# Project: Sound Classification

**Introduction**

In this module, we went through the steps of creating a speech recognition (or keyword spotting) system. The same basic steps can be used to create a device that recognizes and classifies other sounds, too!

In this project, we will start with a simple, pre-made dataset, add our own target sound, and build a classifier for those sounds. I encourage you to find sounds made with something other than your voice!

Please note that at this time, we do not have a way to grade your project. As such, this project is considered optional to complete the course. However, we strongly encourage you to go through these steps to get hands-on experience with embedded machine learning as well as using the Edge Impulse tool.

**Required Hardware**

For collecting sound data, you should have access to a recording device. This can be a smartphone, webcam, laptop, etc.

For deploying, you can use either your smartphone or the [Arduino Nano 33 BLE Sense](https://store.arduino.cc/usa/nano-33-ble-sense).

The smartphone will provide a simple demo only whereas the Arduino board will allow you to change the code to respond to audio events. As a result, we recommend using the Arduino, if you have access to it.

**Collect Data**

We will start with a pre-made dataset from Edge Impulse that includes a generic noise category and the sound of a faucet running. Initially, this dataset was intended to be used as a standalone demo that could detect if someone left a faucet running. You are welcome to use just that data, augment it with your own sounds, or collect your own, entirely new dataset.

To start, download the faucet dataset from [this link](https://github.com/ShawnHymel/ei-faucet-dataset/raw/master/faucet_dataset_v01.zip). Unzip it somewhere on your computer.

If you are going to use your own sounds, I recommend collecting at least 50 1-second audio samples of that sound class (or at least one 50 second recording). Make sure the sound source is in different environments and different distances away from the recording device. This will help create a more robust model that can differentiate that sound from other noises.

For example, I recorded a fan running at different speeds. I moved the microphone to several different angles to collect the sound in front of, behind, and above the fan, as the sound changes as the angles and distance varied.

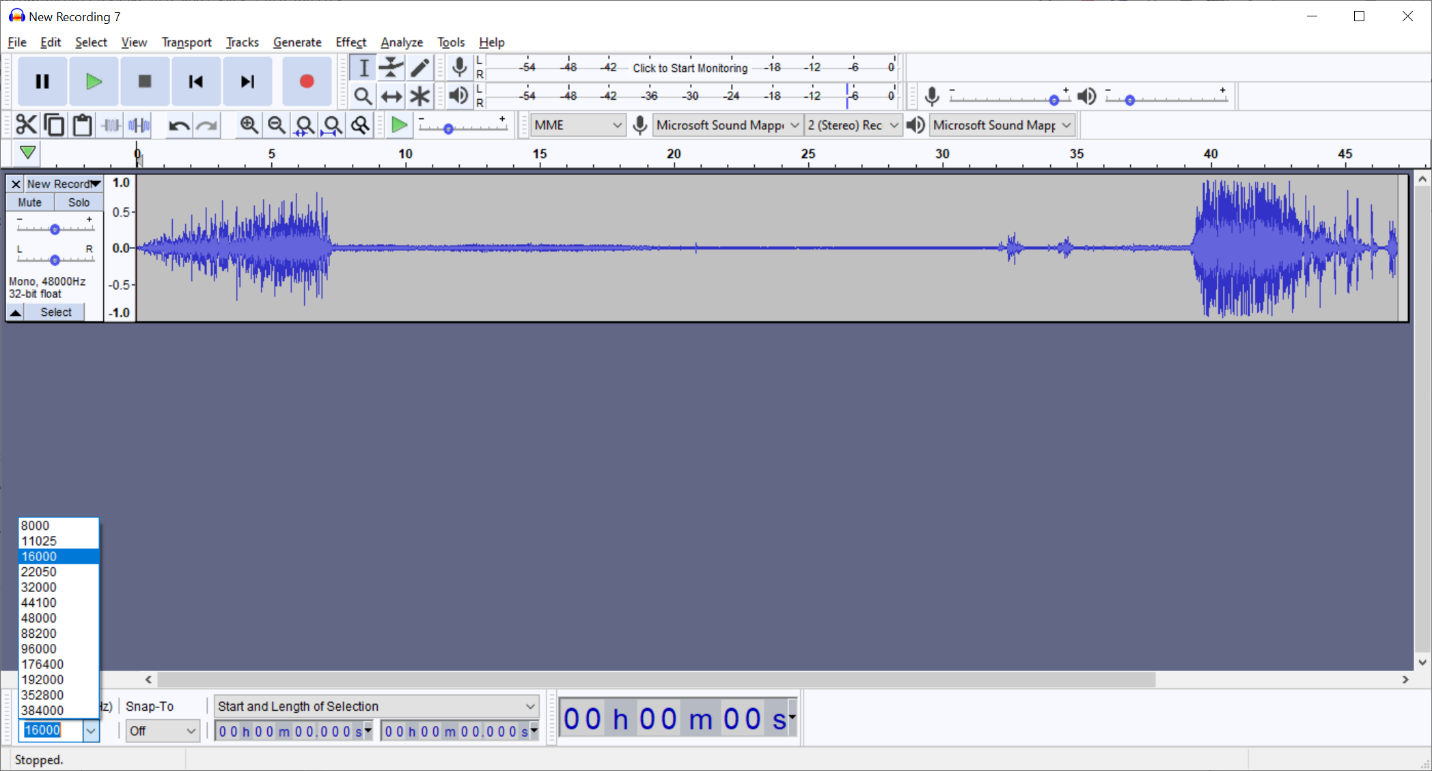


The idea is to have my embedded system identify among 3 different types of sounds: background noise, faucet running, or fan.

