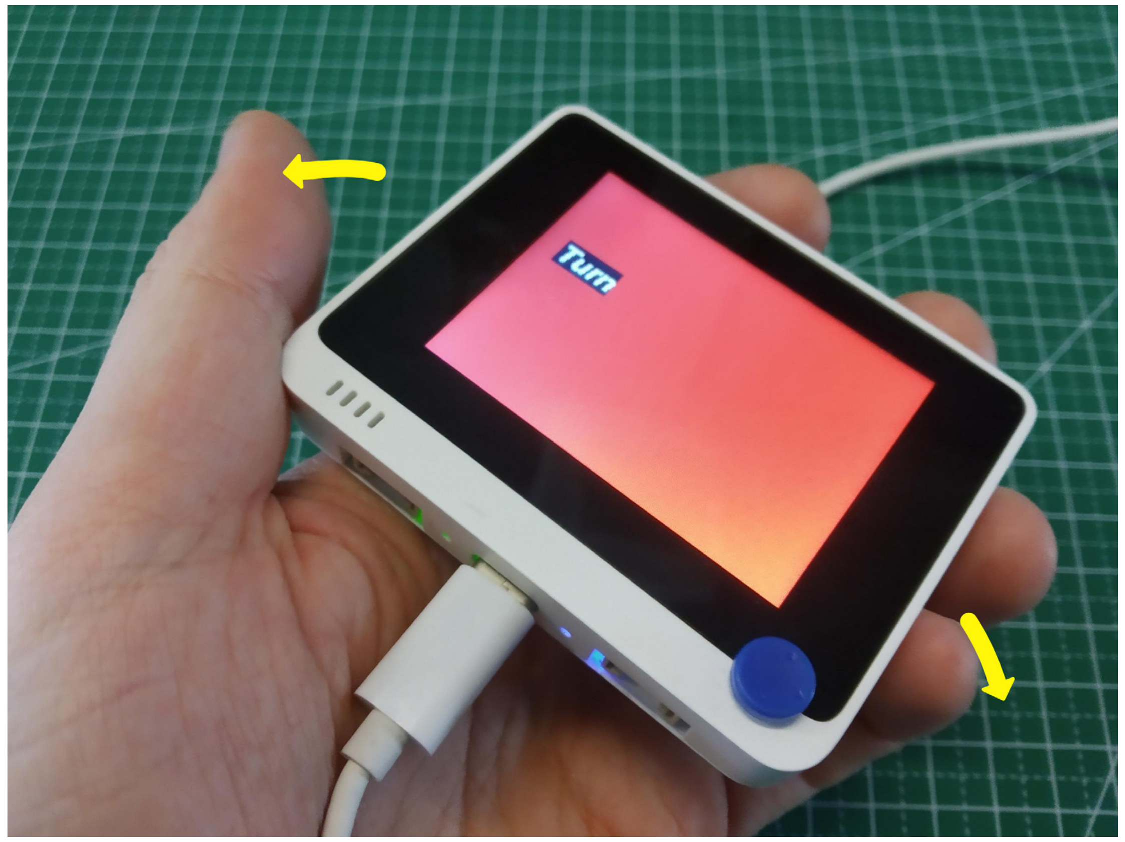
add the following code block, in which we check confidence values of every class and if one of them is higher than threshold, change the color of the screen and display that classes name.

```
if (result.classification[1].value > 0.7) {
    tft.fillScreen(TFT_PURPLE);
    tft.setFreeFont(&FreeSansBoldOblique12pt7b);
    tft.drawString("Shake", 20, 80);
    delay(1000);
    tft.fillScreen(TFT_WHITE);
   }

   if (result.classification[2].value > 0.7) {
    tft.fillScreen(TFT_RED);
    tft.setFreeFont(&FreeSansBoldOblique12pt7b);
    tft.drawString("Turn", 20, 80);
    delay(1000);
    tft.fillScreen(TFT_WHITE);
   }

   if (result.classification[3].value > 0.7) {
    tft.fillScreen(TFT_GREEN);
    tft.setFreeFont(&FreeSansBoldOblique12pt7b);
    tft.drawString("Wave", 20, 80);
    delay(1000);
    tft.fillScreen(TFT_WHITE);
   }
```

Then compile and upload – open the serial monitor and perform one the gestures! You will be able to see the inference results displayed on the Serial monitor and also on LCD screen.







Expansion tasks

Retrain model for other gestures.

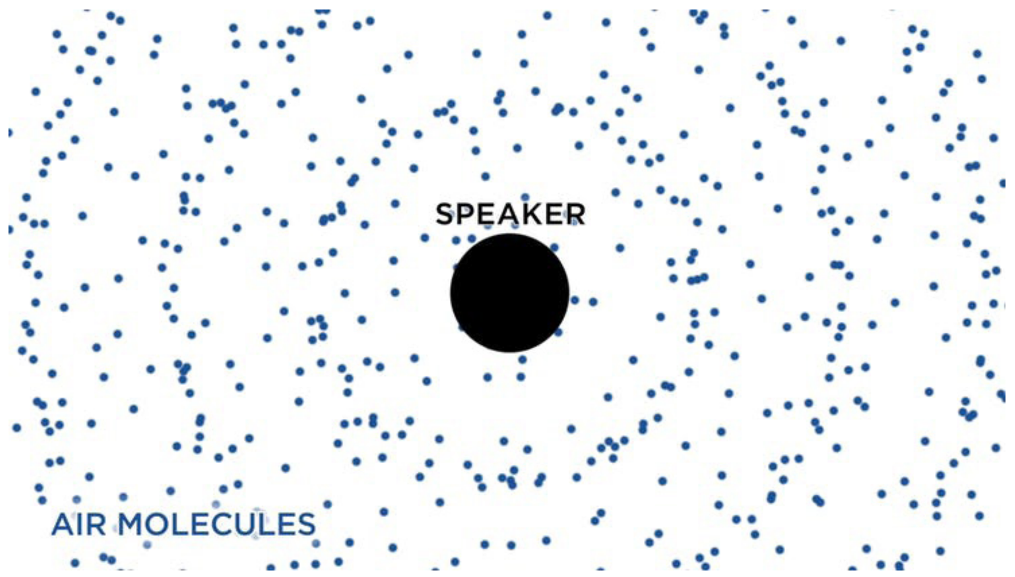
Lesson 06

Project III: Audio scene recognition with microphone: theory and data collection

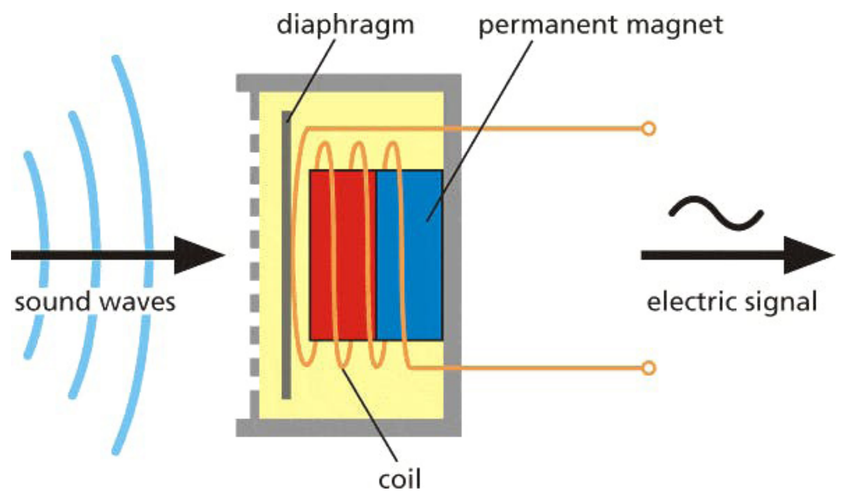
Theory

In this project we will learn how to train and deploy an audio scene classifier with Wio Terminal and Edge Impulse. Audio scene classification is a task, where machine learning model needs to predict a class for audio segment, for example, "a crying baby", "a cough", "a dog barking", etc. Let's learn more about sound processing in computers.

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

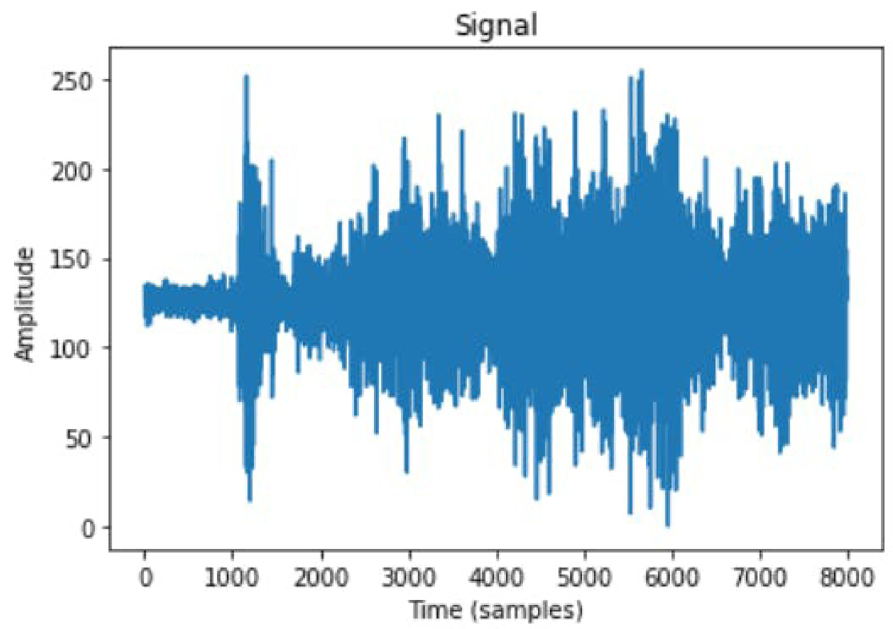


The source of sound pushes the surrounding medium molecules, they push the molecules next to them and so on and so forth. When they reach other object it also vibrates slightly — we use that principle in microphone. The microphone membrane gets pushed inward by the medium molecules and then back to its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

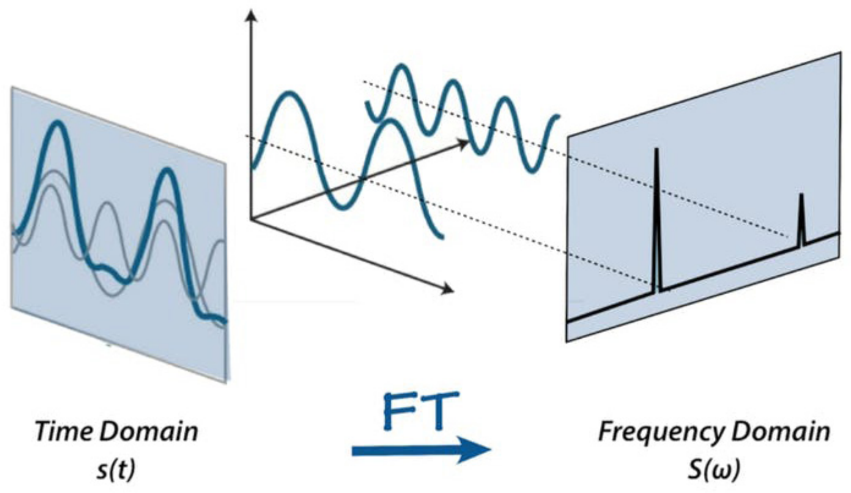


That generates alternating current in the circuit, where voltage is proportional to sound amplitude — the louder the sound, the more it pushes membrane, thus generating higher voltage. We then read this voltage with analog-to-digital converter and record at equal intervals — the number of times we take measurement of sound in one second is called a sampling rate, for

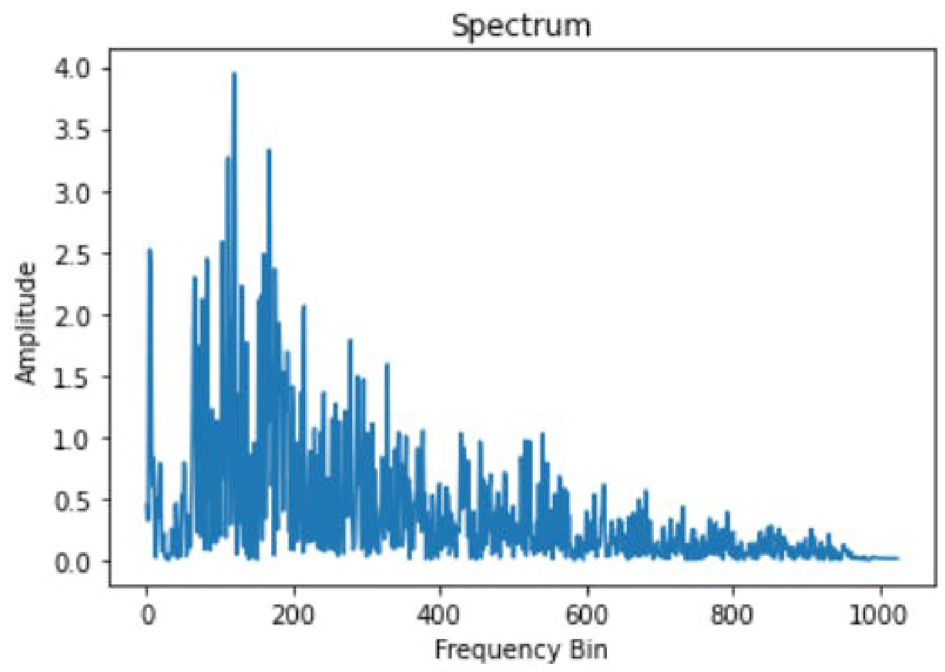
example 8000 Hz sampling rate is taking measurement 8000 times per second. Sampling rate obviously matters a lot for quality of the sound — if we sample too slow we might miss important bits and pieces. The numbers used for recording sound digitally also matter — the larger range of a number used, the 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 is says on the tin — for 8-bit sound an unsigned 8-bit integers are used, which have range from 0 to 255. For 16-bit sound a 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).



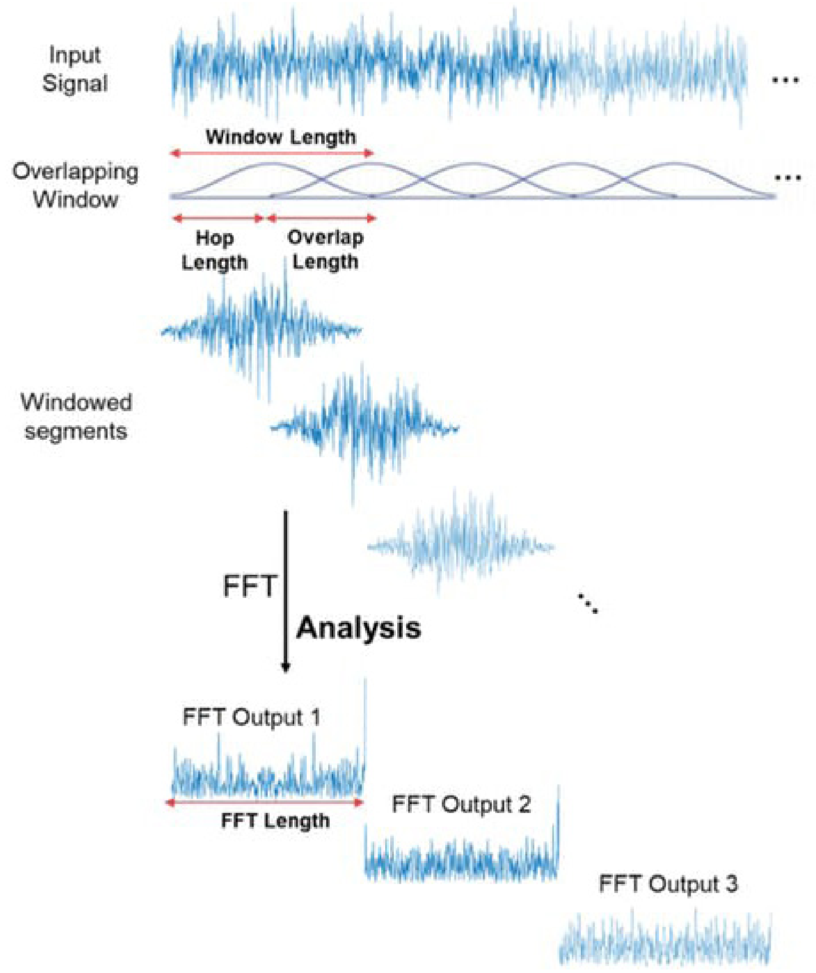
We can't do much with this raw sound representation though — yes, we can cut and paste the parts or make it quilter or louder, but for analyzing the sound, it is, well, too raw. Here is where Fourier transform, Mel scale, spectrograms and cepstrum coefficients come in. For purpose of this project, we'll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



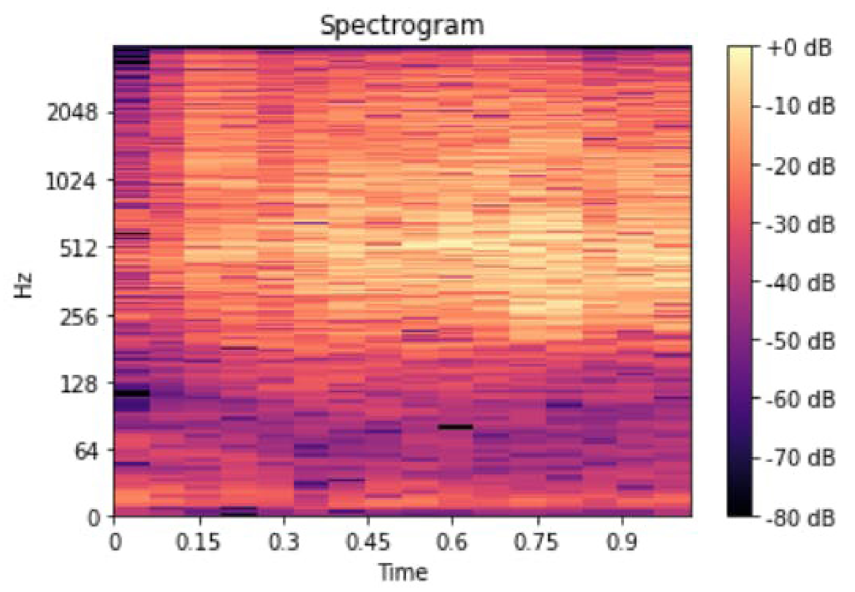
Or, if you'd like to use a metaphor — given the smoothie, it outputs the recipe. That is how our sound looks like after applying Fourier transform — the higher bars correspond to larger amplitude frequencies.



That's great! Now we can do more interesting things with audio signal — for example eliminate the least important frequencies to compress the audio file or remove the noise or maybe the sound of voice, etc. But it is still not good enough for audio and speech recognition — by doing Fourier transform we **lose all the time domain information**, which is not good for non-periodic signals, such as human speech. We are smart cookies though and just take Fourier transform multiple times on the signal sample, essentially slicing it and then stitching the data from multiple Fourier transforms back together in form of spectrogram.



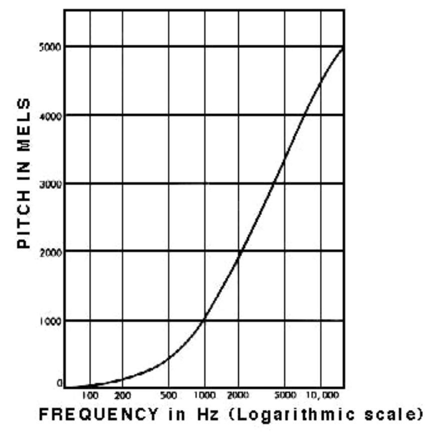
Here x-axis is the time, y-axis is frequency and the amplitude of a frequency is expressed through a color, brighter colors correspond to larger amplitude.



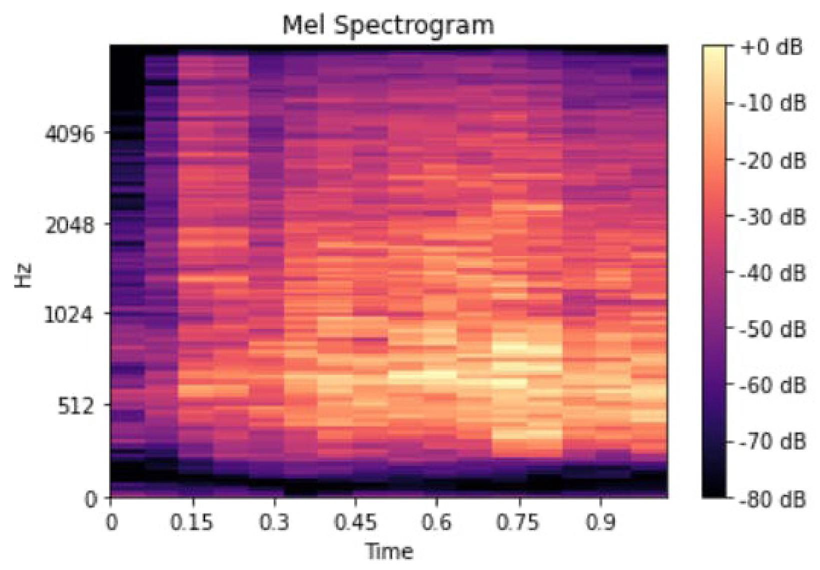
Very well! Can we do sound recognition now? No! Yes! Maybe!

Normal spectrogram contains too much information if we only care about recognizing sounds that human ear can hear. Studies have shown that humans do not perceive frequencies on a linear scale. We are better at detecting differences in lower frequencies than higher frequencies. For example, we can easily tell the difference between 500 and 1000 Hz, but we will hardly be able to tell a difference between 10000 and 10500 Hz, even though the distance between the two pairs are the same.

In 1937, Stevens, Volkmann, and Newmann proposed a unit of pitch such that equal distances in pitch sounded equally distant to the listener. This is called the mel scale.



A mel spectrogram is a spectrogram where the frequencies are converted to the mel scale.



There are more steps involved for recognizing speech — for example cepstrum coefficients, that we mentioned above — we will discuss them in later lessons of the course. It is time to finally start with practical implementation.

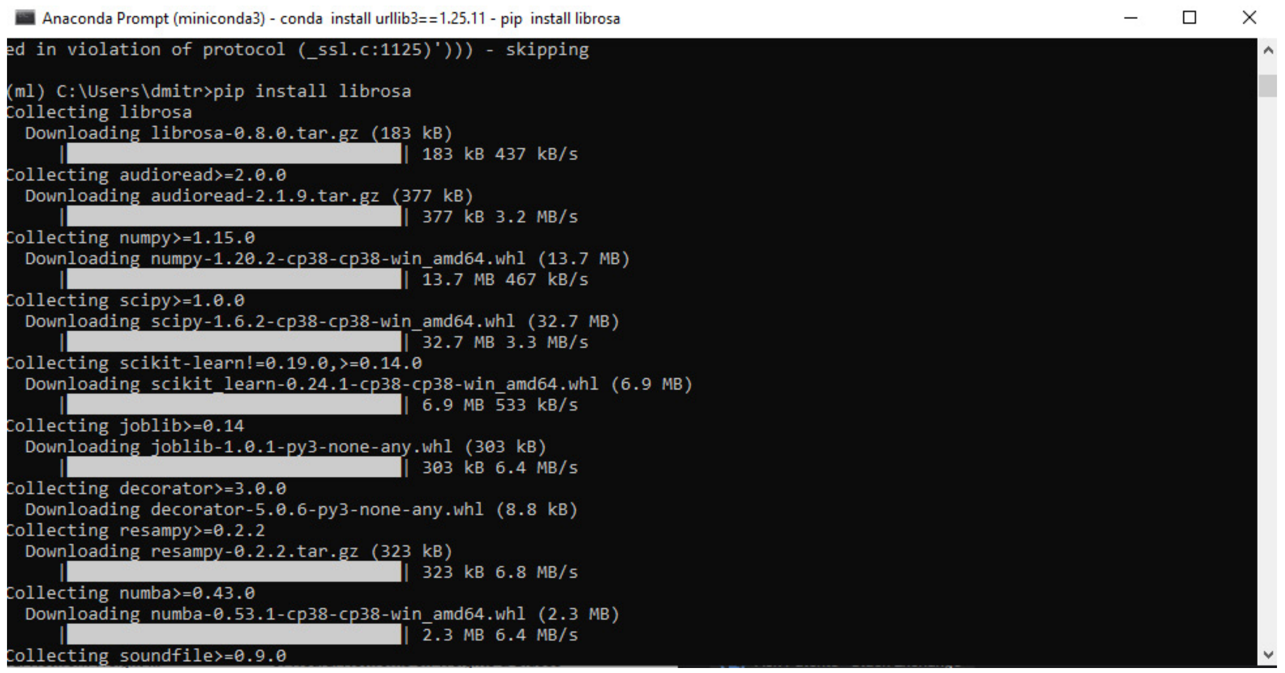
Preparation

Install Anaconda environment manager if you didn't install it in the first lesson – see Lesson Lesson 1 Introduction to TinyML with Wio Terminal for information on how to install Anaconda and create a virtual environment. Then, in a virtual environment install librosa with

`pip install librosa`

and

`conda install –c conda–forge ffmpeg`



Anaconda Prompt (miniconda3) - conda install urllib3==1.25.11 - pip install librosa

```
ed in violation of protocol (_ssl.c:1125)'))) - skipping

(ml) C:\Users\dmitr>pip install librosa
Collecting librosa
  Downloading librosa-0.8.0.tar.gz (183 kB)
     |████████████████████████████████| 183 kB 437 kB/s
Collecting audioread>=2.0.0
  Downloading audioread-2.1.9.tar.gz (377 kB)
     |████████████████████████████████| 377 kB 3.2 MB/s
Collecting numpy>=1.15.0
  Downloading numpy-1.20.2-cp38-cp38-win_amd64.whl (13.7 MB)
     |████████████████████████████████| 13.7 MB 467 kB/s
Collecting scipy>=1.0.0
  Downloading scipy-1.6.2-cp38-cp38-win_amd64.whl (32.7 MB)
     |████████████████████████████████| 32.7 MB 3.3 MB/s
Collecting scikit-learn!=0.19.0,>=0.14.0
  Downloading scikit_learn-0.24.1-cp38-cp38-win_amd64.whl (6.9 MB)
     |████████████████████████████████| 6.9 MB 533 kB/s
Collecting joblib>=0.14
  Downloading joblib-1.0.1-py3-none-any.whl (303 kB)
     |████████████████████████████████| 303 kB 6.4 MB/s
Collecting decorator>=3.0.0
  Downloading decorator-5.0.6-py3-none-any.whl (8.8 kB)
Collecting resampy>=0.2.2
  Downloading resampy-0.2.2.tar.gz (323 kB)
     |████████████████████████████████| 323 kB 6.8 MB/s
Collecting numba>=0.43.0
  Downloading numba-0.53.1-cp38-cp38-win_amd64.whl (2.3 MB)
     |████████████████████████████████| 2.3 MB 6.4 MB/s
Collecting soundfile>=0.9.0
```

Practice

Audio signal needs to be sampled at very high sampling rate, 8000 Hz or, ideally, 16000 Hz. Edge Impulse Data Forwarder tool is too slow to handle this sampling rate, so we will need to use dedicated data collection firmware to get the data for this project. Download a new version of Wio Terminal Edge Impulse firmware with microphone support and flash it to your device, as described in Lesson 4. After that create a new project on Edge Impulse platform, launch edge–impulse ingestion service

```
edge–impulse–daemon
```

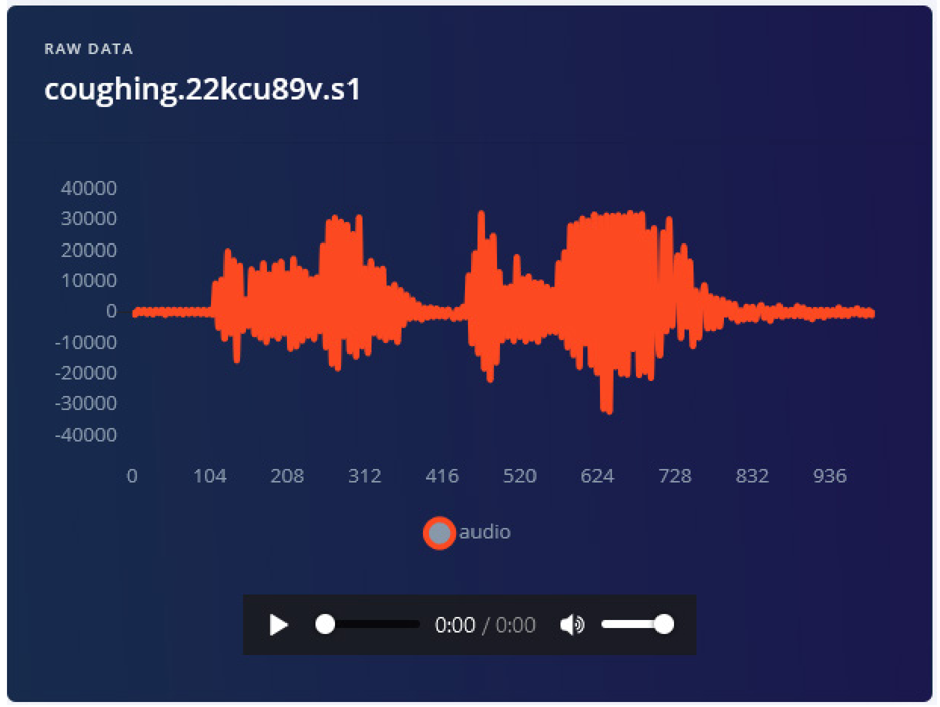
If you used edge–impulse–daemon before, you will need to add ––clean to the command above to clean project data.

