



**Electrical and Computer Engineering
Erik Jonsson School of Engineering & Computer Science
The University of Texas at Dallas
Dr. Tooraj Nikoubin**

Lab 5- TinyML - Continuous Motion Recognition

Contents

Project Objectives.....	3
Texas Instruments CC1352P LaunchPad	4
Edge Impulse	4
Installing Procedure.....	5
1. Edge Impulse CLI.....	5
2. Installation - Windows	5
3. Texas Instruments UniFlash	6
Connecting To Edge impulse: Configure your hardware	6
1. Connect the development board to your computer.....	7
2. Update the firmware.....	7
3.Setting the Keys.....	8
IV. Connection to which device	8
Project Procedure.....	8
1. Verifying that the device is connected:.....	8
2. Record New Data.....	9
3. Designing an Impulse	11
4. Configuring the spectral analysis block.....	12
5. Configuring the Neural Network	13
6. Classifying the new Data	15
7. Anomaly Detection.....	16
8. Deploying back to device	18
Troubleshooting	19
Instructions	20
Expected Outcome.....	20
1. Report.....	20
2. Video report.....	21

TinyML - Continuous Motion Recognition

Project Objectives

- Understand about the concept of TinyML.
- A brief understanding about ML algorithms
- A brief understanding about TI Launchpad (CC1352P) and Booster Sensors.

Introduction

Machine learning is a subfield of artificial intelligence, which is broadly defined as the capability of a machine to imitate intelligent human behavior. Artificial intelligence systems are used to perform complex tasks in a way that is similar to how humans solve problems.

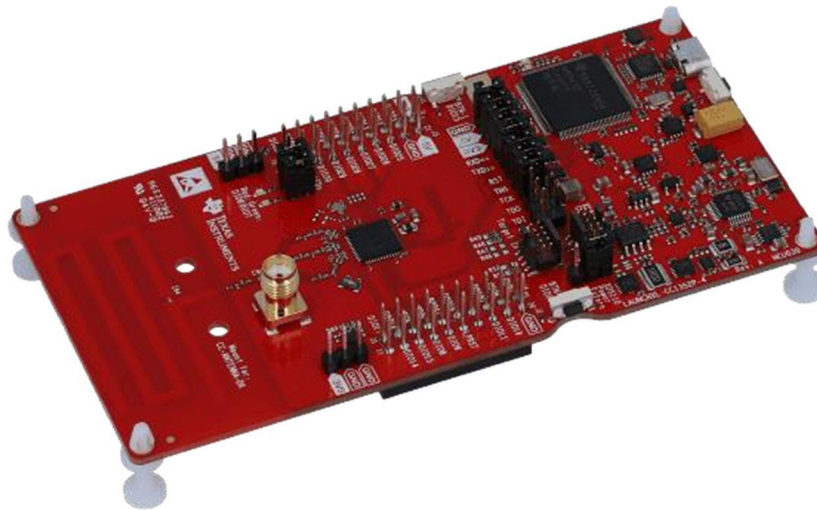
Tiny machine learning, or TinyML, is an emerging field that is at the intersection of machine learning and embedded systems. An embedded system is a computing device that usually is small, or tiny, that operates with low power, extremely low power. So much so that some of these devices can run for days, weeks, months, sometimes even years on something like a coin cell battery. TinyML is a type of machine learning that shrinks deep learning networks to fit on tiny hardware. It brings together Artificial Intelligence and intelligent devices.

Edge computing brings computation and data storage closer to the origin of data. Majority of the edge devices that are integrated with IoT-based ecosystems are initially designed to collect sensor data and transmission of the data to neighborhood or remote cloud.

In this project, machine learning is used to build a gesture recognition system that runs on a microcontroller. 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.

Texas Instruments CC1352P LaunchPad

This LaunchPad speeds development on devices with integrated power amplifier and multi-band radio support for concurrent Sub-1GHz and 2.4-GHz operation. Protocols supported include Bluetooth Low Energy, Sub-1 GHz, Thread, Zigbee®, 802.15.4, and proprietary RF with the compatible CC13x2-CC26x2 SDK. It has Broad band antenna support for Sub-1 GHz (868 MHz / 915 MHz / 433 MHz) and 2.4 GHz frequency bands. [Following is the link to the datasheet.](#)



Edge Impulse

It is a cloud service for developing machine learning models in the TinyML targeted edge devices. This supports AutoML processing for edge platforms. It also supports several boards including smart phones to deploy learning models in such devices. Training is done on the cloud platform and the trained model can be exported to an edge device by following a data forwarder enabled path. The impulse can be run in local machine by the help from the in-built C++, Node.js, Python, and Go SDKs. Impulses are also deployable as a WebAssembly library.

Installing Procedure

Please watch this [video](#) for detailed instructions

1. Installing Dependencies

To set this device up in Edge Impulse, you will need to install the following software:

1. 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. The CLI consists of the following tools:

- **edge-impulse-daemon** - configures devices over serial and acts as a proxy for devices that do not have an IP connection.
- **edge-impulse-uploader** - allows uploading and signing local files.
- **edge-impulse-data-forwarder** - a very easy way to collect data from any device over a serial connection and forward the data to Edge Impulse.
- **edge-impulse-run-impulse** - show the impulse running on your device.
- **edge-impulse-blocks** - create organizational transformation blocks.