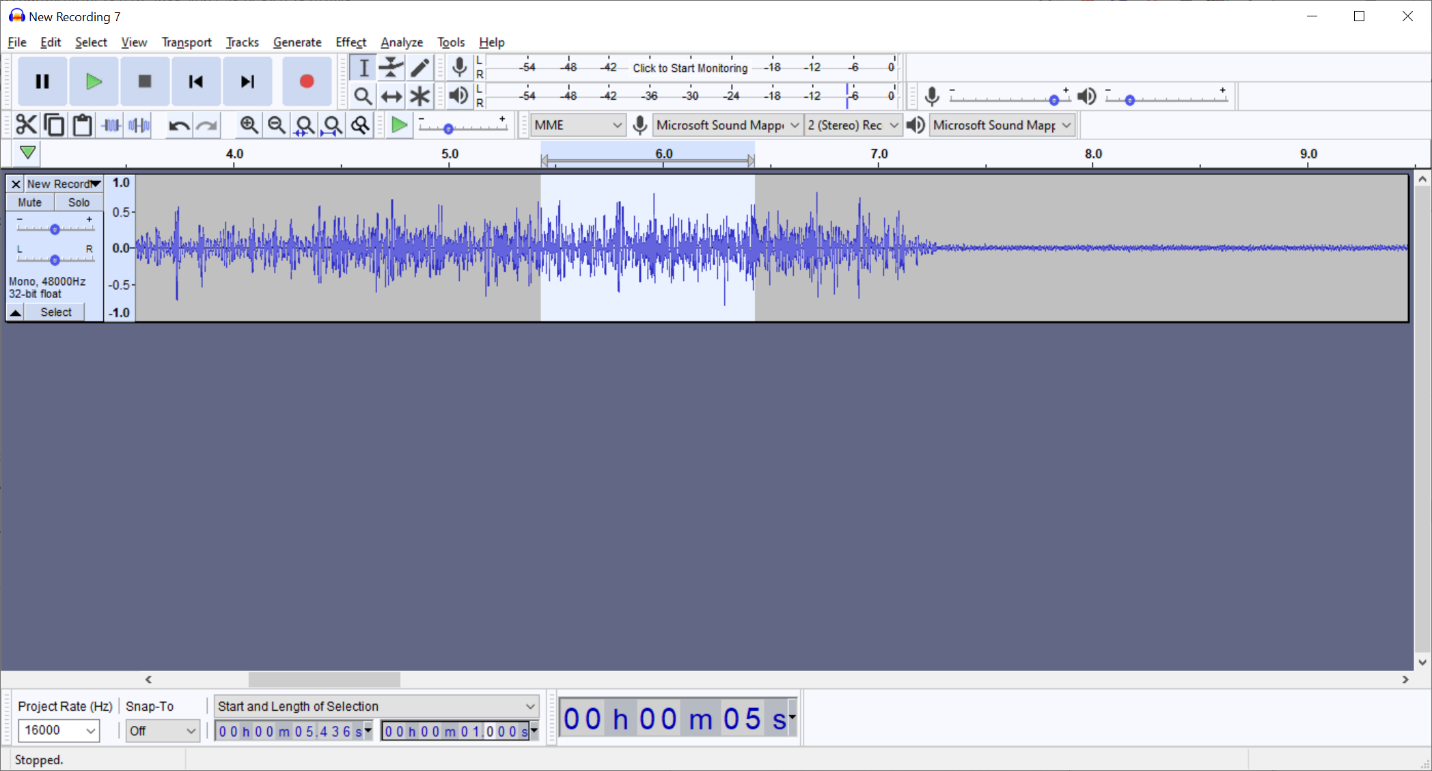
Feel free to collect your own sounds! These can be constant noises, such as an air conditioner running, a car engine, music, etc. or single events, like a clap or a dog barking.

**Curate Data**

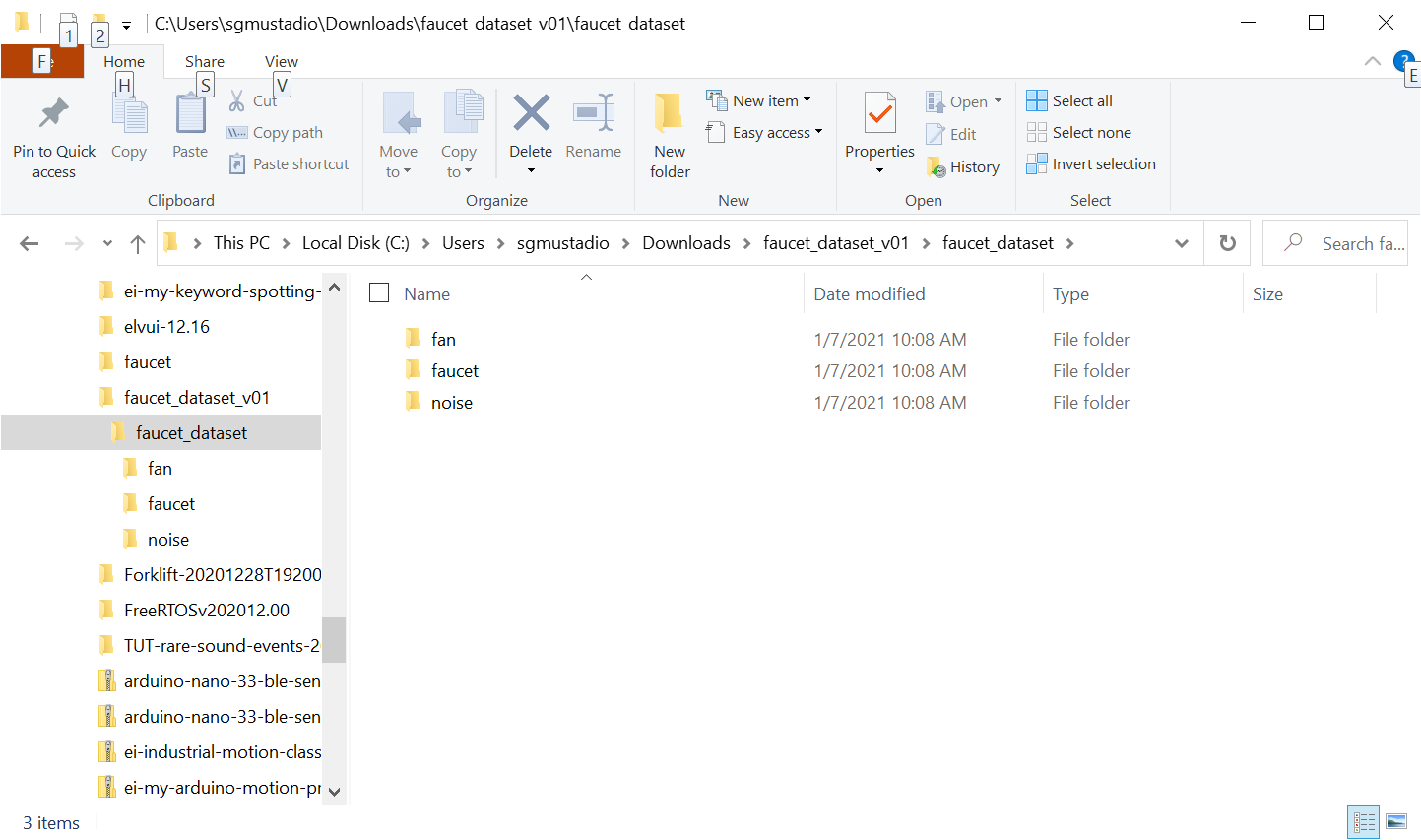
When you’re done, transfer the sounds over to your computer. If you have not done so already, install [Audacity](https://www.audacityteam.org/). Open your audio file in Audacity. Select the drop-down menu, *Project Rate (Hz)*  at the bottom of the screen. Change the sampling rate to **16000 Hz**.