Then log in with your credentials and choose a project you have just created. Go to Data Acquisition tab and you can start getting data samples.

We will have three classes of data:

- background
- coughing
- crying

Record 10 samples for each class, 5000 milliseconds duration each. You can recording the sounds played from the computer speakers (except for background class), but if you have the opportunity to record real sounds, that would be even better.

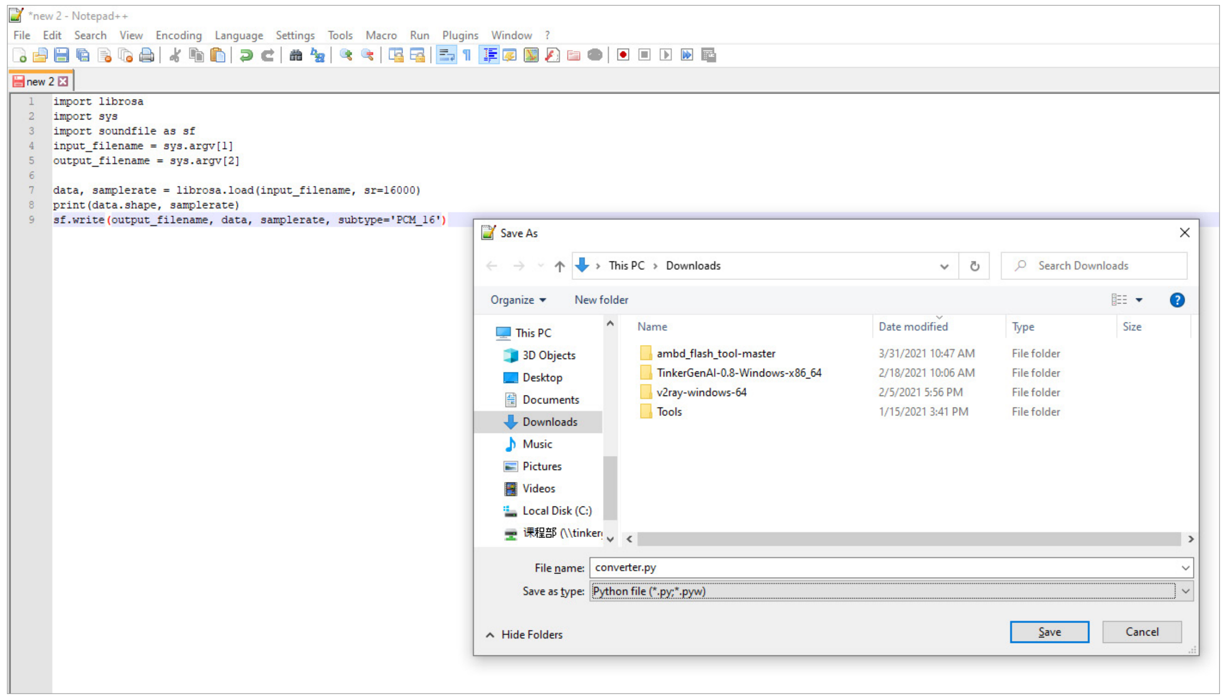


For background class record sounds that should not be classified as coughing or crying, e.g. people talking, no sounds, air conditioning/fan and so on and so forth.

30 samples is abysmally small, so we're also going to upload some more data. Simply download the sounds from the Internet, resample them to 16000 Hz and save them to .wav format with this converter script

```
import librosa
import sys
import soundfile as sf
input_filename = sys.argv[1]
output_filename = sys.argv[2]

data, samplerate = librosa.load(input_filename, sr=16000)
print(data.shape, samplerate)
sf.write(output_filename, data, samplerate, subtype='PCM_16')
```

Copy the code and paste it in a text document (use Notepad++, IDLE IDE or other suitable IDE. Do not use Windows default Notepad).


*new 2 - Notepad++

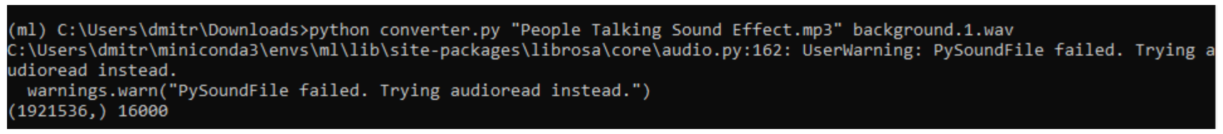
```
import librosa
import sys
import soundfile as sf
input_filename = sys.argv[1]
output_filename = sys.argv[2]

data, samplerate = librosa.load(input_filename, sr=16000)
print(data.shape, samplerate)
sf.write(output_filename, data, samplerate, subtype='PCM_16')
```

Save As — This PC › Downloads

Name	Date modified	Type	Size
ambd_flash_tool-master	3/31/2021 10:47 AM	File folder	
TinkerGenAI-0.8-Windows-x86_64	2/18/2021 10:06 AM	File folder	
v2ray-windows-64	2/5/2021 5:56 PM	File folder	
Tools	1/15/2021 3:41 PM	File folder	

File name: converter.py
Save as type: Python file (*.py;*.pyw)

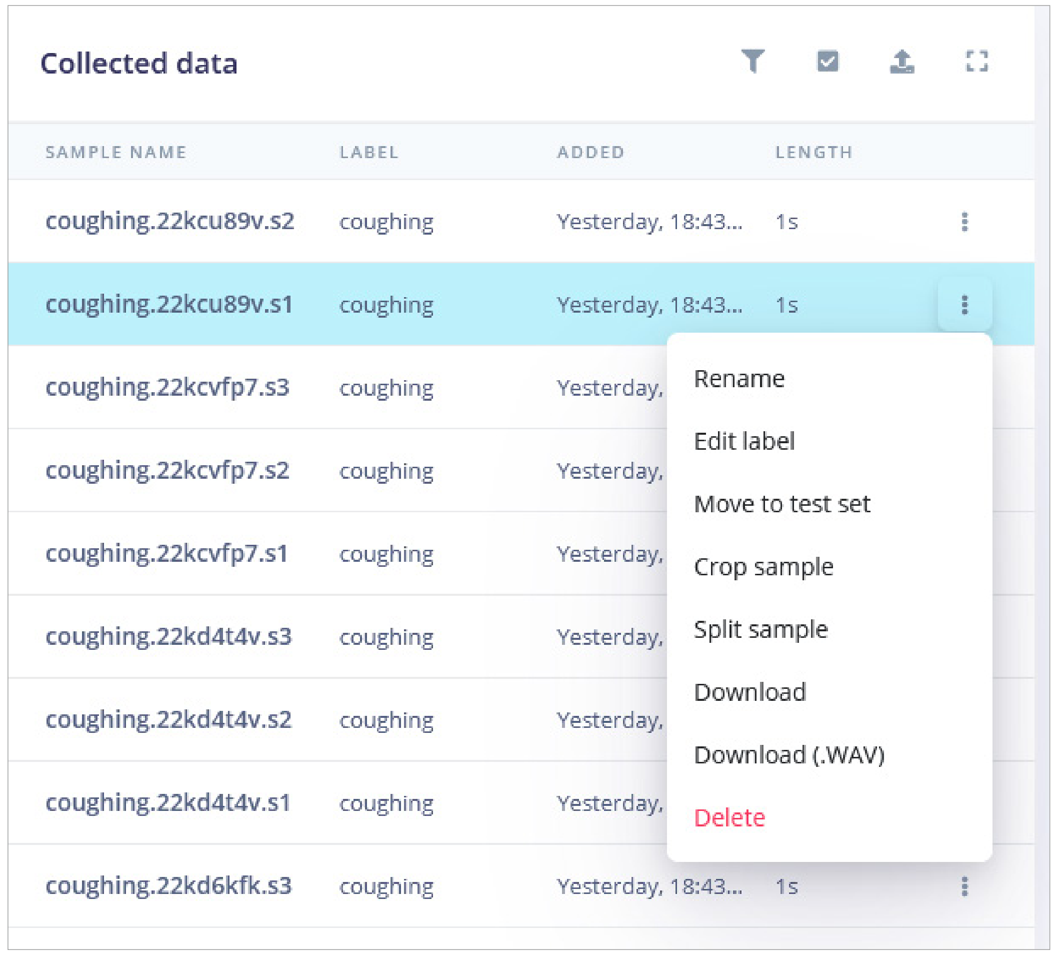
Save document as converter.py and then from Anaconda environment run
python converter.py name–of–the–downloaded–file class_name.number.wav



```
(ml) C:\Users\dmitr\Downloads>python converter.py "People Talking Sound Effect.mp3" background.1.wav
C:\Users\dmitr\miniconda3\envs\ml\lib\site-packages\librosa\core\audio.py:162: UserWarning: PySoundFile failed. Trying a
udioread instead.
  warnings.warn("PySoundFile failed. Trying audioread instead.")
(1921536,) 16000
```

You can find example sound files already converted to right format in materials for this course.

Then split all the sound samples to leave only "interesting" pieces — do that for every class, except for background.


Collected data

SAMPLE NAME	LABEL	ADDED	LENGTH
coughing.22kcu89v.s2	coughing	Yesterday, 18:43...	1s
coughing.22kcu89v.s1	coughing	Yesterday, 18:43...	1s
coughing.22kcvfp7.s3	coughing	Yesterday,	
coughing.22kcvfp7.s2	coughing	Yesterday,	
coughing.22kcvfp7.s1	coughing	Yesterday,	
coughing.22kd4t4v.s3	coughing	Yesterday,	
coughing.22kd4t4v.s2	coughing	Yesterday,	
coughing.22kd4t4v.s1	coughing	Yesterday,	
coughing.22kd6kfk.s3	coughing	Yesterday, 18:43...	1s

After the data collection is done, it is time to choose processing blocks and define our neural network model.

Expansion tasks

Record in different environments (outside on the street, inside of the classroom, etc).

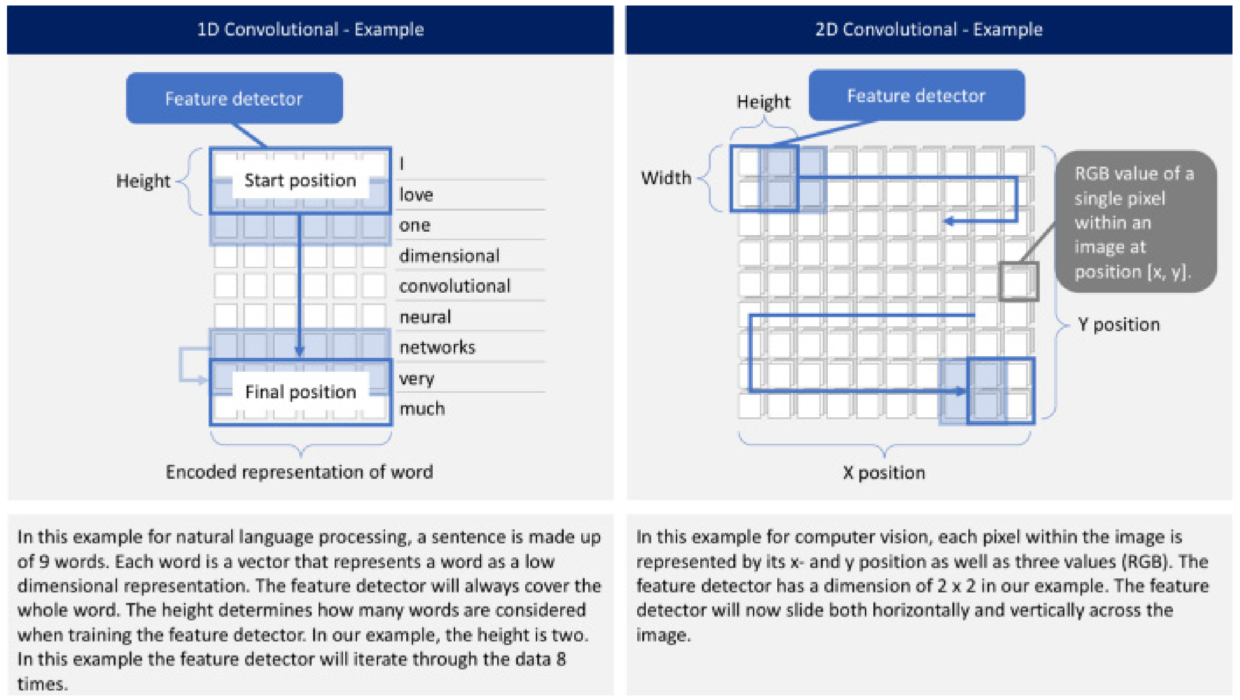
Lesson 07

Project III: Audio scene recognition with microphone: model training and deployment

Theory

Convolutional layers are the major building blocks used in convolutional neural networks.

A convolution is the simple application of a filter to an input that results in an activation. Repeated application of the same filter to an input results in a map of activations called a feature map, indicating the locations and strength of a detected feature in an input, such as a sequence of data (1D), an image(2d) or a point cloud(3D).



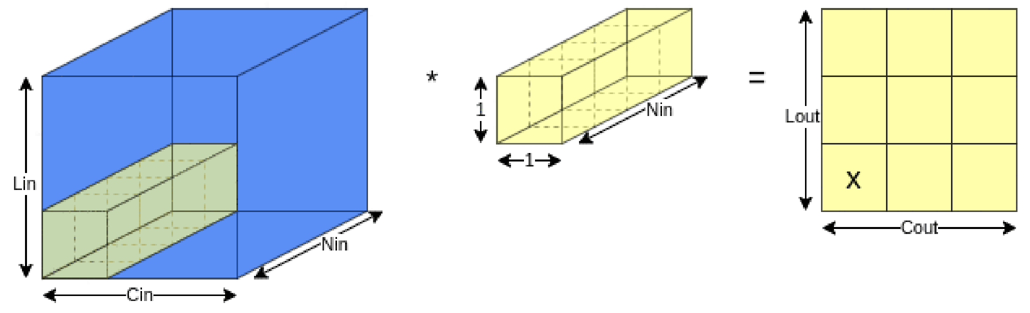
In this example for natural language processing, a sentence is made up of 9 words. Each word is a vector that represents a word as a low dimensional representation. The feature detector will always cover the whole word. The height determines how many words are considered when training the feature detector. In our example, the height is two. In this example the feature detector will iterate through the data 8 times.

In this example for computer vision, each pixel within the image is represented by its x- and y position as well as three values (RGB). The feature detector has a dimension of 2 x 2 in our example. The feature detector will now slide both horizontally and vertically across the image.

"1D versus 2D CNN"by Nils Ackermann is licensed under Creative Commons CC BY-ND 4.0

The innovation of convolutional neural networks is the ability to automatically learn a large number of filters in parallel specific to a training dataset under the constraints of a specific predictive modeling problem, such as image classification. The result is highly specific features that can be detected anywhere on input data.

A CNN works well for identifying simple patterns within your data which will then be used to form more complex patterns within higher layers. A 1D CNN is very effective when you expect to derive interesting features from shorter (fixed-length) segments of the overall data set and where the location of the feature within the segment is not of high relevance.



Preparation

For expansion task, make sure that Wio Terminal has latest WiFi firmware and WiFi libraries – you can check it by uploading the following code

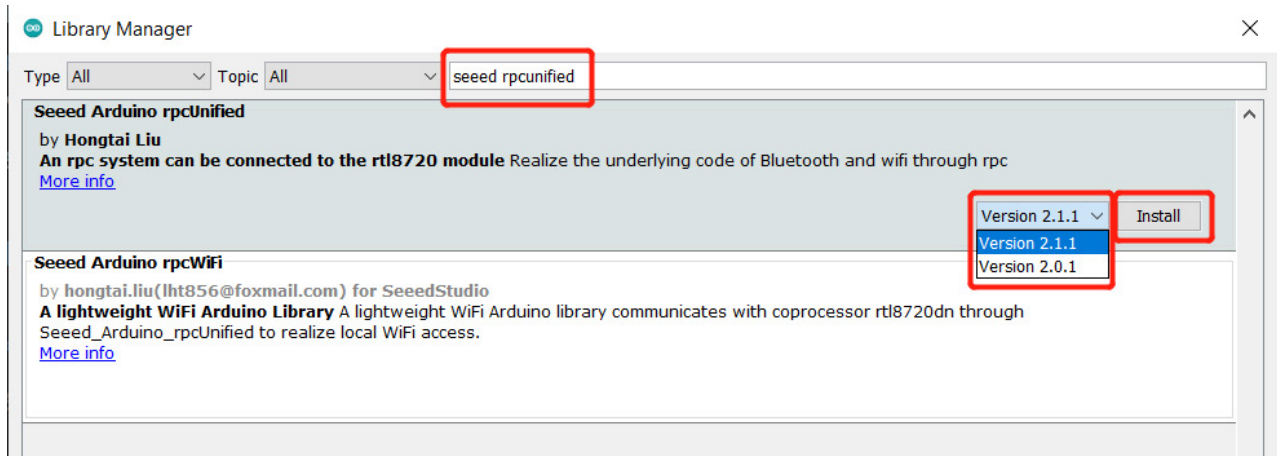
```
#include "rpcWiFi.h"

void setup() {
    Serial.begin(115200);
    while(!Serial); // Wait to open Serial Monitor
    Serial.printf("RTL8720 Firmware Version: %s", rpc_system_version());
}

void loop() {
}
```

Then open Serial monitor and you should see firmware version displayed as output on the screen. If there is no output, install latest WiFi library.

STEP 1: Open the **Arduino IDE**, and click Sketch –> Include Library –> Manage Libraries...

STEP 2: Type the **name of the library** that we need and select the **latest version** from the drop-down menu (if available)

STEP 3: Click **Install**



Seeed Arduino rpcUnified
by **Hongtai Liu**
An rpc system can be connected to the rtl8720 module Realize the underlying code of Bluetooth and wifi through rpc
More info

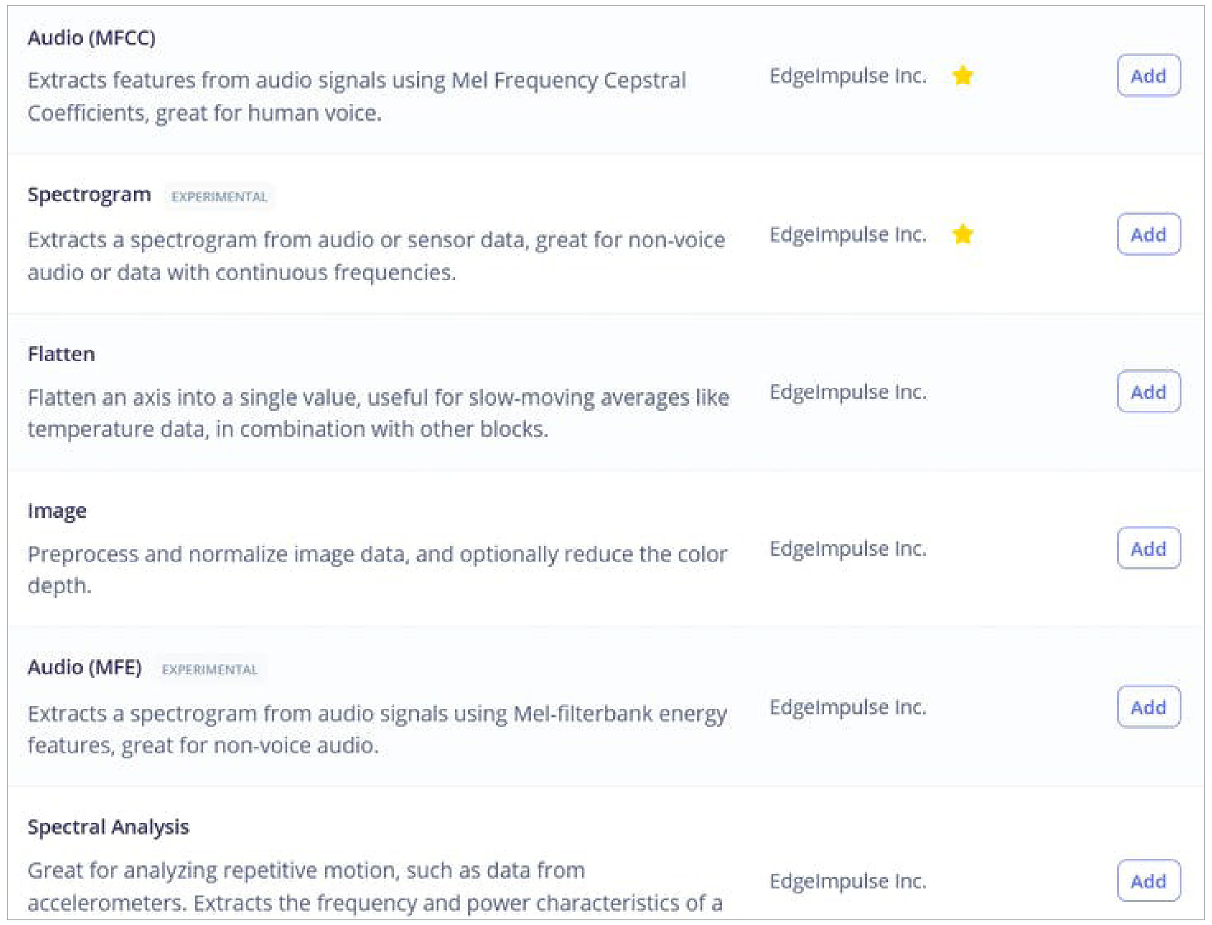
Seeed Arduino rpcWiFi
by **hongtai.liu(lht856@foxmail.com) for SeeedStudio**
A lightweight WiFi Arduino Library A lightweight WiFi Arduino library communicates with coprocessor rtl8720dn through Seeed_Arduino_rpcUnified to realize local WiFi access.
More info

Also install the following libraries –y ou can search for these libraries by typing the library name in the search box of Arduino Library Manager.

- Seeed_Arduino_rpcWiFi – search for "seeed rpcwifi"
- Seeed_Arduino_rpcUnified – search for "seeed rpcunified"
- Seeed_Arduino_mbedtls – search for "seeed mbedtls"
- Seeed_Arduino_FS – search for "seeed fs"
- Seeed_Arduino_SFUD – search for "seeed sfud"

Practice

Among the processing blocks we see three familiar options — namely Raw, Spectral Analysis, which is essentially Fourier transform of the signal, Spectrogram and MFE (Mel–Frequency Energy banks) — which correspond to four stages of audio processing we described earlier!



Audio (MFCC)
Extracts features from audio signals using Mel Frequency Cepstral Coefficients, great for human voice. — EdgeImpulse Inc. ★ — Add

Spectrogram EXPERIMENTAL
Extracts a spectrogram from audio or sensor data, great for non-voice audio or data with continuous frequencies. — EdgeImpulse Inc. ★ — Add

Flatten
Flatten an axis into a single value, useful for slow-moving averages like temperature data, in combination with other blocks. — EdgeImpulse Inc. — Add

Image
Preprocess and normalize image data, and optionally reduce the color depth. — EdgeImpulse Inc. — Add

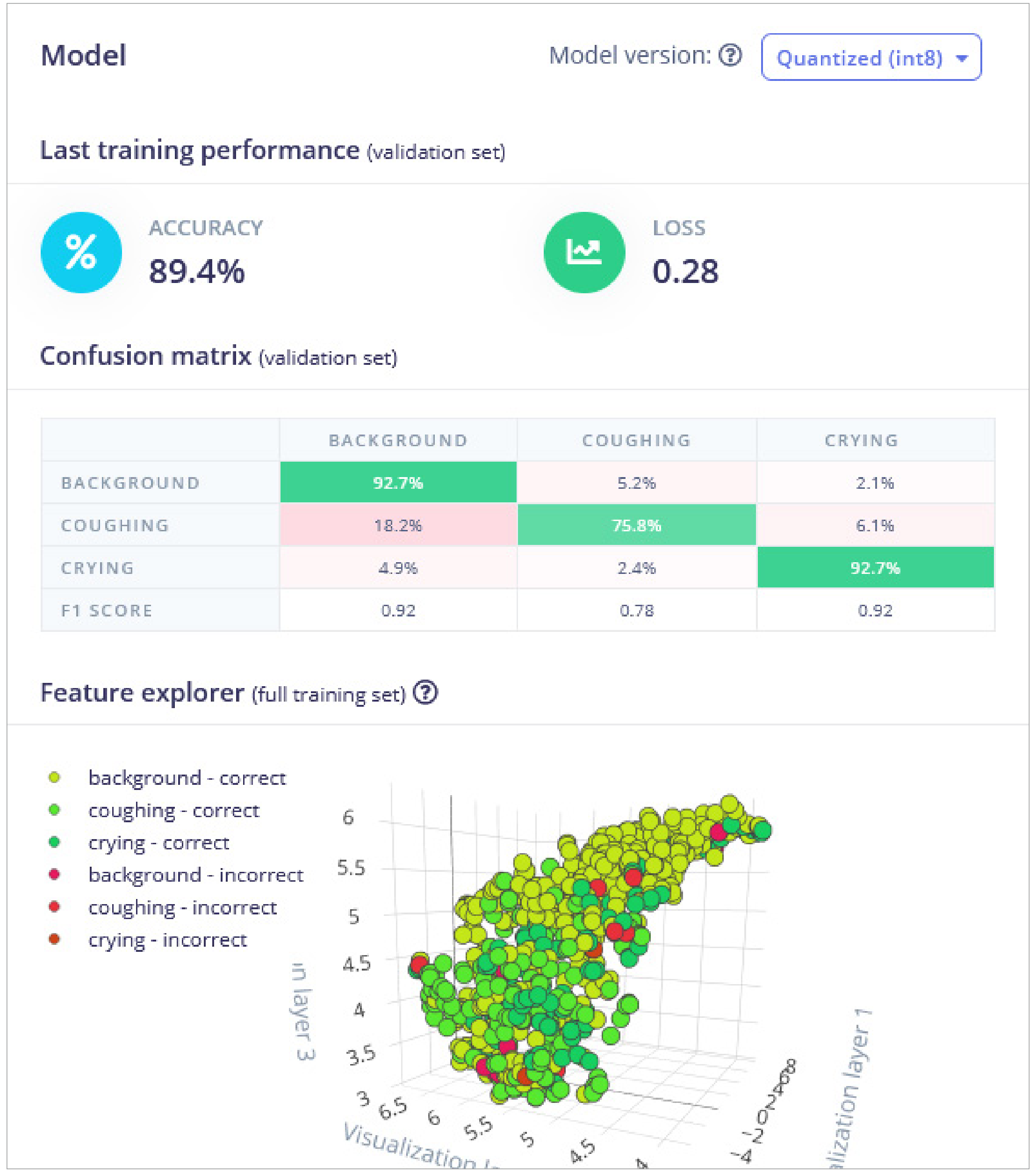
Audio (MFE) EXPERIMENTAL
Extracts a spectrogram from audio signals using Mel-filterbank energy features, great for non-voice audio. — EdgeImpulse Inc. — Add

Spectral Analysis
Great for analyzing repetitive motion, such as data from accelerometers. Extracts the frequency and power characteristics of a — EdgeImpulse Inc. — Add

If you like experimenting, you can try using all of them on your data, except for maybe Raw, which will have too much data for our small–ish neural network. For the purpose of this lesson we will just go with the best option for this task, which is MFE or Mel–Frequency Energy banks. After computing the features, go to NN classifier tab and choose a suitable model architecture. The two choices we have are using 1D Conv and 2D Conv. Both will work, but If possible, we should always go for smaller model, since we will want to deploy it to embedded device. When writing the materials for this course we ran 4 different experiments, 1D Conv/2D Conv with MFE and MFCC features and the results for them are in this table.