2. Installation - Windows

1. Install [Python 3](#) on your host computer.
2. Install [Node.js](#) v14 or higher on your host computer.
 - For Windows users, install the **Additional Node.js tools** (called **Tools for Native Modules** on newer versions) when prompted.

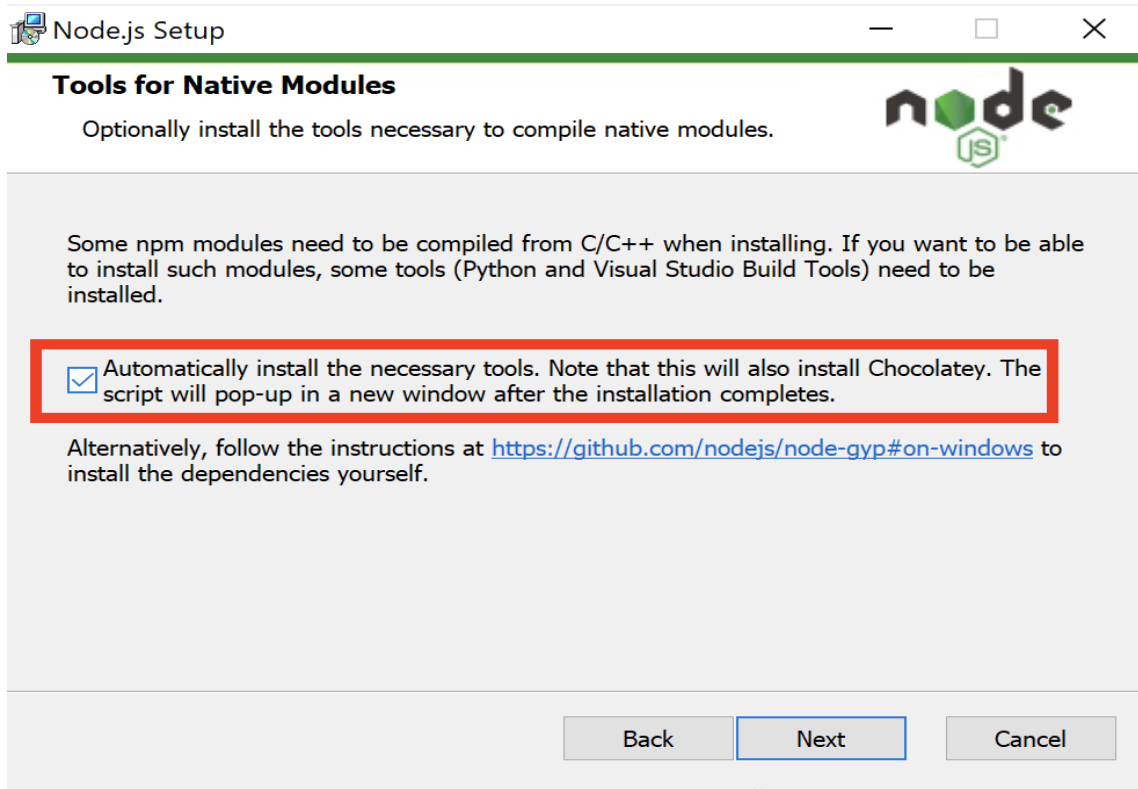


Fig: Install the additional Node.js tools

3. Install the CLI tools via:

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

You should now have the tools available in your PATH.

4. If you haven't already, create an [edge impulse account](#). Many of our CLI tools require the user to log in to connect with the Edge Impulse Studio.

3. Texas Instruments UniFlash

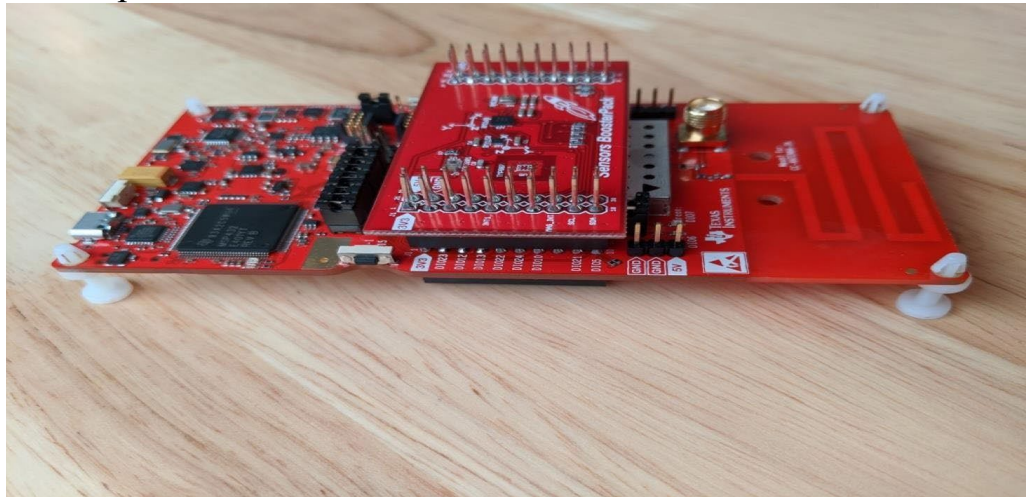
1. Install the desktop version for your operating system [here](#)
2. Add the installation directory to your PATH

Connecting To Edge impulse: Configure your hardware

With all the software in place it's time to connect the development board to Edge Impulse.