Highlight 1-second sections of audio from your recording. I recommend selecting exactly 1 second. You can enter a “1.000” for the seconds in the *Start and Length of Selection* box to select exactly 1 second.

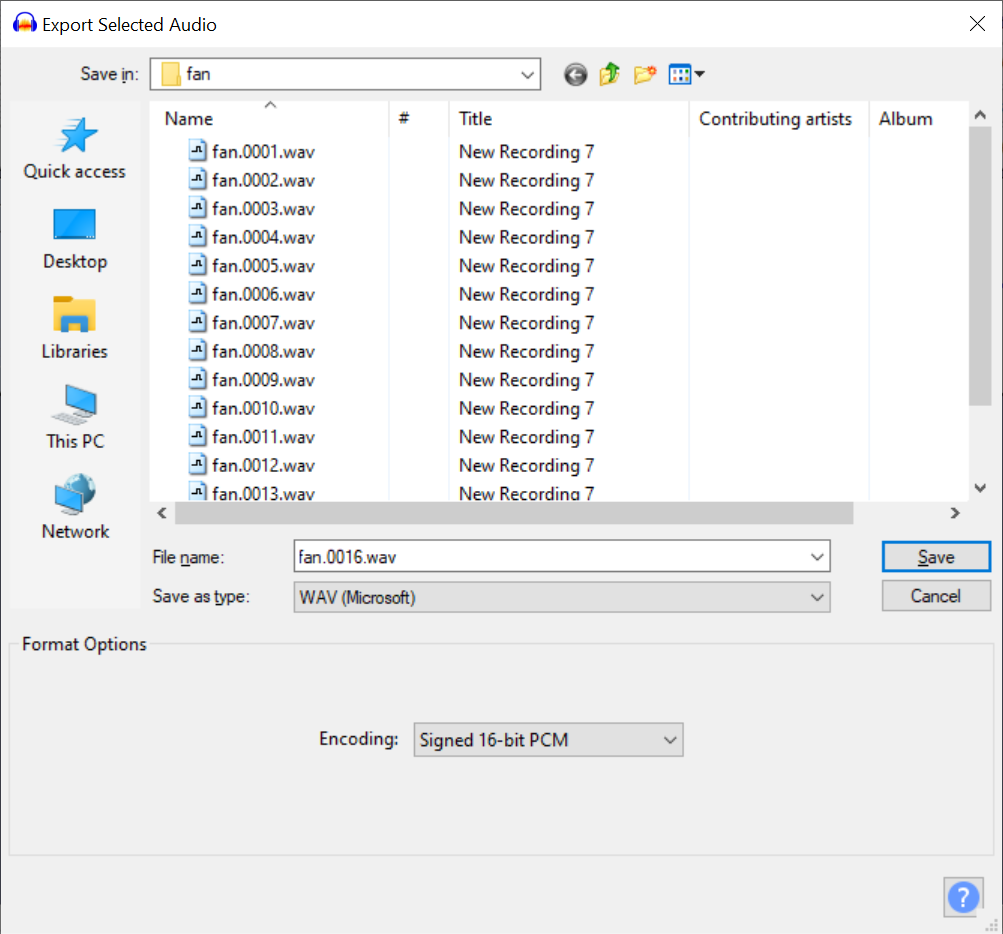


In the faucet dataset, create a new folder with the name of your class (e.g. “fan”).



Click **File > Export > Export Selected Audio**. Set the bit depth to **Signed 16 bit PCB**. Save the audio selection as a WAV file.

I recommend using the naming convention of *<label>.xxxx.wav*. The <label> before the first period (.) can be used by Edge Impulse to automatically determine the class. A number (xxxx) is used to uniquely identify the files. The ordering and exact number is not important (i.e. you could also use a hash).



As mentioned, try to get at least 50 1-second clips from your recording(s). More is better (for reference, I got over 100 1-second clips for the fan sound).

**Important!** Sound clips must be *at least* 1 second long. If you make them shorter than the window size set by Edge Impulse (default of 1 second), the entire sample will be discarded. Because of this, I recommend selecting audio samples of *exactly* 1 second. The sampling rate among all the samples must be the same (16 kHz), and the bit depth should be the same (16-bit PCM).

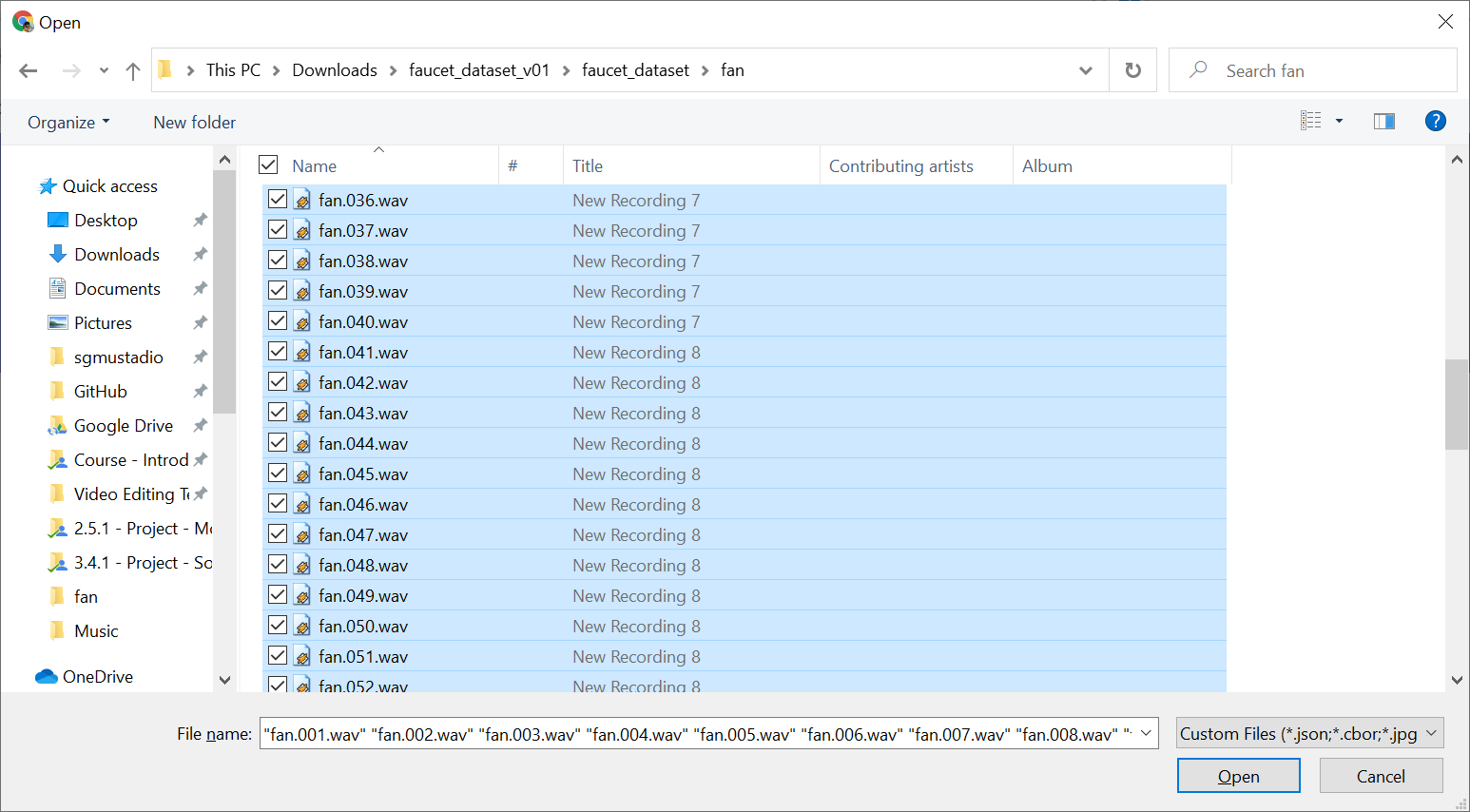
Repeat this process for any other recordings you would like to be included in your dataset. Don’t forget to set the sample rate to 16000 Hz, set the clip length to 1 second, and choose 16-bit PCM as the bit depth!

