Edge Impulse Audio Inference Guide

Mike Ostrowka
Kevin Dhaliwal
Jason Li
Benjamin Howard

Contents

Summary 2
Hardware Requirements 2
Software Used 2
Microphone Setup 2
  Starting and Stopping Sampling 3
  Reading Audio Samples (C++) 4
Training Edge Impulse 5
Implementing Edge Impulse 5
  Building 5
  Running the Inference 6
Troubleshooting 7
References 7

Summary

This guide will show you how to use the MAX 9814 microphone amplifier to run continuous audio sampling and run collected samples through a machine-learning audio inference model. One important thing to note is that you cannot smoothly read from other sensors using the ADC while doing continuous audio sampling. Doing this requires turning off the continuous sampling, reading from the sensor, and enabling continuous sampling, which leads to gaps in the playback.

Hardware Requirements

- MAX9814 microphone amplifier

Software Used

- Edge Impulse: Tool for training and deploying machine learning models targetting embedded systems - https://www.edgeimpulse.com/
- This guide uses C++. Since the edge impulse library primarily uses C++, this makes using it easiest with C++. If using C, it may still be possible to get this to work using <extern “C”> but we have not verified this. You can follow this guide to try this for yourself: https://isocpp.org/wiki/faq/mixing-c-and-cpp

Microphone Setup

We followed this wiring guide by a previous student group (https://opencoursehub.cs.sfu.ca/bfraser/grav-cms/cmpt433/links/files/2022-student-howtos/MAX9814-SetupForPlayableAudio.pdf) and modified it to use p9_33. To wire the microphone please follow this guide.

Starting and Stopping Sampling

To start audio sampling (with the microphone attached to p9_33) run this script on the target:

start.sh

```
#!/bin/sh

# start audio sampling
sudo chmod 0666 "/sys/bus/iio/devices/iio:device0/scan_elements/in_voltage4_en"
echo 1 > "/sys/bus/iio/devices/iio:device0/scan_elements/in_voltage4_en"
sudo chmod 0666 "/sys/bus/iio/devices/iio:device0/buffer/length"
echo 4000 > "/sys/bus/iio/devices/iio:device0/buffer/length"
sudo chmod 0666 "/sys/bus/iio/devices/iio:device0/buffer/enable"
echo 1 > "/sys/bus/iio/devices/iio:device0/buffer/enable"
sudo chmod 0666 /dev/iio:device0

```

To end audio sampling, run this script:

stop.sh

```
#!/bin/sh

#end audio sampling
echo 0 > "/sys/bus/iio/devices/iio:device0/buffer/enable"
echo 0 > "/sys/bus/iio/devices/iio:device0/scan_elements/in_voltage4_en"

```

These scripts were modified from the previously mentioned student guide to work with p9_33.

As mentioned in the summary, single-shot reading from the ADC cannot be done while reading continuous audio samples. This was not something that was mentioned in the guide so please keep this in mind. If you want to read from the ADC then run the stop script, read your sample, then run the start script again. This approach is discouraged as it will lead to gaps in the audio samples, which in our case impacted the accuracy of the audio inference substantially. If choosing to follow this approach, write 0 and 1 directly to "/sys/bus/iio/devices/iio:device0/buffer/enable" instead of calling the start.sh and stop.sh scripts to reduce the latency.

Reading Audio Samples (C++)

We used this function to collect the samples from the buffer in a thread. The code on lines 100 and 113 is related to Edge Impulse, and will be covered later in this guide. The main lines of code to focus on here are line 93, which opens the buffer file; line 103, which reads all samples present in the buffer, and line 117, which saves the sample into an array. The number 1.7 on line 117 is a software gain on the audio – adjust this to whatever number works for you to increase detection accuracy, though be cognizant of distortion that may be caused by overflow.

```
86  void AudioSampler::run()
87  {
88      /*
89      * Adapted from:
90      * https://opencoursehub.cs.sfu.ca/bfraser/grav-cms/cmpt433/links/files/2022-student-howtos/MAX9814-SetupForPlayableAudio.pdf
91      */
92      int16_t buffer[AUDIO_READ_BUFFER_SIZE];
93      int fd = open("/dev/iio:device0", O_RDONLY | O_NONBLOCK);
94      if(fd == -1) {
95          printf("ERROR: did you remember to run the mic start script?");
96          exit(1);
97      }
98      int count = 0;
99      // Discard old samples and add new ones to the window.
100     run_classifier_init();
101     while (!shutdownManager->isShutdownRequested()) {
102
103         read(fd, buffer, AUDIO_READ_BUFFER_SIZE * sizeof(uint16_t));
104
105         for(int i = 0; i < AUDIO_READ_BUFFER_SIZE; i++) {
106
107             //after collecting enough samples to run the classifier
108             //disable audio sampling and unlock mutex to allow single-shot adc reading
109             if(count >= AUDIO_BUFFER_SIZE) {
110                 count = 0;
111                 writeToFile(BUFFER_PATH, "0");
112                 adc_lock.unlock();
113                 audioClassifier();
114                 adc_lock.lock();
115                 writeToFile(BUFFER_PATH, "1");
116             }
117             sound[count++] = buffer[i] * 1.7;
118         }
119         sleepForDoubleMs(AUDIO_BUFFER_SLEEP);
120     }
121     adc_lock.unlock();
122 }
```