	MFE	MFCC
Conv1D	89.4%	84.7%
Conv2D	88.8%	81.2%

The best model was 1D Conv network with MFE processing block. By tweaking MFE parameters (namely increasing stride to 0.02 and decreasing low frequency to 0) we have achieved accuracy of 89.4% on validation dataset.



Model Model version: ⓘ Quantized (int8) ▾

Last training performance (validation set)

ACCURACY 89.4%

LOSS 0.28

Confusion matrix (validation set)

	BACKGROUND	COUGHING	CRYING
BACKGROUND	92.7%	5.2%	2.1%
COUGHING	18.2%	75.8%	6.1%
CRYING	4.9%	2.4%	92.7%
F1 SCORE	0.92	0.78	0.92

Feature explorer (full training set) ⓘ

You can find the trained model in the course materials and test it out yourself. While it is good at distinguishing crying sounds from background, **coughing** sound detection accuracy is a bit low low and might require obtaining more samples.

After we have our model and satisfied with its accuracy in training, we can test it on new data in Live classification tab and then Deploy it to Wio terminal. We'll download it as Arduino library, put it in Arduino libraries folder and open Examples –> name of your project –> nano_33_ble_sense_microphone_continuous. The demo is based on Arduino Nano 33 BLE and uses PDM library. 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 inference buffer without involvement of MCU.

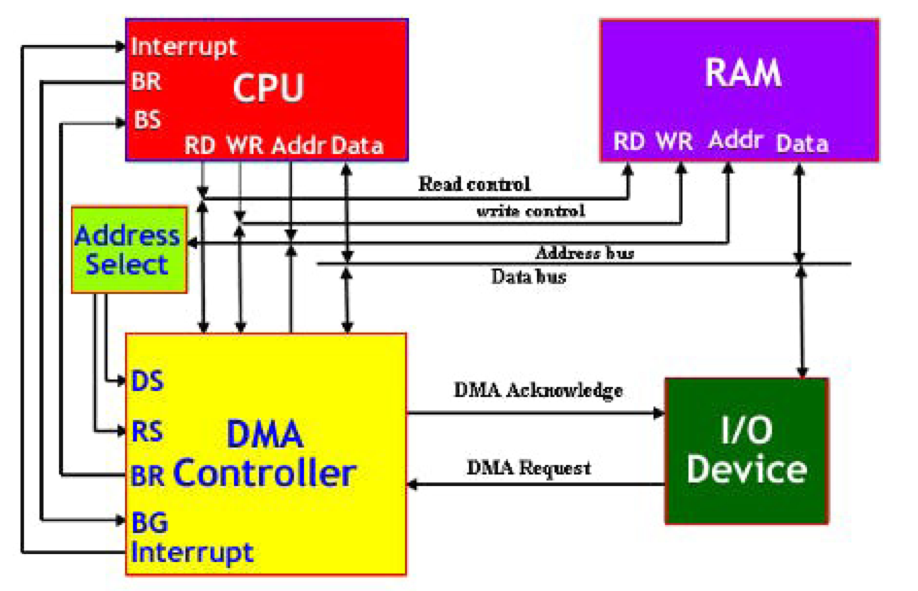


Fig: Showing DMA Mode of Data Transfer

That will allow us to collect the sound samples and perform inference at the same time. There is quite a few changes we need to make in order to change sound data collection from PDM library to DMA, if you feel a bit lost during the explanation, have a look at the full sample code, which you can find in the course materials.

Delete PDM library from the sketch

```
#include <PDM.h>
```

Add DMA descriptor structure, and other settings constants right after last include statement

```
// Settings
#define DEBUG 1                  // Enable pin pulse during ISR
enum {ADC_BUF_LEN = 4096};    // Size of one of the DMA double buffers
static const int debug_pin = 1; // Toggles each DAC ISR (if DEBUG is set to 1)

// DMAC descriptor structure
typedef struct {
  uint16_t btctrl;
  uint16_t btcnt;
  uint32_t srcaddr;
  uint32_t dstaddr;
  uint32_t descaddr;
} dmacdescriptor;
```

Then right before setup function create variables for buffer arrays, volatile variables for passing the values between ISR callback and the main code and also High pass Butterworth filter, which we will apply to signal to eliminate most of DC component in microphone signal.

```
// Globals – DMA and ADC
volatile uint8_t recording = 0;
volatile boolean results0Ready = false;
volatile boolean results1Ready = false;
uint16_t adc_buf_0[ADC_BUF_LEN];    // ADC results array 0
uint16_t adc_buf_1[ADC_BUF_LEN];    // ADC results array 1
volatile dmacdescriptor wrb[DMAC_CH_NUM] __attribute__ ((aligned (16)));          // Write-back DMAC descriptors
dmacdescriptor descriptor_section[DMAC_CH_NUM] __attribute__ ((aligned (16)));    // DMAC channel descriptors
dmacdescriptor descriptor __attribute__ ((aligned (16)));                          // Place holder descriptor

//High pass butterworth filter order=1 alpha1=0.0125
class  FilterBuHp1
{
  public:
    FilterBuHp1()
    {
      v[0]=0.0;
    }
  private:
    float v[2];
  public:
    float step(float x) //class II
    {
      v[0] = v[1];
      v[1] = (9.621952458291035404e-1f * x)
         + (0.92439049165820696974f * v[0]);
      return
         (v[1] - v[0]);
    }
};

FilterBuHp1 filter;
```

Add three blocks of code after that – the first one is a callback function, called by ISR (Interrupt Service Routine) every time one of the two buffers filled. Inside that function we read elements from recording buffer (the one that was filled just now), process them and put into an inference buffer.

```
/****************************************************************************
 * Interrupt Service Routines (ISRs)
 */

/**
 * @brief     Copy sample data in selected buf and signal ready when buffer is full
 *
 * @param[in]  *buf  Pointer to source buffer
 * @param[in]  buf_len  Number of samples to copy from buffer
 */
static void audio_rec_callback(uint16_t *buf, uint32_t buf_len) {

  static uint32_t idx = 0;

  // Copy samples from DMA buffer to inference buffer
  if (recording) {
    for (uint32_t i = 0; i < buf_len; i++) {

      // Convert 12-bit unsigned ADC value to 16-bit PCM (signed) audio value
      inference.buffers[inference.buf_select][inference.buf_count++] = filter.step(((int16_t)
buf[i] - 1024) * 16);
      // Swap double buffer if necessary
      if (inference.buf_count >= inference.n_samples) {
        inference.buf_select ^= 1;
        inference.buf_count = 0;
        inference.buf_ready = 1;
      }
    }
  }
}
```

Next block contains the ISR itself – it is executed by a timer at certain period of time, inside of that function we check if DMAC channel 1 has been suspended – if it has been suspended it means that one of the buffers for microphone data has filled and we need to copy the data from it, switch to another buffer and restart DMAC ADC.

```
/**
 * Interrupt Service Routine (ISR) for DMAC 1
 */
void DMAC_1_Handler() {
```

```

  static uint8_t count = 0;

  // Check if DMAC channel 1 has been suspended (SUSP)
  if (DMAC->Channel[1].CHINTFLAG.bit.SUSP) {

     // Debug: make pin high before copying buffer
#if DEBUG
    digitalWrite(debug_pin, HIGH);
#endif

    // Restart DMAC on channel 1 and clear SUSP interrupt flag
    DMAC->Channel[1].CHCTRLB.reg = DMAC_CHCTRLB_CMD_RESUME;
    DMAC->Channel[1].CHINTFLAG.bit.SUSP = 1;

    // See which buffer has filled up, and dump results into large buffer
    if (count) {
      audio_rec_callback(adc_buf_0, ADC_BUF_LEN);
    } else {
      audio_rec_callback(adc_buf_1, ADC_BUF_LEN);
    }


    // Flip to next buffer
    count = (count + 1) % 2;

    // Debug: make pin low after copying buffer
#if DEBUG
    digitalWrite(debug_pin, LOW);
#endif
  }
}
```

Next block contains configuration data for ADC DMAC and timer controlling ISR (interrupt Service Routine)

```
// Configure DMA to sample from ADC at regular interval
void config_dma_adc() {

  // Configure DMA to sample from ADC at a regular interval (triggered by timer/
counter)
  DMAC->BASEADDR.reg = (uint32_t)descriptor_section;                // Specify the location of the descriptors
  DMAC->WRBADDR.reg = (uint32_t)wrb;                                // Specify the location of the write back descriptors
  DMAC->CTRL.reg = DMAC_CTRL_DMAENABLE | DMAC_CTRL_LVLEN(0xf);      // Enable the DMAC peripheral
  DMAC->Channel[1].CHCTRLA.reg = DMAC_CHCTRLA_TRIGSRC(TC5_DMAC_ID_OVF) |      // Set DMAC to trigger on TC5 timer overflow
                                 DMAC_CHCTRLA_TRIGACT_BURST;        // DMAC burst transfer

  descriptor.descaddr = (uint32_t)&descriptor_section[1];           // Set up a circular descriptor
  descriptor.srcaddr = (uint32_t)&ADC1->RESULT.reg;                 // Take the result from the ADC0 RESULT register
  descriptor.dstaddr = (uint32_t)adc_buf_0 + sizeof(uint16_t) * ADC_BUF_LEN;  // Place it in the adc_buf_0 array
  descriptor.btcnt = ADC_BUF_LEN;                                   // Beat count
  descriptor.btctrl = DMAC_BTCTRL_BEATSIZE_HWORD |                  // Beat size is HWORD (16-bits)
                      DMAC_BTCTRL_DSTINC |                          // Increment the destination address
                      DMAC_BTCTRL_VALID |                           // Descriptor is valid
                      DMAC_BTCTRL_BLOCKACT_SUSPEND;                 // Suspend DMAC channel 0 after block transfer
  memcpy(&descriptor_section[0], &descriptor, sizeof(descriptor));  // Copy the descriptor to the descriptor section

  descriptor.descaddr = (uint32_t)&descriptor_section[0];           // Set up a circular descriptor
  descriptor.srcaddr = (uint32_t)&ADC1->RESULT.reg;                 // Take the result from the ADC0 RESULT register
  descriptor.dstaddr = (uint32_t)adc_buf_1 + sizeof(uint16_t) * ADC_BUF_LEN;  // Place it in the adc_buf_1 array
  descriptor.btcnt = ADC_BUF_LEN;                                   // Beat count
  descriptor.btctrl = DMAC_BTCTRL_BEATSIZE_HWORD |                  // Beat size is HWORD (16-bits)
                      DMAC_BTCTRL_DSTINC |                          // Increment the destination address
                      DMAC_BTCTRL_VALID |                           // Descriptor is valid
                      DMAC_BTCTRL_BLOCKACT_SUSPEND;                 // Suspend DMAC channel 0 after block transfer
  memcpy(&descriptor_section[1], &descriptor, sizeof(descriptor));  // Copy the descriptor to the descriptor section

  // Configure NVIC
  NVIC_SetPriority(DMAC_1_IRQn, 0);    // Set the Nested Vector Interrupt Controller (NVIC) priority for DMAC1 to 0 (highest)
  NVIC_EnableIRQ(DMAC_1_IRQn);         // Connect DMAC1 to Nested Vector Interrupt Controller (NVIC)

  // Activate the suspend (SUSP) interrupt on DMAC channel 1
  DMAC->Channel[1].CHINTENSET.reg = DMAC_CHINTENSET_SUSP;

  // Configure ADC
  ADC1->INPUTCTRL.bit.MUXPOS = ADC_INPUTCTRL_MUXPOS_AIN12_Val; // Set the analog input to ADC0/AIN2 (PB08 - A4 on Metro M4)
  while(ADC1->SYNCBUSY.bit.INPUTCTRL);                // Wait for synchronization
  ADC1->SAMPCTRL.bit.SAMPLEN = 0x00;                  // Set max Sampling Time Length to half divided ADC clock pulse (2.66us)
  while(ADC1->SYNCBUSY.bit.SAMPCTRL);                 // Wait for synchronization
  ADC1->CTRLA.reg = ADC_CTRLA_PRESCALER_DIV128;       // Divide Clock ADC GCLK by 128 (48MHz/128 = 375kHz)
  ADC1->CTRLB.reg = ADC_CTRLB_RESSEL_12BIT |          // Set ADC resolution to 12 bits
                    ADC_CTRLB_FREERUN;                // Set ADC to free run mode
  while(ADC1->SYNCBUSY.bit.CTRLB);                    // Wait for synchronization
  ADC1->CTRLA.bit.ENABLE = 1;                         // Enable the ADC
  while(ADC1->SYNCBUSY.bit.ENABLE);                   // Wait for synchronization
  ADC1->SWTRIG.bit.START = 1;                         // Initiate a software trigger to start an ADC conversion
  while(ADC1->SYNCBUSY.bit.SWTRIG);                   // Wait for synchronization

  // Enable DMA channel 1
  DMAC->Channel[1].CHCTRLA.bit.ENABLE = 1;

  // Configure Timer/Counter 5
  GCLK->PCHCTRL[TC5_GCLK_ID].reg = GCLK_PCHCTRL_CHEN |        // Enable perhipheral channel for TC5
```

```
                          GCLK_PCHCTRL_GEN_GCLK1;   // Connect generic clock 0 at 48MHz

    TC5->COUNT16.WAVE.reg = TC_WAVE_WAVEGEN_MFRQ;               // Set TC5 to Match Frequency (MFRQ) mode
    TC5->COUNT16.CC[0].reg = 3000 - 1;                          // Set the trigger to 16 kHz: (4Mhz / 16000) - 1
    while (TC5->COUNT16.SYNCBUSY.bit.CC0);                      // Wait for synchronization

    // Start Timer/Counter 5
    TC5->COUNT16.CTRLA.bit.ENABLE = 1;                          // Enable the TC5 timer
    while (TC5->COUNT16.SYNCBUSY.bit.ENABLE);                   // Wait for synchronization
}
```

Add the debug condition on top of the setup function:

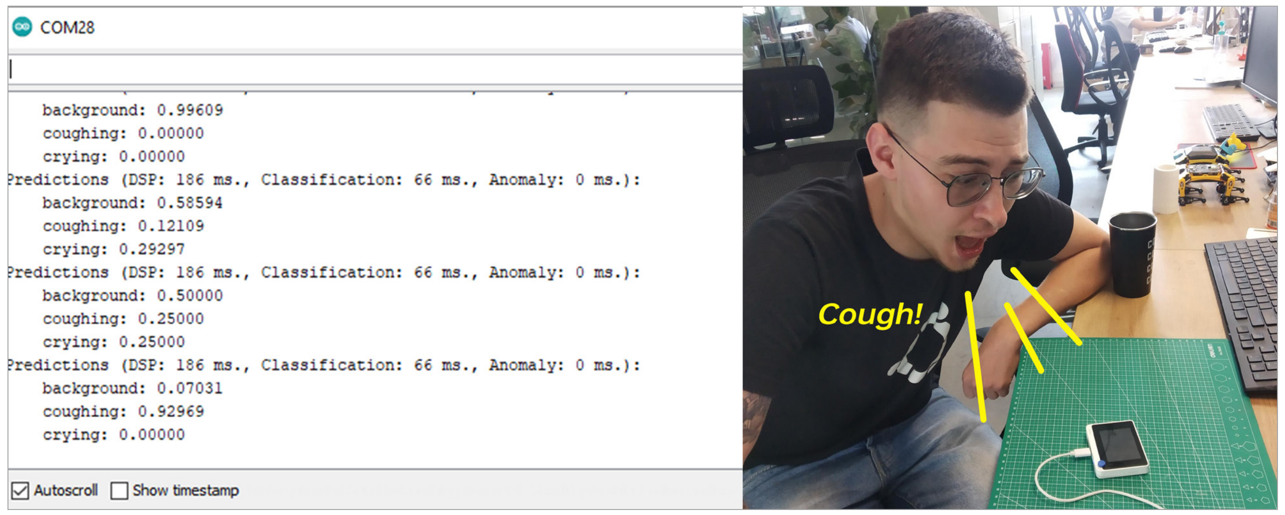
```
// Configure pin to toggle on DMA interrupt
#if DEBUG
  pinMode(debug_pin, OUTPUT);
#endif
```

Then in the setup function, after run_classifier_init(); add the following code that creates inference buffers, configures DMA and starts the recording by setting volatile global variable recording to 1.

```
 // Create double buffer for inference
  inference.buffers[0] = (int16_t *)malloc(EI_CLASSIFIER_SLICE_SIZE * sizeof(int16_t));

  if (inference.buffers[0] == NULL) {
    ei_printf("ERROR: Failed to create inference buffer 0");
    return;
  }
  inference.buffers[1] = (int16_t *)malloc(EI_CLASSIFIER_SLICE_SIZE *
      sizeof(int16_t));
  if (inference.buffers[1] == NULL) {
    ei_printf("ERROR: Failed to create inference buffer 1");
    free(inference.buffers[0]);
    return;
  }

  // Set inference parameters
  inference.buf_select = 0;
  inference.buf_count = 0;
  inference.n_samples = EI_CLASSIFIER_SLICE_SIZE;
  inference.buf_ready = 0;

  // Configure DMA to sample from ADC at 16kHz (start sampling immediately)
  config_dma_adc();

  // Start recording to inference buffers
  recording = 1;
}
```

Delete PDM.end(); and free(sampleBuffer); from microphone_inference_end(void) function and also microphone_inference_start(uint32_t n_samples) and pdm_data_ready_inference_callback(void) functions, since we're not using them.

After compiling and uploading the code, open the Serial monitor and you will see probabilities for every classes printed out. Play some sounds or cough at Wio Terminal to check the accuracy!



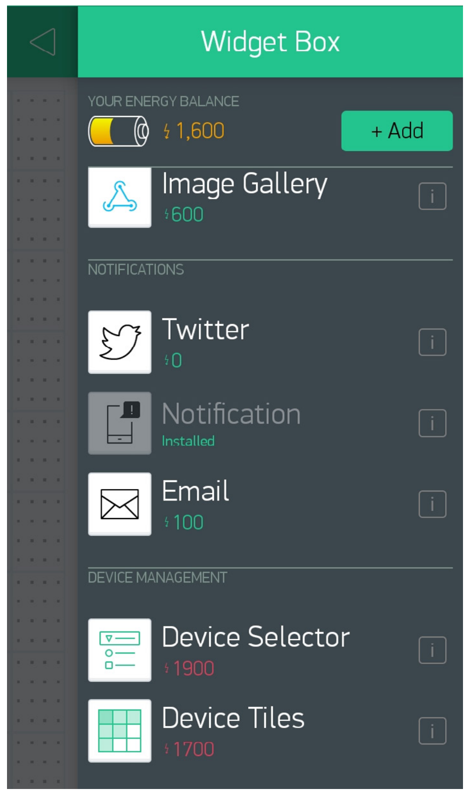
```
    background: 0.99609
    coughing: 0.00000
    crying: 0.00000
Predictions (DSP: 186 ms., Classification: 66 ms., Anomaly: 0 ms.):
    background: 0.58594
    coughing: 0.12109
    crying: 0.29297
Predictions (DSP: 186 ms., Classification: 66 ms., Anomaly: 0 ms.):
    background: 0.50000
    coughing: 0.25000
    crying: 0.25000
Predictions (DSP: 186 ms., Classification: 66 ms., Anomaly: 0 ms.):
    background: 0.07031
    coughing: 0.92969
    crying: 0.00000
```

Expansion tasks

Since WioTerminal can connect to the Internet, we can take this simple demo and make it into a real IoT application with Blynk.

Blynk is a platform that allows you to quickly build interfaces for controlling and monitoring your hardware projects from your iOS and Android devices. In this case we will use Blink to push notification to our smartphone if Wio Terminal detects any sounds we should worry about.

To [get started with Blink](), download the app, register a new account and create a new project. Add a push notification element to it and press play button.



Widget Box

YOUR ENERGY BALANCE
1,600 + Add

Image Gallery
600

NOTIFICATIONS

Twitter
0

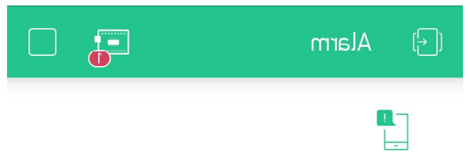
Notification
Installed

Email
100

DEVICE MANAGEMENT

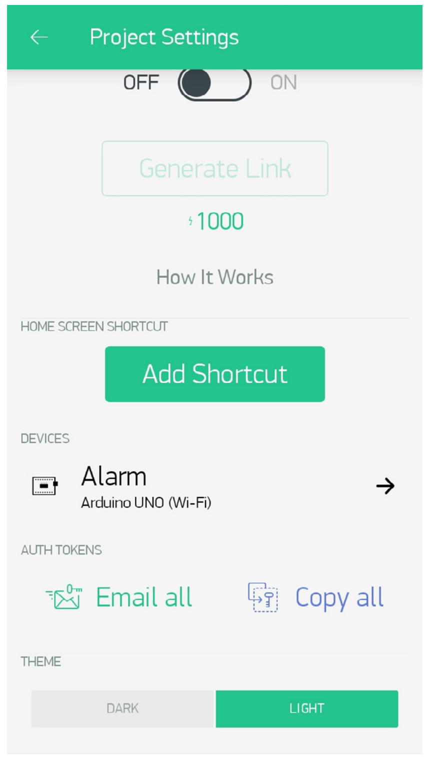
Device Selector
1900

Device Tiles
1700



Then test your setup by compiling and uploading simple push button example — make sure you change WiFi SSID, password and your Blynk API token, which you can get in the app.

```
#define BLYNK_PRINT Serial
#include <rpcWiFi.h>
#include <WiFiClient.h>
#include <BlynkSimpleWioTerminal.h>
char auth[] = ”token”;
char ssid[] = ”ssid”;
char pass[] = ”password”;
void checkPin()
{
  int isButtonPressed = !digitalRead(WIO_KEY_A);
  if (isButtonPressed) {
    Serial.println(”Button is pressed.”);
    Blynk.notify(”Yaaay... button is pressed!”);
  }
}
void setup()
{
  Serial.begin(115200);
  Blynk.begin(auth, ssid, pass);
  pinMode(WIO_KEY_A, INPUT_PULLUP);
}
void loop()
{
  Blynk.run();
  checkPin();
}
```



Project Settings

OFF ON

Generate Link

1000

How It Works

HOME SCREEN SHORTCUT

Add Shortcut

DEVICES

Alarm
Arduino UNO (Wi-Fi)

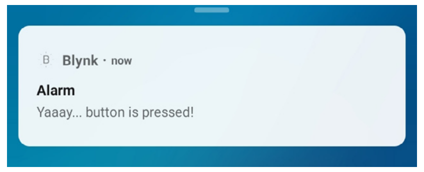
AUTH TOKENS

Email all Copy all

THEME

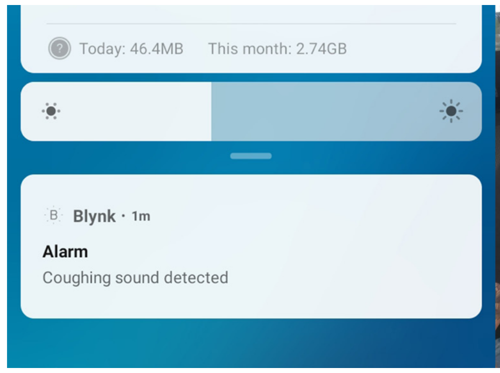
DARK LIGHT

If code compiles and the test is successful (pressing top left button on Wio Terminal causes a push notification to appear on your phone), then you can move to the next stage.



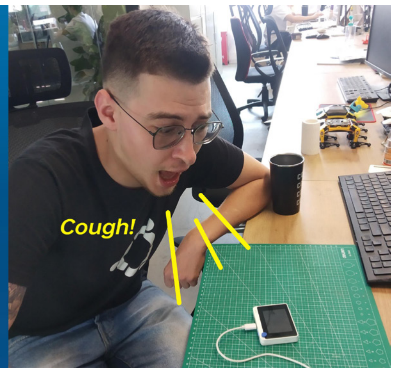
Blynk · now
Alarm
Yaaay... button is pressed!

We’re going to move all the neural network inference code in a separate function and call it in the loop() function right after Blynk.run(). Similar to what we did before, we check the neural network prediction probabilities and if they are higher than threshold for a certain class, we call Blynk.notify() function, which as you might have guessed pushes a notification to your mobile device.