Alternate method: you can also collect audio data straight from your Edge Impulse project! Go to *Data acquisition* in a new project and connect your smartphone or Arduino board. Just like we did in the first project, you can collect audio data from these connected devices. However, I outlined the steps above so you could see how to create your own dataset manually and how to adjust sound files to have matching sample rates, bit depth, etc.

**Upload Data**

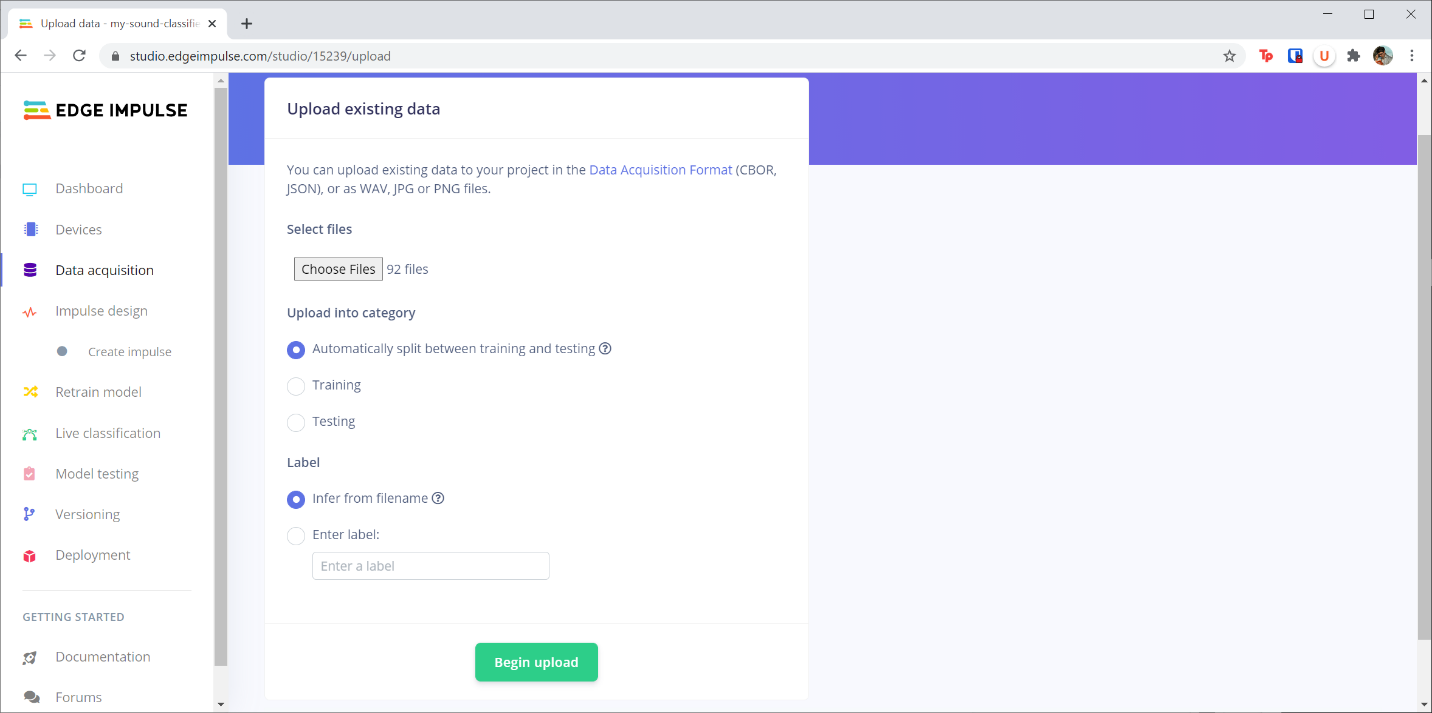
Create a new project in Edge Impulse. Head to the **Data Acquisition** page. Click **Let’s collect some data**. Select the **Go to the uploader** option.

On this new page, click **Choose files**. Select all of the files from your curated sample set.



Click **Open**.

