Introduction to Machine Learning with Edge Impulse – Speech Recognition Lab

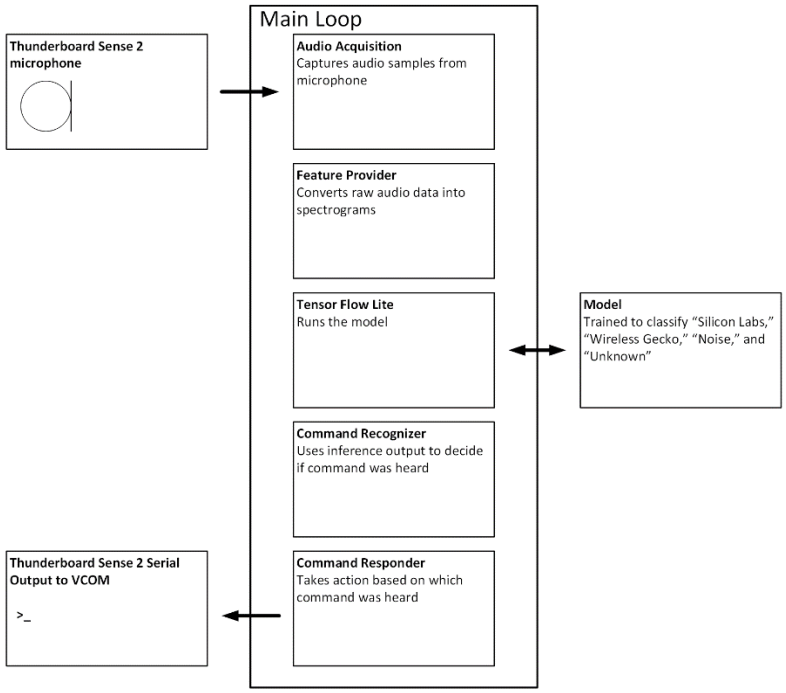
KEY POINTS

- Learn machine learning fundamental concepts.
- Gain experience using Edge Impulse's online studio.
- Collect data, build a machine learning model, and train the model in the cloud-based studio.
- Deploy the trained model and run real-time inference on an embedded device (Thunderboard Sense 2).

The goal of this lab is to provide a firm understanding of the development steps to build a machine learning application. This lab procedure walks through the steps to create a speech recognition project utilizing Edge Impulse's platform. Machine learning will be used to build a system that can recognize audible events, specifically your voice through audio classification. The system you create will recognize keywords such as "Hey Tyson" , even in the presence of other background noise or background chatter.

This lab will also go over the techniques of collecting audio data from microphones, use signal processing to extract the most important information, and train a deep neural network that can predict whether the keyword was heard in a given stream of audio. Finally, this system will be deployed to an EFR32MG12 on a Thunderboard Sense 2.

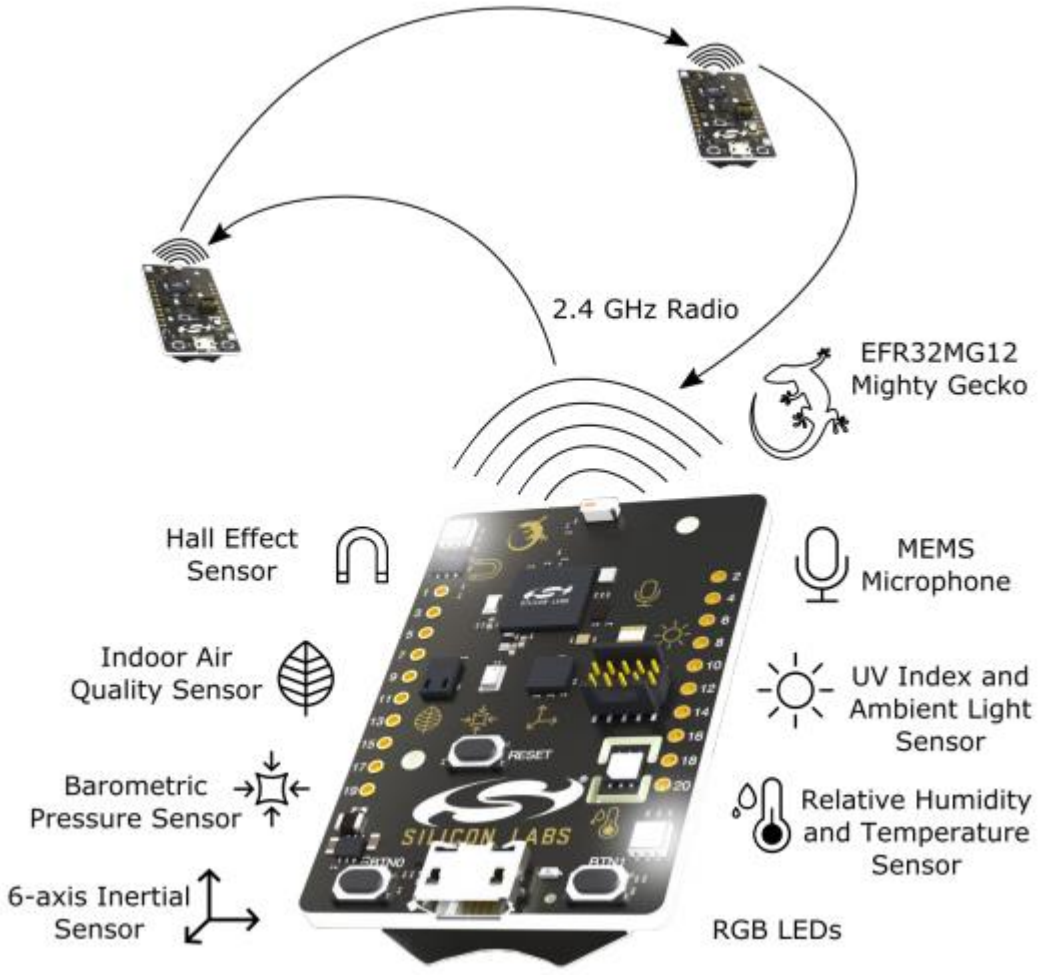
This machine learning model takes in one second's worth of data at a time. It outputs four probability scores, one for each of these four classes ("Silicon Labs," "Wireless Gecko," noise, and unknown), predicting how likely it is that the data represented is one of them.



1 Prerequisites

For this lab you will need the following:

- EFR32MG12 Thunderboard Sense 2 (SLTB004A)
- Micro-USB to USB Type-A cable (Not included with Thunderboard)
- A computer running Windows or Mac
- An account created with Edge Impulse using @silabs.com email address
- Any mobile phone or computer - for recording audio samples
- Prebuilt dataset provided with prework lab package

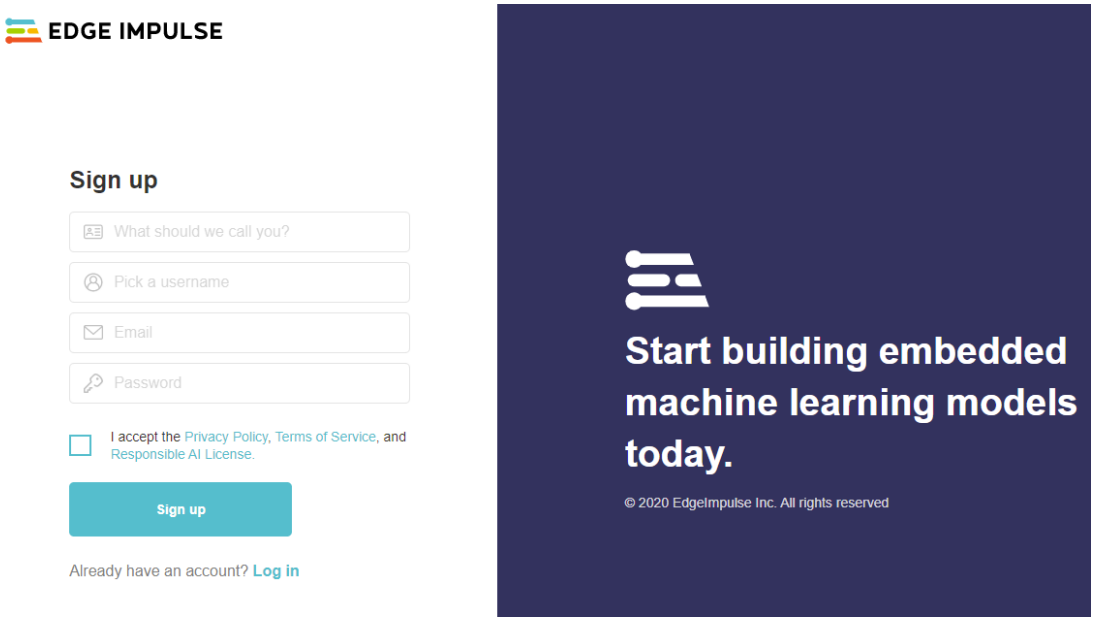


2 Edge Impulse Introduction and Setup

Edge Impulse is a development platform that can be used to create intelligent device solutions with machine learning on embedded devices. This section will go over getting set up with Edge Impulse.

2.1 Create an account with Edge Impulse

Sign up with the corresponding @silabs.com address on Edge Impulse's website here.



