CSE 145 / CSE 237D
Final Report

Underwater Acoustics

Team 1: Chao (Jack) Li, Ryan Wagner, Jiashen Wang

Team 2: Jeremy Smith, Rochelle Manongdo, Luis Llobrera, Tiffany Huang

Spring 2017

Abstract

It is often unclear to what extent human activities can damage ocean environments and how effective our actions are in repairing damaged habitats and restoring marine populations to their former states. Our project aims to address this issue by allowing researchers to monitor sea life trends in areas that have been adversely affected by human activities. We are working with seven years of acoustic data that contain fish noises from the Deepwater Horizon oil spill area to better understand how many, how often, and what kinds of fish are passing through the area as time progresses. With this information, we can gain a better understanding of how damaging human activities to ocean environments are and how helpful efforts to restore damaged habitats are. To help oceanographer with their research, our project was greenlit to automate the fish call detection as well as fish call classification. This allows researcher to spent more time analysing the data, instead of manually annotating the data.

Introduction

This project is implemented under the guidance of Dr. Ana Sirovic, a researcher at the Marine Acoustics Lab of the Scripps Institute of Oceanography. Using the seven years worth of acoustic data, our team will develop a signal detection algorithm to parse out the fish calls from the background noise. The outputs from the detection algorithm will then be feed into a Convolutional Neural Network (CNN) that can classify recorded fish noises according to species.

The current legacy baseline code for the CNN is written in MATLAB and is severely lacking in terms of documentation and code organization. We hope to fix this with the following:

- Delivering a single, intuitive application that is accessible to researchers as well as thoroughly documenting application processes as development continues.
- Determining proper metrics for our CNN's performance while improving the CNN according to those metrics.
- Generalizing the current Convolutional Neural Network so that it may be used in other fields of study as well; expressed as an interest by Dr. Sirovic

Problem

Following the Deepwater Horizon Oil Spill of 2010, researchers at the Scripps Institute of Oceanography decided to deploy a detector which recorded ocean sounds. It was left in the area for 7 years and has since collected a significant amount of acoustic data. Much of this data is comprised of fish calls. Of the 7 years of acoustic data, researchers have been able to successfully annotate 2 years worth, classifying recurring acoustic patterns by the fish species that are believed to have produced them. This however, is not an efficient process and even if researchers were to successfully parse all of the data, the question remains: What should be done with all of this information?

Solution

By taking the 2 years of annotated data and feeding it into a custom Convolutional Neural Network as training data, the remaining data can be systematically organized and classified. Further, by developing a detector in parallel, the Neural Network can be used as a research tool for the long-term surveillance of sea-life populations.

Our Deliverables

We present the following three major deliverables to be accomplished by the end of the quarter:

1. *An Automated Signal Detector* - takes in audio data and outputs parsed audio data of fish call signals.
2. *A User Friendly Interface* - a means of interfacing between detector and classifier. Additionally, a graphical executable that will allow researchers without a deep knowledge of MATLAB to use the tool.
3. *The Classifier* - a Convolutional Neural Network designed to classify fish species by the noises they produce. Continued development necessitates improved accuracy and runtime. Dr. Sirovic has also shown interest in generalizing the CNN as a tool for signal and acoustic classification.

Technical Material

Team 1:

Team 1 - Detection Overview
For this project, we used a few technologies to achieve our goals. This section will overview the method used to detect fish noises in a WAV file. Our initial experiments used DFT with average thresholding in combination with the Teager-Kaiser Energy Operator (Used for baseline detection in [1]). This method was unsuccessful in providing adequate results. Most notable on the table is the runtime for the algorithms used in detection, as well as the return values in the DFT and Teager-Kaiser Methods when compared to ours. We found that the DFT method was outputting an overly narrow array of results, returning many false negatives. We had similar results with the Teager-Kaiser Method.