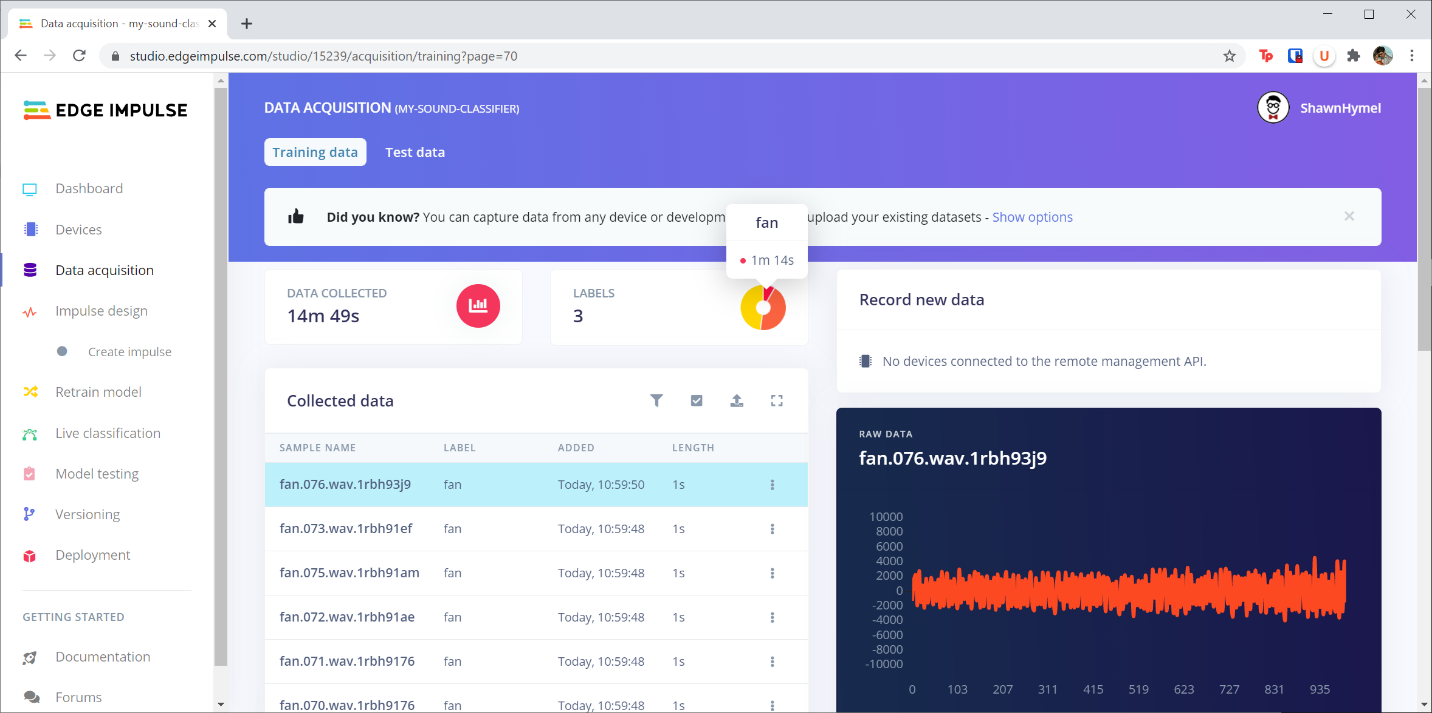
Leave *Automatically split between training and testing* selected. If you used the file naming scheme I outlined above, leave *Infer from filename* selected. If not, select *Enter label* and give your samples a label (e.g. “fan”).



Click **Begin upload**.

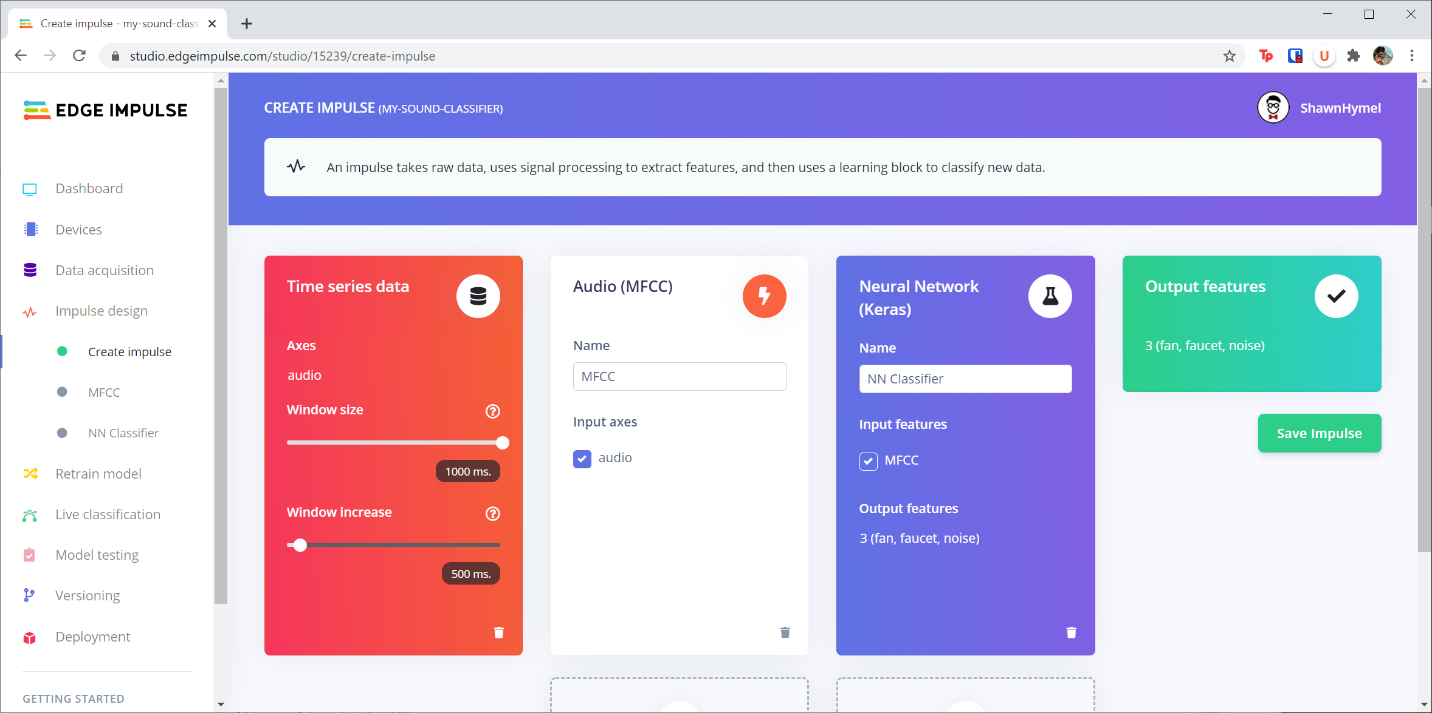
Repeat this process for the *faucet* and *noise* sets that you downloaded in the *faucet\_dataset* ZIP file you downloaded at the beginning. The WAV files in that set should follow the naming scheme I outlined, letting you leave *Infer from filename* selected in order to add labels to the samples.

Click on the **Data acquisition** link to go back to the Data Acquisition page. Here, make sure that all of your samples are present and that they are divided between the training and test sets (there should be about 20% of the samples in the test set).