Sign in using the newly created credentials.

2.2 Installing Dependencies for Thunderboard Sense 2

To set up Thunderboard Sense 2 in Edge Impulse, the following software will need to be installed:

- Node.js v12 or higher. Note that you may need to check the box to install "chocolatey." This may be easy to overlook so watch out and avoid clicking through or skipping this.
- On Linux:
    - GNU Screen – install for example via `sudo apt install screen`
- The Edge Impulse CLI. Install by opening a command prompt or terminal and run the node package manager:

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

2.3 Adding Software Components

With all the software in place, we can proceed to connect the development board to Edge Impulse.

2.3.1 Connect the development board to your computer

1. Use a micro-USB cable to connect the development board to your computer.
2. The development board should mount as a USB mass-storage device (like a USB flash drive), with the name `TB004`. Make sure you can see this drive.

2.3.2 Update the firmware

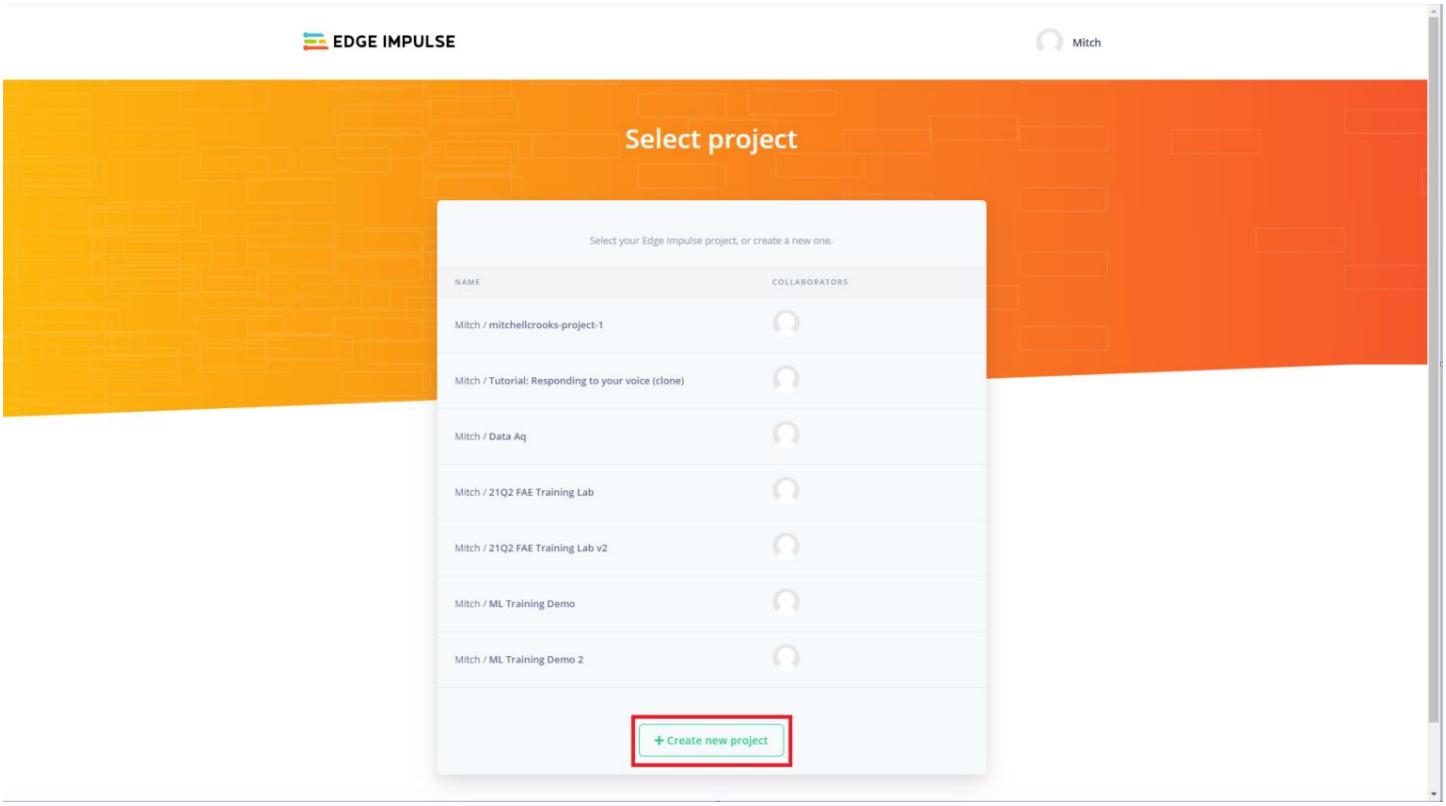
The development board must be flashed with the right firmware, so it is detected by Edge Impulse Studio. To flash the firmware:

1. Download the latest Edge Impulse firmware.
2. Drag the `silabs-thunderboard-sense2.bin` file (the file downloaded in step #1) to the `TB004` drive.
3. Wait 30 seconds.

2.3.3 Setting keys

Note: Before completing this step, please be sure that you have created a new project in Edge Impulse Studio. To do this:

1. Go to https://studio.edgeimpulse.com/ and log in
2. Click on [+ Create new project]:



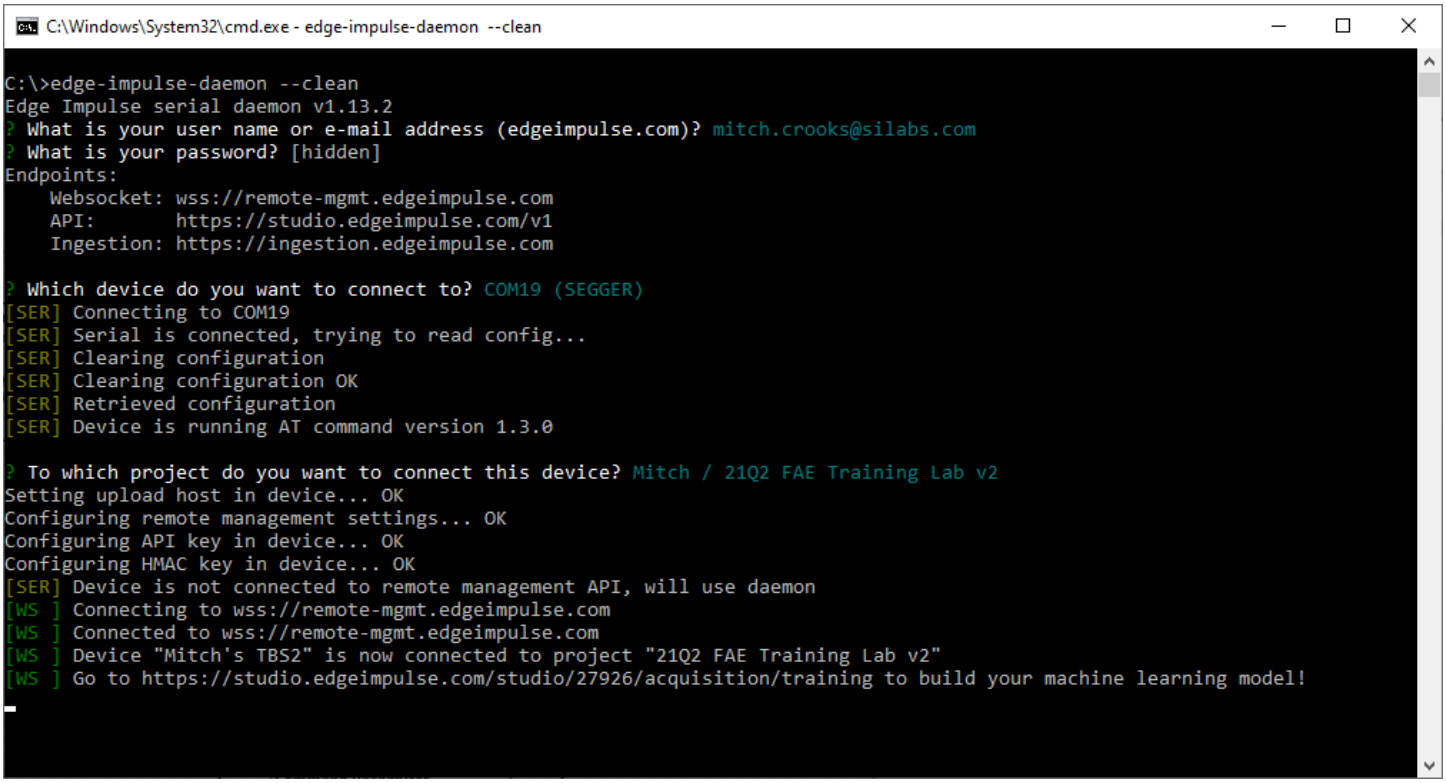
3. Name your project. This will ensure that you have a project to connect to using the Edge Impulse CLI.

The remainder of this section will simply ensure that you can connect to your device and Edge Impulse account using the CLI.