Training Edge Impulse

We followed this guide - https://medium.com/@teswar159/audio-recognition-using-edge-impulse-using-machine-learning-c5a5cb1ea9ee - to train edge impulse to recognize a fire alarm sound. The summary is

1. Collect samples – used youtube to get samples of fire alarms and background noise
2. Used MFCC (Mel Frequency Cepstral Coefficients) block to generate audio features with default settings.
3. Trained Neural Network with default settings
4. Deployed using “C++ Library” option

Some things to keep in mind that were not mentioned in the guide:

- If the accuracy of your model reads as 100%, the model is likely overfitted (not enough training data) or your testing data is too similar to the training data
- You can quickly get an idea of the real-world accuracy of your model by deploying it to your phone and running it in the browser

Implementing Edge Impulse

Building

After exporting the C++ library you are given a folder containing your model and the edge impulse library. To build it, you must use the Edge Impulse Makefile. The easiest way to integrate the library with the rest of your project is to just add your source code files to the Edge Impulse Makefile under CXXSOURCES after putting them in a folder in the same directory as the Makefile.

Important: The library triggers some compiler warnings so it will not build using the -Werror compiler flag. To fix this, turn off errors for the specific warnings that trigger errors, such as -Wno-error=nonnull-compare.

The library is quite large so the first time you build it might take up to 5 minutes.

Running the Inference

All the code to run the inference is found in edge-impulse-sdk/classifier/ei_run_classifier.h. After including this file, <run_classifier_init()> needs to be run before using the classifier. Next, fill up your buffer with samples until there are EI_CLASSIFIER_SLICE_SIZE samples (defined in ei_run_classifier.h) Once the buffer is full, run a function like this:

```
44
45  void AudioSampler::audioClassifier() {
46      signal_t signal; //wrapper for raw data
47      static ei_impulse_result_t result; //classifier return
48      signal.total_length = EI_CLASSIFIER_SLICE_SIZE;
49      signal.get_data = &getSound;
50
51      EI_IMPULSE_ERROR res = run_classifier_continuous(&signal, &result, false, false);
52
53      //sets alarm value
54      updateAverage(result.classification[0].value);
55      printf("%s: %f\n", result.classification[0].label, result.classification[0].value);
56      printf("\n%f", alarmValue);
57
58      if (alarmValue >= requiredCertainty) {
59          std::cout << "Alarm detected!" << std::endl;
60      }
61
62  }
63
```

This will run the audio through the classifier and return a float value of the certainty that a sound is a fire alarm, where 0.0 is 0% and 1.0 is 100%. The audio buffer is fed to the classifier using <signal.get_data = &getSound;> where getSound is:

```
40
41  static int getSound(size_t offset, size_t length, float *out_ptr) {
42      return numpy::int16_to_float(sound + offset, out_ptr, length);
43  }
44
```

Where <sound> in this function is the audio buffer containing your samples.

When you close the thread remember to run <run_classifier_deinit()> to free all memory used by Edge Impulse.

Troubleshooting

- If the inference returns a constant value, then the microphone likely isn't collecting samples correctly.
    - Try checking the connection on the wires
    - Ensure the start.sh script was run
    - Ensure you are using p9_33 or correctly modified the script to use a different pin
- If the inference is wildly inaccurate:
    - Try changing the gain of the microphone using the "MAX9814 Setup for playable audio" guide
    - Try retraining your model with a larger set of diverse training data
- If an error message along the lines of "resource is busy" is shown when attempting to read from ADC:
    - You are most likely trying to read in single-shot mode while doing continuous sampling. Turn off continuous sampling to take your single-shot sample then enable it again.

References

- MAX9814 Setup for playable audio: https://opencoursehub.cs.sfu.ca/bfraser/grav-cms/cmpt433/links/files/2022-student-howtos/MAX9814-SetupForPlayableAudio.pdf
- Edge Impulse Audio Inference Guide: https://medium.com/@teswar159/audio-recognition-using-edge-impulse-using-machine-learning-c5a5cb1ea9ee