To interface the Launchpad with sensor hardware, you will need to connect the BOOSTXL-SENSORS to collect accelerometer data. Follow the steps below:

Connecting the BOOSTXL-SENSORS board to the Launchpad is simple. Just orient the sensor board such that the 3V3 and GND markings on the booster pack line up with the Launchpad, and then attach the booster pack to the top header pins of the Launchpad, as shown below:



1. Connect the development board to your computer

Use a micro-USB cable to connect the development board to your computer.

2. Update the firmware

The development board does not come with the right firmware yet. To update

- I. [Download the latest Edge Impulse firmware](#), and unzip the file.
- II. Open the flash script for your operating system (`flash_windows.bat`) to flash the firmware.
- III. Wait until flashing is complete and press the RESET button once to launch the new firmware.

If there are problems flashing the firmware onto the LaunchPad, see the Troubleshooting section

3.Setting the Keys

From a command prompt or terminal, run:

edge-impulse-daemon

This will start a wizard which will ask you to log in and choose an Edge Impulse project. If you want to switch projects run the command with **--clean**.

IV. Connection to which device

The Launchpad enumerates two serial ports. The first is the Application/User UART, which the edge-impulse firmware communications through. The other is an Auxiliary Data Port, which is unused.

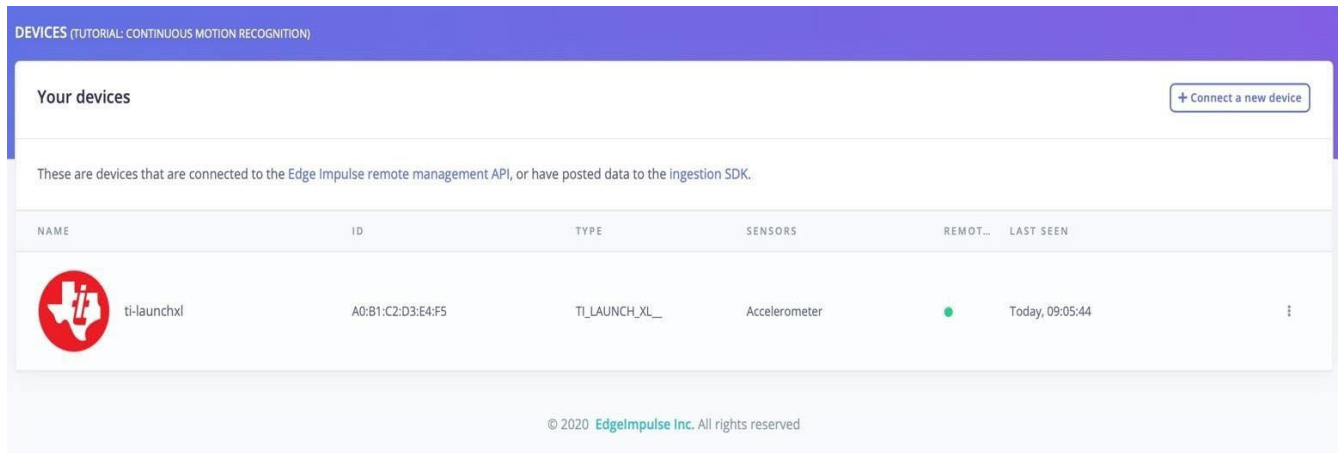
When running the edge-impulse-daemon you will be prompted on which serial port to connect to.

Generally, select the lower numbered serial port. This usually corresponds with the Application/User UART. On windows, the serial port may also be verified in the device manager.

Project Procedure

1. Verifying that the device is connected:

To verify this, go to your Edge Impulse project, and click **Devices**. The device will be listed here.





2. Record New Data

With your device connected we can collect some data. In the studio go to the Data acquisition tab. This is the place where all your raw data is stored, and - if your device is connected to the remote management API - where you can start sampling new data.

Under Record new data, select your device, set the label to updown, 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 updown. You can later edit these labels if needed.

Collect data



Device ?

Motion

Label

updown

Sample length (ms.)