Since we prefer false POSITIVES over false negatives, we decided on using our own method. It is important to note that the Teager-Kaiser method is very valuable for single-call detection. If we were searching for a single aquatic call within the input stream, this would be the ideal approach to take in detection. The problem is that we do not know what calls might exist on the recording, so we require a very broad detection scheme.

Our implementation was focused on finding average amplitude thresholds and returning values above this. The specific method is detailed as follows:

- Split the file into several 1-5 minute tracks (to reduce the memory footprint of detection).
- Increase the contrast of the wave amplitude.

- Take the absolute value of all amplitude samples.
- Find all peaks discovered in 0.5 second intervals.
- Take the mean of those peak values.
- Filter out all values below the discovered mean.
- Generate intervals by calculating the time between each peak and the current running mean.
    - All values found within an interval get combined into the interval of the highest peak.

The chief benefit to this implementation is that it would not remove fish noises that may have been at a much lower loudness or frequency than the “average” noise, and returns a much larger breadth of possible noises to classify. Although it does have a high rate of false positives, it has two chief benefits, which reflects our focus in this project:

- Very unlikely going to miss a call, since it picks up based on local averages.
- Precomputes short calls for the classifier used in the next step efficiently.

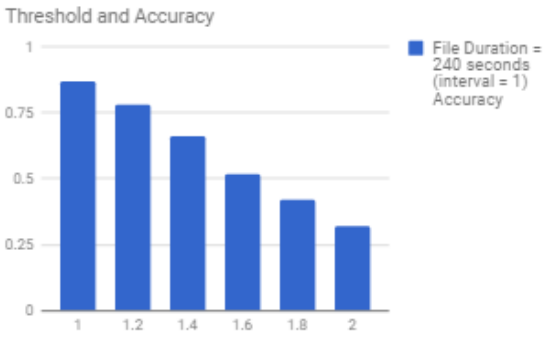


Figure 1: Minimum Amplitude Threshold vs. Detection Accuracy

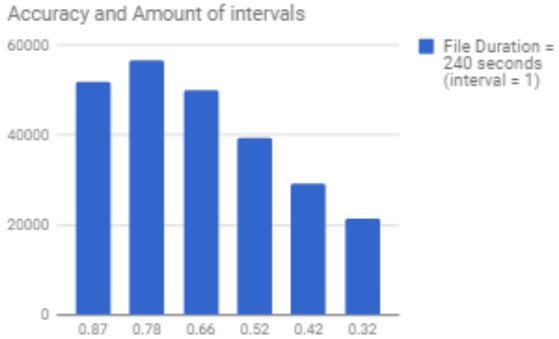


Figure 2: Detection Accuracy vs. Number of Generated Sound Intervals

Pictured on figure 1 and figure 2 is our accuracy, number of intervals (possible calls), and thresholding values. We found that the higher our threshold (threshold = n*mean) was for adding a value to the interval list, the fewer intervals we were left with and, consequently, the more calls we missed. This makes sense because not all calls will be of uniform amplitude. Note that as the threshold decreases, number of false negatives decreases. As we are primarily interested in ensuring no calls are missed, this is the most important value to keep track of. On a 4 minute sample, we saw our greatest algorithmic accuracy (87%) with the most forgiving threshold. We repeated these tests on 60 seconds, 120 seconds, 10 minutes, 1 hour, and 60 hours with very similar-looking results. We found that the smaller we cut up the sound file, the more accurate our results were. This is shown in figure 3.

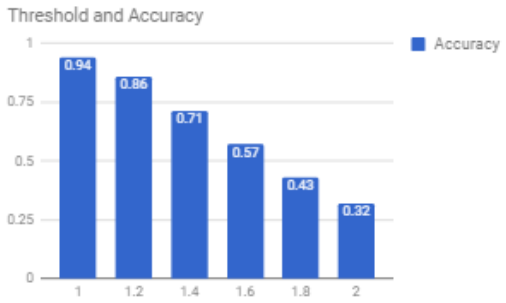


Figure 3: Thresholding Value vs. Accuracy for a 60 second file

The following details the runtime of each method, as this demonstrates the practicality of automation over the traditional by-hand method. We found that using our method on a 60 hour file, detection takes a total of 4 minutes to complete. As a comparison, detection by hand on a file of any size tends to take about 3 times long as the file length (60 hours = 180 hours of labor). Note that this number is not set in stone since all researchers tend to take differing amounts of time to detect and classify calls. A 10 minute WAV file took 10 seconds to process via the DFT method. Finally, the integrated Teager-Kaiser method takes 11 seconds to process a 10 minute file.