Today: 46.4MB This month: 2.74GB

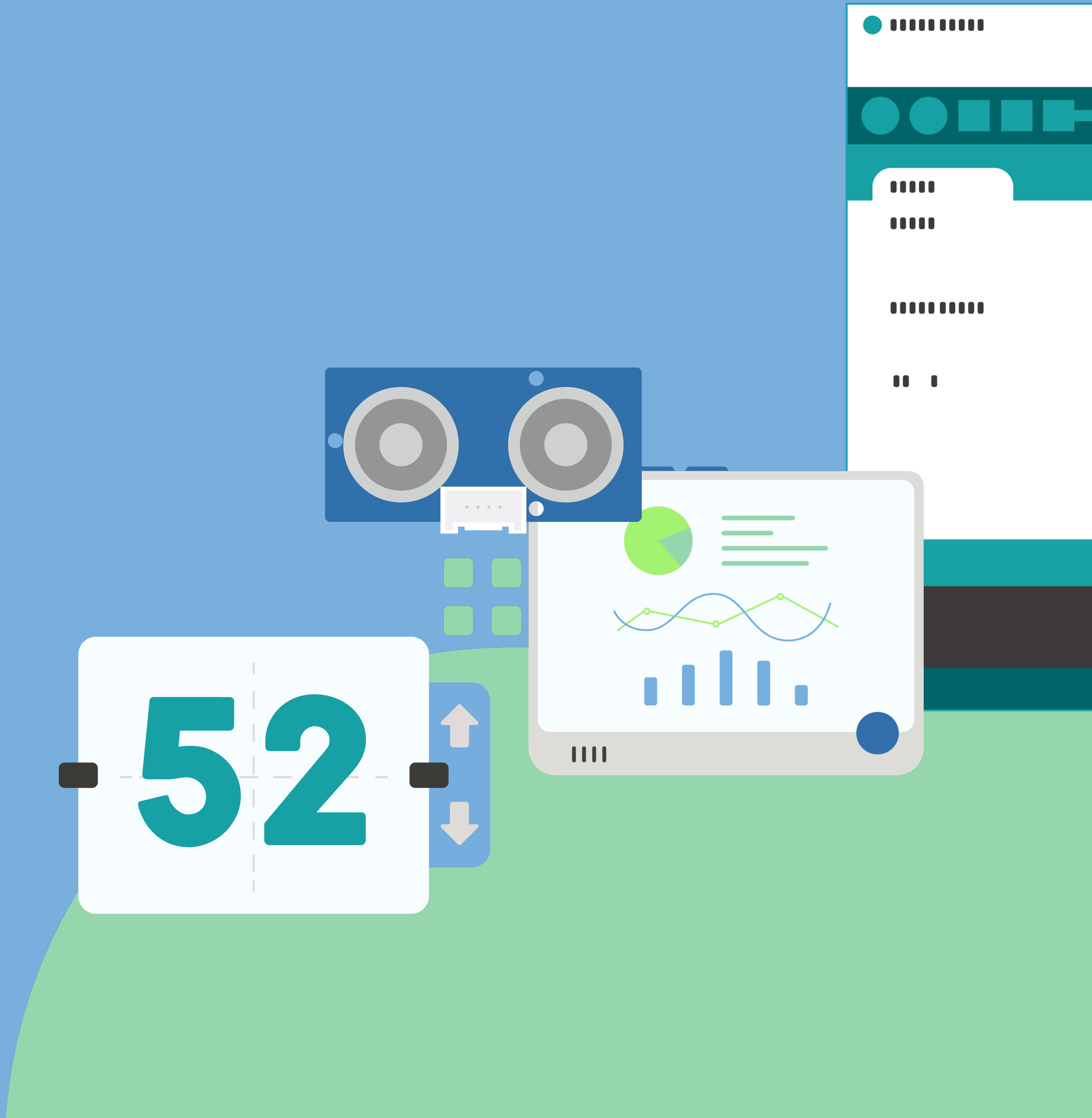
Blynk · 1m
Alarm
Coughing sound detected



Find the full code for NN inference + Blynk notification in course materials.

Lesson 08

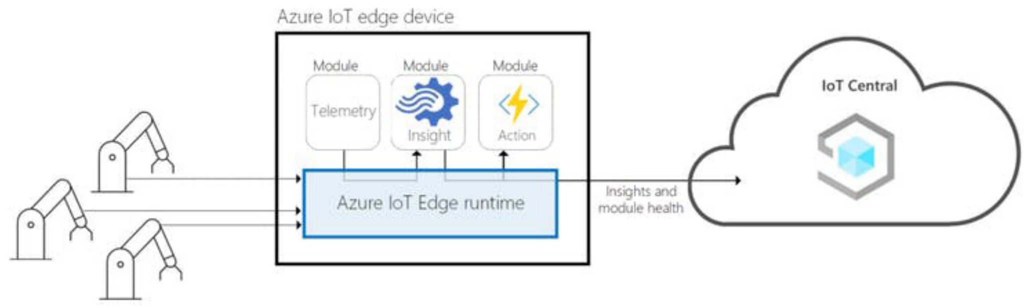
Project IV: People counting with Ultrasonic sensor: theory and data collection



Theory

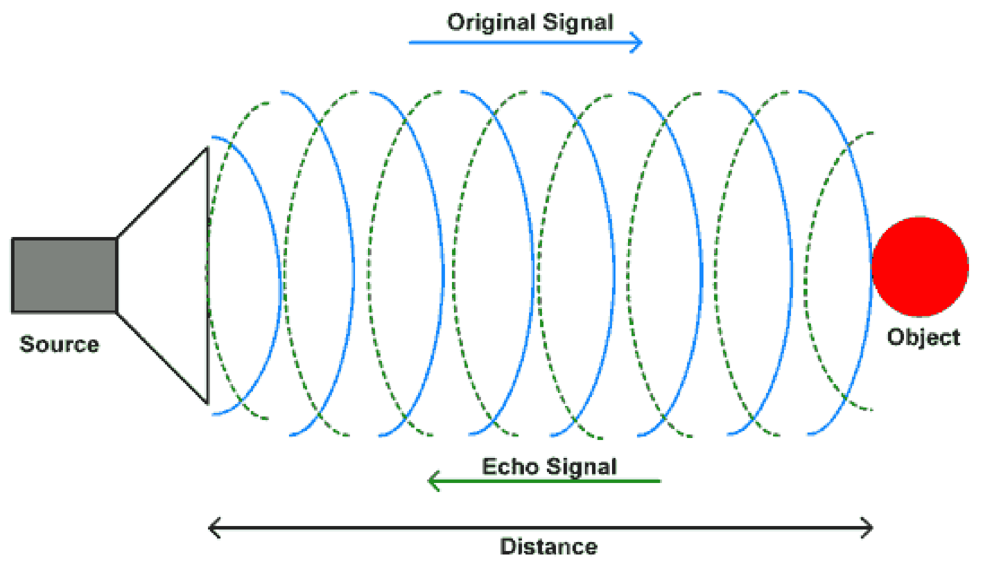
In this article we will create a people counting system by using Wio Terminal, an ordinary Ultrasonic ranger and special Deep Learning sauce to top it off and actually make it work.

We will also utilize Microsoft Azure IoT Central service to store the room occupancy data in the cloud and visualize it on PC.



First, let's understand the data we can get from 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 which works at 40KHz. When we provide a pulse trigger signal with more than 10uS through 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.



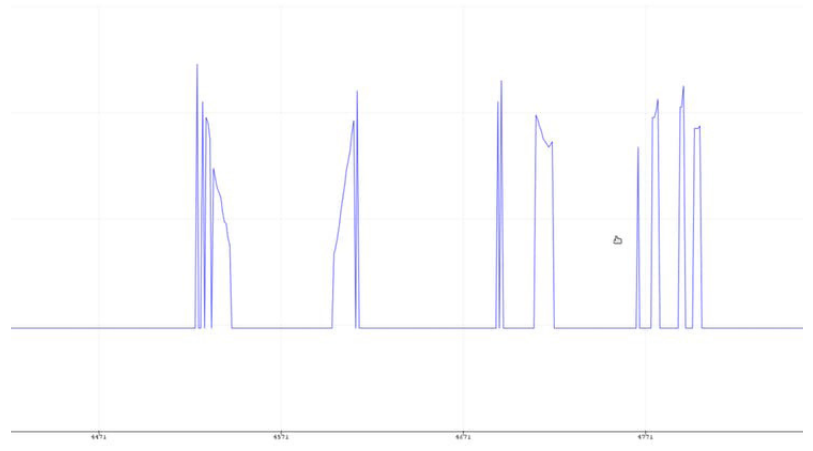
Warning

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

After installing Grove – Ultrasonic Ranger library for Arduino IDE and connecting Ultrasonic Ranger to Wio Terminal D1/D2 port, we can upload this simple script to Wio Terminal connected to Grove Ultrasonic Ranger and then walk in and walk out of the room.

```
#include "Ultrasonic.h"
#define INTERVAL_MS 50
Ultrasonic ultrasonic(0);
void setup() {
    Serial.begin(115200);
    }
void loop() {
    static unsigned long last_interval_ms = 0;
    float distance;
    if (millis() > last_interval_ms + INTERVAL_MS) {
        last_interval_ms = millis();
        distance = ultrasonic.MeasureInCentimeters();
        if (distance < 200.0) {
        Serial.println(distance);
        }
        else
        Serial.println(-1);
        //Serial.print('\t');
    }
}
```

We can immediately see that for walking it, 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.

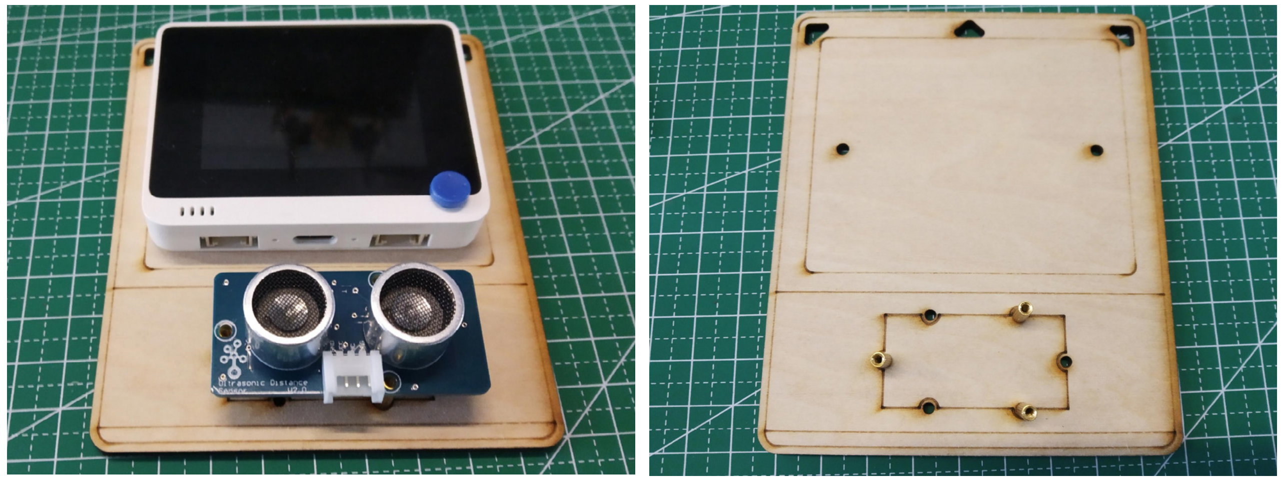
Preparation

Install Grove – Ultrasonic Ranger library to Arduino IDE.

Step 1. Download the UltrasonicRanger Library from Github.

Step 2. Extract the archive and place it inside your libraries folder.

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.

Practice

Let's create a new project in Edge Impulse Dashboard and prepare to get the data. For gathering the data, since we don't need very high sampling frequency, we can use data forwarder tool from edge-impulse-cli. Upload the **ei_people_counter_data_collection.ino** script (exactly the same script as pasted above) to Wio Terminal — the following steps assume that you have already installed Edge Impulse CLI as described in Lesson 1.

In this particular script we filter out all the values above 200 cm, setting them to -1.

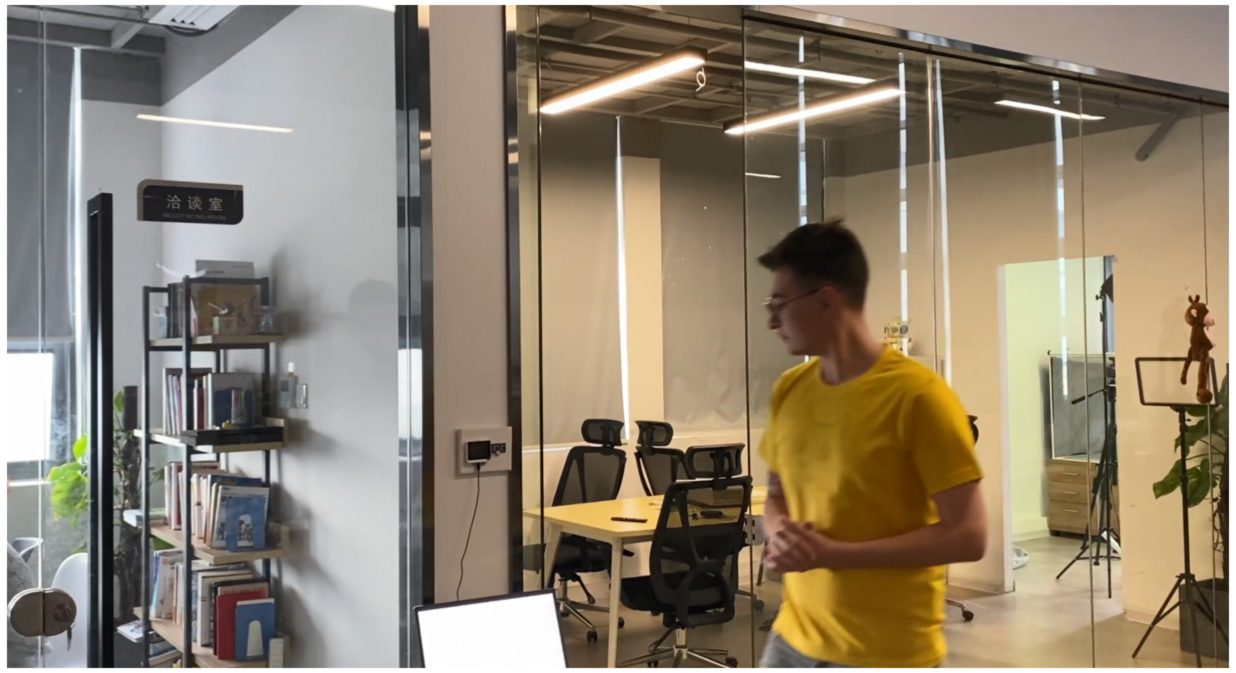
```
if (distance < 200.0) {
    Serial.println(distance);
}
else {
    Serial.println(-1);
}
```

For your application you might need to set this value lower or higher, depending on the set up. Then start walking.

Walking in

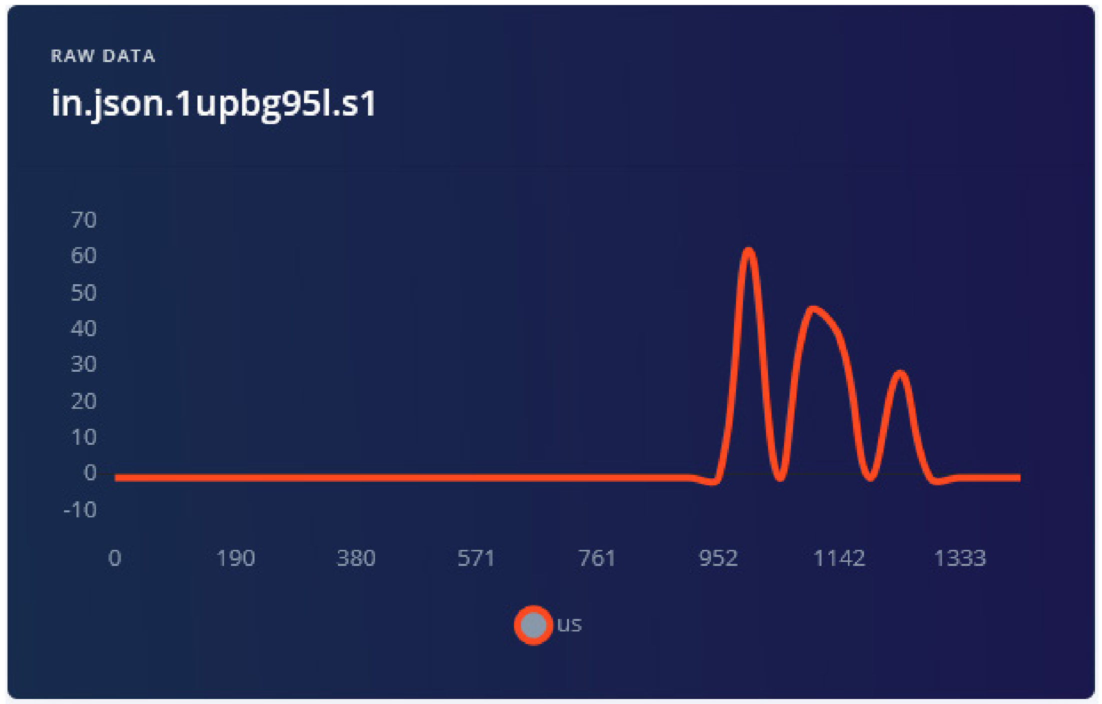
Walking out

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

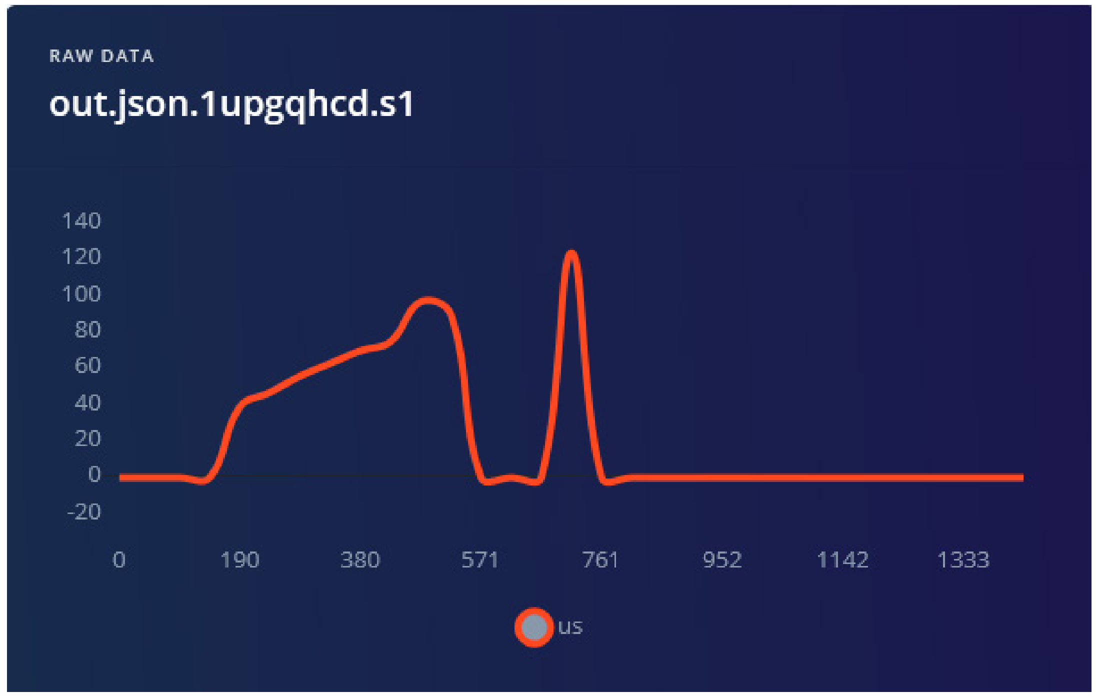


For this lesson we recorded 1 minute 30 seconds of data for every class, each time recording 5000 ms samples and then cropping them to get 1500 ms samples — remember that variety is very important in the dataset, so make sure you have samples where you (or other people) walk fast, slow, run, etc.

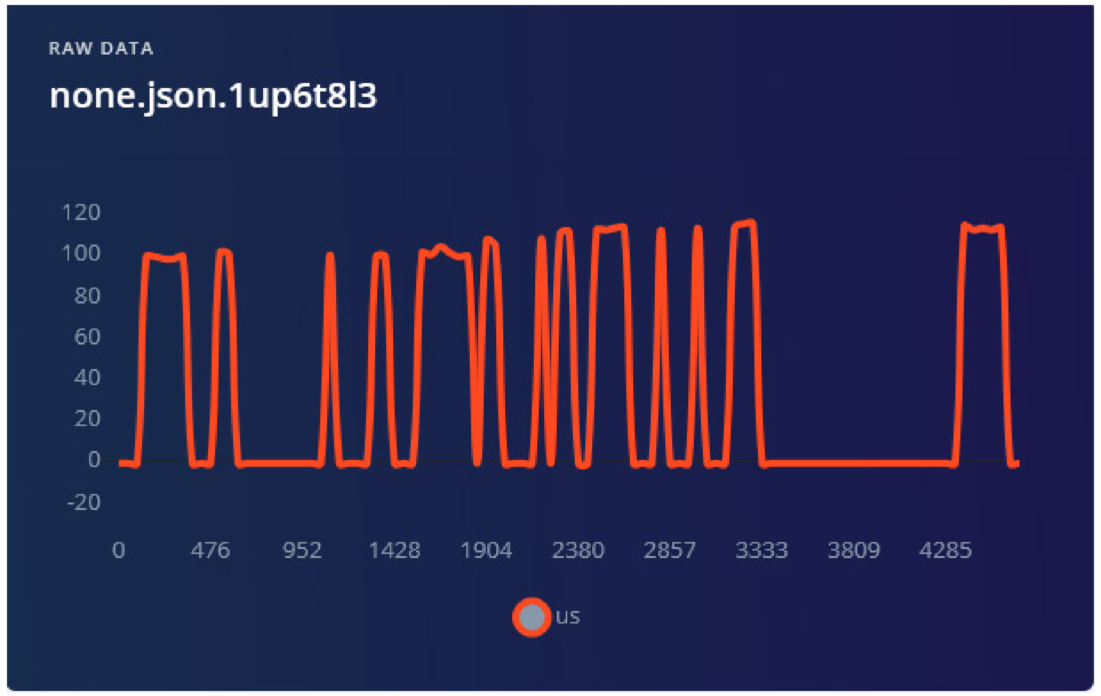
Walking in



Walking out



None



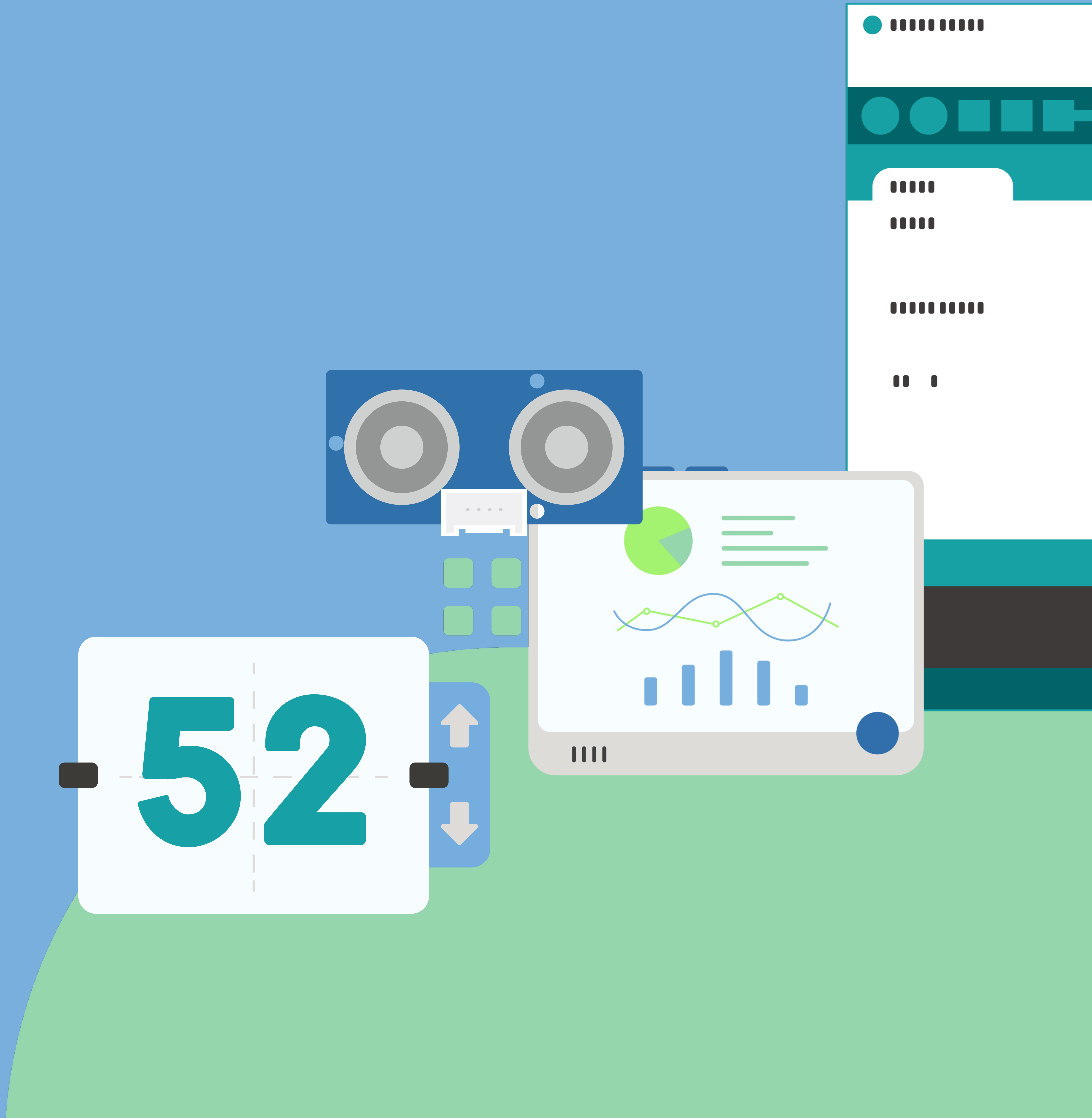
For none category apart from samples that have nobody in front of the device, it is a good idea to include samples that have a person just **standing close** to the device and **walking beside** it, to avoid any movement being falsely classified as in or out.

Expansion tasks

Change the position of Ultrasonic sensor, installing it lower or higher, or possibly near another door frame.

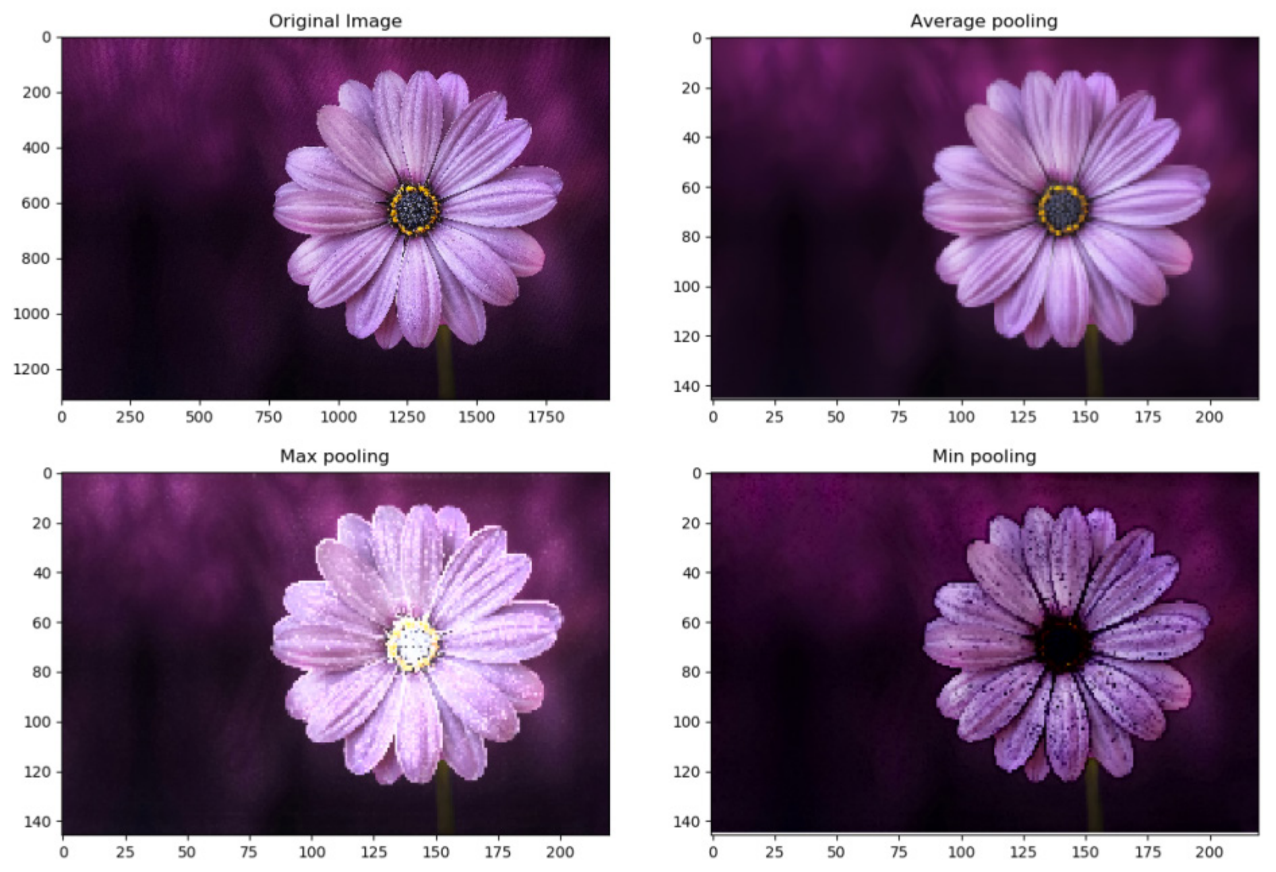
Lesson 09

Project IV: People counting with Ultrasonic sensor: model training and deployment



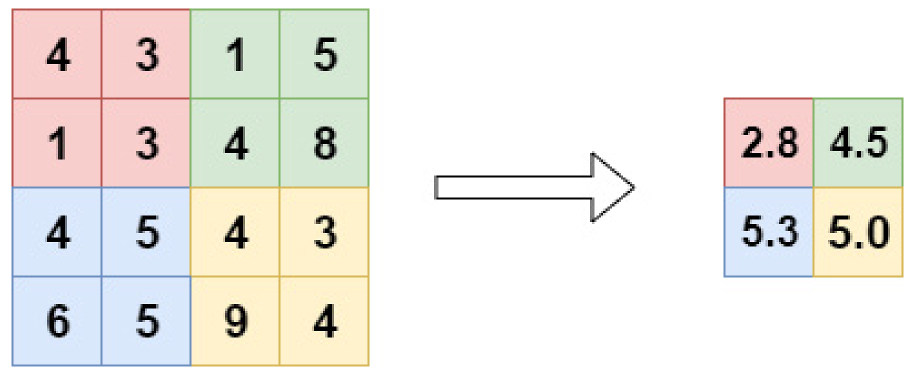
Theory

Convolutional layers significantly reduce computational cost, as compared to fully connected layers. However when amount of data becomes larger, for example when we perform inference on higher resolution images or longer periods of time-series data, training a deep convolutional neural network still requires a lot of compute. Another problem with models consisting of purely Convolutional layers is that during training it also learns the locations of features, which is not a desired effect – a cat is still a cat, doesn't matter if it's in the upper or lower part of the image. We can apply down sampling operation during model training to solve both problems at once, reducing number of connections and thus size of the model and allowing model to learn the featurs irrespective of their location in the data, we can add Pooling layers between Convolutional layers: MaxPooling, MinPooling and AveragePooling.