10000

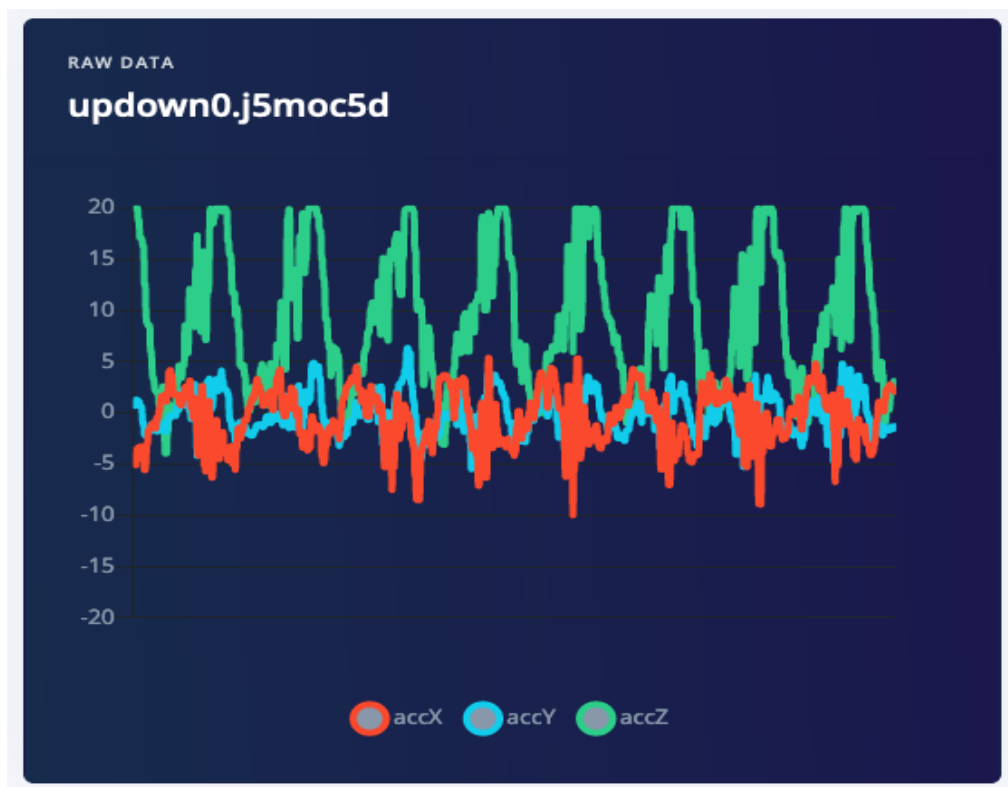
Sensor

Accelerometer

Frequency

62.5Hz

Start sampling



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.
- Snake - moving the device over your desk as a snake.
- Wave - waving the device from left to right.
- Updown - moving the device up and down.

3. Designing an Impulse

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 project 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 'Classifier' learning block, that takes these spectral features and learns to distinguish between the four (idle, snake, wave, updown) classes.

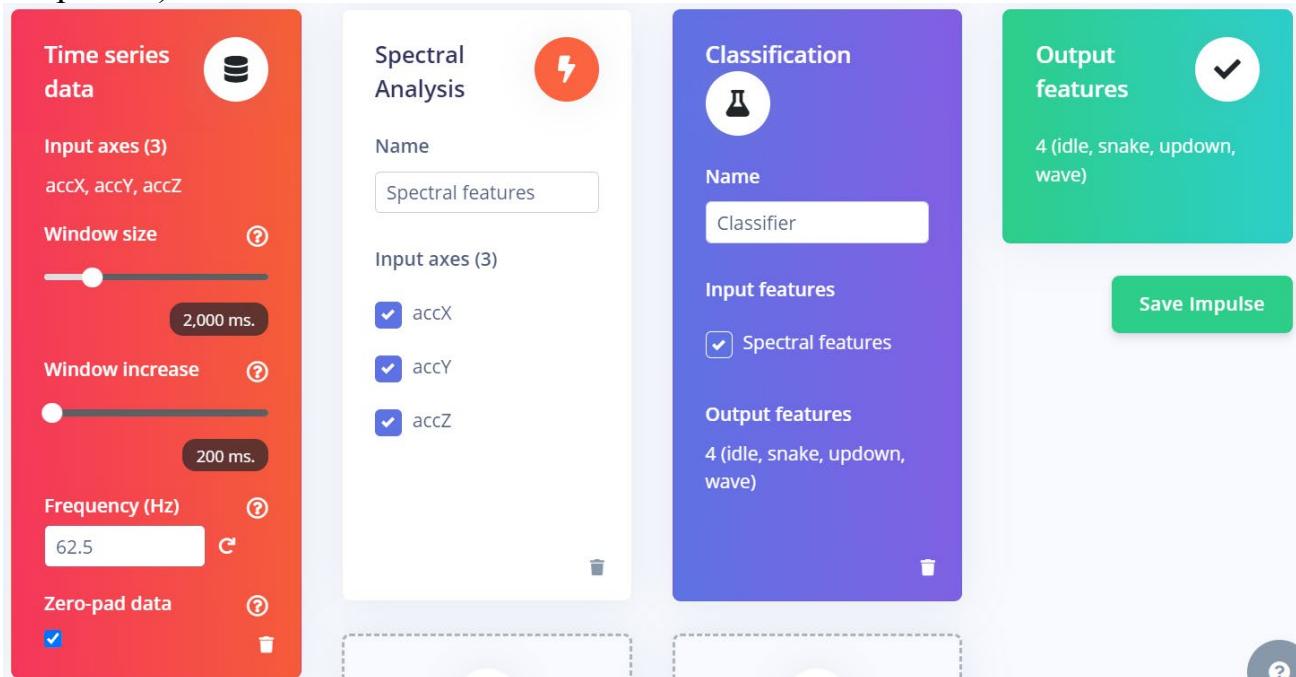


Fig: First impulse, with one processing block and one learning block.

4. Configuring the spectral analysis block

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. You can keep the parameter values as default for the initial training and if you get less training and testing accuracy you can come back and adjust these values.