Team 1 - Hardware Platform

We chose to synthesize our algorithm onto a ZedBoard, as a proof of concept that it synthesized small enough to be put onto a FPGA. The board that we chose was the Zynq-7000, as it was readily available for use in our testing. In the future, a board much more similar to the one being used in the sound recording will be preferable so that the algorithm can more readily be integrated into the real-time system. Below is an image of one such board.

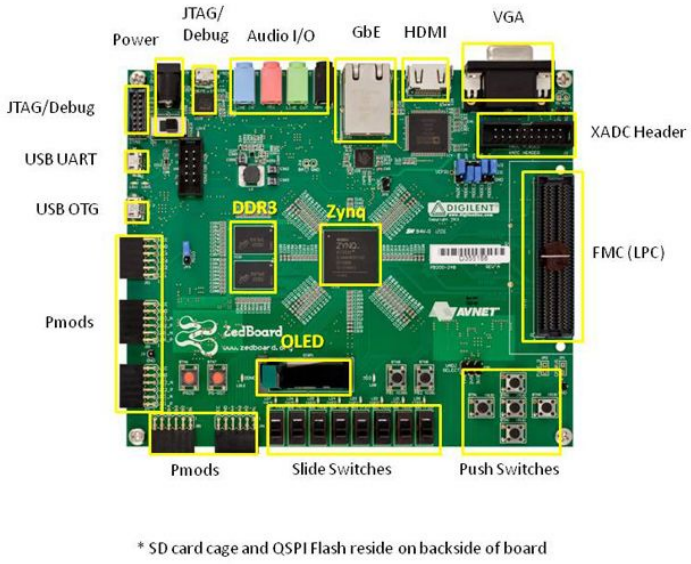


Figure 4: ZedBoard ZYNQ-7000 Board

The initial algorithm that we used, as well as our final implementation, are compared on figure 2. Note that the initial implementation was much larger than what would be usable on the Zynq board. Our final algorithm was much simpler, and our decision to reduce the sampling frame size from the entire input sound file to segments of 1-5 minute files to reduce computation size made a very large difference in the resultant hardware.

One frequently asked question is why our team chose to use a FPGA accelerator. We decided to implement a hardware accelerator because eventually our team would like to put the CNN into the accelerator. So the whole parsing and classification process can be done in one program. However, due to time constraints, we decided it is best to not be over ambitious and selected only the signal detection algorithm.

Team 1 - Hardware Implementation

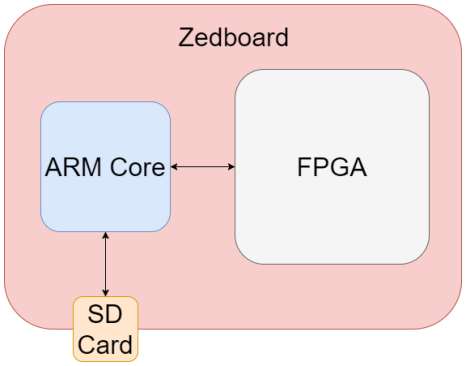


Figure 5: Communication Overview

Figure 5 shows our integration design. The ARM core will communicate with the SD card and the FPGA. The SD card contains the input audio data. The ARM core will take in an audio data file and a duration variable on which the user wants to analysis the data. The audio data will then be passed into the FPGA for parsing out the fish calls. Once the FPGA completes its task, the results are passed back into the ARM core, which will write to the SD card.