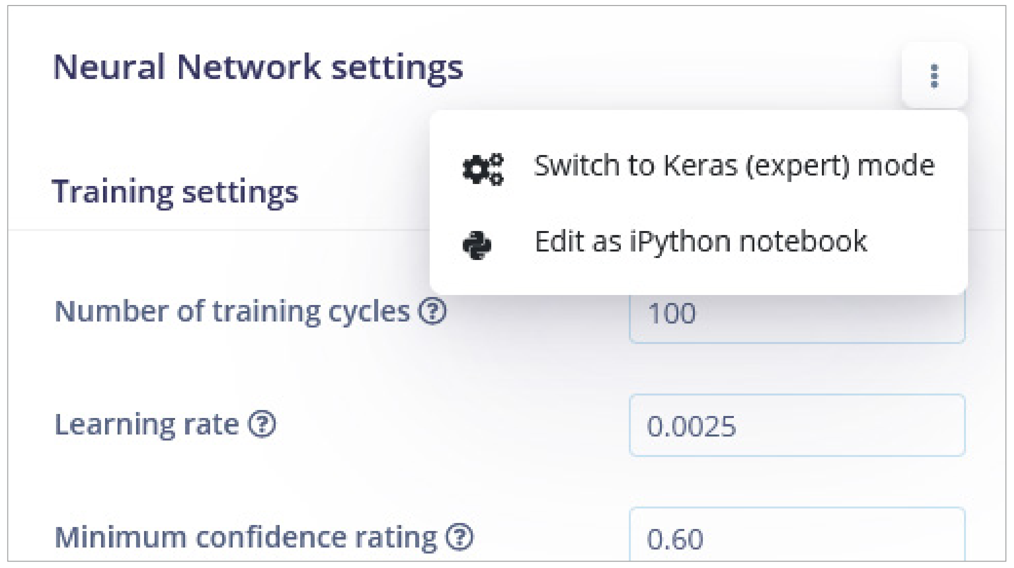
Pooling involves selecting a pooling operation, much like a filter to be applied to feature maps. The size of the pooling operation or filter is smaller than the size of the feature map; specifically, it is almost always 2×2 pixels applied with a stride of 2 pixels.

This means that the pooling layer will always reduce the size of each feature map by a factor of 2, e.g. each dimension is halved, reducing the number of pixels or values in each feature map to one quarter the size. For example, a pooling layer applied to a feature map of 4×4 (16 pixels) will result in an output pooled feature map of 2×2 (4 pixels). See the result of Average Pooling operation applied to 4x4 matrix below:



$$\begin{bmatrix} 4 & 3 & 1 & 5 \\ 1 & 3 & 4 & 8 \\ 4 & 5 & 4 & 3 \\ 6 & 5 & 9 & 4 \end{bmatrix} \Rightarrow \begin{bmatrix} 2.8 & 4.5 \\ 5.3 & 5.0 \end{bmatrix}$$

When using Convolutional layers in Edge Impulse, a MaxPooling layer is automatically applied after Convolutional layer – you can delete it or change its parameters if switching to expert mode, when training a model.



Neural Network settings

Training settings

Switch to Keras (expert) mode

Edit as iPython notebook

Number of training cycles 100

Learning rate 0.0025

Minimum confidence rating 0.60

Preparation

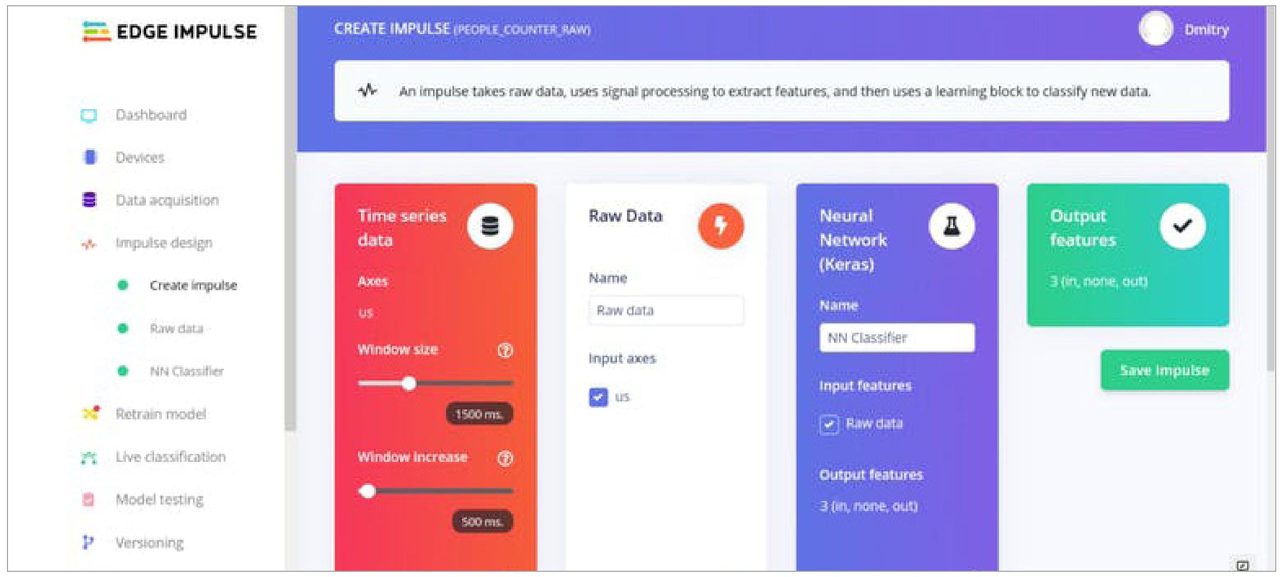
For expansion task, make sure you have WiFi libraries installed and RTL Fimrware is the latest version of Wio Terminal. You will also need to download the following libraries from Github:

https://github.com/Seeed-Studio/Seeed_Arduino_rpcWiF
https://github.com/Seeed-Studio/Seeed_Arduino_rpcUnified
https://github.com/Seeed-Studio/Seeed_Arduino_mbedtls#dev
https://github.com/Seeed-Studio/Seeed_Arduino_FS
https://github.com/Seeed-Studio/Seeed_Arduino_SFUD
https://github.com/sstaub/NTP
https://github.com/ciniml/ExtFlashLoader
https://github.com/Seeed-Studio/Seeed_Arduino_LIS3DHTR
https://github.com/bxparks/AceButton

And then install the following libraries using Arduino libraries manger:
PubSubClient
MsgPack

Practice

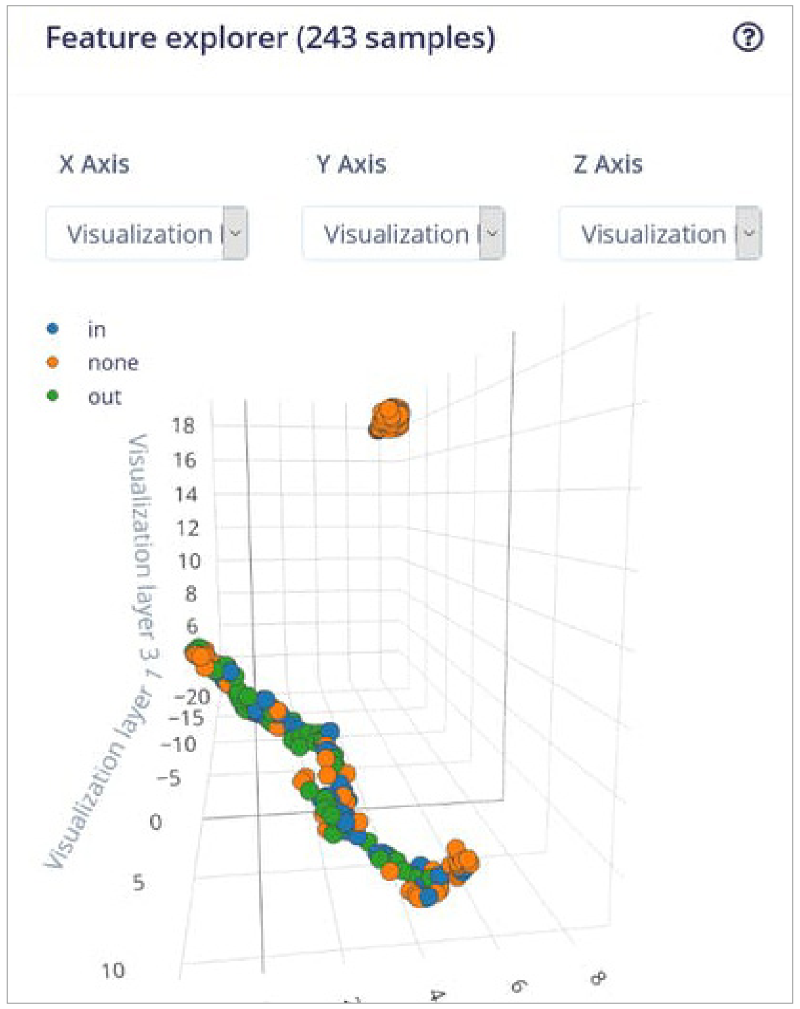
When you are done with data collection, create your impulse — set window length to 1500 ms and windows size increase to 500 ms.



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

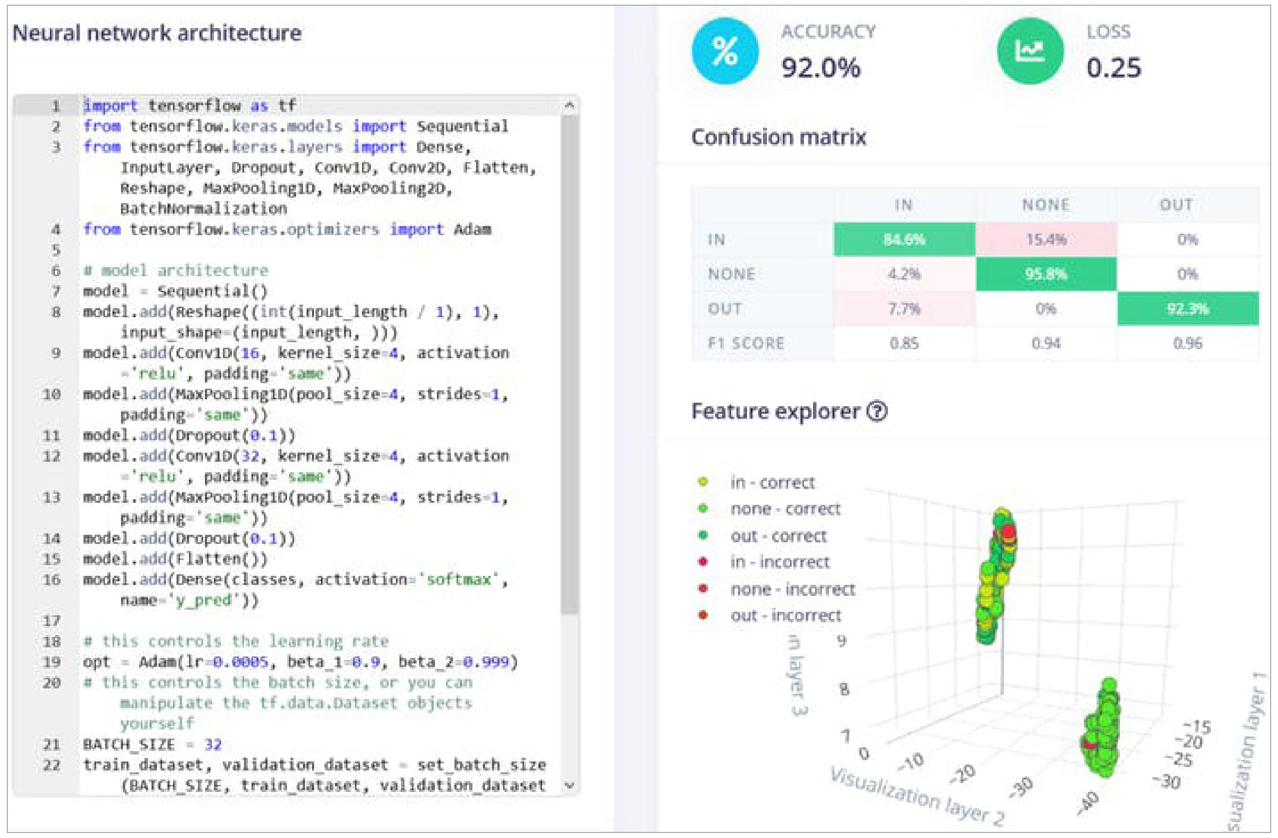
Window size: 1500 ms.
Window increase: 500 ms.
Raw Data Name: Raw data
Input axes: ✓ us
Neural Network (Keras) Name: NN Classifier
Input features: ✓ Raw data

1500 ms is more than enough to cover time duration person takes, when walking in the door or walking out, except if moving extremely slow. For processing blocks, we only have two blocks this time to experiment with — **Raw data** or **Spectral analysis**. Flatten block will erase all the time-domain information from the data, making it completely useless in determining direction, so we won't use it.



Spectral analysis block applies Fast Fourier transform to data samples, converting signal from time domain to frequency domain. While FFT can work for other types of signals, such as sounds or accelerometer data, in our case the frequency of signal actually also doesn't matter that much, since we cannot judge if person is coming in or going out of the room based on frequency. If you look at the data visualization after Spectral analysis block, it is clear that it's hard to separate in and out data samples.

Changing processing block to Spectrogram doesn't really alleviate the problem and resulting accuracy still stays fairly low — the highest we could get was 79.6 %, with a lot of confusion between in and out classes. And the winner, once again is Raw data (with scaling) + 1D Convolutional network. 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.



Neural network architecture

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,
    InputLayer, Dropout, Conv1D, Conv2D, Flatten,
    Reshape, MaxPooling1D, MaxPooling2D,
    BatchNormalization
from tensorflow.keras.optimizers import Adam

# model architecture
model = Sequential()
model.add(Reshape((int(input_length / 1), 1),
    input_shape=(input_length, )))
model.add(Conv1D(16, kernel_size=4, activation
    ='relu', padding='same'))
model.add(MaxPooling1D(pool_size=4, strides=1,
    padding='same'))
model.add(Dropout(0.1))
model.add(Conv1D(32, kernel_size=4, activation
    ='relu', padding='same'))
model.add(MaxPooling1D(pool_size=4, strides=1,
    padding='same'))
model.add(Dropout(0.1))
model.add(Flatten())
model.add(Dense(classes, activation='softmax',
    name='y_pred'))

# this controls the learning rate
opt = Adam(lr=0.0005, beta_1=0.9, beta_2=0.999)
# this controls the batch size, or you can
    manipulate the tf.data.Dataset objects
    yourself
BATCH_SIZE = 32
train_dataset, validation_dataset = set_batch_size
    (BATCH_SIZE, train_dataset, validation_dataset
```

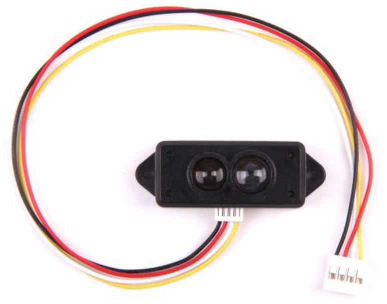
Confusion matrix

	IN	NONE	OUT
IN	84.6%	15.4%	0%
NONE	4.2%	95.8%	0%
OUT	7.7%	0%	92.3%
F1 SCORE	0.85	0.94	0.96

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, 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 percent 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?

• Ultrasonic sensor is 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.



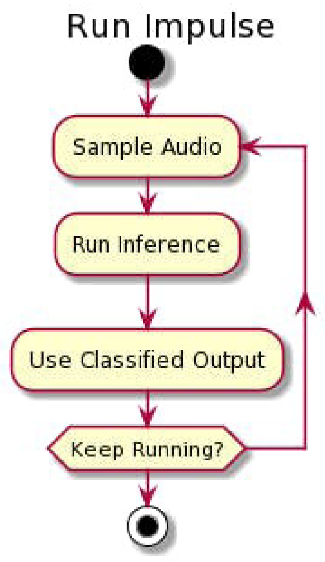
• Get more data and possibly place the sensor lower, at normal human waist level to make sure it can detect shorter than normal height people and children.

• 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 point 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 device — here we found that 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. To deploy the model to Wio Terminal go to deployment tab, choose Arduino library, download it, extract the archive and put it inside of your Arduino libraries folder.

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

If you remember, in the first lesson of the course, 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 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.

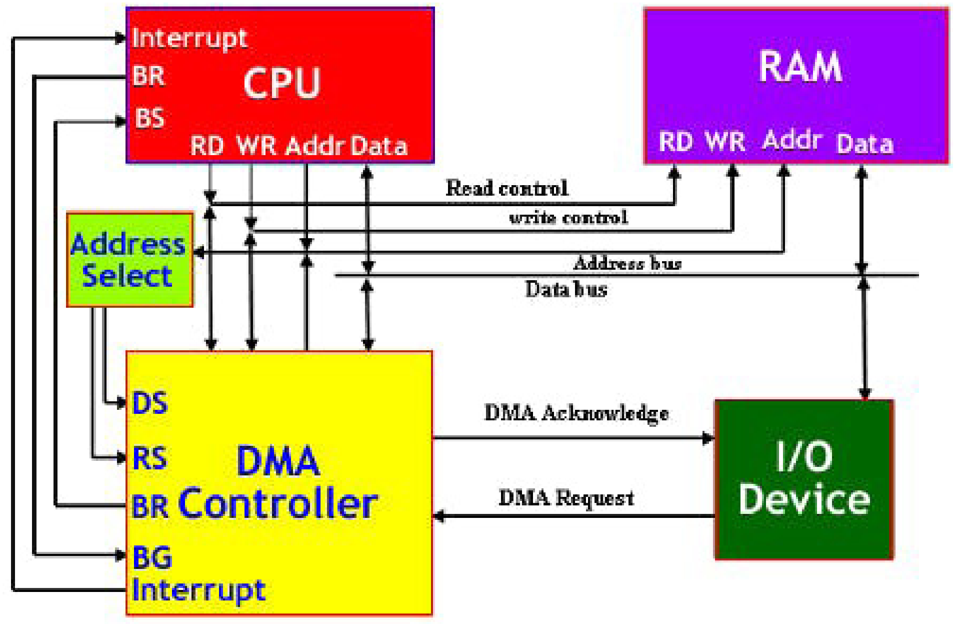
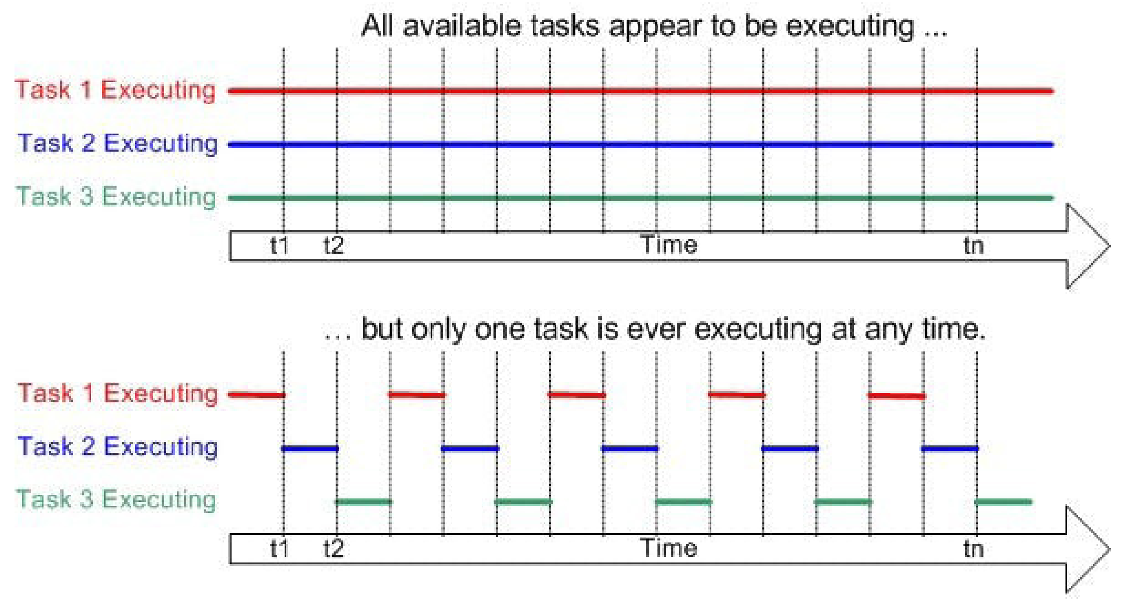
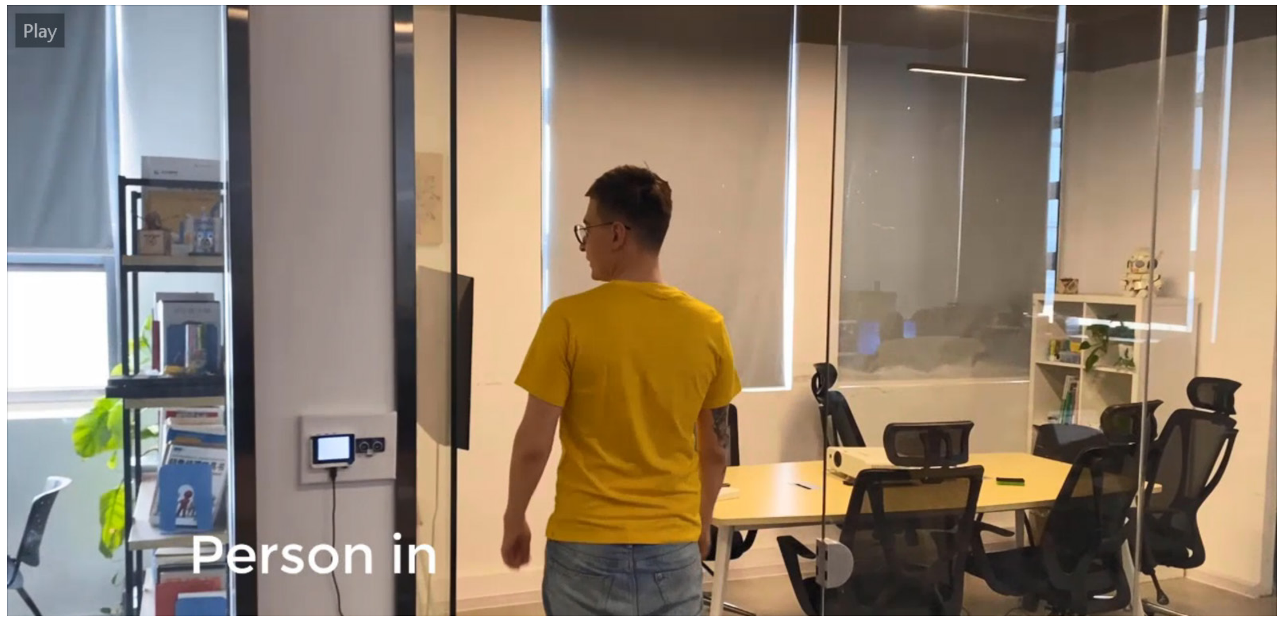


Fig: Showing DMA Mode of Data Transfer

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 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 person is entering the room and turns it off when a person exits.



Open Examples –> name of your project – > nano_ble33_sense_accelerometer_continuous sketch and replace everything (including run_inference_background function declaration) above setup function with the following code block:

```
/* Includes ---------------------------------------------------------------- */
#include <people_counter_raw_inference.h>
#include <Seeed_Arduino_FreeRTOS.h>
#include "Ultrasonic.h"
#include "TFT_eSPI.h"

#define ERROR_LED_LIGHTUP_STATE HIGH

/* 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;

TaskHandle_t Handle_aTask;
TaskHandle_t Handle_bTask;
Ultrasonic ultrasonic(0);
TFT_eSPI tft;
```

In setup function, initialize LCD screen

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

and delete all the lines that are related to accelerometer. Then in place of `inference_thread.start(mbed::callback(&run_inference_background));` paste the following code block – the reason we need to replace this line is because Thread initialization is done differently in Arduino BLE33 Sense and Wio Terminal.