From a command prompt or terminal, run:

```
$ edge-impulse-daemon
```

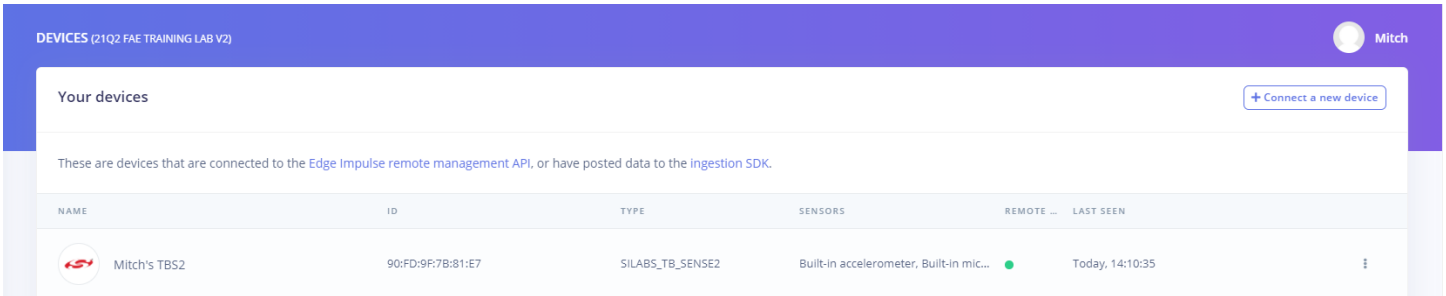
This will start a wizard which will ask you to log in and choose an Edge Impulse project. If you want to switch projects, run the command with `--clean`.



2.3.4 Verifying that the device is connected

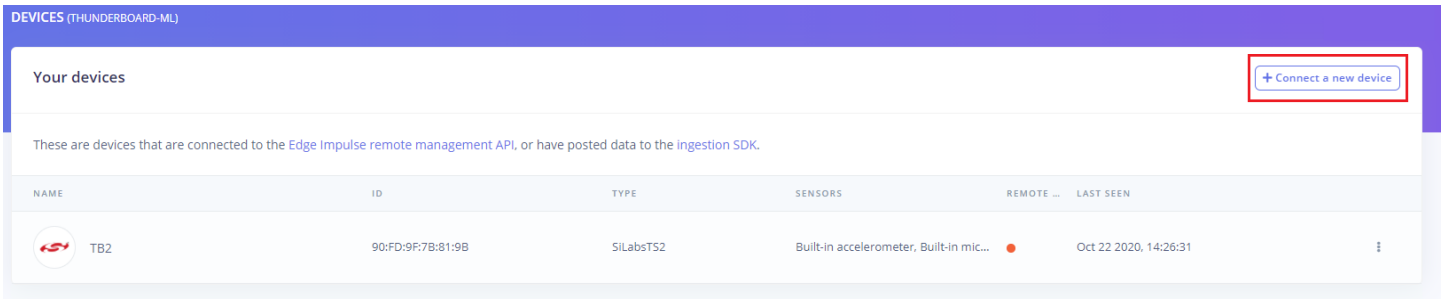
To verify that your device is connected, log on to studio.edgeimpulse.com with the credentials created in section 2.1. Click Devices.

The device should be listed here.

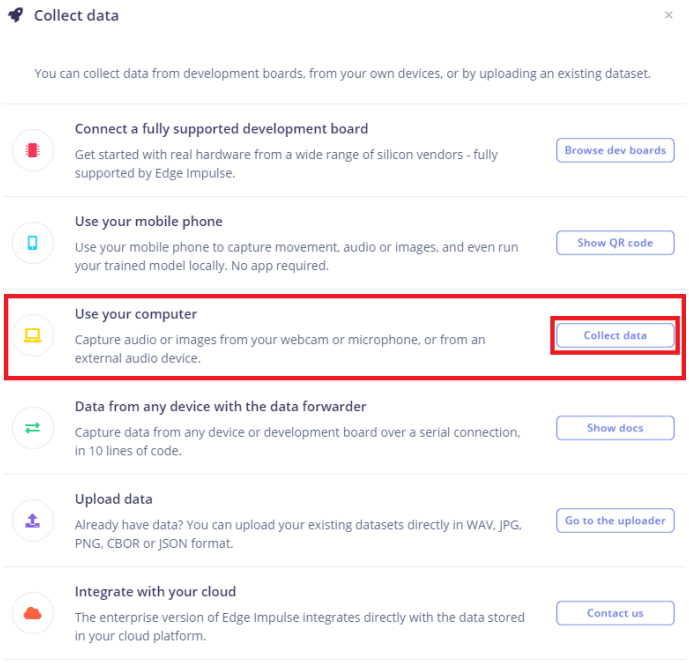


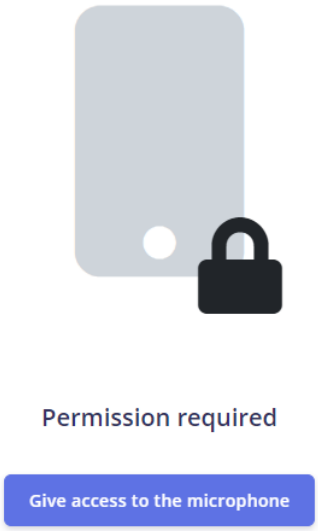
2.3.5 Use your phone or computer to collect data using Edge Impulse data acquisition

A phone or computer's microphone can be used to collect audio samples while building the dataset. To connect a mobile phone or computer to Edge Impulse, in the **Devices** page from section 2.3.4. Then click "**Connect a new device**".



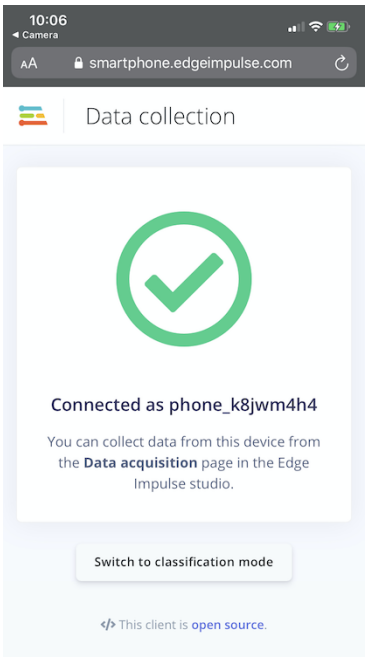
- To connect a computer, click [Collect data] next to the “Use your computer” option. Click [Give access to micrphone] to allow Edge Impulse to access the computer’s microphone.



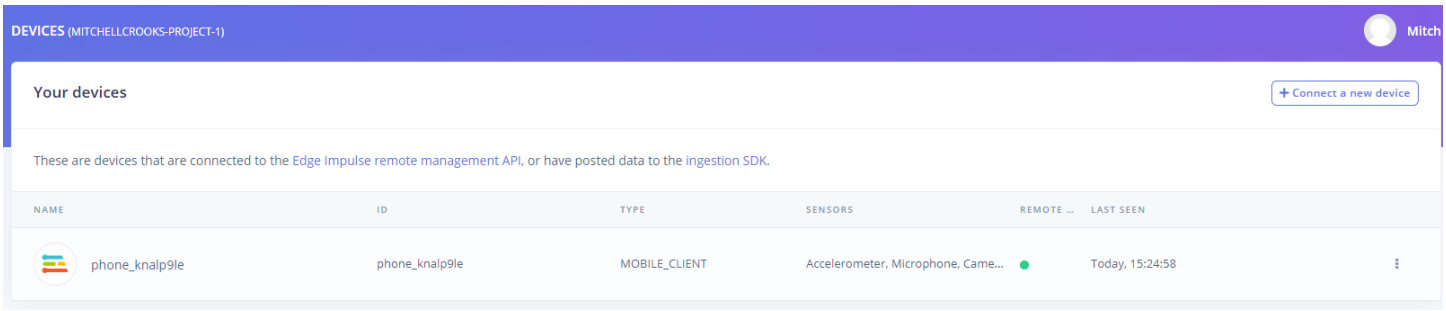


- To connect a mobile phone, click [Show QR code] next to the “Use your mobile phone” option and a QR code will appear. Either scan the QR code with the camera of the phone – many phones will automatically recognize the code and offer to open a browser window – or click on the link above the QR code to open the mobile client.

This opens the mobile client and registers the device directly. A *Connected* message should appear on the phone.



On returning to the **Devices** page in Edge Impulse Studio, the phone or computer should should show as connected.



NOTE: Why use a mobile phone at all? Data collection from a development board might be slow, so using a phone or computer speeds up the process of data collection.

3 Collect and Label a Dataset

Since we want to build a system that recognizes keywords, the first task is to pick a keyword. For this lab, we will be training our model to recognize the keyoword phrases “Silicon Labs” and “Wireless Gecko.”

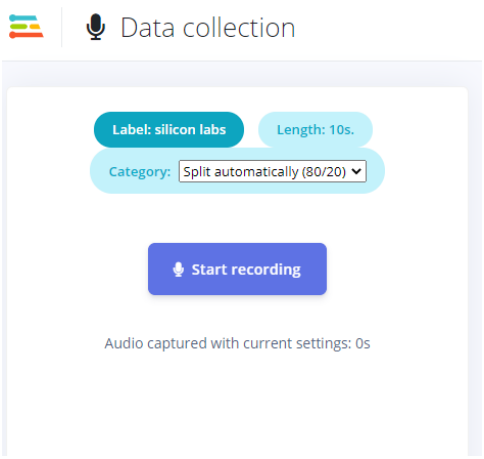
Since the data acquisition requires about 12-25 minutes of data collection (as varied as possible, for best results), we are providing a dataset of pre-recorded samples of these keywords to get you started with your sample set. In this lab step, you will augment the pre-recorded dataset with your own recorded samples. A training data set that is varied and diverse in its sources will produce a trained model that is better able to accurately classify new, unheard input sounds. For this reason, the sample data provided in this lab were collected by Silicon Labs teams using multiple individuals across our global sites.