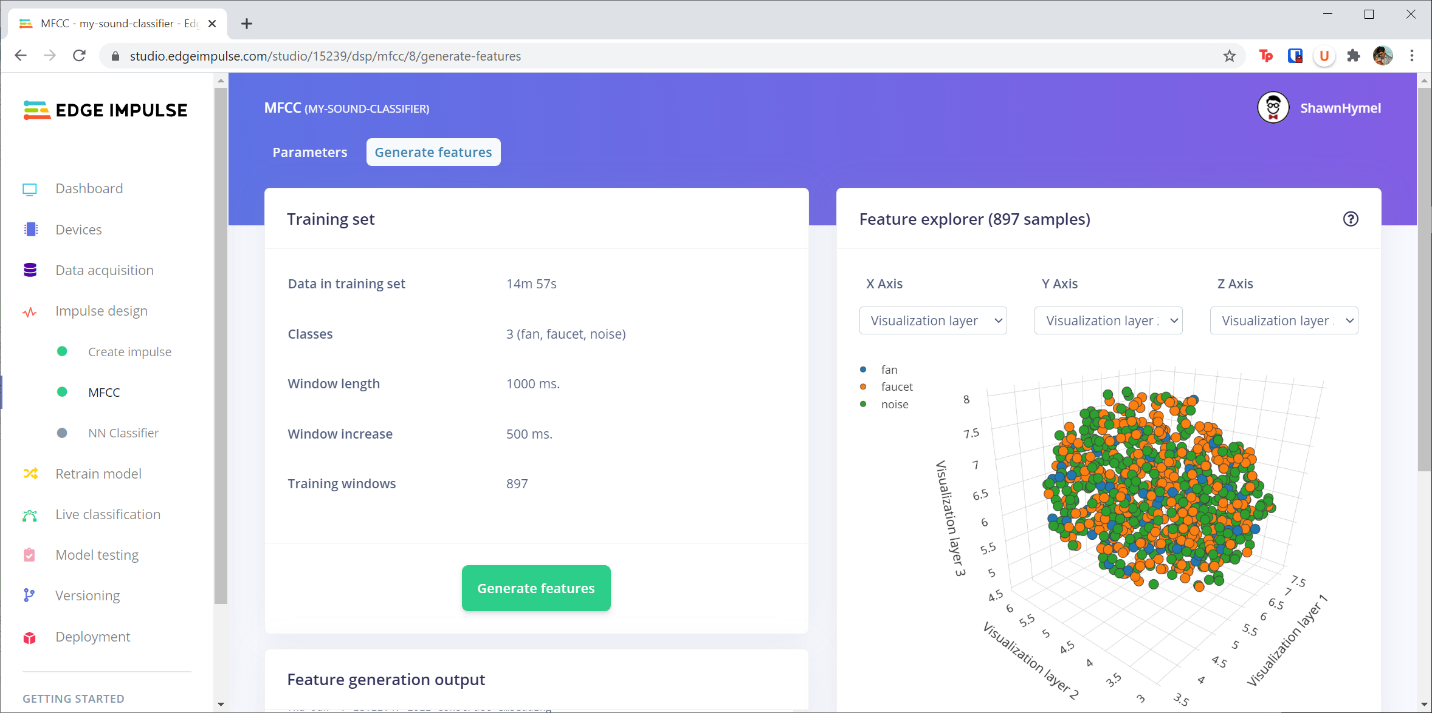
**Feature Extraction**

Navigate to the **Impulse design** page of your project. Add an **Audio (MFCC)** processing block and a **Neural Network (Keras)** learning block.



Click **Save Impulse.**

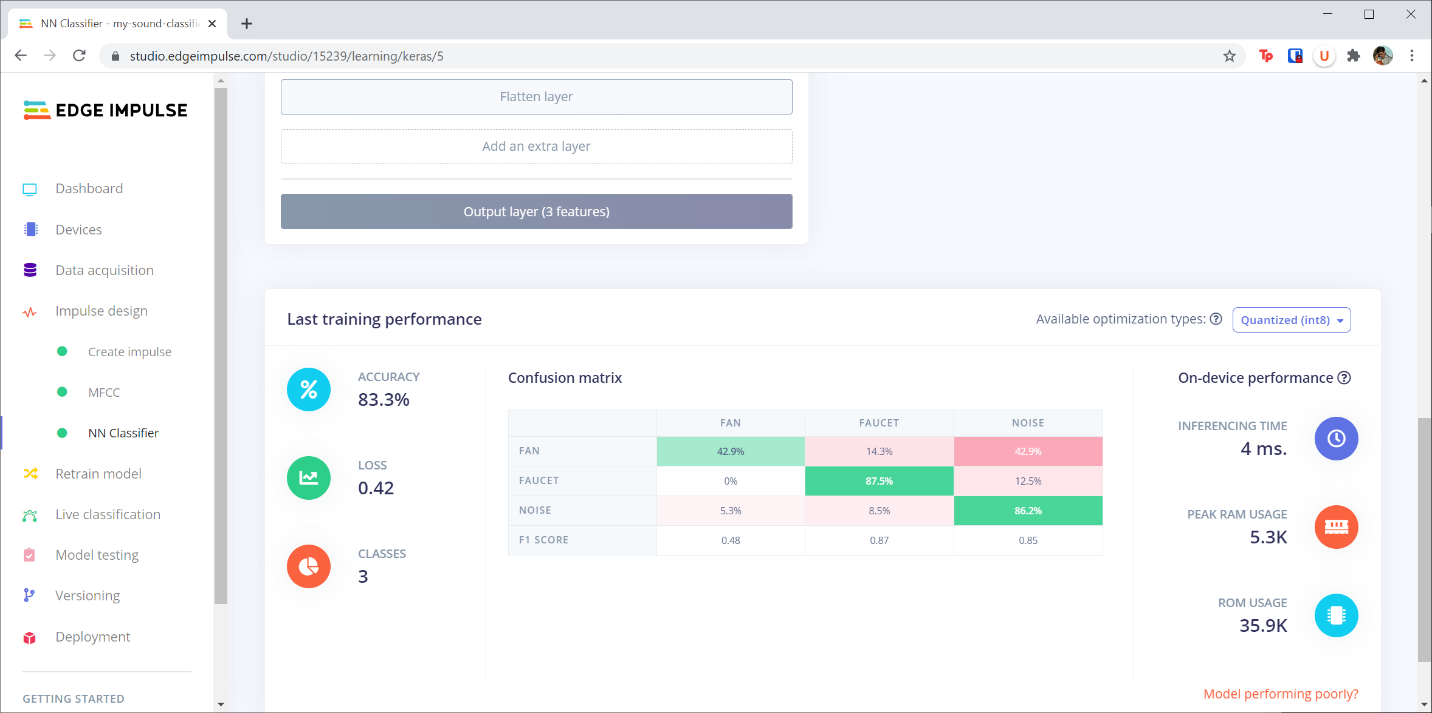
Go to the **MFCC** page and click on the **Generate Features** tab. Click the **Generate Features** button, and wait a moment while your audio samples are converted into spectrograms. When it’s done, take a look at the *Feature explorer* to see if you can identify separation among your classes.