Parameters
Autotune parameters

Filter

Scale axes ? 1

Input decimation ratio ? 1

Type ? none

Analysis

Type ? FFT

FFT length ? 16

Take log of spectrum? ? ☒

Overlap FFT frames? ? ☒

Improve low frequency resolution? ? ☐

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.. A good rule of thumb is that if you can visually separate the data, then the machine learning model will be able to do so as well.

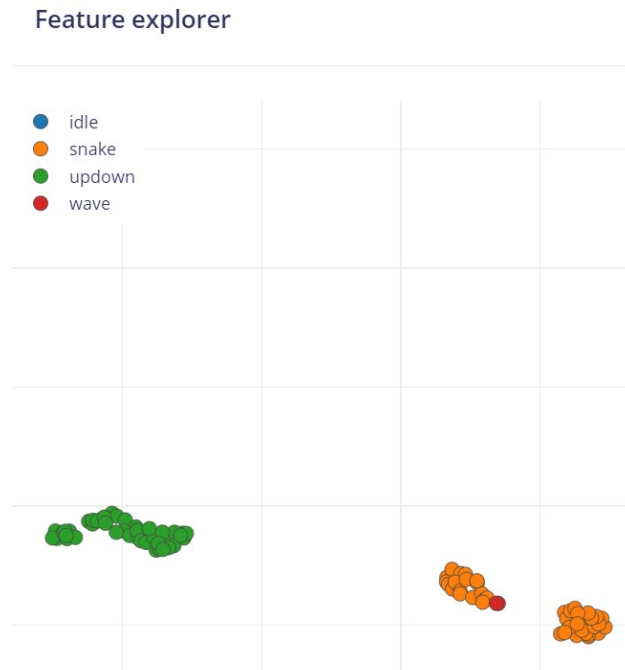


Fig: Examining your full dataset in the feature explorer.

5. Configuring the Neural Network

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 four 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 **Classifier** in the menu.

Keep the Neural Network parameter as default and click on **Start training**, if the training accuracy is less than 80% you need to adjust these parameters.

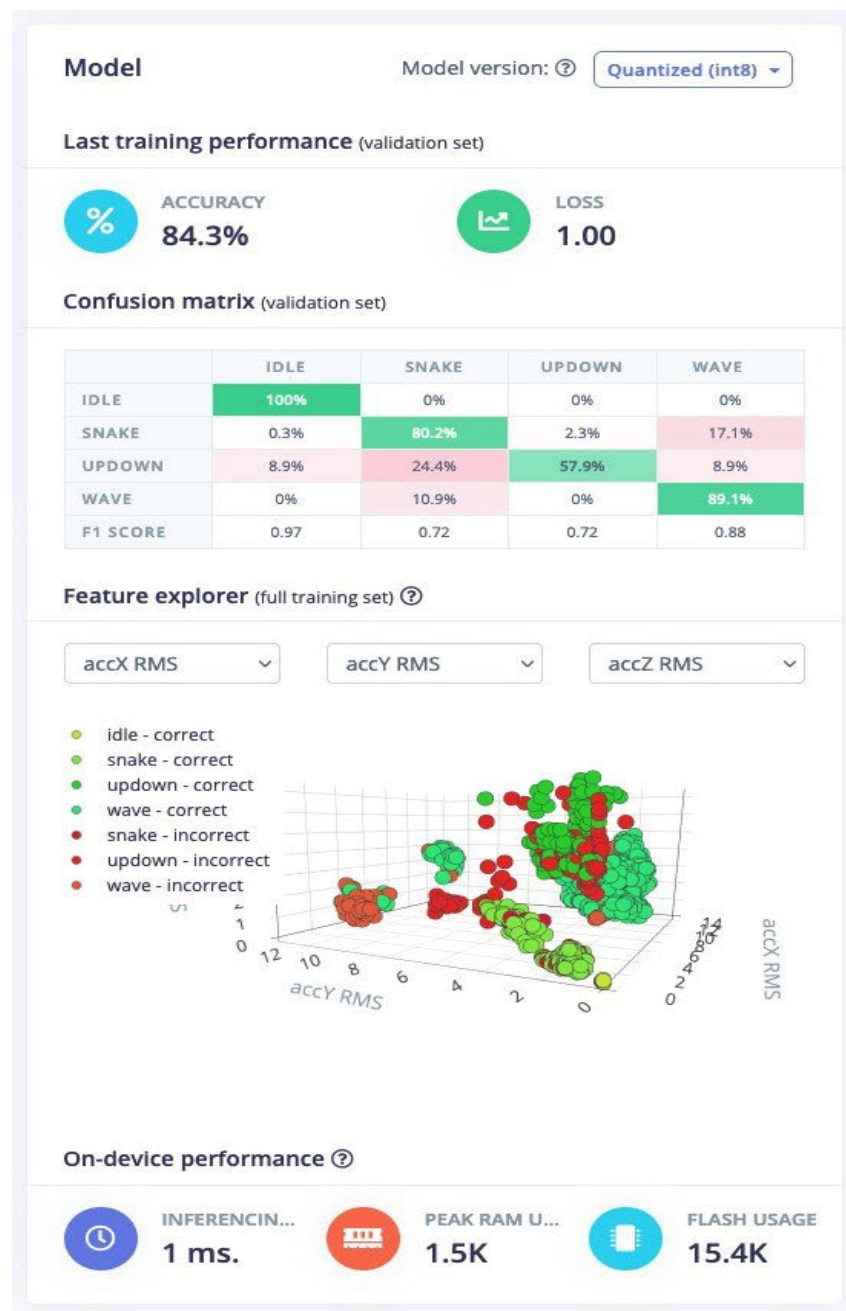


Fig: Training performance after a single iteration.