Keep in mind that some keywords are harder to distinguish from others, and especially keywords with only one syllable (like 'One') might lead to false positives (e.g. when you say 'Gone'). This is the reason that Apple, Google, and Amazon all use at least three-syllable keywords ('Hey Siri', 'OK, Google', 'Alexa'). Our chosen key phrases (“Silicon Labs” and “Wireless Gecko”) are sufficiently complex and contain enough syllables to be recognizeable to a machine learning inference model.

3.1 Collecting your first data

Use the instructions above in section 2.3.5 to connect either your phone or computer to Edge Impulse Studio in the new project you have created. Once you have connected either your phone or computer, you can begin to collect data.

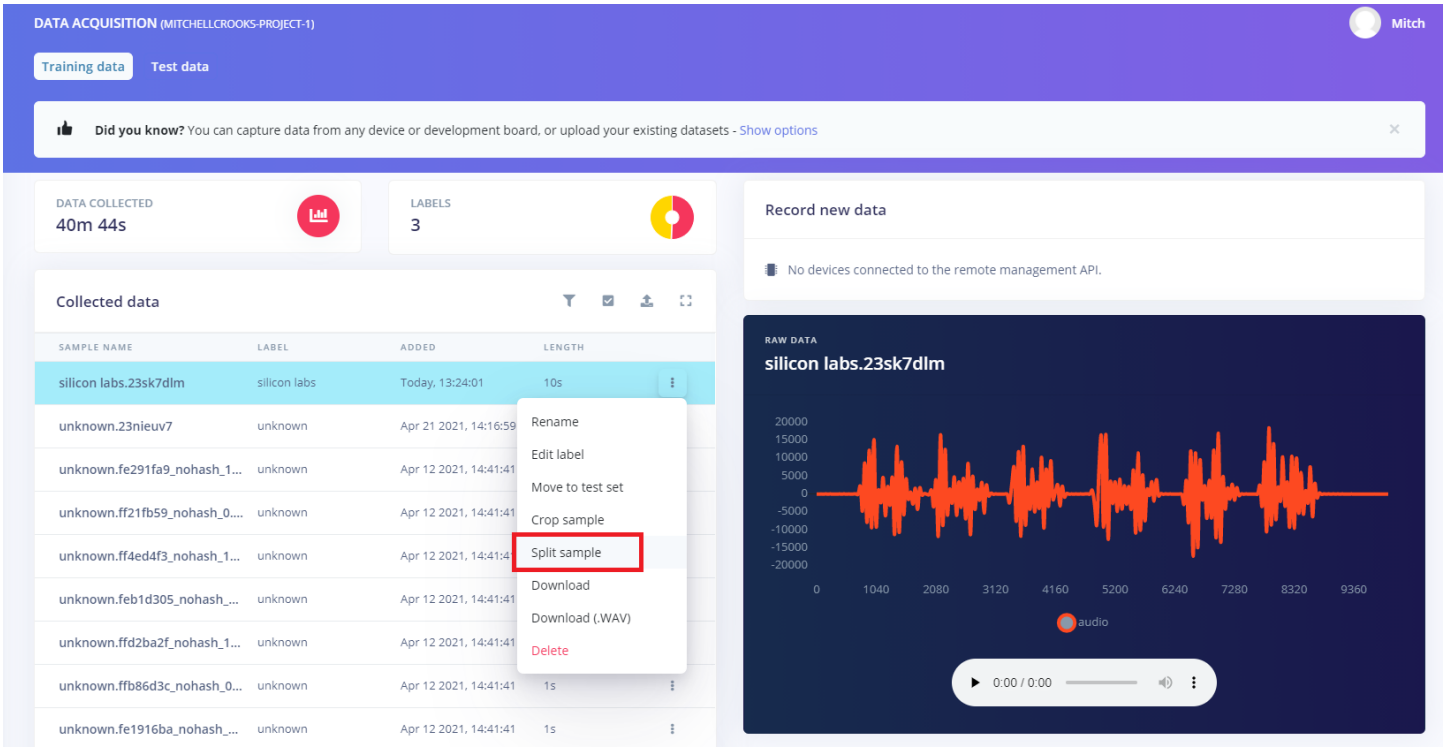
To collect your first data, set your keyword as the label, set your sample length to 10 s, and select “Category: Split automatically (80/20).” This will automatically split your collected samples between the training (80%) and validation (20%) sets. Then click Start sampling and start saying your keyword repeatedly (with some pause in between). Be sure to only speak the keyword/phrase that you used as the label for the recording.



After the first recording, a file like this will be created -

Collect and Label a Dataset

This data is not suitable for Machine Learning yet. We need to split the parts where keyword is mentioned. This is important because only the actual keyword should be labeled as such, and noise, or incomplete sentences (e.g. only "OK") should not be labeled. Click ⋮ next to the sample and select **Split sample**.



Once the samples are split, the following window will be seen –

If you have a short keyword, enable *Shift samples* to randomly shift the sample around in the window, and then click **Split**. Shifting the samples will introduce an element of randomness with regard to where the keyword waveform appears in the sample window, reducing the likelihood that the model will learn to identify keywords only if they appear in the same location in the sample window. Since our keywords are relatively long, it is likely that they will take up most of the sample window and are likely to be longer than 1 second each. Thus, you may not need to shift the samples.

The samples in the provided dataset are between 1 and 3 seconds each.

3.2 Building the dataset

In a voice recognition system, in addition to the keywords generated, a dataset needs to be built for humans saying other words ('unknown' class) and for background noise such as the TV playing ('noise' class). This is required because a machine learning model has no idea about right and wrong, but only learns from the data fed to it. The more varied the data, the better the model will work.

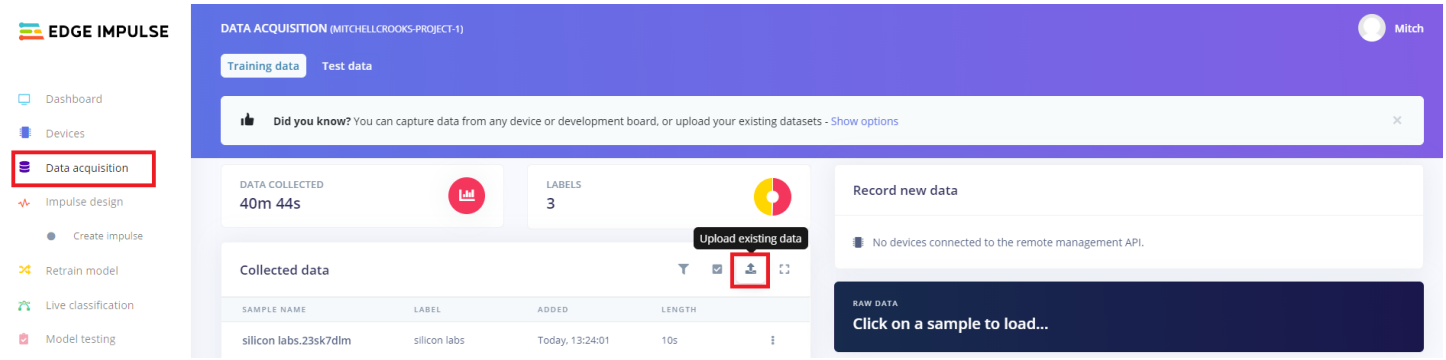
Each of these four classes ("Silicon Labs," "Wireless Gecko," noise, and unknown) requires the user to capture an even amount of data because balanced datasets work better - and for a decent keyword spotting model it is recommended to have at least 10 minutes of data for each class.

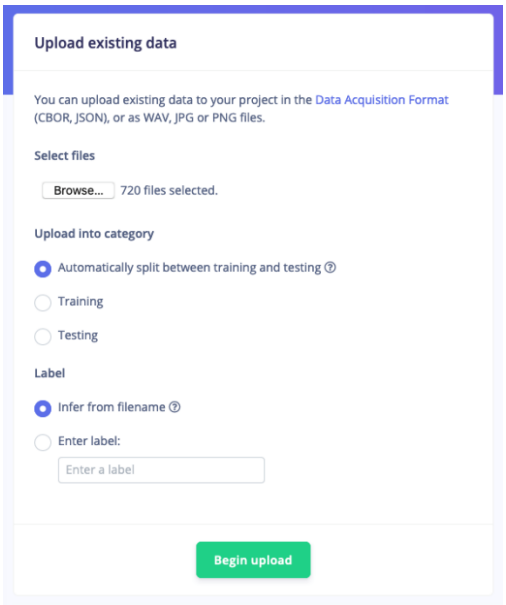
Follow the same procedure to collect at least 10 minutes of samples for the keyword - do this in the same manner as section 3.1. Use a mobile phone or computer to collect 10 second or 1 minute clips, then split this data as described above.