```
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(run_inference_background,"Inference", 512, NULL, tskIDLE_PRIORITY + 1, &Handle_aTask);
    xTaskCreate(read_data, "Data collection", 256, NULL, tskIDLE_PRIORITY + 2, &Handle_bTask);

    // Start the RTOS, this function will never return and will schedule the tasks.

    vTaskStartScheduler();
```

run_inference_continuous function is largely unchanged, the only two things that need to be changed here are

- `void run_inference_background()` to `static void run_inference_background(void* pvParameters)`
- `ei_classifier_smooth_init(&smooth, 10 /* no. of readings */, 7 /* min. readings the same */, 0.8 /* min. confidence */, 0.3 /* max anomaly */);` to
  `ei_classifier_smooth_init(&smooth, 3 /* no. of readings */, 2 /* min. readings the same */, 0.6 /* min. confidence */, 0.3 /* max anomaly */);`

The line above controls averaging (or smoothing) parameters, that we apply to output of the model. You can experiment with the values to see what values allow for best performance in terms of true positives/false positives rate.

While in the original code, data collection happens in loop function, for Wio Terminal FreeRTOS port, it is better to implement data collection in a thread and leave the loop function empty. Delete the loop function in original code and replace it with the following code block

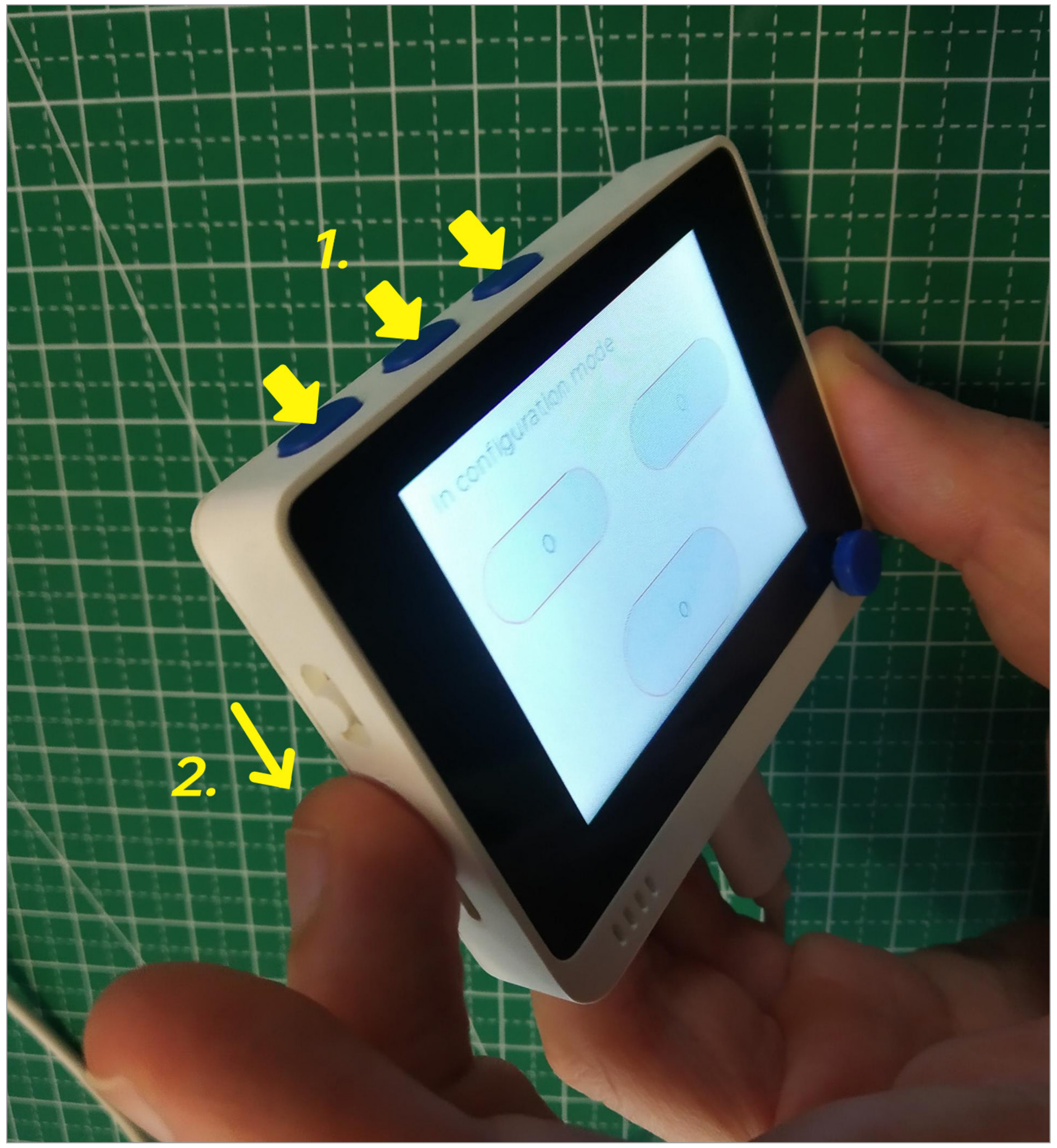
```
/**
* @brief      Get data and run inferencing
*
* @param[in]  debug  Get debug info if true
*/
static void read_data(void* pvParameters)
{
    while (1) {
        // Determine the next tick (and then sleep later)
        uint64_t next_tick = micros() + (EI_CLASSIFIER_INTERVAL_MS * 1000);

        // 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
}
```

Here we wait until it is time to get the data, then take distance measurement with ultrasonic sensor and copy it to inference buffer. Remember that since it is a thread, having delay here doesn't affect the whole system and just temporarily "stops" the thread – until it is time to take next reading. FreeRTOS can perform tasks in other threads while data collection thread is inactive.

★ Expansion tasks

Okay, the model works, but again all in by itself it is not suitable for actually applying it in the real world. Let's add two elements to make it into a full-fledged application — a **simple GUI** and **data upload to cloud** with pretty graphs. We will use LVGL library for creating graphical user interface and Microsoft Azure IoT Central service for sending data to and visualization. The resulting sketch is 693 lines long and has 3 concurrent threads running in RTOS. Here is a quick recap of steps you need to make it work with IoT central.

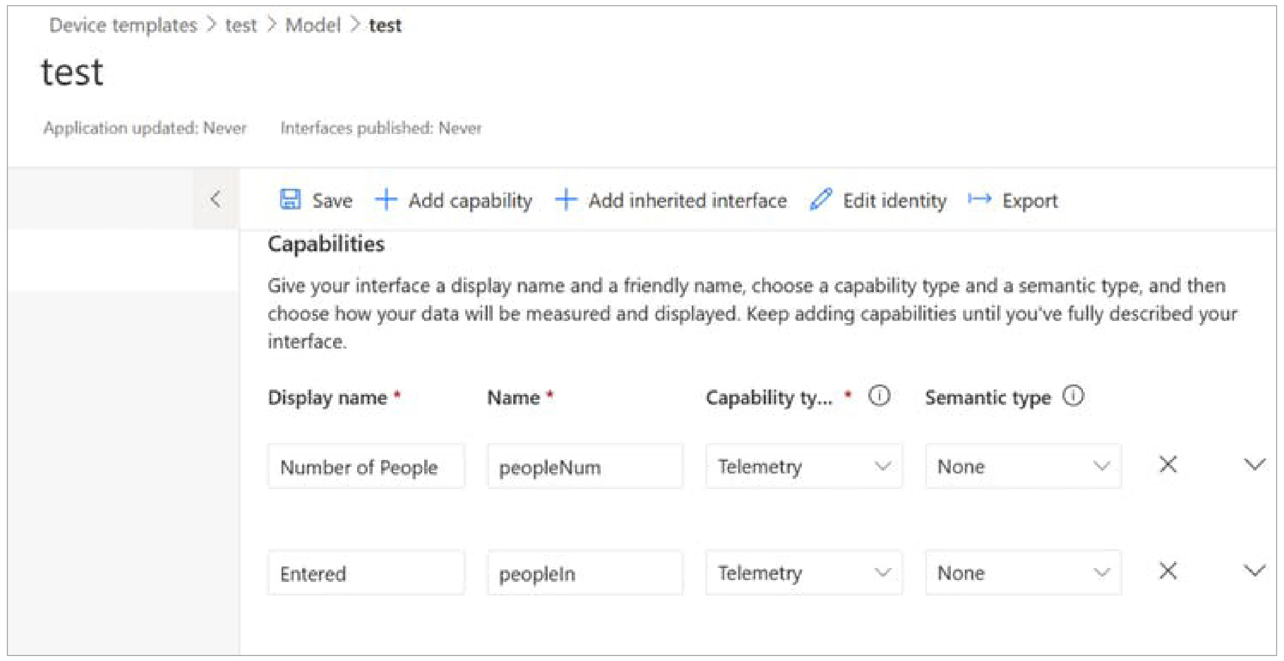
Find the project in course materials, under name WioTerminal_Azure_Central.ino and open it in Arduino IDE. After the sketch is uploaded, enter configuration mode by pressing three buttons on top of Wio Terminal and resetting the device.



"In configuration mode" will be displayed on device screen. Connect to device with Serial monitor (baud rate 115200, carriage return) and set WiFi SSID, password and Azure IoT Central credentials (in the following format **set_az_iotc**your_ID_scopeyour_primary_keyyour_device_ID), which you can get by following these steps:

- Go to https://apps.azureiotcentral.com/

- If you don't have a Microsoft account yet, register one.
- Go to Build –> Custom app. Enter the app name and unique URL (can be similar to app name). Choose Free plan.
- After an app is created, go to Device Templates. Make a new template of IoT device type. Choose custom model, add three capabilities as in the below screenshot and two interfaces (press on Views –> Visualizing the device). After finishing that and making sure everything is correct, publish the template.
- Create a new device from template by going to Devices and pressing on New, remember to choose the Template you just created and published!
- Get the ID scope from Administration –> Device connection, Primary key from Administration –> Device connection –> SAS–IoT–Devices and device ID from Devices tab, where you created your device on Step 5.


Device templates > test > Model > test

test

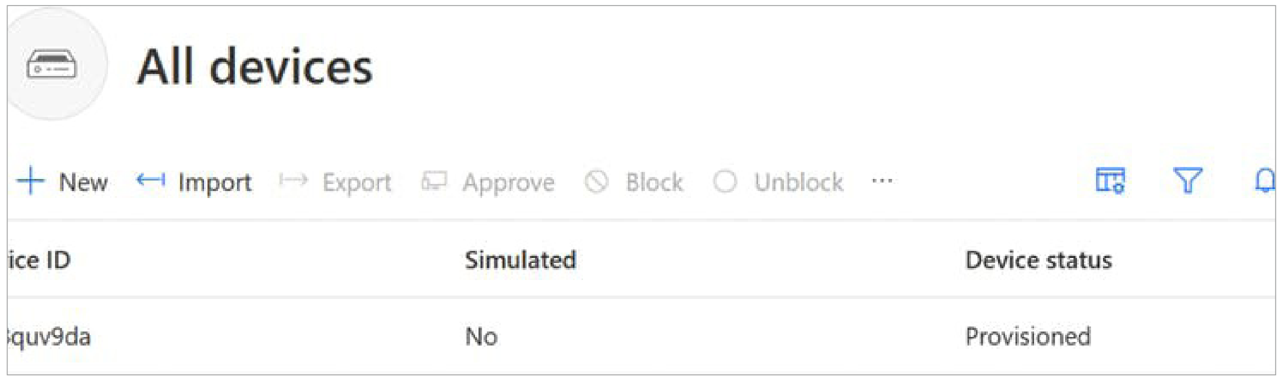
Application updated: Never Interfaces published: Never

Capabilities

Give your interface a display name and a friendly name, choose a capability type and a semantic type, and then choose how your data will be measured and displayed. Keep adding capabilities until you've fully described your interface.

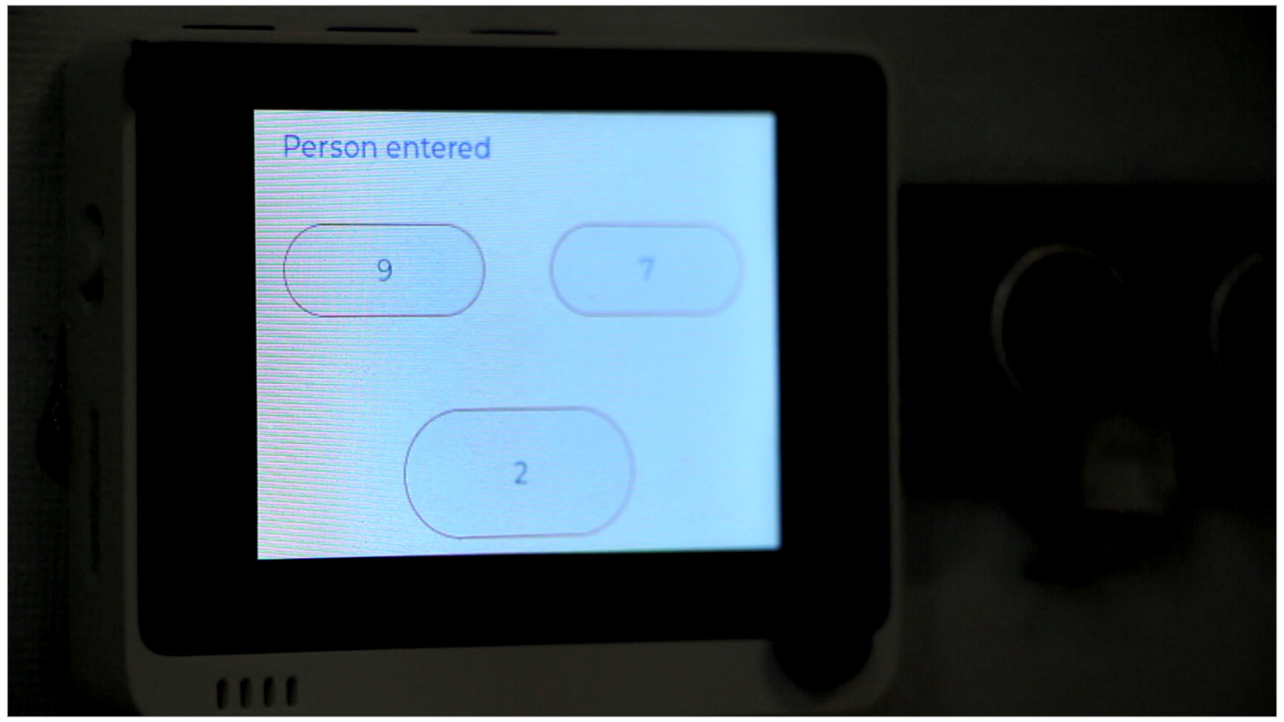
Display name *	Name *	Capability ty... *	Semantic type
Number of People	peopleNum	Telemetry	None
Entered	peopleIn	Telemetry	None

After configuration is successful, restart Wio Terminal and it will start connecting to Azure IoT Central, you can watch the detailed progress feedback on the Serial Terminal. You will then be able to see a) Device status on dashboard has changed to Provisioned b) Telemetry data from Accelerometer sensor in Device –> Raw data.


All devices

ice ID	Simulated	Device status
quv9da	No	Provisioned

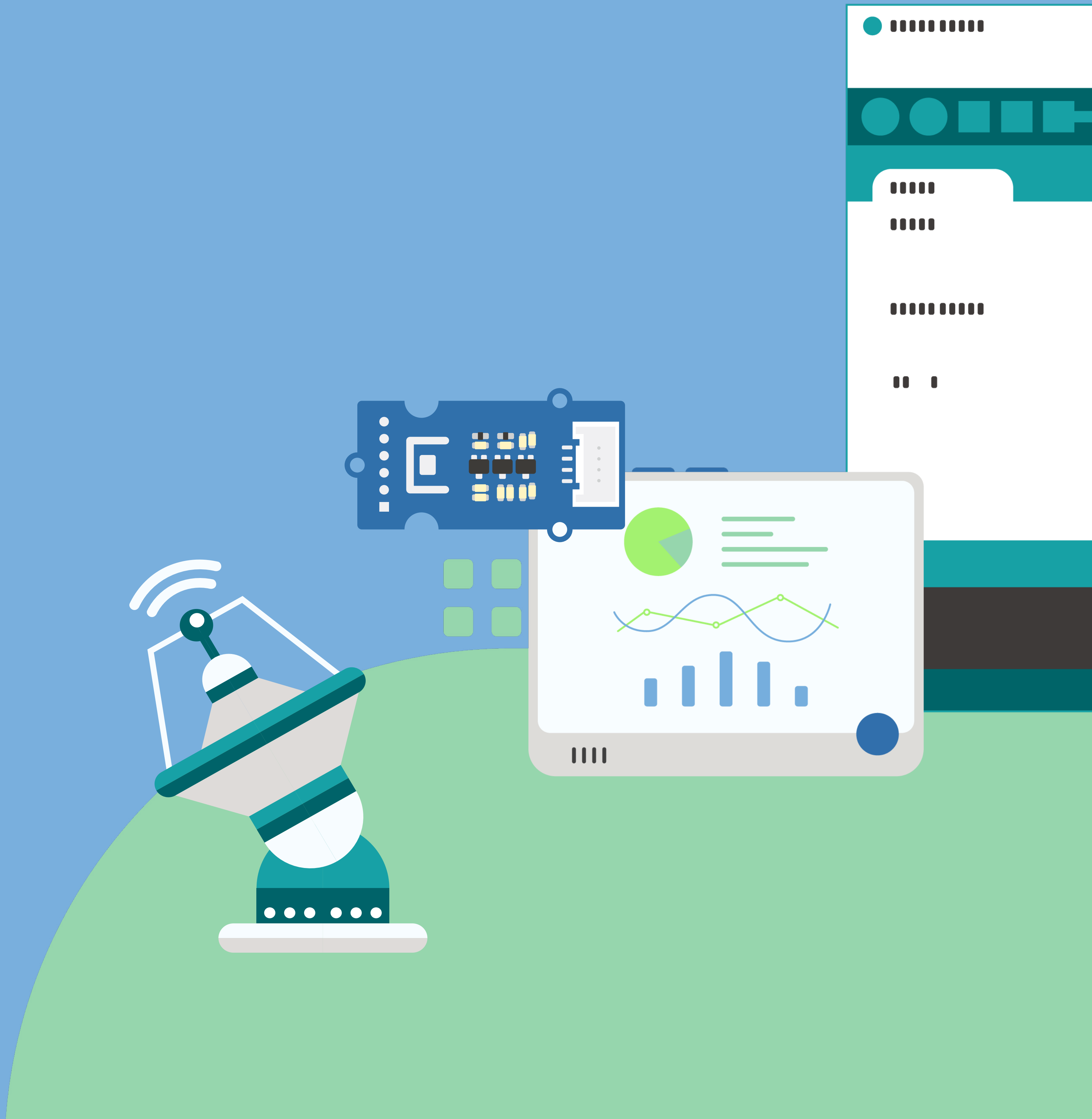
We then add the parts responsible for Edge Impulse model inference, threading and modify send telemetry function to send values for number of people entered, people left and total number of people in the room. We also add simple GUI consisting of three buttons and a text field, which displays information updates — you can see the resulting sketch by opening **WioTerminal_EI_People_Counting_Azure_Central_LVGL**.ino from the course materials.



The hardest part was really making sure everything works normally in each separate thread and does not influence other threads. Sacrifices were made in order to accommodate that without over–complicating the code too much, for example placing LVGL task update function right after interface updates and not letting it run periodically.

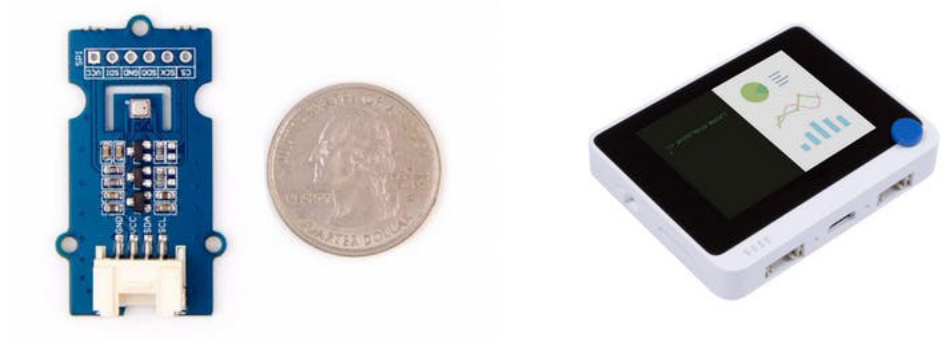
Lesson 10

Project V: Intelligent meteostation with BME280: theory and data collection

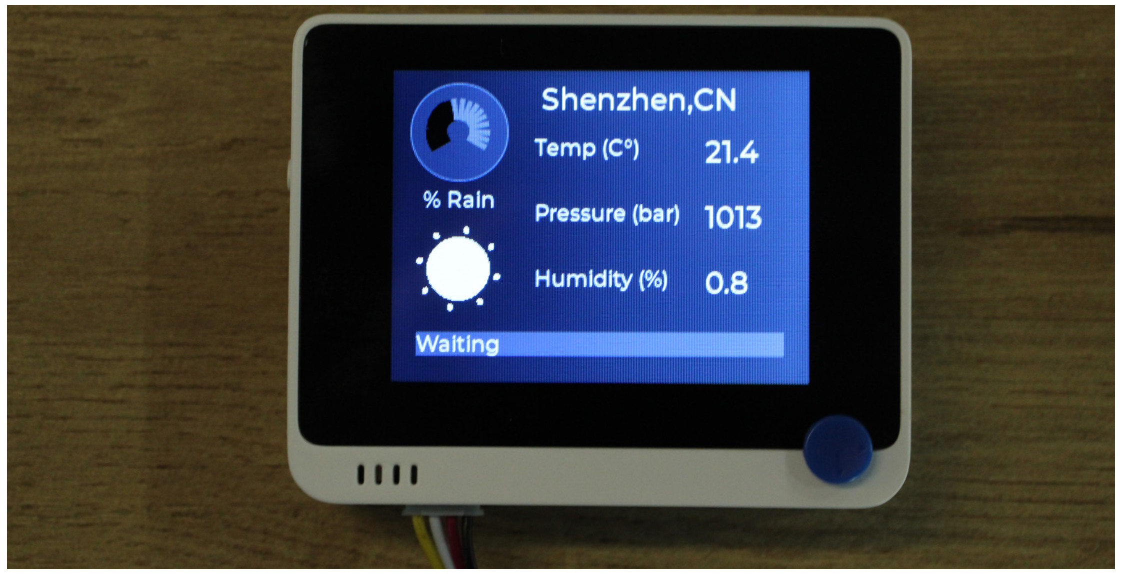


Theory

In this lesson we're going to use Wio Terminal and Tensorflow Lite for Microcontrollers to create an **intelligent meteostation**, able to predict the weather and precipitation for next 24 hours based on local data from BME280 environmental sensor.



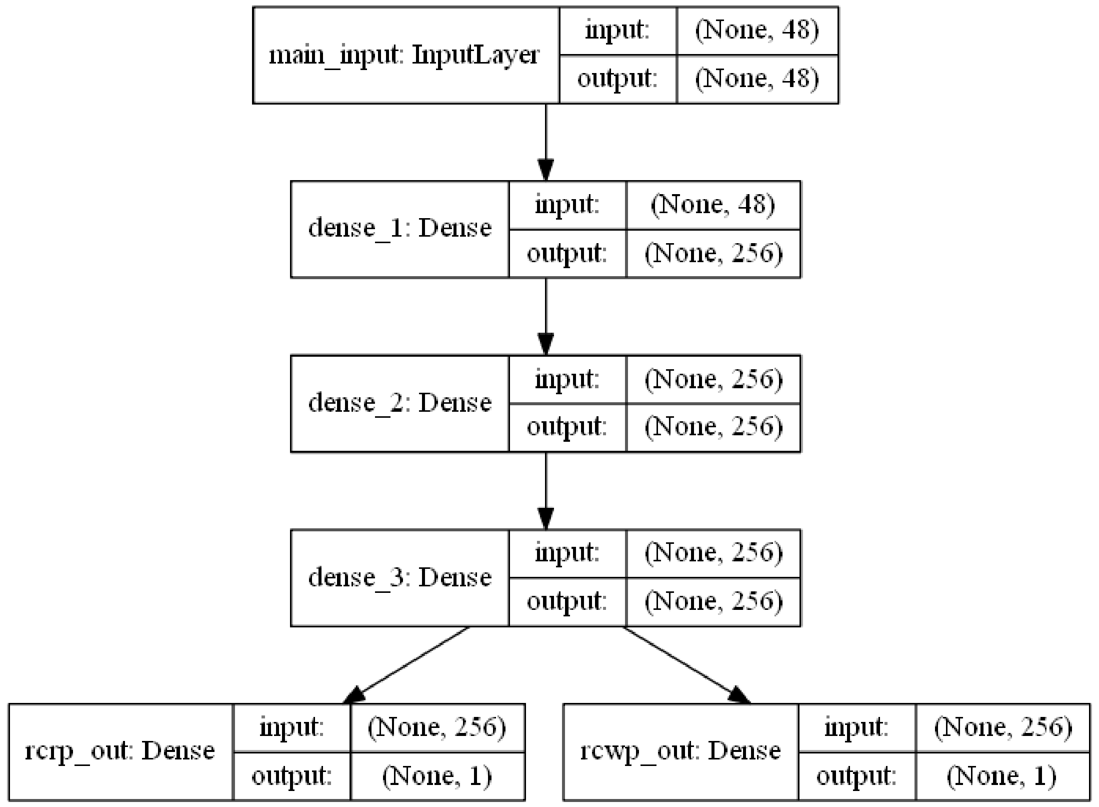
You will learn how to apply model optimization techniques, that will allow not only to run medium-sized Convolutional neural network, but also to have this sleeky GUI and WiFi connection all running at the same time for days and month at the time!



This is the end result, you can see there are current temperature, humidity and atmospheric pressure values displayed on the screen, together with city name, predicted weather type and predicted precipitation chance — and in the bottom of the screen there is a log output field, which you can easily re-purpose for displaying extreme weather information or other relevant information. While it looks good and useful as it is, there is a lot of things you can add yourself — for example above mentioned news/tweets output on the screen or using deep sleep mode to conserve energy and make it battery powered and so on.

In this project we will be dealing with time series data, as we did multiple times before – the only big difference this time is that the time period is much larger for weather prediction. We are going to take a measurement every hour and perform prediction on 24 hours of data. Also since

we're going to predict the average weather type for next 24 hours, additionally we will predict a precipitation chance for next 24 hours, with the same model. In order to do that we will utilize Keras Functional API and multi-output model.


main_input: InputLayer | input: (None, 48) | output: (None, 48)
dense_1: Dense | input: (None, 48) | output: (None, 256)
dense_2: Dense | input: (None, 256) | output: (None, 256)
dense_3: Dense | input: (None, 256) | output: (None, 256)
rcrp_out: Dense | input: (None, 256) | output: (None, 1)
rcwp_out: Dense | input: (None, 256) | output: (None, 1)

Within multi-output model there is going to be "a stem", common for both outputs, which going to "branch out" to two different outputs. Main benefit of using multi-output model as compared to two independent models here is that the data and learned features used for predicting weather type and precipitation chance are highly related.

Preparation

If you are making this project on Windows, **first thing** you'll need to do is to download nightly version of Arduino IDE, since current stable version 1.18.3 will not compile sketches with a lot of library dependencies (the issue is that linker command during compilation exceeds maximum length on Windows).

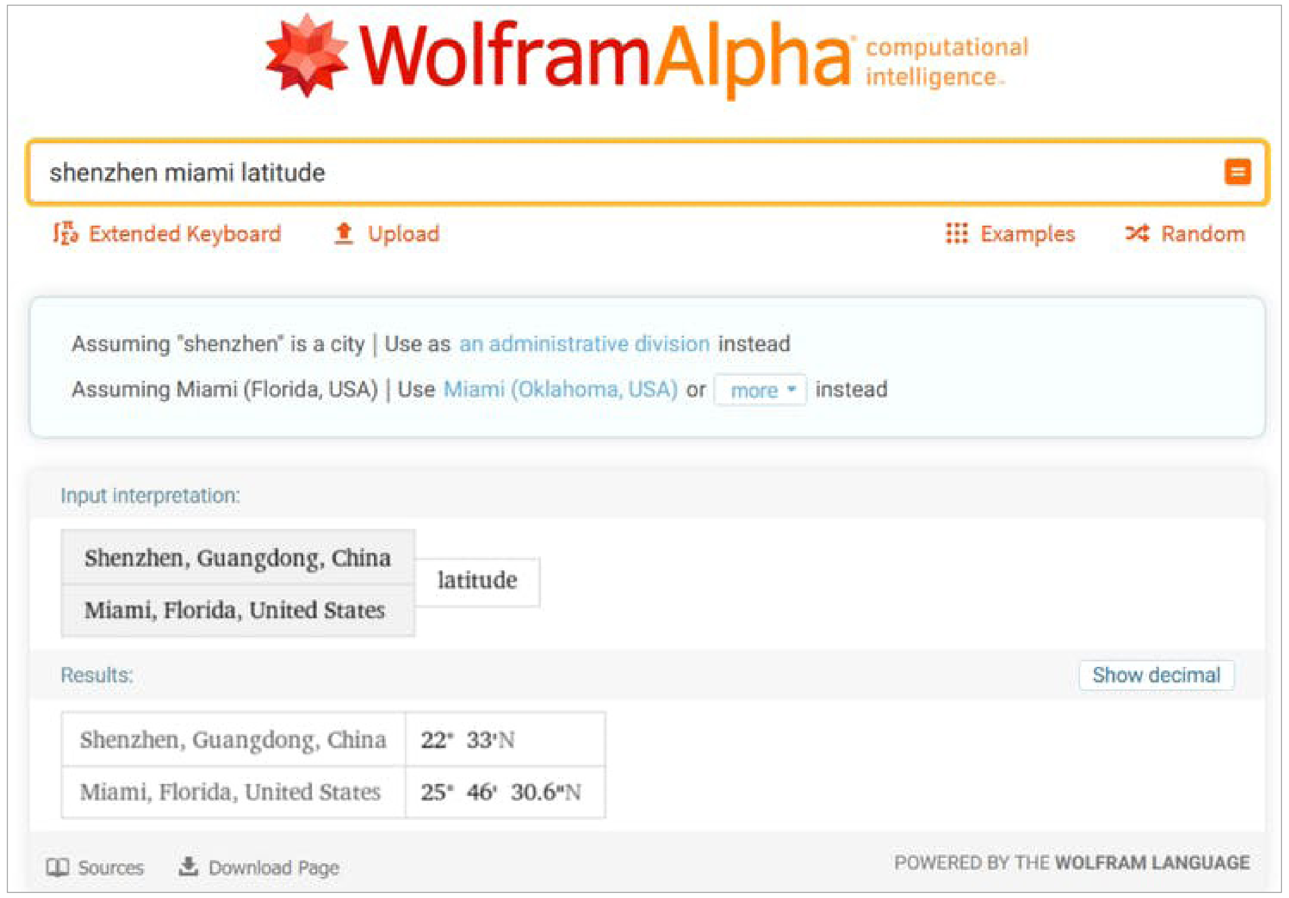
Second, you need to make sure you have 1.8.2 version of Seeed SAMD board definitions in Arduino IDE.

Finally, since we're using a Convolutional neural network and build it with Keras API, it contains an operation not supported by current stable version of Tensorflow Micro. Browsing Tensorflow issues on Github I found that there is a pull request for adding this op (EXPAND_DIMS) to list of available ops, but it was not merged into master at the time of making this video.

You can git clone the Tensorflow repository, switch to PR branch and compile Arduino library by executing./tensorflow/lite/micro/tools/ci_build/test_arduino.sh on Linux machine — the resulting library can be found in tensorflow/lite/micro/tools/make/gen/arduino_x86_64/prj/tensorflow_lite.zip. **Alternatively**, you can download already compiled library from this project Github repository and place it into your Arduino sketches libraries folder — just make sure you only have one Tensorflow lite library at the time!

Practice

It all starts with data of course. For this tutorial we will use a readily available weather dataset from Kaggle, Historical Hourly Weather Data 2012–2017. Seeed EDU headquarters are located in Shenzhen, a city in Southern China — and that city is absent from the dataset, so we picked a city that is located on the similar latitude and also has a subtropical climate — Miami.

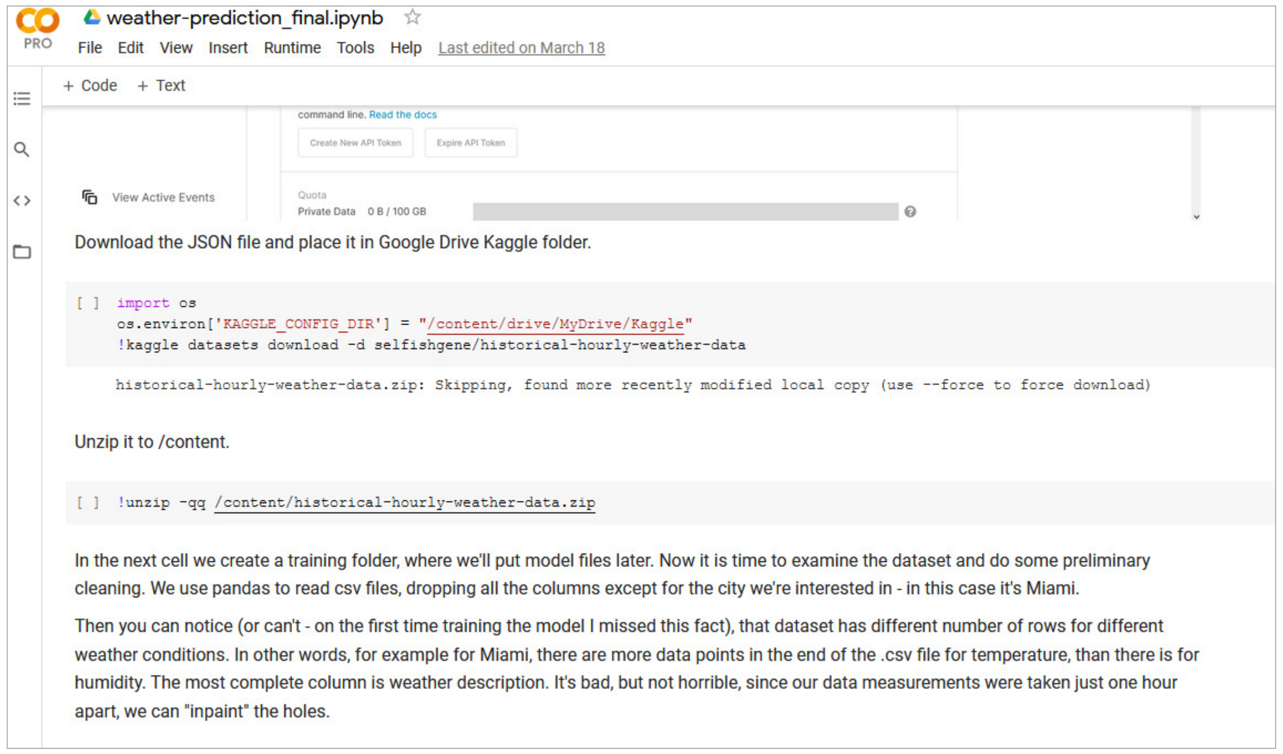

WolframAlpha computational intelligence.
shenzhen miami latitude
Extended Keyboard
Upload
Examples
Random
Assuming "shenzhen" is a city | Use as an administrative division instead
Assuming Miami (Florida, USA) | Use Miami (Oklahoma, USA) or more instead
Input interpretation:
Results:
Show decimal
Sources
Download Page
POWERED BY THE WOLFRAM LANGUAGE

Input interpretation:	
Shenzhen, Guangdong, China	latitude
Miami, Florida, United States	

Results:	
Shenzhen, Guangdong, China	22° 33'N
Miami, Florida, United States	25° 46' 30.6"N

You'll need to pick a city that at least resembles the climate where you live — it goes without saying that the model trained on data from Miami and then deployed in Chicago in winter is not going to output correct predictions.

For data processing and model training step, let's open Jupyter Notebook you can find in course materials. The easiest way to run this notebook is to upload it to Google Colab, since it already has all the packages installed and ready to run.


CO PRO
weather-prediction_final.ipynb
File Edit View Insert Runtime Tools Help Last edited on March 18
+ Code + Text
command line. Read the docs
Create New API Token
Expire API Token
View Active Events
Quota
Private Data 0 B / 100 GB

Download the JSON file and place it in Google Drive Kaggle folder.