My features do not look like they are separated very well. As a result, I will likely have a difficult time creating an accurate model. However, it is enough to get started for a demo project like this.

**Model Training**

Navigate to the **NN Classifier** page. Leave all of the hyperparameters at their defaults and click **Start training**. When it’s done, scroll down to view the *Confusion matrix* of the validation data.

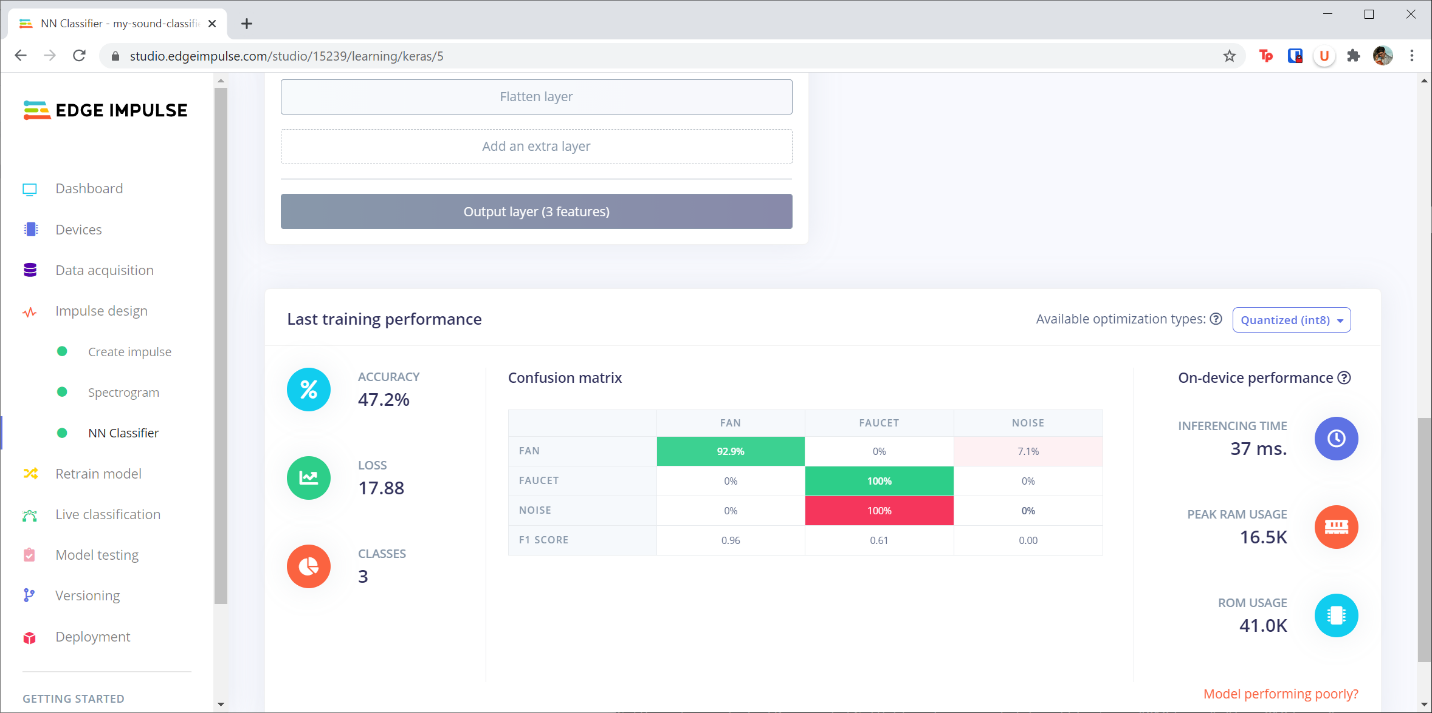


Write down the *F1 scores* of each class and the *total accuracy*.

Go back to the **Create Impulse** page. Remove the **Audio (MFCC)** processing block and replace it with a **Spectrogram** processing block. Click **Save Impulse**.

Go to the **Spectrogram** page and generate features again. When that’s done, head to the **NN Classifier** page and re-train your model.

Compare your *F1 scores* and *total accuracy* to those from the previous model (where you used MFCCs as your features). How do they compare? Which feature extraction method provided better results?



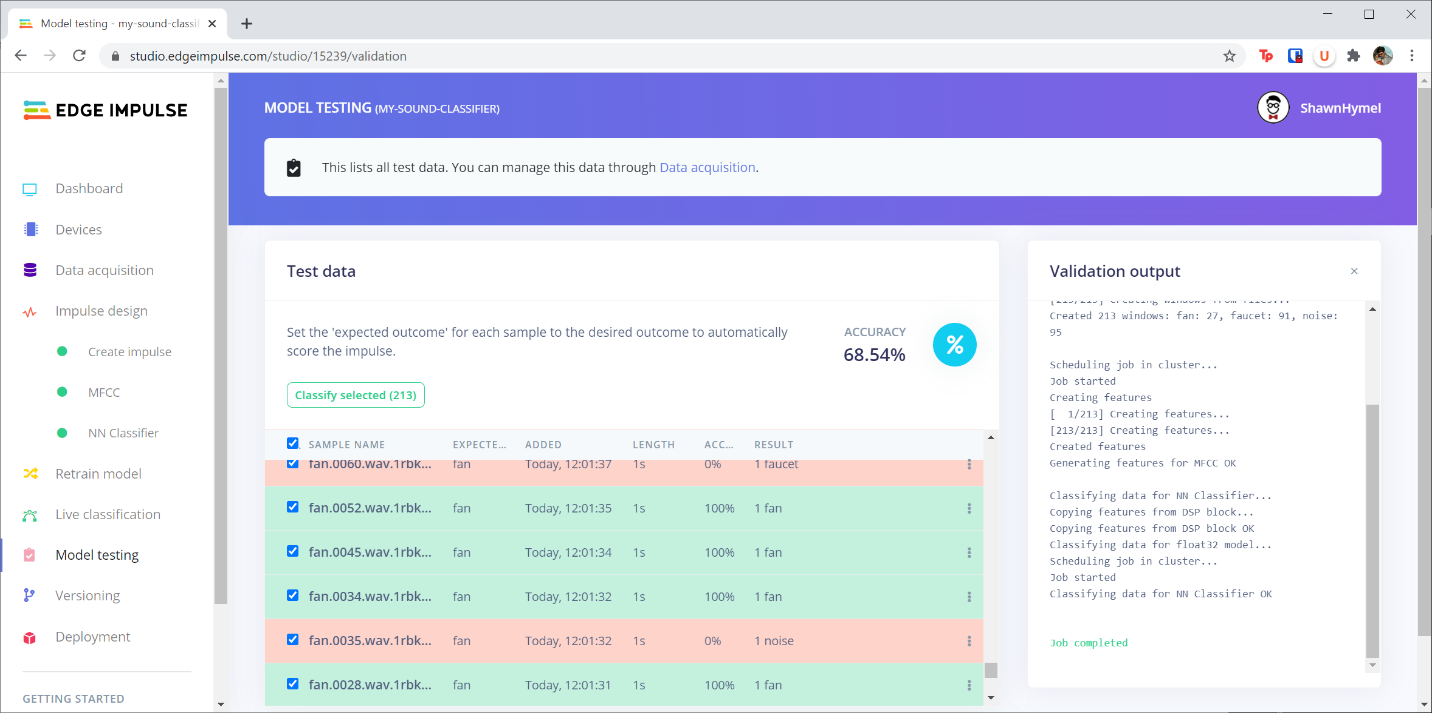
Use whichever feature extraction method (MFCCs or Spectrogram) provided the best results.