Note – It is important to make sure to capture wide variations of the keyword: leverage your family and your colleagues to help you collect the data, make sure you cover high and low pitches, and slow and fast speakers.

For the noise and unknown datasets as well as the prerecorded keyword dataset, use the pre-built dataset available in the lab materials download. Note that the noise and unknown samples in cluded in this dataset are obtained from the Google Speech Commands Dataset.

To import this data, go to [**Data acquisition]**, click the Upload icon, click **[Browse…]**, navigate to where you have unzipped the provided dataset, and select all sample files you wish to upload. Select the option **[Automatically split between training and testing],** and click **[Begin upload].** The data is automatically labeled and added to your project.



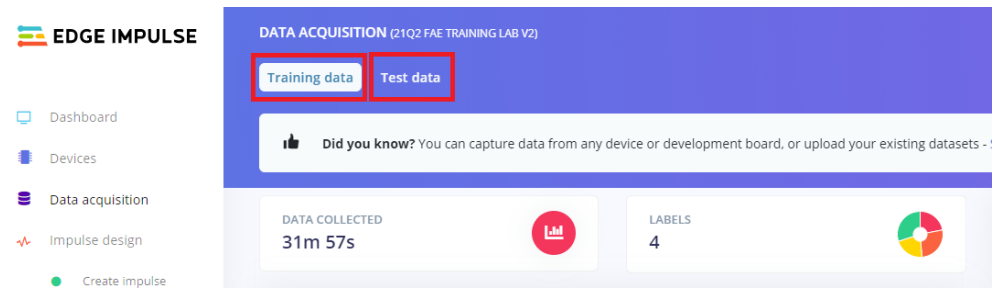


3.3 Augmenting the dataset with your collected data

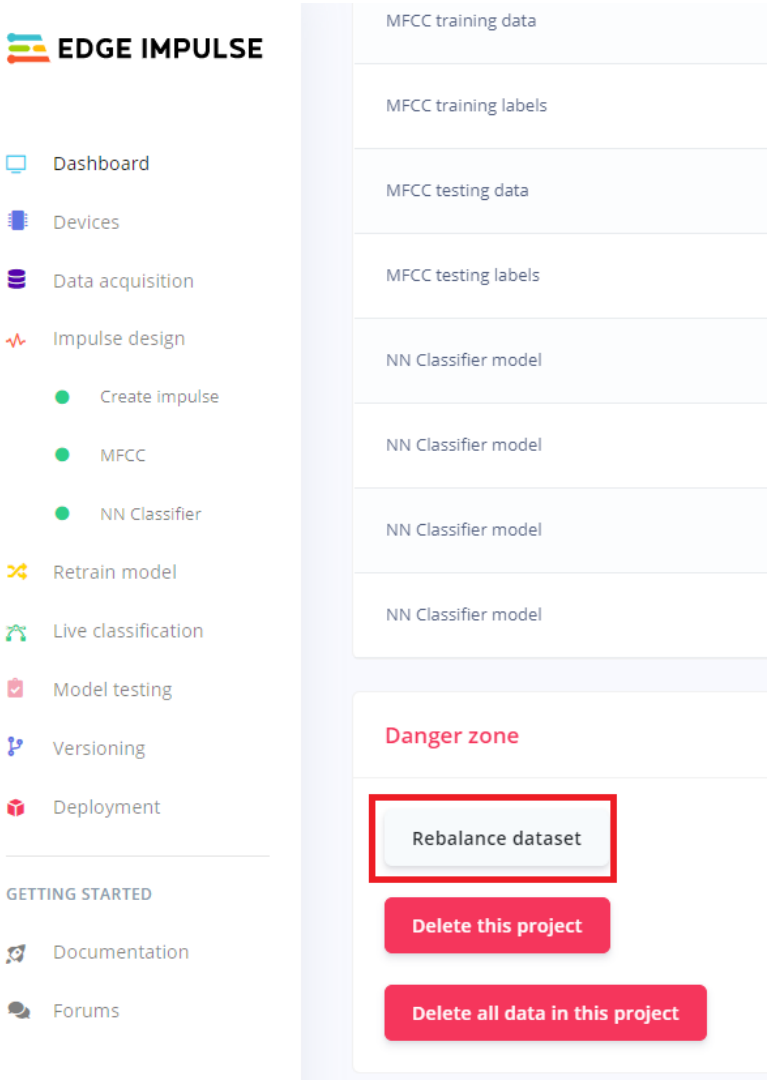
Continue with data collection for the keywords “Silicon Labs” and “Wireless Gecko” until you have at least 12 minutes of training data for each. Collect data as described above in section 3.1.

3.4 Rebalancing your dataset

If all the training data has been collected through the 'Record new data' widget, all the keywords will be present in the ‘Training’ dataset. The data needs to be split (80-20, at least) across ‘Training’ and ‘Test’ data. ‘Test’ data is needed to validate the machine learning model. Go to **[Dashboard]** and select **[Rebalance dataset]**. This will automatically split your data between a training class (80%) and a testing class (20%).



Collect and Label a Dataset



4 Design a model architecture

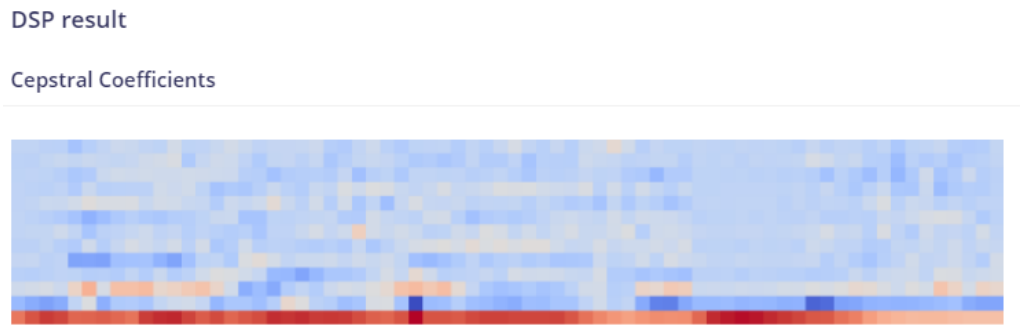
While thinking of a model architecture for the speech recognition lab, the following factors must be kept in mind:

- It takes audio data as an input. This requires heavy processing before it can be fed into a model.
- Its model is a classifier, outputting class probabilities. This needs to be parsed to make sense of the output.
- It is designed to perform inference continually, on live data. Hence the model needs to make sense of a stream of inferences.
- The model is larger and more complex.

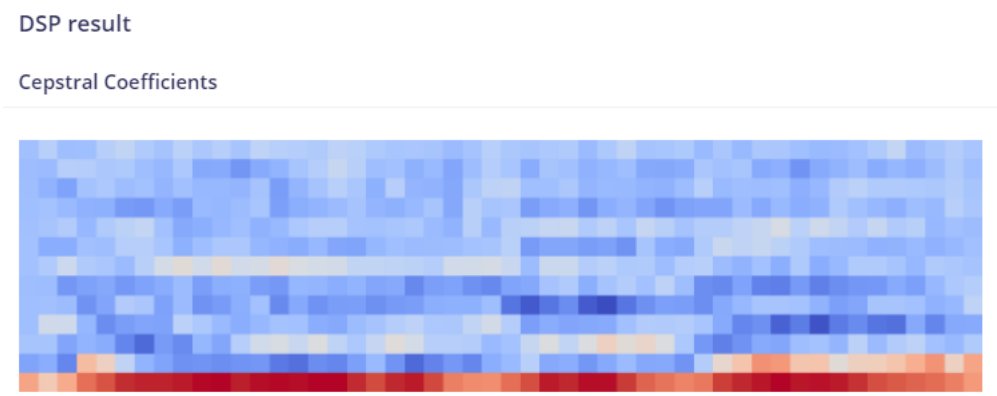
4.1 Model Architecture Details

It is important to note that the model does not take in raw audio sample data. Instead, it works with spectrograms, which are two dimensional arrays that are made up of slices of frequency information, each taken from a different time window. The figures shown below give the visual representation of spectrograms generated from ~ one-second audio clips of the phrases “Silicon Labs” and “Wireless Gecko”.

“Silicon Labs”:



“Wireless Gecko”:



By splitting the samples in section 3.1 during preprocessing, the frequency information is isolated which makes processing audio easier for the model. During training, the model doesn’t need to learn how to interpret raw audio data; instead, it gets to work with a higher-layer abstraction (i.e. the spectrogem) that distills the most useful information.

A type of neural network architecture that is specifically designed to work well with multidimensional tensors in which information is contained in the relationships between groups of adjacent values is called a convolutional neural network (CNN). CNNs are very well suited to work with spectrogram data.