```
[ ]  import os
     os.environ['KAGGLE_CONFIG_DIR'] = "/content/drive/MyDrive/Kaggle"
     !kaggle datasets download -d selfishgene/historical-hourly-weather-data

     historical-hourly-weather-data.zip: Skipping, found more recently modified local copy (use --force to force download)
```

Unzip it to /content.

```
[ ]  !unzip -qq /content/historical-hourly-weather-data.zip
```

In the next cell we create a training folder, where we'll put model files later. Now it is time to examine the dataset and do some preliminary cleaning. We use pandas to read csv files, dropping all the columns except for the city we're interested in - in this case it's Miami.

Then you can notice (or can't - on the first time training the model I missed this fact), that dataset has different number of rows for different weather conditions. In other words, for example for Miami, there are more data points in the end of the .csv file for temperature, than there is for humidity. The most complete column is weather description. It's bad, but not horrible, since our data measurements were taken just one hour apart, we can "inpaint" the holes.

Alternatively you can execute the notebook locally – to do that first install all the required dependencies in the virtual environment by running

```
pip install –r requirements.txt
```

with ml virtual environment you have created before activated. Then run `jupyter notebook` command in the same environment, which will open notebook server in your default browser.

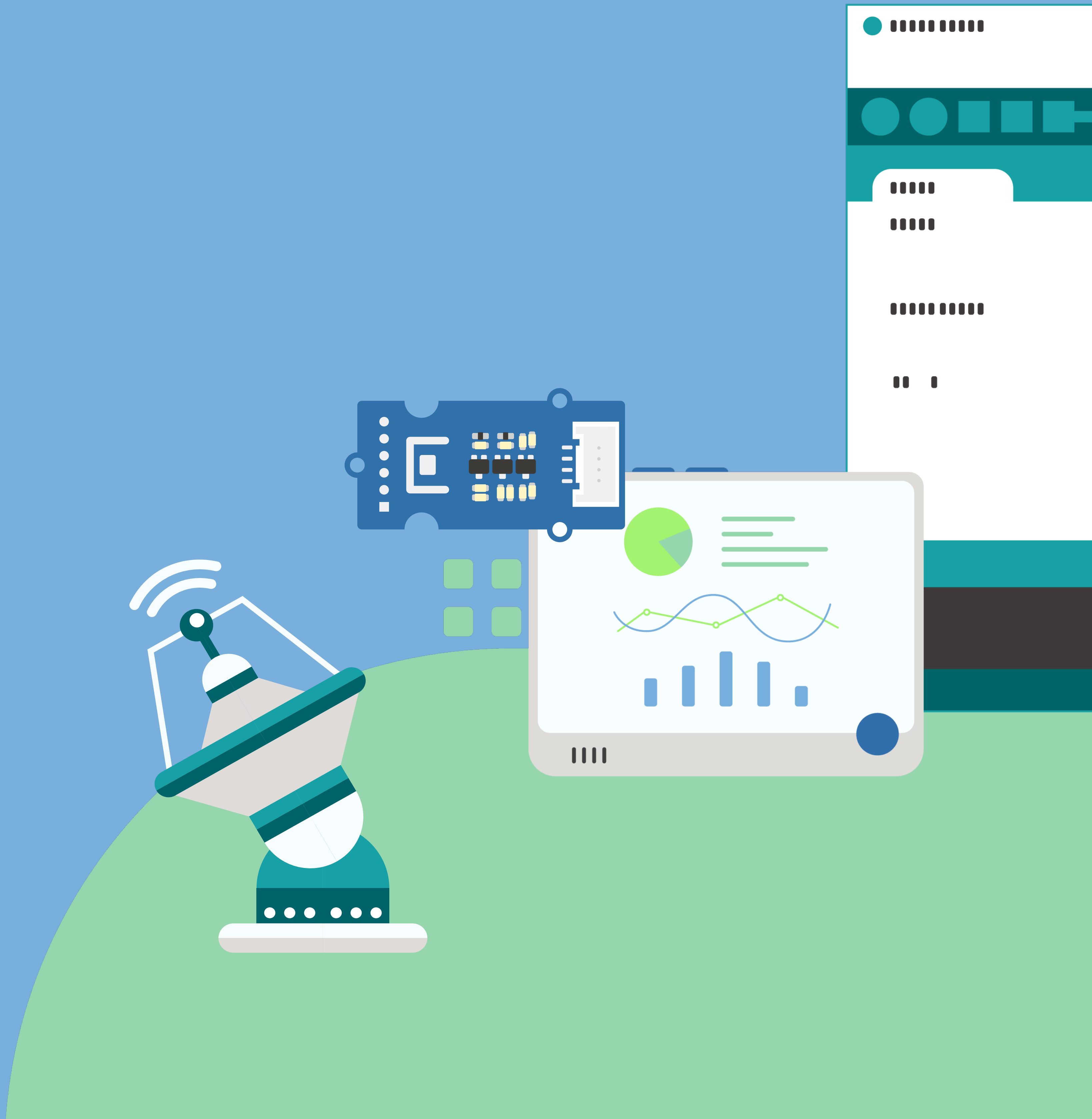
Jupyter Notebooks are great way to explore and present data, since they allow having both text and executable code in the same environment. The general workflow is explained in the Notebook text sections.

Expansion tasks

Experiment with different length time window in the notebook to see how it affects model accuracy.

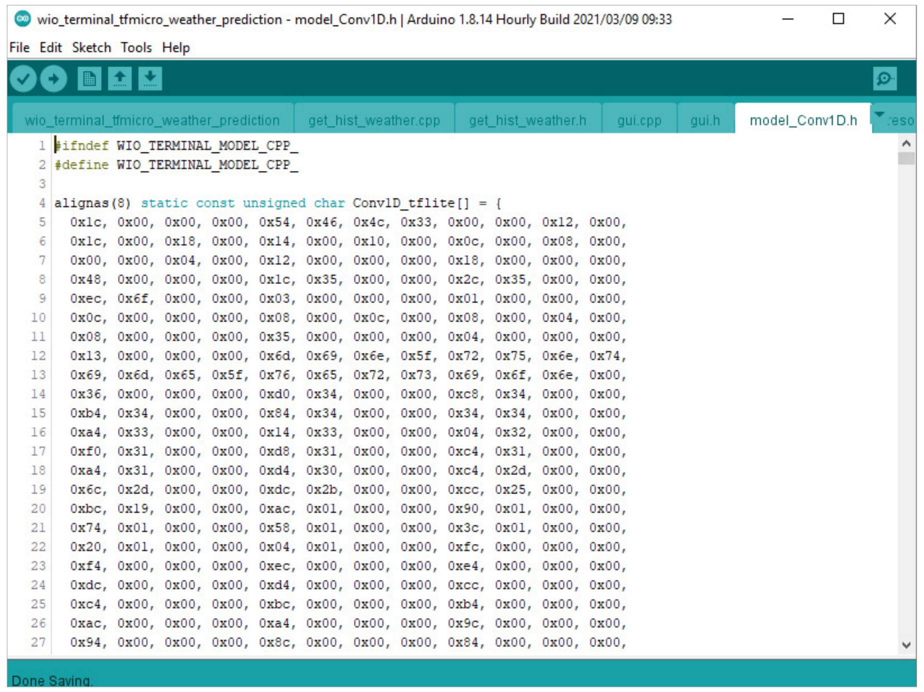
Lesson 11

Project V: Intelligent meteostation with BME280: model training and deployment (tf.keras)



Theory

The model you have trained in the last step was converted to a byte array, which contains model structure and wights and can now be loaded to Wio Terminal together with C++ code.



```
#ifndef WIO_TERMINAL_MODEL_CPP_
#define WIO_TERMINAL_MODEL_CPP_

alignas(8) static const unsigned char Conv1D_tflite[] = {
  0x1c, 0x00, 0x00, 0x00, 0x54, 0x46, 0x4c, 0x33, 0x00, 0x00, 0x12, 0x00,
  0x1c, 0x00, 0x18, 0x00, 0x14, 0x00, 0x10, 0x00, 0x0c, 0x00, 0x08, 0x00,
  0x00, 0x00, 0x04, 0x00, 0x12, 0x00, 0x00, 0x00, 0x18, 0x00, 0x00, 0x00,
  0x48, 0x00, 0x00, 0x00, 0x1c, 0x35, 0x00, 0x00, 0x2c, 0x35, 0x00, 0x00,
  0xec, 0x6f, 0x00, 0x00, 0x03, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00,
  0x0c, 0x00, 0x00, 0x00, 0x08, 0x00, 0x0c, 0x00, 0x08, 0x00, 0x04, 0x00,
  0x08, 0x00, 0x00, 0x00, 0x35, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00,
  0x13, 0x00, 0x00, 0x00, 0x6d, 0x69, 0x6e, 0x5f, 0x72, 0x75, 0x6e, 0x74,
  0x69, 0x6d, 0x65, 0x5f, 0x76, 0x65, 0x72, 0x73, 0x69, 0x6f, 0x6e, 0x00,
  0x36, 0x00, 0x00, 0x00, 0xd0, 0x34, 0x00, 0x00, 0xc8, 0x34, 0x00, 0x00,
  0xb4, 0x34, 0x00, 0x00, 0x84, 0x34, 0x00, 0x00, 0x34, 0x34, 0x00, 0x00,
  0xa4, 0x33, 0x00, 0x00, 0x14, 0x33, 0x00, 0x00, 0x04, 0x32, 0x00, 0x00,
  0xf0, 0x31, 0x00, 0x00, 0xd8, 0x31, 0x00, 0x00, 0xc4, 0x31, 0x00, 0x00,
  0xa4, 0x31, 0x00, 0x00, 0xd4, 0x30, 0x00, 0x00, 0xc4, 0x2d, 0x00, 0x00,
  0x6c, 0x2d, 0x00, 0x00, 0xdc, 0x2b, 0x00, 0x00, 0xcc, 0x25, 0x00, 0x00,
  0xbc, 0x19, 0x00, 0x00, 0xac, 0x01, 0x00, 0x00, 0x90, 0x01, 0x00, 0x00,
  0x74, 0x01, 0x00, 0x00, 0x58, 0x01, 0x00, 0x00, 0x3c, 0x01, 0x00, 0x00,
  0x20, 0x01, 0x00, 0x00, 0x04, 0x01, 0x00, 0x00, 0xfc, 0x00, 0x00, 0x00,
  0xf4, 0x00, 0x00, 0x00, 0xec, 0x00, 0x00, 0x00, 0xe4, 0x00, 0x00, 0x00,
  0xdc, 0x00, 0x00, 0x00, 0xd4, 0x00, 0x00, 0x00, 0xcc, 0x00, 0x00, 0x00,
  0xc4, 0x00, 0x00, 0x00, 0xbc, 0x00, 0x00, 0x00, 0xb4, 0x00, 0x00, 0x00,
  0xac, 0x00, 0x00, 0x00, 0xa4, 0x00, 0x00, 0x00, 0x9c, 0x00, 0x00, 0x00,
  0x94, 0x00, 0x00, 0x00, 0x8c, 0x00, 0x00, 0x00, 0x84, 0x00, 0x00, 0x00,
```

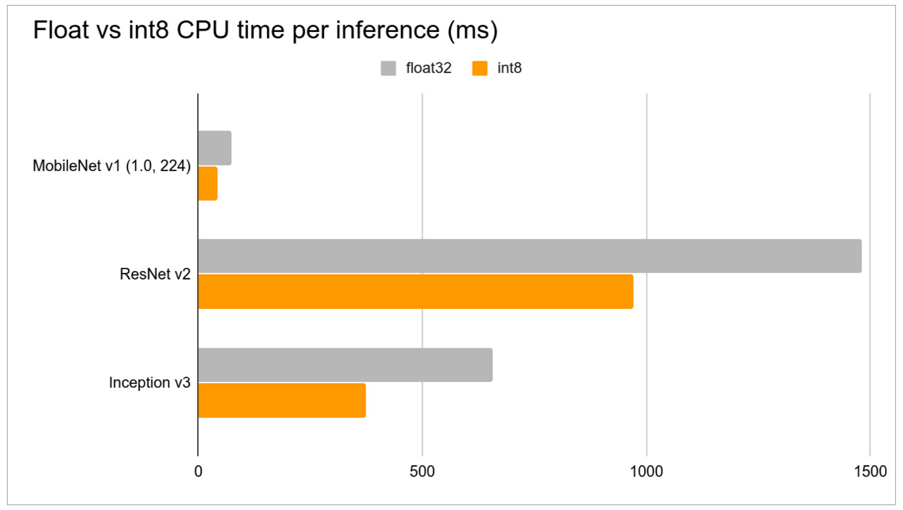
Done Saving.

wio_terminal_tfmicro_weather_prediction - model_Conv1D.h | Arduino 1.8.14 Hourly Build 2021/03/09 09:33
File Edit Sketch Tools Help
wio_terminal_tfmicro_weather_prediction
get_hist_weather.cpp
get_hist_weather.h
gui.cpp
gui.h
model_Conv1D.h

Tensorflow Lite for Microcontrollers includes model Interpreter, which is designed to be lean and fast. The interpreter uses a static graph ordering and a custom (less–dynamic) memory allocator to ensure minimal load, initialization, and execution latency. The data placed in input buffers is fed to the model graph and then after inference is finished results are placed in the output buffer.

In order to reduce the size of the model and decrease inference time, we perform two important optimizations:

• Perform full–integer quantization, changing model weights, inputs and outputs from floating point 32 numbers (each one occupying 32 bits of memory) to integer 8 numbers (each occupying only 8 bits), thus reducing size by factor of 4.



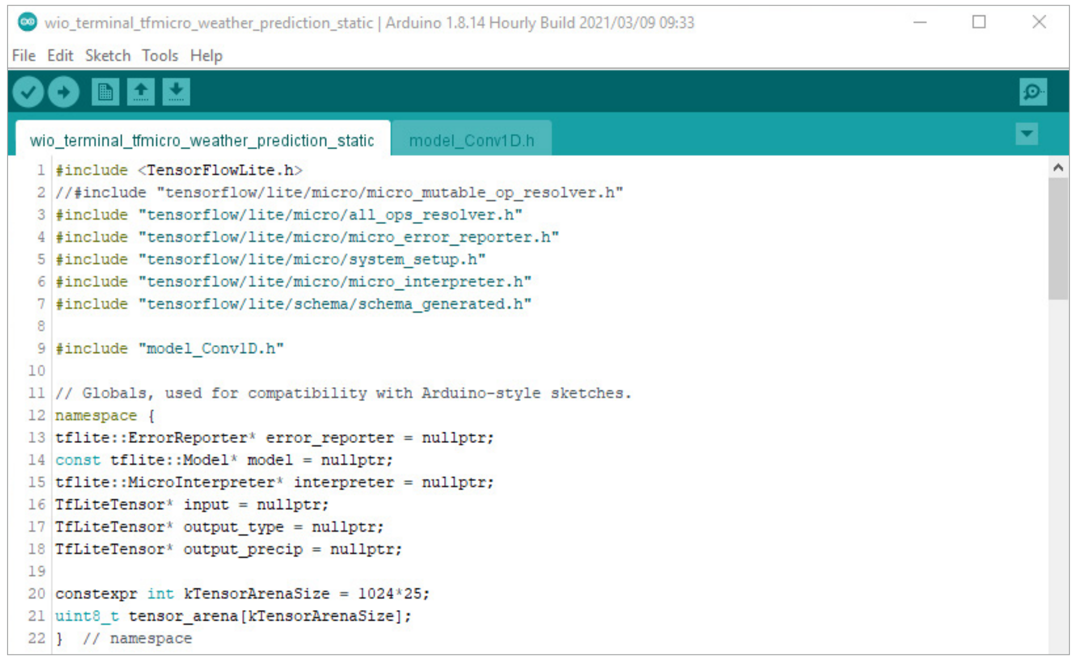
• Use micro_mutable_op_resolver and specify operations that we have in neural network, to compile our code only with the operations needed to run the model, as opposed to using all_ops_resolver, which includes all operations supported by current Tensorflow Lite for Microcontrollers interpreter.

Preparation

Make sure you have libraries for WiFi and LVGL installed – we will need them for expansion task.

Practice

Once the model training is finished, create an empty sketch and save it. Then copy the model you trained to the sketch folder and re–open the sketch. Change the variable name of model and model length to something shorter. Then use the code from wio_terminal_tfmicro_weather_prediction_static.ino, which you can find in course materials for testing.



```
#include <TensorFlowLite.h>
//#include "tensorflow/lite/micro/micro_mutable_op_resolver.h"
#include "tensorflow/lite/micro/all_ops_resolver.h"
#include "tensorflow/lite/micro/micro_error_reporter.h"
#include "tensorflow/lite/micro/system_setup.h"
#include "tensorflow/lite/micro/micro_interpreter.h"
#include "tensorflow/lite/schema/schema_generated.h"

#include "model_Conv1D.h"

// Globals, used for compatibility with Arduino-style sketches.
namespace {
tflite::ErrorReporter* error_reporter = nullptr;
const tflite::Model* model = nullptr;
tflite::MicroInterpreter* interpreter = nullptr;
TfLiteTensor* input = nullptr;
TfLiteTensor* output_type = nullptr;
TfLiteTensor* output_precip = nullptr;

constexpr int kTensorArenaSize = 1024*25;
uint8_t tensor_arena[kTensorArenaSize];
}  // namespace
```

wio_terminal_tfmicro_weather_prediction_static | Arduino 1.8.14 Hourly Build 2021/03/09 09:33
File Edit Sketch Tools Help
wio_terminal_tfmicro_weather_prediction_static
model_Conv1D.h

Let's go over the main steps we have in C++ code

We include the headers for Tensorflow library and the file with model flatbuffer

```
#include <TensorFlowLite.h>
//#include ”tensorflow/lite/micro/micro_mutable_op_resolver.h”
#include ”tensorflow/lite/micro/all_ops_resolver.h”
#include ”tensorflow/lite/micro/micro_error_reporter.h”
#include ”tensorflow/lite/micro/system_setup.h”
#include ”tensorflow/lite/micro/micro_interpreter.h”
#include ”tensorflow/lite/schema/schema_generated.h”
#include ”model_Conv1D.h”
```

Notice how I have micro_mutable_op_resolver.h commented out and all_ops_resolver.h enabled — all_ops_resolver.h header compiles all the operations currently present in Tensorflow Micro and convenient for testing, but once you finished testing it is much better to switch to micro_mutable_op_resolver.h to save devices memory — it does make a big difference.

Next we define the pointers for error reporter, model, input and output tensors and interpreter. Notice how our model has two outputs — one for precipitation amount and another one for weather type. We also define tensor arena, which you can think of as a scratch board, holding input, output, and intermediate arrays — size required will depend on the model you are using, and may need to be determined by experimentation.

```
// Globals, used for compatibility with Arduino-style sketches.
namespace {
tflite::ErrorReporter* error_reporter = nullptr;
const tflite::Model* model = nullptr;
tflite::MicroInterpreter* interpreter = nullptr;
TfLiteTensor* input = nullptr;
TfLiteTensor* output_type = nullptr;
TfLiteTensor* output_precip = nullptr;
constexpr int kTensorArenaSize = 1024*25;
uint8_t tensor_arena[kTensorArenaSize];
}  // namespace
```

Then in setup function, there is more boilerplate stuff, such as instantiating error reporter, op resolver, interpreter, mapping the model, allocating tensors and finally checking the tensor shapes after allocation. Here is when code might throw an error during runtime, if some of model operations are not supported by current version of Tensorflow Micro library. In case you have unsupported operations, you can either changed the model architecture or add the support for the operator yourself, usually by porting it from Tensorflow Lite.