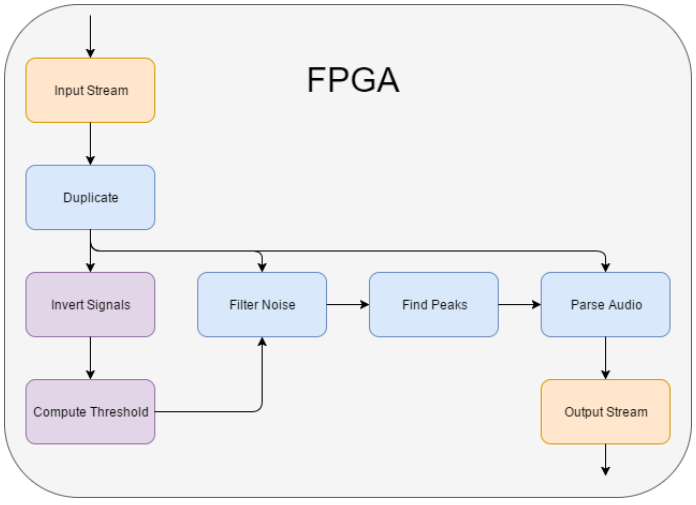


Figure 6: Signal Detection Block Diagram

The hardware accelerator description is as follows: It takes in an input stream of the audio data. Then it duplicates the data three times which will be used for computing the threshold, filtering the noise, and parsing the potential candidate fish calls. The duplication is needed because reading from the data stream is destructive and the data will not be available for reuse. The first part is inverting the signals from the negative y-axis to the positive y-axis. Then it computes the threshold by taking the mean and a nudge factor. The threshold is then used filter out noise. Once the filtering is complete, the find peak module will compute the local maxima. For each peaks, we take a stride of t+stride and t-stride at that peak's time. This interval is the potential fish call candidate.

Team 1 - Output Specification

For our part of the project, another important design decision was to decide on an output format that could be easily read by the classification team. We chose to output a vector of positions where possible calls had been detected, which was tentatively output to a text file as a string of 32-bit floating point numbers. How this output information is used will be discussed further in the Team 2 Technical Material.

Team 2:

Team 2 - Classification Process

For the classification process, we are making use of MatConvNet, a MATLAB toolbox that implements Convolutional Neural Networks for computer vision applications. While MatConvNet has several pretrained networks, we are working with a custom-designed architecture that is better suited towards our needs in classifying fish noises.

Our custom CNN is composed of the following architecture:

- Convolutional Layer 1
- Batch Normalization Layer 1
- Pooling Layer 1
- ReLU 1
- Convolutional Layer 2
- Batch Normalization Layer 2
- Pooling Layer 2
- ReLU 2
- Convolutional Layer 3
- Batch Normalization Layer 3
- Pooling Layer 3
- ReLU 3
- Convolutional Layer 4
- Batch Normalization Layer 4
- ReLU 4
- Convolutional Layer 5
- Softmax Loss

The Convolutional Layers perform the convolution operation in the network. A series of multidimensional filters is convolved with the input map.

Here are the parameters for each of the convolutional layers. f is the constant 1/100, and randn is a MATLAB function that returns an output matrix with normally distributed pseudorandom numbers of the same dimensions of the input parameters. zeros is a MATLAB function that returns a zero matrix of the same dimensions as the input parameters.