4.1.1 Designing your impulse on Edge Impulse Studio

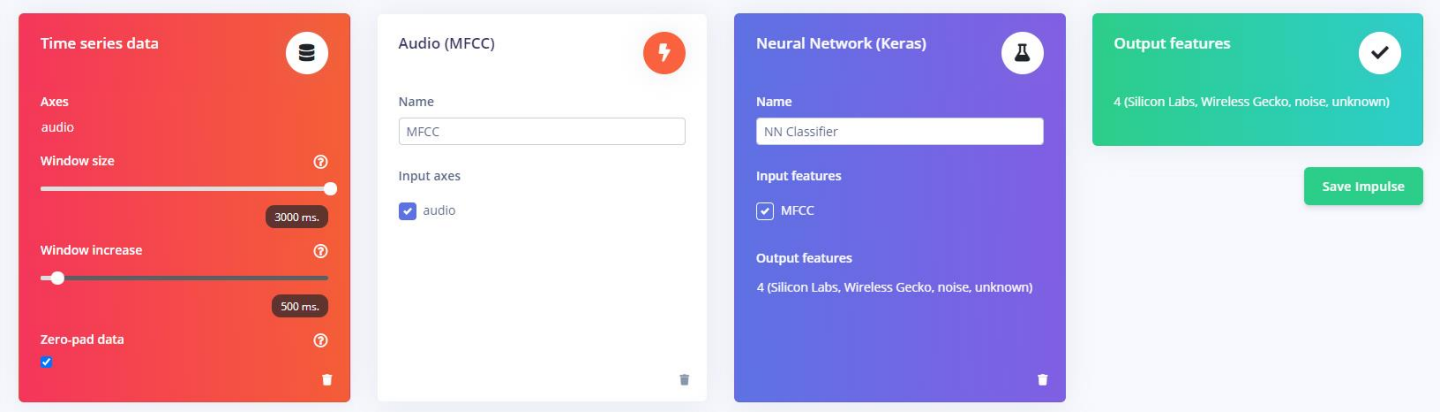
An impulse takes the raw data, slices it up in smaller windows, uses signal processing blocks to extract features, and then uses a learning block to classify new data. **Signal processing blocks** always return the same values for the same input and are used to make raw data easier to process, while **learning blocks** learn from past experiences.

This tutorial uses the “MFCC” signal processing block. MFCC stands for Mel Frequency Cepstral Coefficients. This is a way of turning raw audio—which contains a large amount of redundant information—into simplified form. Edge Impulse has many other processing blocks for audio, including "MFE" and the "Spectrogram" blocks for non-voice audio, but the "MFCC" block is ideal for dealing with human speech.

This simplified audio data is then passed into a Neural Network block which will learn to distinguish between the four classes of audio.

In the Studio,

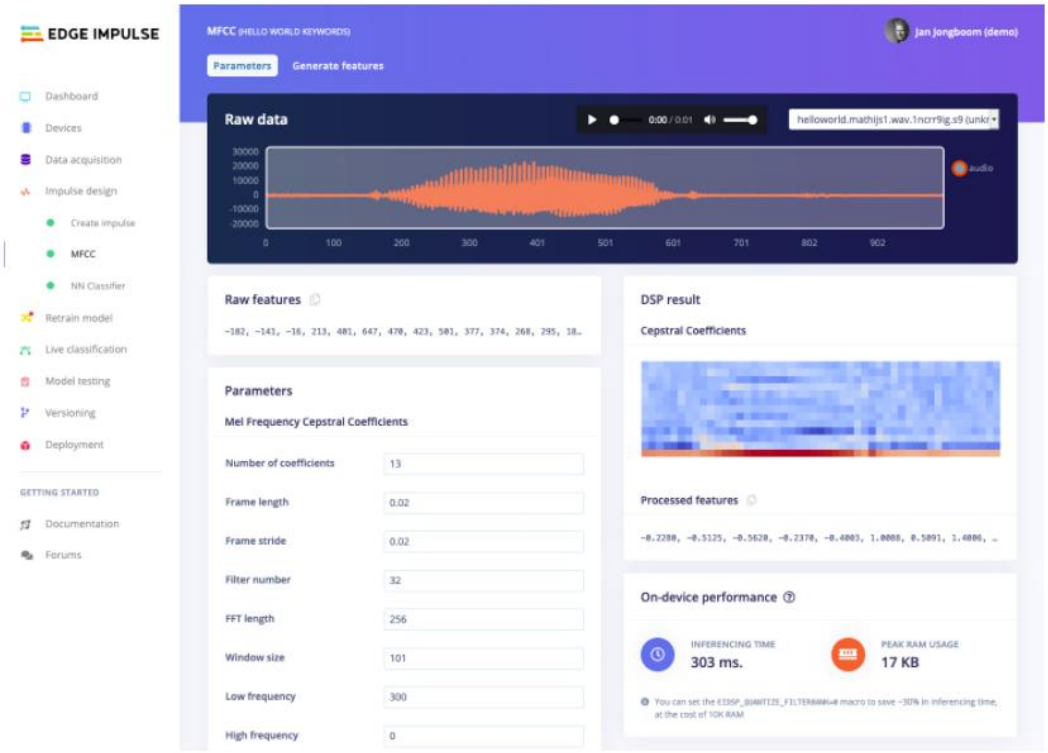
1. Go to the **[Impulse Design]/[Crete Impulse]** tab
2. Add a Time series data, an Audio (MFCC) and a Neural Network (Keras) block.
3. Set window size to 1.5 second (as that's the maximum length of our audio samples in the dataset) and click Save Impulse.
4. Click to enable the “Zero-pad data” option, which will zero-pad any data we have in our dataset that is less than the window size.



4.1.2 Configure the MFCC block on Edge Impulse Studio

1. Click on the MFCC tab in the left-hand navigation menu.

Design a model architecture

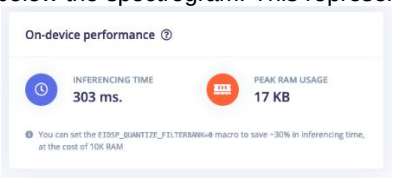


MFCC block looking at an audio file

2. The MFCC block can be configured here, if required. This page can also be used to view the visualization of the MFCC's spectrogram for a piece of audio. In the spectrogram, the vertical axis represents the frequencies (the number of frequency bands is controlled by the "Number of coefficients" parameter) and the horizontal axis represents time (controlled by 'frame stride' and 'frame length'). We will not be changing any of the default values for this example, but spend some time altering the fields to see the change in the spectrogram.
3. Explore your data and look at the types of spectrograms associated with it. Use the dropdown box near the top right of the page to choose between different audio samples to visualize or play with the parameters to see how the spectrogram changes.

Design a model architecture

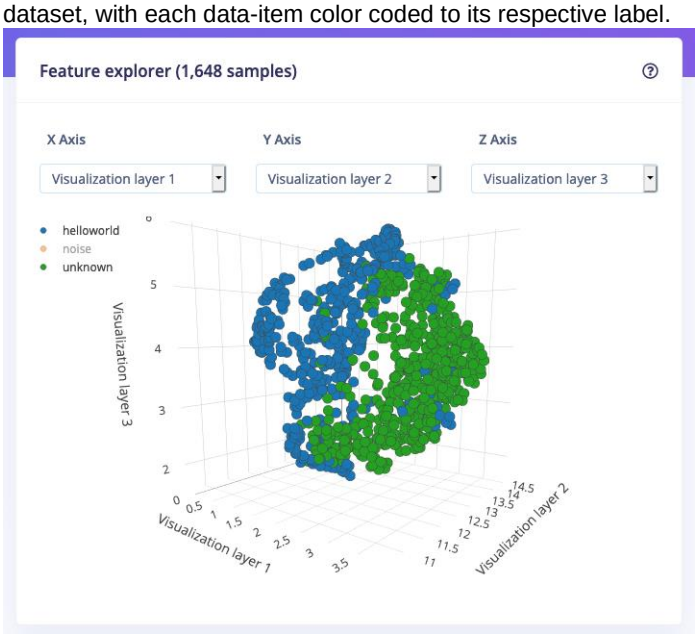
4. Observe the performance of the MFCC block on the Thunderboard Sense 2 below the spectrogram. This represents the complete time that it takes on a low-power microcontroller to analyze 1 second of data.



Note: Based on the numbers above, it looks like only 2-3 classification windows are available every second. Edge impulse continuously builds the spectrogram, which takes less time, so it is possible to continuously listen for events 5-6x a second, even on a 40 MHz processor.MFCC Feature Explorer on Edge Impulse Studio

The spectrograms generated by the MFCC block will be passed into a neural network architecture that is good at learning to recognize patterns in this type of 2D data. Before training the neural network, the MFCC blocks for all of windows of audio should be generated.

1. Click the **Generate features** button/tab at the top of the page,
2. Click the green **Generate features** button
3. This will take a minute or so to complete. The feature explorer window should then show a 3D representation with the complete dataset, with each data-item color coded to its respective label.