6. Classifying the new Data

From the statistics in the previous step we know that the model works against our training data, but how well would the network perform on new data? Click on **Live classification** in the menu to find out. Your device should (just like in step 2) show as online under 'Classify new data'. Set the 'Sample length' to 5000 (5 seconds), click **Start sampling** and start doing movements. Afterwards you'll get a full report on what the network thought that you did.

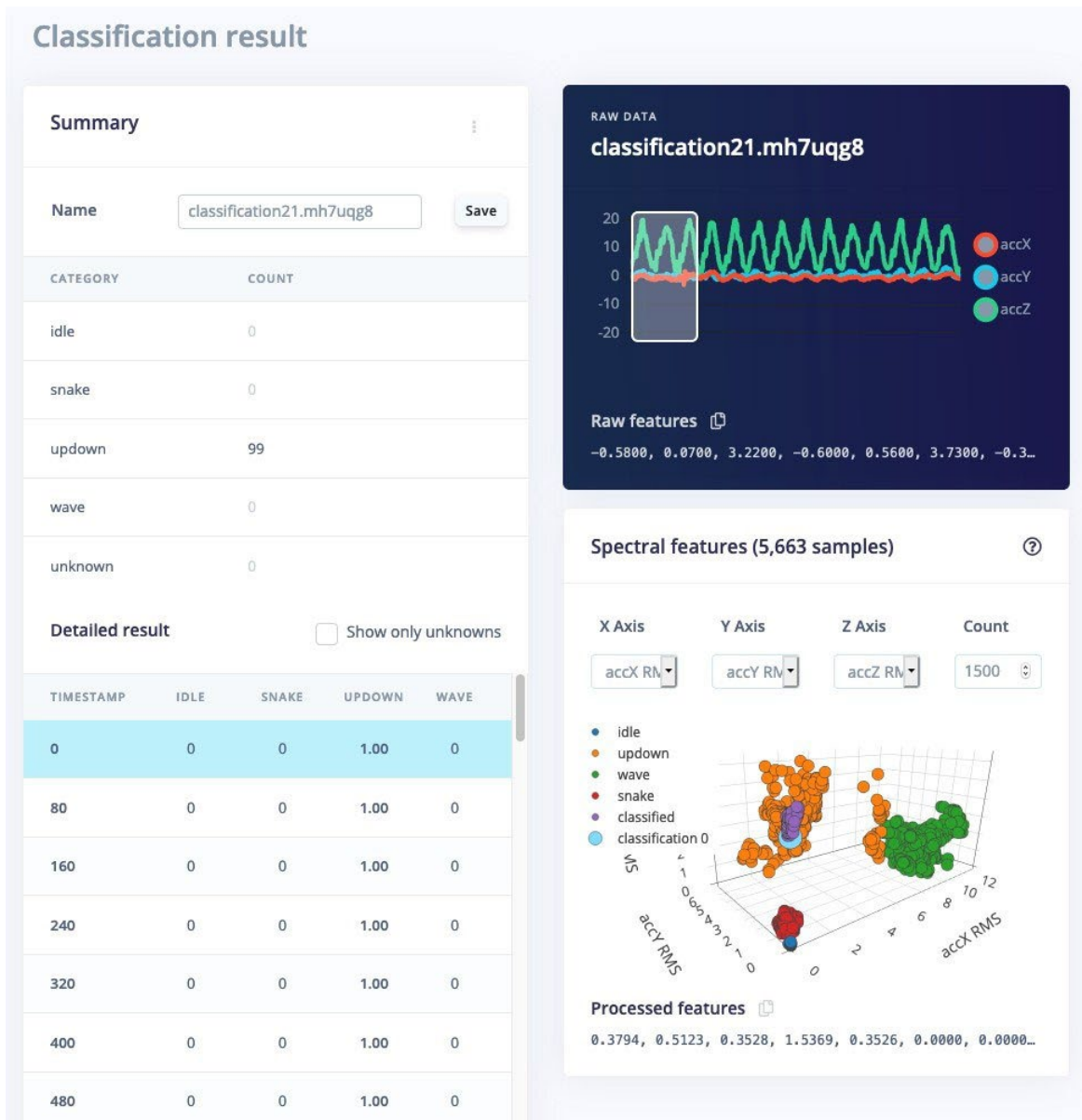


Fig: Classification result

If the network performed great, fantastic! But what if it performed poorly? There could be a variety of reasons, but the most common ones are:

1. There is not enough data. Neural networks need to learn patterns in data sets, and the more data the better.
2. The data does not look like other data the network has seen before. This is common when someone uses the device in a way that you didn't add to the test set. You can add the current file to the test set by clicking :, then selecting **Move to training set**. Make sure to update the label under 'Data acquisition' before training.
3. The model has not been trained enough. Up the number of epochs to 200 and see if performance increases (the classified file is stored, and you can load it through 'Classify existing validation sample').
4. The model is overfitting and thus performs poorly on new data. Try reducing the learning rate or add more data.
5. The neural network architecture is not a great fit for your data. Play with the number of layers and neurons and see if performance improves.