```
void setup() {
  Serial.begin(115200);
  while (!Serial) {delay(10);}

  // Set up logging. Google style is to avoid globals or statics because of
  // lifetime uncertainty, but since this has a trivial destructor it's okay.
  // NOLINTNEXTLINE(runtime-global-variables)
  static tflite::MicroErrorReporter micro_error_reporter;
  error_reporter = &micro_error_reporter;
  // Map the model into a usable data structure. This doesn't involve any
  // copying or parsing, it's a very lightweight operation.
  model = tflite::GetModel(Conv1D_tflite);
  if (model->version() != TFLITE_SCHEMA_VERSION) {
    TF_LITE_REPORT_ERROR(error_reporter,
                         "Model provided is schema version %d not equal "
                         "to supported version %d.",
                         model->version(), TFLITE_SCHEMA_VERSION);
    return;
  }
  // This pulls in all the operation implementations we need.
  // NOLINTNEXTLINE(runtime-global-variables)
  //static tflite::MicroMutableOpResolver<1> resolver;
  static tflite::AllOpsResolver resolver;
  // Build an interpreter to run the model with.
   static tflite::MicroInterpreter static_interpreter(model, resolver, tensor_arena,
kTensorArenaSize, error_reporter);
  interpreter = &static_interpreter;
  // Allocate memory from the tensor_arena for the model's tensors.
  TfLiteStatus allocate_status = interpreter->AllocateTensors();
  if (allocate_status != kTfLiteOk) {
    TF_LITE_REPORT_ERROR(error_reporter, "AllocateTensors() failed");
    return;
  }
  // Obtain pointers to the model's input and output tensors.
  input = interpreter->input(0);
  output_type = interpreter->output(1);
  output_precip = interpreter->output(0);

  Serial.println(input->dims->size);
  Serial.println(input->dims->data[1]);
  Serial.println(input->dims->data[2]);
  Serial.println(input->type);
  Serial.println(output_type->dims->size);
  Serial.println(output_type->dims->data[1]);
  Serial.println(output_type->type);
  Serial.println(output_precip->dims->size);
  Serial.println(output_precip->dims->data[1]);
  Serial.println(output_precip->type);
}
```

Finally in the loop function we define a placeholder for quantized INT8 values and an array with float values, which you can copy paste from Colab notebook for comparison of model inference on device vs. in Colab.

```
void loop() {
  int8_t x_quantized[72];
  float x[72] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0};
```

We quantize the float values to INT8 in for loop and place them in the input tensor one by one:

```
for (byte i = 0; i < 72; i = i + 1) {
        input–>data.int8[i] = x[i] / input–>params.scale + input–>params.zero_point;
  }
```

Then inference is performed by Tensorflow Micro interpreter and if no errors are reported, values are placed in the output tensors.

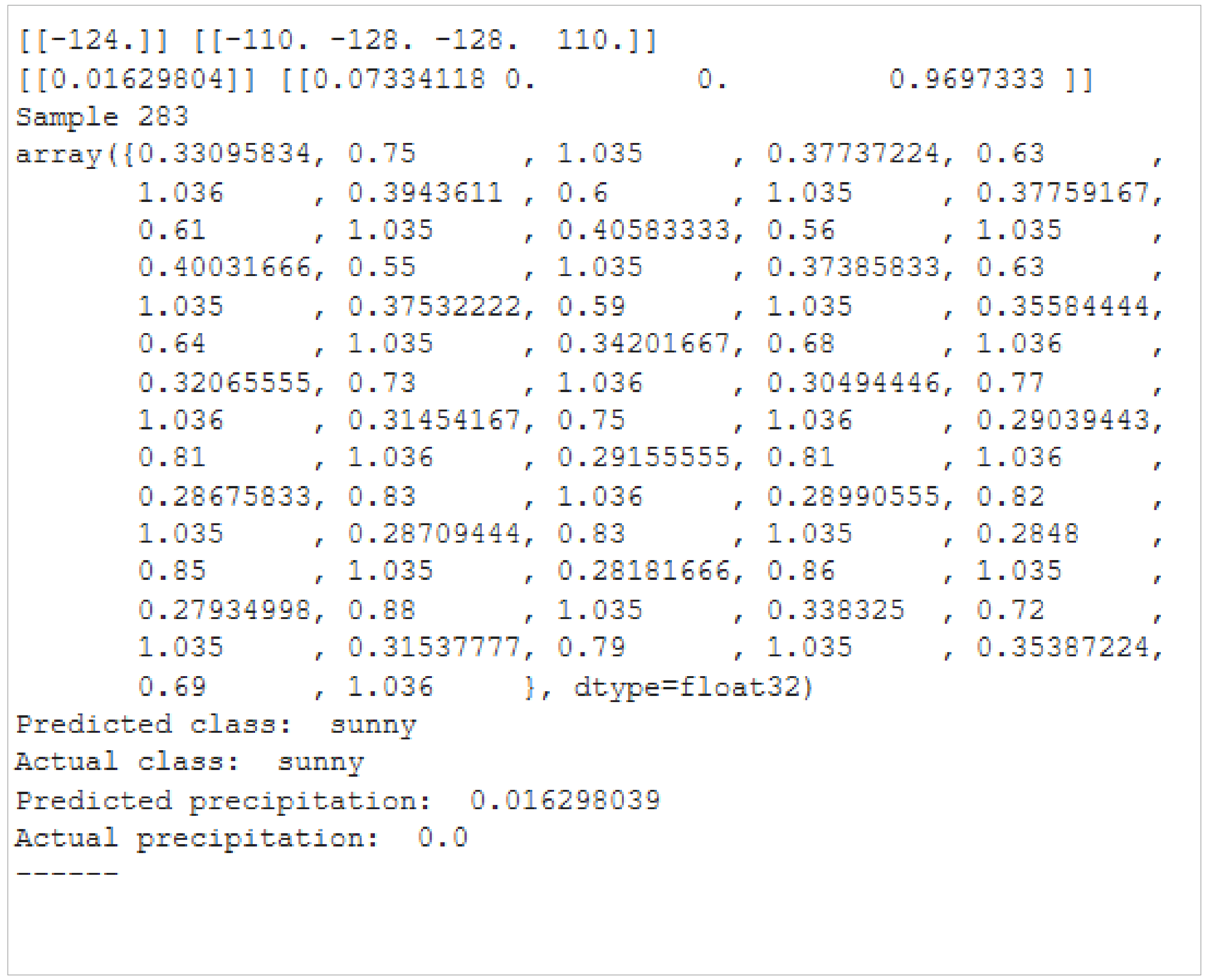
```
// Run inference, and report any error
  TfLiteStatus invoke_status = interpreter–>Invoke();

  if (invoke_status != kTfLiteOk) {
    TF_LITE_REPORT_ERROR(error_reporter, ”Invoke failed”);
    return;
  }
```

Similar to input, the output of the model is also quantized, so we need to perform the reverse operation and convert it from INT8 to float.

```
// Obtain the quantized output from model’s output tensor
  float y_type[4];
  // Dequantize the output from integer to floating–point
  int8_t y_precip_q = output_precip–>data.int8[0];
  Serial.println(y_precip_q);
   float y_precip = (y_precip_q – output_precip–>params.zero_point) * output_precip–
```

```
>params.scale;
    Serial.print(”Precip: ”);
    Serial.print(y_precip);
    Serial.print(”\t”);
    Serial.print(”Type: ”);
    for (byte i = 0; i < 4; i = i + 1) {
       y_type[i] = (output_type–>data.int8[i] – output_type–>params.zero_point) * output_
type–>params.scale;
      Serial.print(y_type[i]);
      Serial.print(” ”);
    }
    Serial.print(”\n”);
  }
```

Check and compare the values for the same data point, they should be the same for quantized Tensorflow Lite model in Colab notebook and Tensorflow Micro model running on your Wio Terminal.

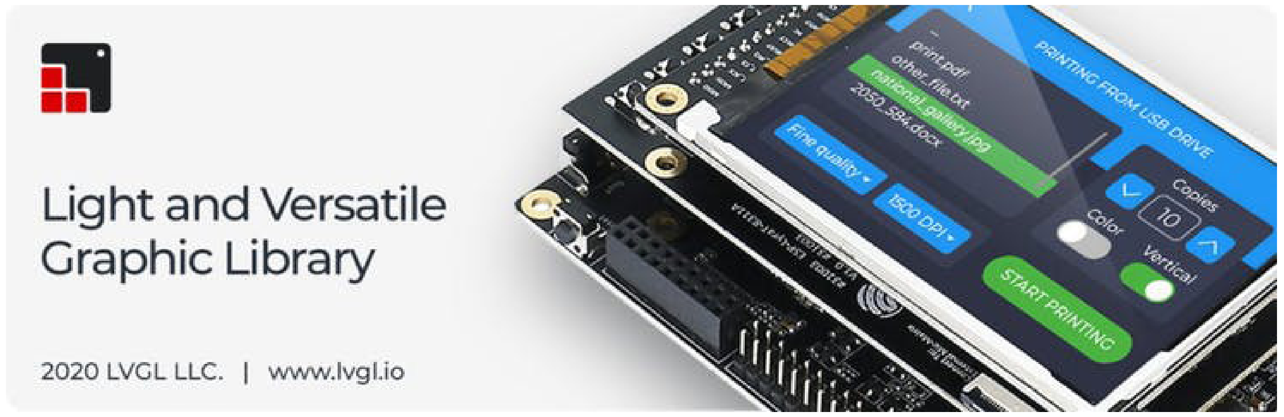


```
[[-124.]] [[-110. -128. -128.  110.]]
[[0.01629804]] [[0.07334118 0.         0.         0.9697333 ]]
Sample 283
array({0.33095834, 0.75      , 1.035     , 0.37737224, 0.63      ,
       1.036     , 0.3943611 , 0.6       , 1.035     , 0.37759167,
       0.61      , 1.035     , 0.40583333, 0.56      , 1.035     ,
       0.40031666, 0.55      , 1.035     , 0.37385833, 0.63      ,
       1.035     , 0.37532222, 0.59      , 1.035     , 0.35584444,
       0.64      , 1.035     , 0.34201667, 0.68      , 1.036     ,
       0.32065555, 0.73      , 1.036     , 0.30494446, 0.77      ,
       1.036     , 0.31454167, 0.75      , 1.036     , 0.29039443,
       0.81      , 1.036     , 0.29155555, 0.81      , 1.036     ,
       0.28675833, 0.83      , 1.036     , 0.28990555, 0.82      ,
       1.035     , 0.28709444, 0.83      , 1.035     , 0.2848    ,
       0.85      , 1.035     , 0.28181666, 0.86      , 1.035     ,
       0.27934998, 0.88      , 1.035     , 0.338325  , 0.72      ,
       1.035     , 0.31537777, 0.79      , 1.035     , 0.35387224,
       0.69      , 1.036     }, dtype=float32)
Predicted class:  sunny
Actual class:  sunny
Predicted precipitation:  0.016298039
Actual precipitation:  0.0
------
```

Expansion tasks

Now the next step is to make it from a demo into actually useful project. Open the sketch wio_terminal_tfmicro_weather_prediction_static.ino from course materials and have a look at its content.

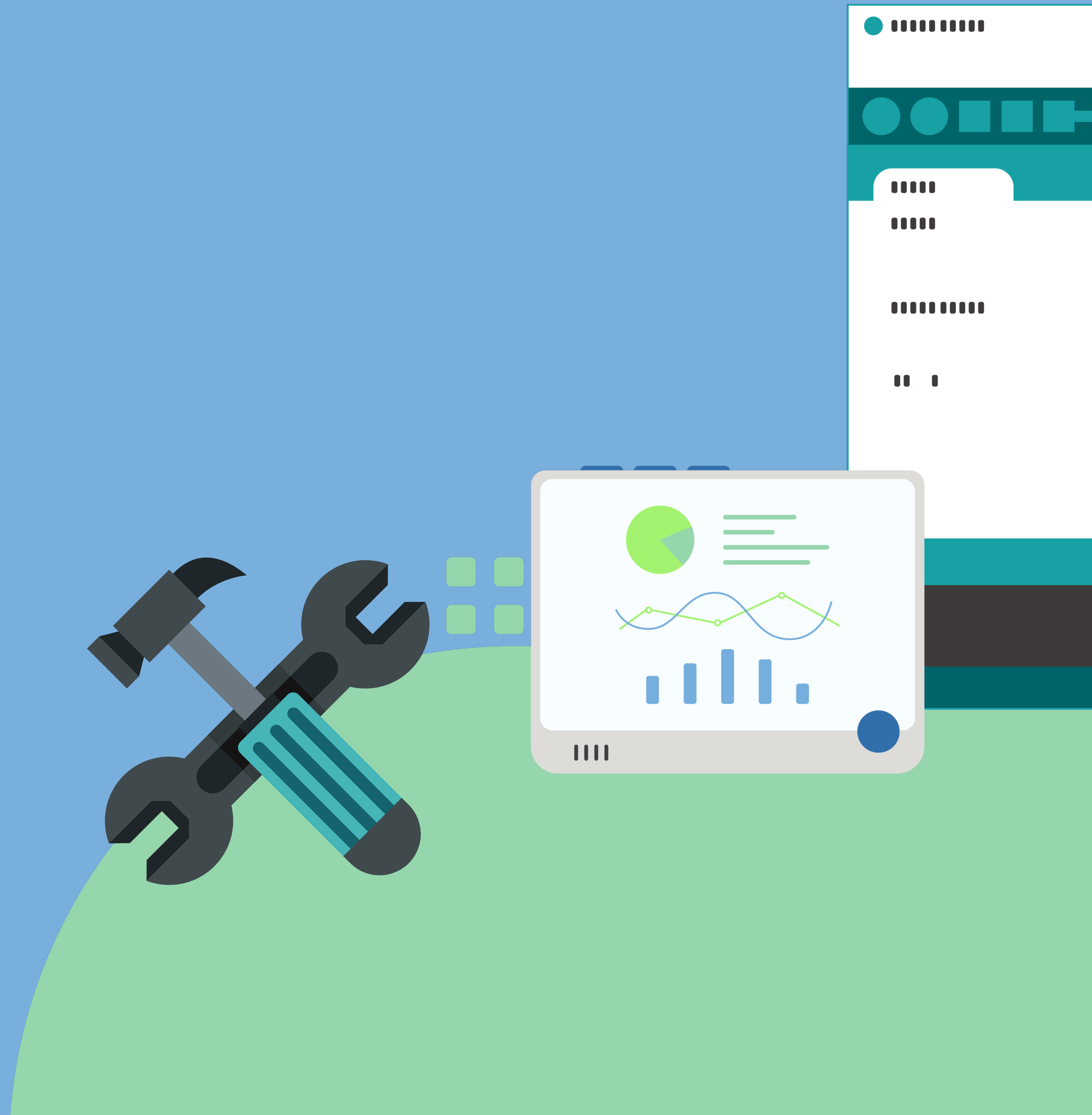
The code is divided into main sketch, get_historical_data and GUI parts. Since our model requires the data for past 24 hours we would need to wait 24 hours to perform the first inference, which is a lot — to solve this problem we get the weather for past 24 hours from openweathermap.com API and can perform the first inference immediately after device boots up and then replace the values in the circular buffer with temperature, humidity and pressure from BME280 sensor connected to Wio Terminal I2C Grove socket. For GUI we used LVGL, a Little and Versatile Graphics Library.



Compile and upload the code, make sure you change WiFi credentials, your location and openweathermap.com API key in sketch before uploading. After upload the device will connect to the Internet, get the data for last 24 hours for your location and perform the first inference. Then it will wait for 1 hour before getting the values from BME280 sensor connected to Wio Terminal – if no sensor connected, the program will not initialize.

Lesson 12

Student project



In this course so far we have learned how to use Edge Impulse and Tensorflow Lite for Microcontrollers to train and deploy highly–optimized models for inference with Wio Terminal. You also should have an understanding of how to combine model inference with integration to Cloud services and/or graphical user interface.

But of course learning and making doesn't stop here – for the last lesson of the coruse, we encourage you to think outside of the box and create a project of your own. Number of projects with TinyML are growing by day and below are some of the most interesting ones our team found on the Internet.

Community projects

TinyML Water Sensor – Based on Edge Impulse & Arduino Sense

https://www.hackster.io/enzo2/tinyml–water–sensor–based–on–edge–impulse–arduino–sense–f8b133

TinyML implementation to identify running water tap sound and once heard one, a buzzer + LED timer is triggered.

The device is listening continuously for any sound, and once a running tape is identified a timer is

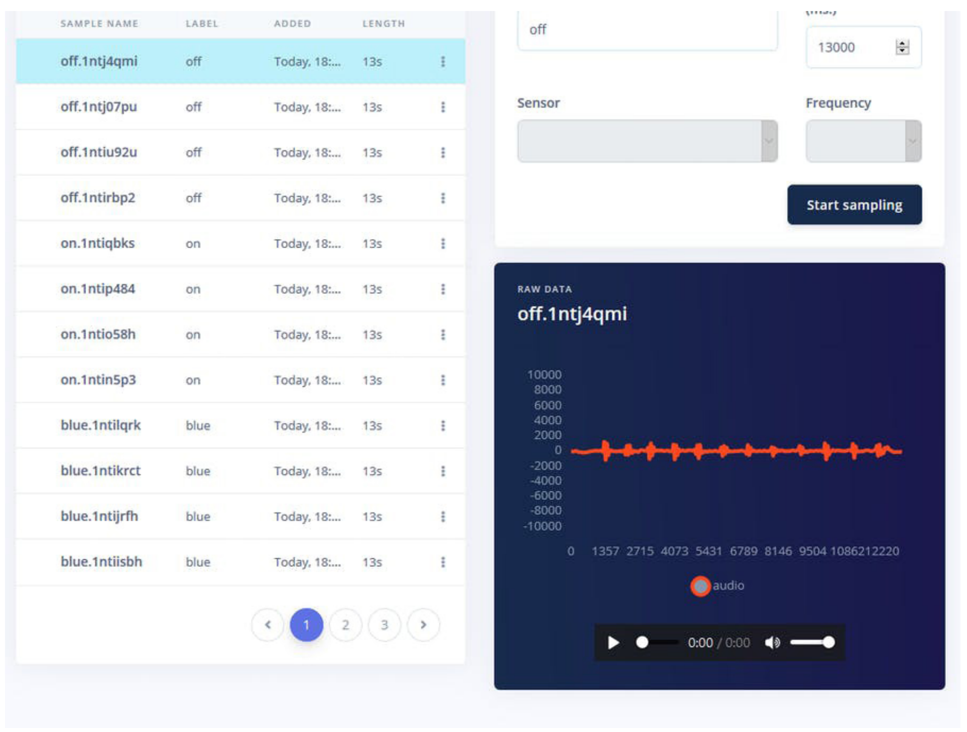
triggered to produce 20 seconds buzzing sound and a flashing LED.

The project is based on the two components an Arduino Nano 33 BLE Sense and the Edge Impulse Studio as developing platform.

TinyML Keyword Detection for Controlling RGB Lights

https://www.hackster.io/gatoninja236/tinyml–keyword–detection–for–controlling–rgb–lights–9f51e9

Train a TensorFlow model to recognize certain keywords and control an RGB light strip using an Arduino Nano 33 BLE Sense.



SAMPLE NAME	LABEL	ADDED	LENGTH
off.1ntj4qmi	off	Today, 18:...	13s
off.1ntj07pu	off	Today, 18:...	13s
off.1ntiu92u	off	Today, 18:...	13s
off.1ntirbp2	off	Today, 18:...	13s
on.1ntiqbks	on	Today, 18:...	13s
on.1ntip484	on	Today, 18:...	13s
on.1ntio58h	on	Today, 18:...	13s
on.1ntin5p3	on	Today, 18:...	13s
blue.1ntilqrk	blue	Today, 18:...	13s
blue.1ntikrct	blue	Today, 18:...	13s
blue.1ntijrfh	blue	Today, 18:...	13s
blue.1ntiisbh	blue	Today, 18:...	13s

off

13000

Sensor

Frequency

RAW DATA

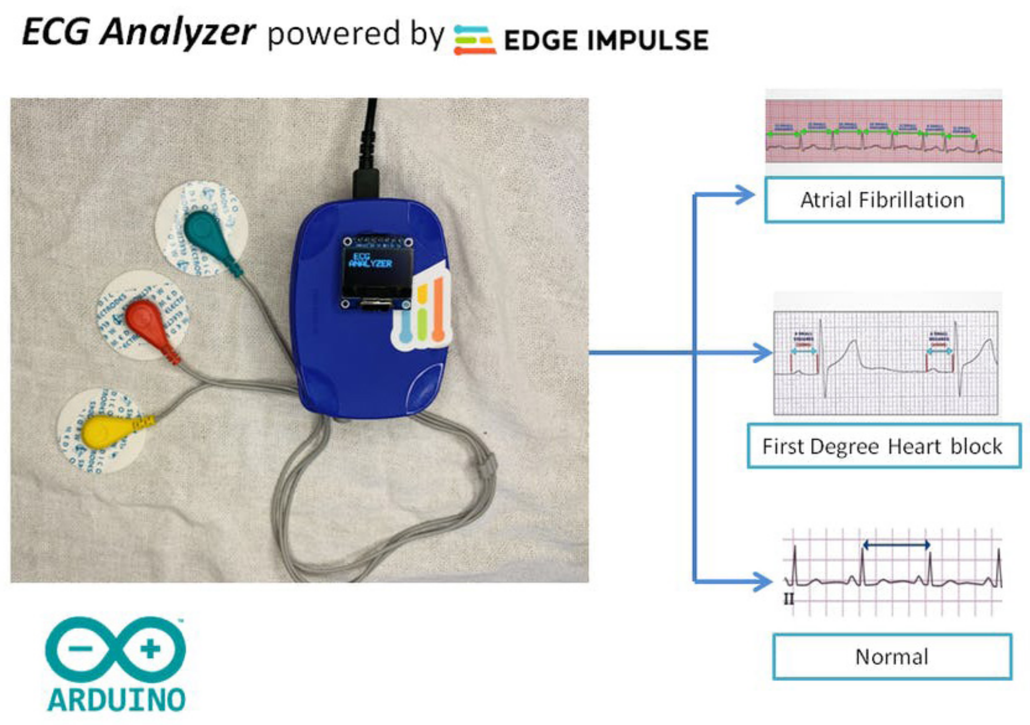
off.1ntj4qmi

ECG Analyzer Powered by Edge Impulse

https://www.hackster.io/manivannan/ecg–analyzer–powered–by–edge–impulse–24a6c2

A TinyML based Medical device powered by Edge Impulse to predict Atrial fibrillation, AV Block 1 and Normal ECG with >90%.

TinyML application powered by Edge Impulse to develop a mini–Diagnosis ECG analyzer device which can fit in a pocket and it can diagnose heart diseases independently without a cloud connectivity.

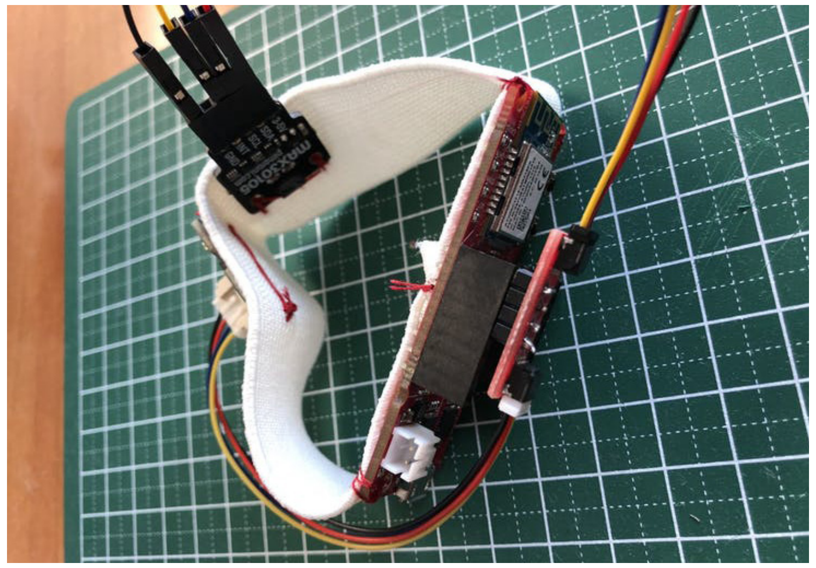


Fall Detection and Heart Rate Monitoring Using AVR–IoT

https://www.hackster.io/naveenbskumar/fall-detection-and-heart-rate-monitoring-using-avr-iot-75fb16

A wearable low–powered device detects falls using machine learning at the edge. It also monitors the heart rate.

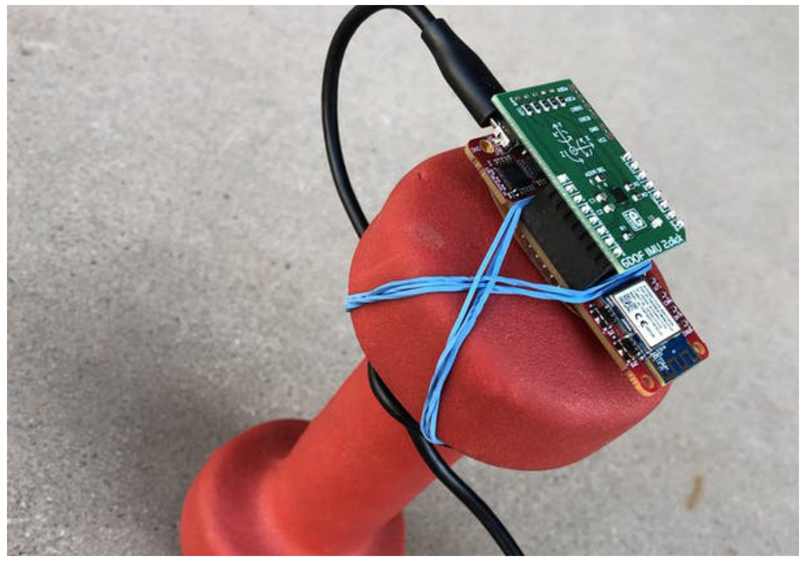
Falls can result in physical and psychological trauma, especially for the elderly. In order to improve the quality of life of these patients this project presents the development of a fall detection and heart rate monitoring system. The wearable device obtains information from accelerometer and heart rate sensor and sends them to the AWS IoT Core. The fall detection is done offline on the chip using a machine learning algorithm.



Build a Smart Dumbbell with the SAMD21 Machine Learning EVK

https://www.hackster.io/alex-jagger/build-a-smart-dumbbell-with-the-samd21-machine-learning-evk-c86cae

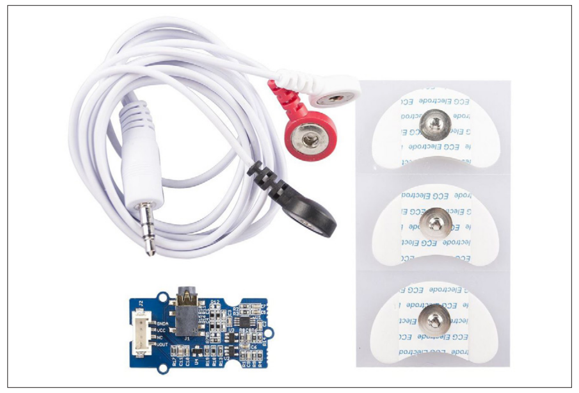
Turn your dumbbell into a smartbell with the Microchip SAMD21 ML EVK and an embedded ML classifier built in the Edge Impulse Studio.



Ideas from Seeed EDU team

Pet sounds translator

Use Wio Terminal internal microphone to record the dataset of sounds of your pet being hungry, angry, lonely, happy, excited, etc and then train the model that would help you (and other people) to understand it better!



Robot control with EMG detector

Utilize Grove – EMG Detector module to analyze arm musucle activity and control your own robot with arm gestures.

Intelligent plant care

Employ BME280 Temperature & Humidity & Pressure, Wio Terminal Internal Light sensor and possibly Grove Moisture Senor to monitor and control plant environment for optimal growth.

Wio Terminal on Bittle

Install Wio Terminal on Bionic robotic dog from Petoi LLC. and Seeed Edu – both Bittle and Wio Terminal have Raspberry Pi pin connectors, which makes interfacing them easy.

TinkerGen

STEM made simple

TinkerGen STEM made simple | seeed studio

To know more about TinkerGen

Please contact us:

contact@chaihuo.org 86–0755–86716703 www.tinkergen.com