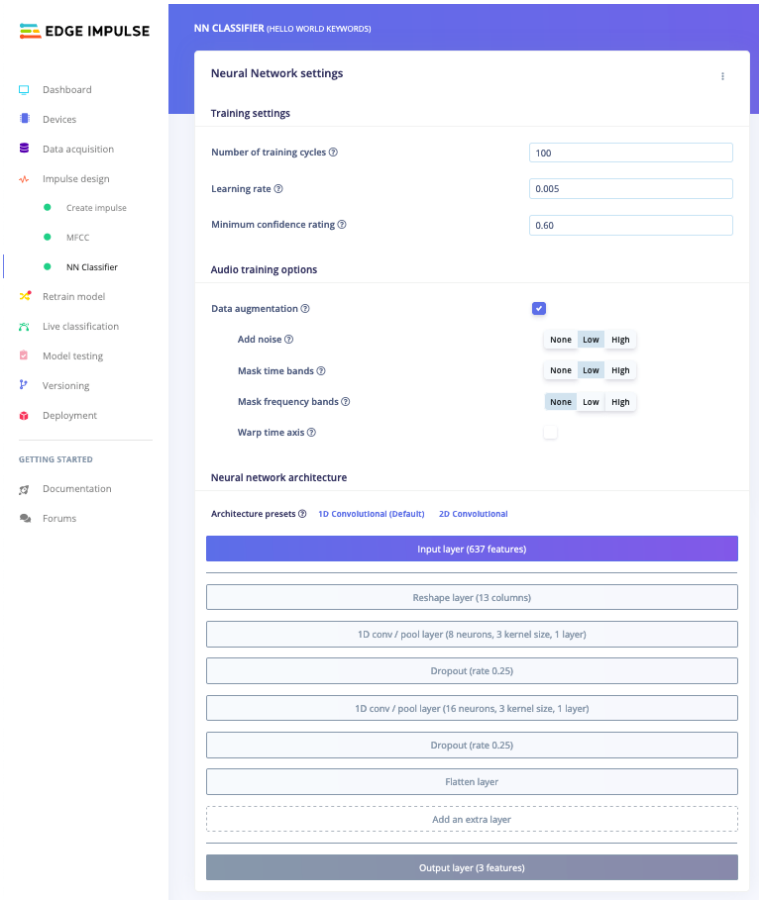
4. Zoom in to some items, find items in a wrong cluster, if any, i.e., anomalies.
5. Click on items to listen to the sample.
6. For the dataset to be suitable for ML, it should be separated well in this plot. This is a good way to check whether the dataset contains incorrect items and to validate the dataset's suitability.

5 Train the model

5.1 Configure the neural network

Neural networks are algorithms, modeled loosely after the human brain, that can learn to recognize patterns that appear in their training data. The network that is being trained now will take the MFCC as an input and try to map this to one of four classes— “Silicon Labs,” “Wireless Gecko,” noise, or unknown.

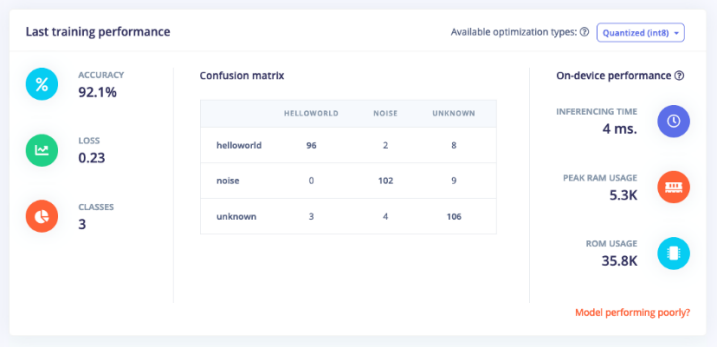
1. Click on NN classifier in the left-hand menu.



   A neural network is composed of layers of virtual "neurons", which is represented on the left-hand side of the NN Classifier page. An input—an MFCC spectrogram—is fed into the first layer of neurons, which filters and transforms it based on each neuron's unique internal state. The first layer's output is then fed into the second layer, and so on, gradually transforming the original input into something radically different. In this case, the spectrogram input is transformed over four intermediate layers into just four numbers: the probability that the input represents the keyword “Silicon Labs,” the keyword “Wireless Gecko,” noise, or unknown.
2. Change the Minimum confidence rating to 0.6. This means that when the neural network makes a prediction, Edge Impulse will disregard it unless it is above the threshold of 0.6.
3. Enable 'Data augmentation'. When enabled, data is randomly mutated during training. For example, by adding noise, masking time or frequency bands, or warping your time axis. This is a very quick way to make the dataset work better in real life and prevents the neural network from overfitting as the data samples are changed every training cycle.
4. Click “Start training”. Training will take a few minutes.

Train the model

5. When complete, the **Last training performance** will appear at the bottom of the page:



At the start of training, 20% of the training data is set aside for validation. This means that instead of being used to train the model, it is used to evaluate how the model is performing. The **Last training performance** panel displays the results of this validation, providing some vital information about the model and how well it is working. Note: these numbers may be different since they are dependent on the dataset.

On the left-hand side of the panel, **Accuracy** refers to the percentage of windows of audio that were correctly classified. The higher number the better, although an accuracy approaching 100% is unlikely, and is often a sign that your model has overfit the training data. For many applications, an accuracy above 85% can be considered very good.

The **Confusion matrix** is a table showing the balance of correctly versus incorrectly classified windows. To understand it, compare the values in each row.

The **On-device performance** region shows statistics about how the model is likely to run on-device. Inferencing time is an estimate of how long the model will take to analyze one second of data on a typical microcontroller (an Arm Cortex-M4F running at 80MHz). Peak RAM usage gives an idea of how much RAM will be required to run the model on-device.

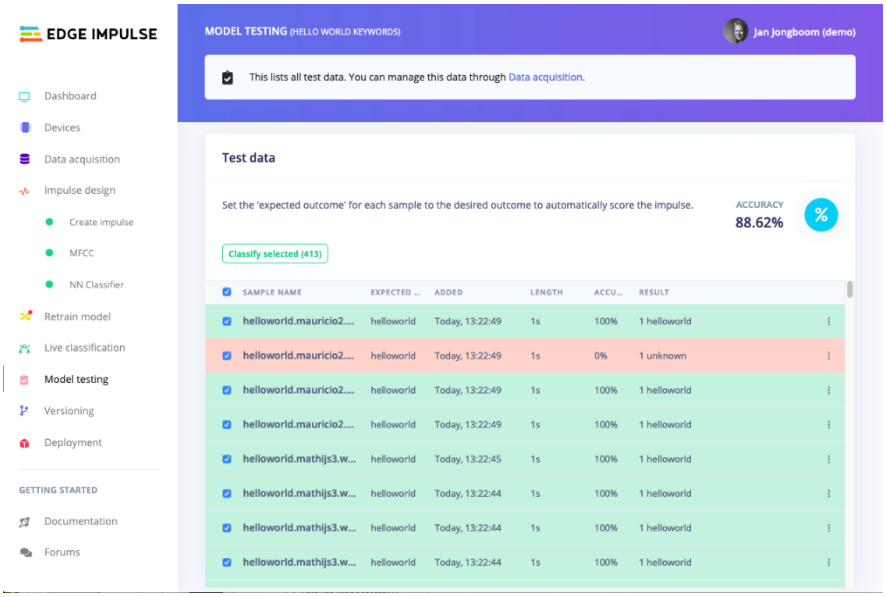
6 Test the model

6.1 Classify test data to validate the model

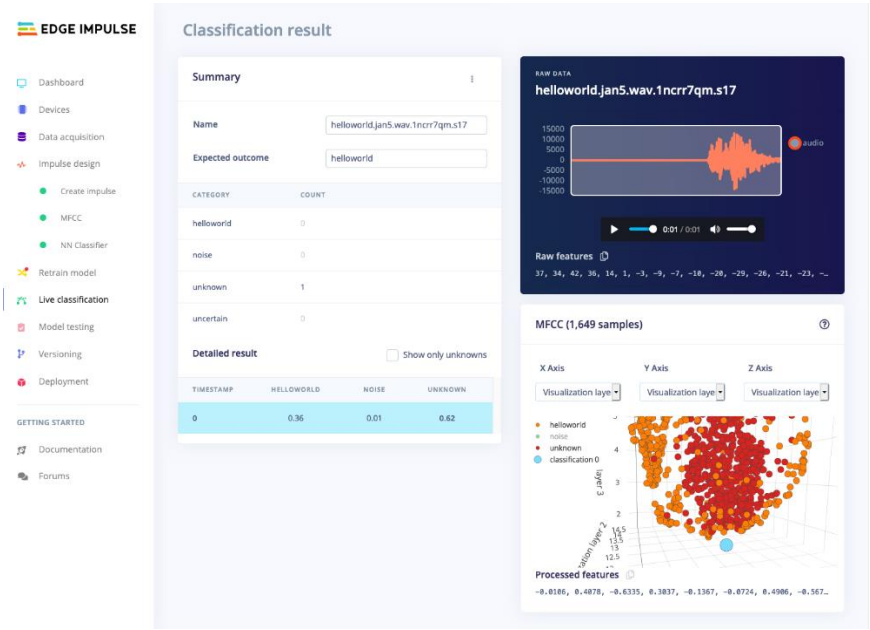
The performance numbers in section 5.1 show that the model is working well on its training data, but it is important to test the model on new, unseen data before deploying it. This will help ensure the model has not learned to overfit the training data, which is a common occurrence.

The 20% of the data that was in the 'Test set' (see section 3.4) will be used to validate if the model works on unseen data.

1. Click **Model testing** in the left navigation panel.
2. Select all items.
3. Click **Classify selected**.
4. The incorrectly classified test data will be highlighted in red



5. To drill down into a misclassified sample, click the three dots (⋮) next to a sample and select **Show classification**.
6. This should open the classification view. Inspect the sample and compare it to the training data. This way, one can inspect if this was a classification failure or whether the data was incorrectly labeled.