As you see there is still a lot of trial and error when building neural networks, but we hope the visualizations help a lot. You can also run the network against the complete validation set through 'Model validation'.

With a working model in place we can look at places where our current impulse performs poorly.

7. Anomaly Detection

Neural networks are great, but they have one big flaw. They're terrible at dealing with data they have never seen before (like a new gesture). Neural networks cannot judge this, as they are only aware of the training data. If you give it something unlike anything it has seen before it'll still classify as one of the four classes.

So... how can we do better? If you look at the feature explorer on the accX RMS, accY RMS and accZ RMS axes, you should be able to visually separate the classified data from the training data. We can use this to our advantage by training a new (second) network that creates clusters around data that we have seen before and compares incoming data against these clusters. If the distance from a cluster is too large you can flag the sample as an anomaly, and not trust the neural network.

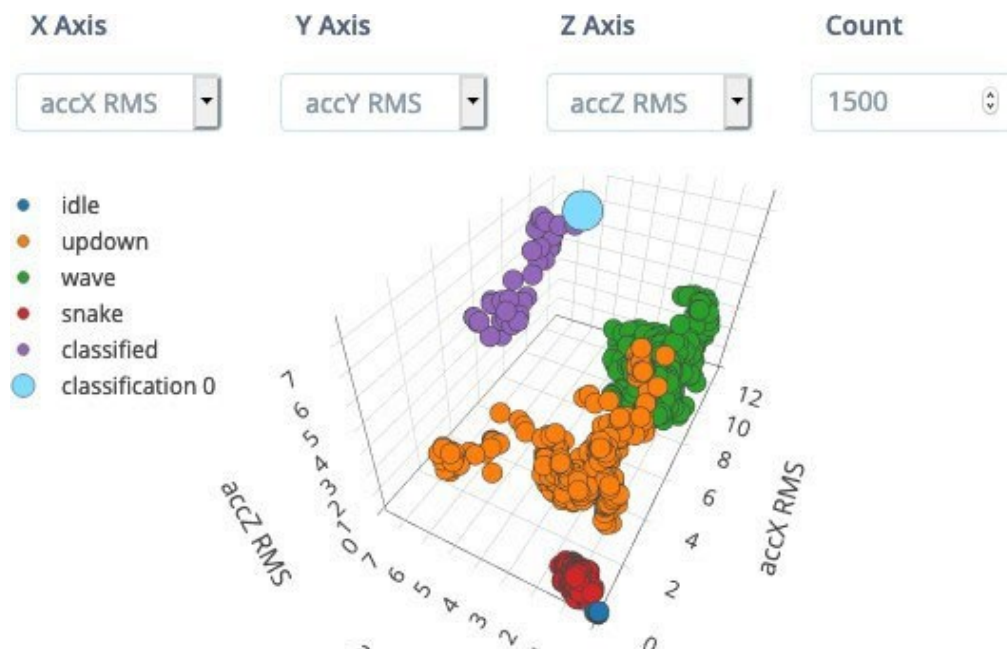


Fig: Shake data is easily separated from the training data.

To add this block, go to **Create impulse**, click **Add learning block**, and select 'Anomaly Detection (K-Means)'. Then click **Save impulse**.

To configure the clustering model, click on **Anomaly detection** in the menu. Here we need to specify:

- The number of clusters. Here use 32.
- The axes that we want to select during clustering. As we can visually separate the data on the accX RMS, accY RMS and accZ RMS axes, select those.

Click **Start training** to generate the clusters. You can load existing validation samples into the anomaly explorer with the dropdown menu.