Feel free to try changing some of the hyperparameters and re-training your model to see if it improves the per-class accuracy. Note that there is no easy solution here: much of creating a better model is trial and error, and sometimes, you simply don’t have enough (or the right kind of) data to train a good model. In those cases, it’s back to the drawing board to gather data!

Once you are happy with the results, move on to the next step. I recommend having all F1 scores be above 0.5. As you can see, my F1 scores are close, but I should probably spend more time gathering more data or adjusting the hyperparameters.

**Testing**

Head to the **Model testing** page and classify all of the test data.



If you’re happy with the test results, continue to the deployment step. 68% is decent, but I should probably go back to collect more data and adjust hyperparameters.

**Deployment**

**If you are using a smartphone:**

Connect your smartphone to your project (if you have not done so already). Head to *smartphone.edgeimpulse.com* on your phone’s browser. Scroll down and click **Switch to classification mode**. If asked, allow the program to access your phone’s microphone.

Hold your phone’s microphone near various sources of sound to see if it can identify them.

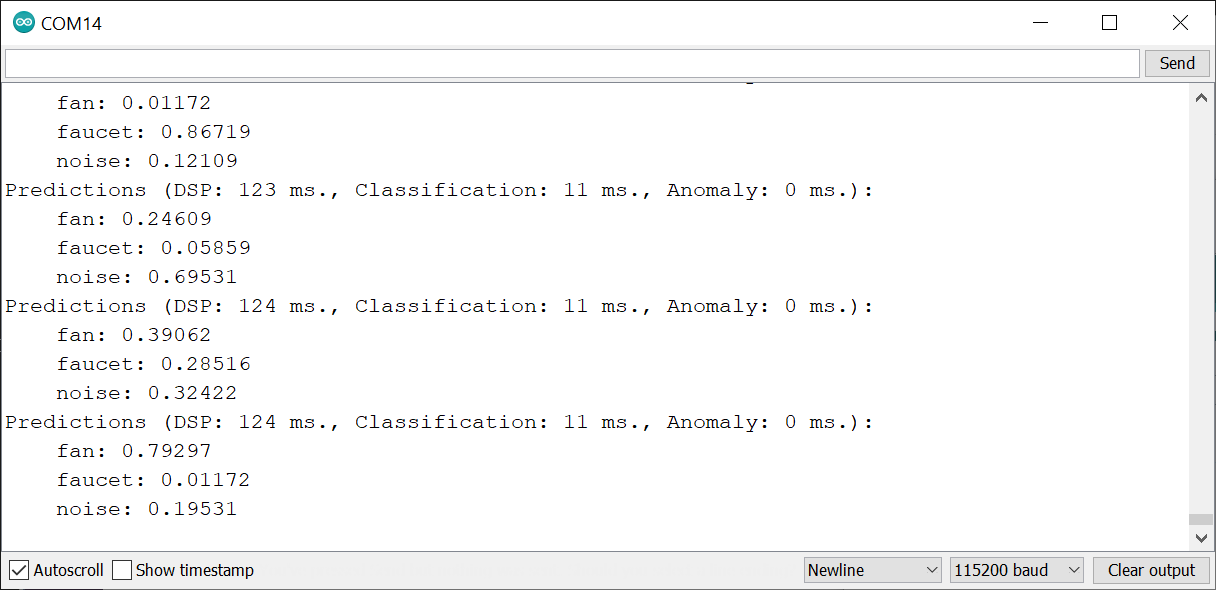


**If you are using the Arduino board:**

Head to the **Deployment** page in your Edge Impulse project. Select the **Arduino library** and click **Build**. When it’s done downloading, open your Arduino IDE. Select **Sketch > Include Library > Add .ZIP Library…** Select the library file you just downloaded.

Go to **File > Examples > <my-project-name> > nano\_ble33\_sense\_microphone\_continuous**. Click **Upload** to compile and send the program to your Arduino board.

When it’s done, open a Serial Monitor. Try holding the Arduino board up to different noise sources. How well does it perform at identifying the noise source?



**(Optional) Arduino Challenge**

If you are using the Arduino board for this project, I encourage you to try creating a *2-point moving average filter (MAF)* for one of your labels. Comment out the following section of code in the *nano\_ble33\_sense\_microphone\_continuous* sketch:

if (++print\_results >= (EI\_CLASSIFIER\_SLICES\_PER\_MODEL\_WINDOW)) {

        // print the predictions

        ei\_printf("Predictions ");

        ei\_printf("(DSP: %d ms., Classification: %d ms., Anomaly: %d ms.)",

            result.timing.dsp, result.timing.classification, result.timing.anomaly);

        ei\_printf(": \n");

        for (size\_t ix = 0; ix < EI\_CLASSIFIER\_LABEL\_COUNT; ix++) {

            ei\_printf("    %s: %.5f\n", result.classification[ix].label,

                      result.classification[ix].value);

        }

#if EI\_CLASSIFIER\_HAS\_ANOMALY == 1

        ei\_printf("    anomaly score: %.3f\n", result.anomaly);

#endif

        print\_results = 0;

    }

Replace it with your moving average filter. Choose a threshold appropriate for your application (such as 0.7). When the moving average is above that threshold, print something to the Serial Monitor (e.g. “Event identified!”).

**Conclusion**

I hope this has helped you get started using machine learning with audio data! As you’ve probably seen, choosing a model and adjusting hyperparameters can be a messy process with a lot of trial and error. A lot of time goes into creating a robust model that works in most environments, especially when it comes to identifying sounds!

Please note that this project is just a start. It will require a lot more data and effort to get a working model ready for production.