Test the model

7. Either update the label if the label was wrong or move the item to the training set to refine the model.
8. If the item was moved to the training set, retrain the model.

Note: It's inevitable that even a well-trained machine learning model will sometimes misclassify its inputs. This should be kept in mind while integrating a model into an application.

7 Deploying to Thunderboard Sense 2

With the model designed, trained, and verified, it can be deployed back to the Thunderboard Sense 2 device. This makes the model run without an internet connection, minimizes latency, and runs with minimum power consumption.

7.1 Convert the model

The trained model is essentially a set of instructions that tell an interpreter how to transform data to produce an output. Since we will be running our models on microcontrollers, we need an interpreter that is designed for this specific use case. Edge Impulse provides a couple of techniques of converting and exporting the model for use on microcontrollers.

7.1.1 Model Exported as a Binary

Edge Impulse also allows for the model to be exported as a binary that will run on the Thunderboard Sense 2.

1. To export the model, click on **Deployment** in the menu.
2. Under 'Build firmware', select Thunderboard Sense 2.
3. Click **Build**. This will export the model, build a binary that will run on the Thunderboard Sense 2.
4. After the build is completed, download the binary. Save this on local computer.

8 Run Inference

8.1 Model Exported as a Binary

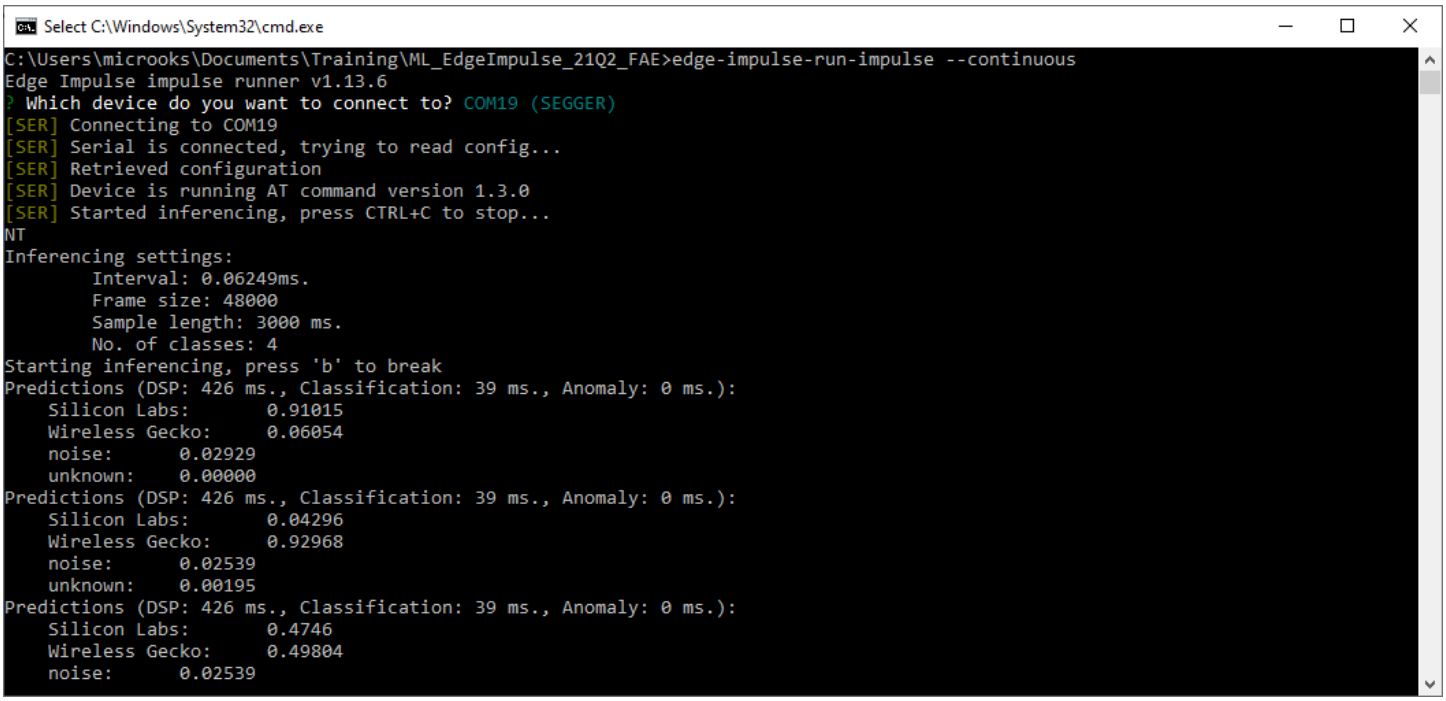
8.1.1 Flashing the device

1. Use a micro-USB cable to connect the development board to the computer where the binary was saved.
2. The development board should mount as a USB mass-storage device (like a USB flash drive), with the name `TB004`. Make sure you can see this drive.
3. Drag and drop the binary file to the `TB004` drive.
4. Wait 30 seconds.

8.1.2 Running the model on the device

The board should be connected to over a serial terminal to observe the output of the newly flashed firmware.

1. Open a command line terminal.
   ```
   $ edge-impulse-run-impulse --continuous
   ```
2. This will capture audio from the microphone, run the MFXX code, and then classify the spectrogram.



8.2 Evaluate and Troubleshoot

If the model is working properly on Studio, but does not recognize the keyword when running in continuous mode on the device, then this could be a symptom of a dataset imbalance, for e.g. – there might be a lot more unknown/noise data compared to the keyword) in combination with the moving average code to reduce false positives.

When running in continuous mode, a moving average is run over the predictions to prevent false positives. E.g. if 3 classifications are done per second, the keyword can be potentially classified three times (once at the start of the audio file, once in the middle, once at the end). However, if the dataset is unbalanced (there's a lot more noise / unknown than in the dataset) the ML model typically manages to only find the keyword in the 'center' window, and thus it is filtered out as a false positive.

Some things you can try to improve your model performance:

1. Adding more data

2. Revisit steps 2-5 (Collect a dataset, design a model architecture, and train the model) as well as the subsequent steps to test, deploy, etc. the model
3. Try adjusting the many parameters exposed to you in Edge Impulse Studio for data collection, data augmentation, neural network architecture, training, optimization, etc.

Alternately, if it is not the above, it is possible that the model has overfit both the training and the test data. This can be fixed by adding more data – the more diverse the data, the better.

8.3 Tips and Tricks

This section presents some tips and tricks to try for achieving the best possible inference results:

1) Collect a training and validation dataset using the hardware used during inference (Thunderboard Sense 2 microphone) and in the environment where inference is expected to be run. Regardless of the collection device, audio samples must be collected at a 16 kHz sample rate. It is possible to use the Edge Impulse Studio interface to record samples directly from the Thunderboard Sense 2.
2) Noise can be added to the samples by clicking on the data augmentation checkbox under the NN classifier. This will make it harder for the model to learn during training (you might have lower results in the training / testing accuracy) but may help improve inference accuracy.
3) When you upload data using the uploader, it may be necessary to NOT select "infer label from name of file". If you do this and select all the files in the training folder you may actually get 1589 labels. You will need to therefore specify the label name and then select only the appropriate files that correspond to that label. Additionally, when you create the impulse, you need to check the "zero pad data" checkbox because some of the samples are 1s and some are 2s long. Otherwise, the "generate features" step won't work.
4) The model has a difficult time trying to differentiate the two key phrases and also differentiate between "noise" and "unknown". It is recommended to merge “noise” and “unknown” into a catch-all "openset" category. You can think of the keywords as "positive" classes and openset as a "negative" class. You can kind of see this from the confusion matrix where there was some uncertainty in whether the utterance belongs in noise or unknown. The confusion matrix becomes a lot more definitive when we use just one negative class. An easy way to move everything over to "openset" is to to use the "Filter your data" and "select multiple item" shortcut icons (on the same line as "Collected Data") on the data acquisition tab. You can then edit the labels for noise and unknown and rename the labels to openset.
5) It may help to change the window size to more accurately trap the length of the samples because the utterances for "Silicon Labs" and "Wireless Gecko" present in the training samples range from **1 to 1.5 seconds**. Without this change, you'll have some empty dead space that will enter the model training, which is not desired.
6) This is more of an observation, but you can see that from the feature map that there seems to be a split between higher pitched speakers (possibly children) and the adult speakers, which may explain why the features are not as clumped as one would expect them to be. This is ok, but just informs that more data would help get a better feature map to cover more of the spectrum.
7) It is recommended to add your own voice samples for each keyword. You can recorded about 30 seconds using the Thunderboard Sense 2, and then use the split sample feature to automatically split it into individual samples. This option is available if you click on the icon that looks like three little dots on each row represented by the file name.
8) When performing inferencing on the board, please use edge-impulse-run-impulse without the --continuous flag, as this will provide a more definitive result as it takes a snapshot recording of like 2 seconds for the voice sample, windows on it, then provides the result.
9) The following are a couple of example impulse projects available for you to clone and experiment with:
    - A high-accuracy version of this lab: https://studio.edgeimpulse.com/public/37838/latest
    - Edge Impulse’s standard “Hello World” example: https://studio.edgeimpulse.com/public/14225/latest