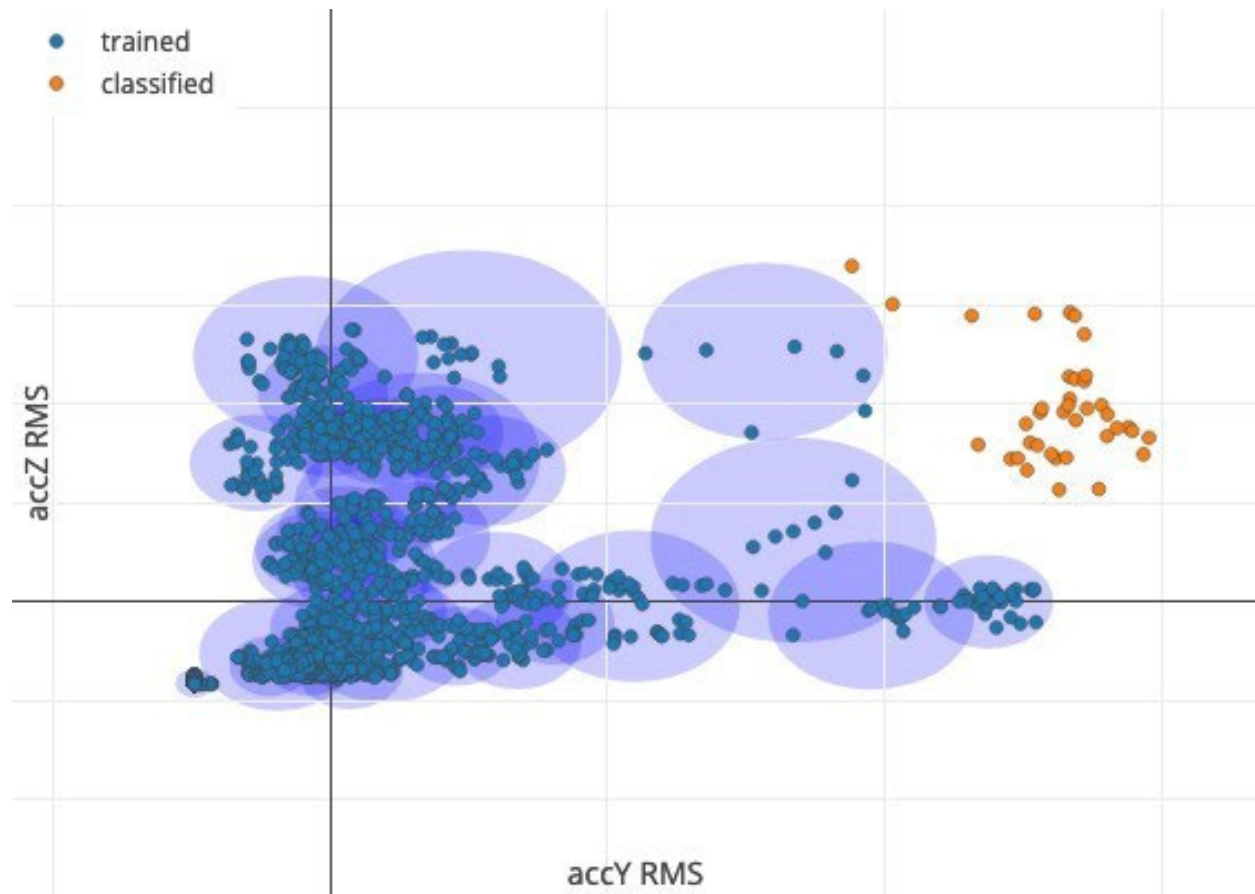


Fig: Known clusters in blue, the live input data in orange.

8. Deploying back to device

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. Edge Impulse can package up the complete impulse - including the signal processing code, neural network weights, and classification code - up in a single C++ library that you can include in your embedded software. To export your model, click on **Deployment** in the menu. Then under 'Build firmware' select your development board and click **Build**. This will export the impulse and build a binary that will run on your development board in a single step. After building is completed, you'll get prompted to download a binary. Save this on your computer.

```
Starting inferencing in 2 seconds...
Sampling... Storing in file name: /fs/device-classification261
Tensor shape: 4
Predictions (DSP: 17 ms., Classification: 1 ms., Anomaly: 0 ms.):
  idle: 0.00004
  snake: 0.00012
  updown: 0.00009
  wave: 0.99976
  anomaly score: 0.032
Finished inferencing, raw data is stored in '/fs/device-classification261'. Use AT+UPLOADFILE to send
```

Fig: Sample output of wave type motion

Troubleshooting

If device Failed to flash :-

If the UniFlash CLI is not added to your PATH, the install scripts will fail. To fix this, add the installation directory of UniFlash to your PATH on:

- Windows: use the following link->
<https://www.architectryan.com/2018/08/31/how-to-change-environment-variables-on-windows-10/>

If during flashing you encounter further issues, ensure:

- The device is properly connected and/or the cable is not damaged.
- You have the proper permissions to access the USB device and run scripts.

Instructions

1. Go through the instructions video provided.
2. Refer to the provided manual for step-by-step instructions to be followed for the project.

Expected Outcome

1. Report

1. Provide a report explaining the output of the project.
2. Your report should contain the following section.
 - I. Explanation of the blocks used while designing the impulse.
[Refer design impulse section.](#)
 - II. Explain performance of your model after training for one cycle. Provide screenshot of training performance after single iteration using feature explorer and confusion matrix.
[Refer configuring the neural network section](#)
 - III. Test your model before deploying on the device, explain classification result.
[Refer classify new data section](#)
 - IV. Deploy the trained model back to device.
Provide screenshot of terminal while the model is running.
Explain what you infer from the output.

Sample output is provided here

```
Starting inferencing in 2 seconds...
Sampling... Storing in file name: /fs/device-classification261
Tensor shape: 4
Predictions (DSP: 17 ms., Classification: 1 ms., Anomaly: 0 ms.):
  idle: 0.00004
  snake: 0.00012
  updown: 0.00009
  wave: 0.99976
  anomaly score: 0.032
Finished inferencing, raw data is stored in '/fs/device-classification261'. Use AT+UPLOADFILE to send
```

2. Video report

Provide a video report of maximum duration – 10 minutes.

1. Explain acquisition of data.
2. Explain designing the impulse.
3. Explain configuring of neural network.
4. Explain the testing of your model.
5. Show the procedure to deploy the model to the device.
6. Show the output on the terminal, explain the output.