Layer No.	Weights	Bias Weights	Stride	Padding
1	f*randn(10,10,1,10, 'single')	zeros(1, 10, 'single')	2	5
2	f*randn(3,3,10,20, 'single')	zeros(1,20,'single')	2	0
3	f*randn(4,4,20,50, 'single')	zeros(1,50,'single')	2	2
4	f*randn(3,3,50,500, 'single')	zeros(1,500,'single')	1	0
5	f*randn(1,1,500,9, 'single')	zeros(1,9,'single')	1	0

The Batch Normalization layers perform a special kind of normalization in between layers of the network, which can transform the normalization layer's inputs to have zero mean and unit variance. The batch normalization algorithm has the capability to decide if the normalization process is beneficial to the network or not, and can undo the normalization if necessary. There are several benefits to performing batch normalization:

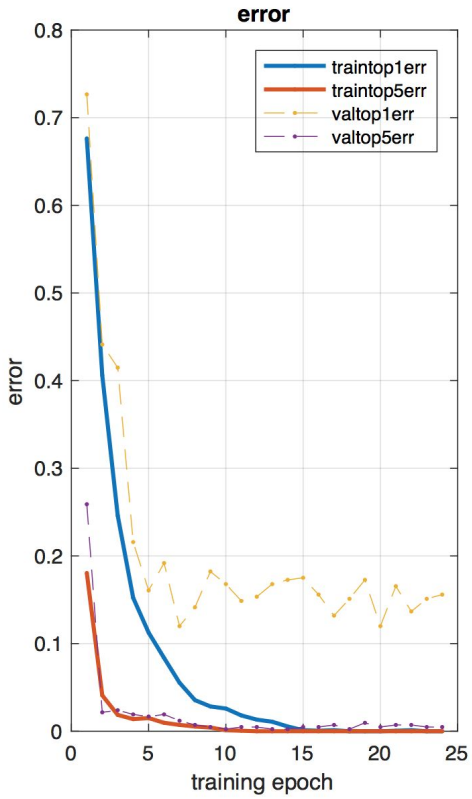
1. Normalized input data to different layers typically performs better for classification tasks.
2. Batch normalization acts as a regularizer, which introduces additional information to combat overfitting.
3. Batch normalization allows us to use higher learning rates, which results in the network being trained faster than a non-normalized network.

The ReLU layers perform the Rectified Linear Unit activation function, which sets a lower cap at 0 for the weighted sum of each node's inputs. This is how nonlinearity is introduced into the network. The Pooling Layers each implement max pooling, with a 2 by 2 size patch. This reduces the size of the processed data by extracting only the maximum value of each spatial neighborhood. Each pooling layer uses a stride of 2, and zero padding. The Softmax Loss Layer combines a softmax layer and a log-loss layer into a single step in order to improve stability. Softmax fits each of the inputs into the range [0,1] such that the total sum is 1. This is especially useful for the image classification task, so that the output layer can indicate its degree of confidence in what the image's class is with a percentage for each possible class. Log-loss, or logarithmic loss, measures the performance of the model using an input value between 0 and 1. A high log-loss would result from a very wrong prediction. For example, if an observed label is 1, but the prediction is 0.001, there would be a very high log-loss. Note that while a classical Convolutional Neural Network would have a fully connected layer, MatConvNet does not distinguish between fully connected layers and convolutional blocks. The equivalent of fully connected layers are obtained by using convolutional layers with a 1x1 output. In this implementation, this is done for the last convolutional layer, where the first two dimensions of the filters are 1x1.

For our program and execution flow, our program flow begins by having the user manually create the training data for the network. 75 second clips of audio data are displayed as a spectrogram, and

the option given to “peek” inside. If this option is chosen, the 75 second clip will be broken up into 7 snapshots. The user will then select the label from one of 9 supported fishcall types. The label is applied to the processed data, and saved as an individual .mat file.

Once all training samples have been obtained, each labeled snapshot is reshaped into a matrix, and standardized by subtracting the mean of the training data from each point. The points are saved into ‘imdb.mat’ for the actual training and validation of the network. The training process extracts the training samples from ‘imdb.mat’ and then uses back propagation with the architecture described in order to determine the values of the convolution kernels as well as the bias weights for each layer of the network. During the process, 24 “epochs” are created, which provide snapshots of the network’s status at different points throughout the training process. After the last epoch has been created, the network’s training has been completed. A portion of the training data is allocated for validation, which can be visualized with the following graph.



In this image, the top 5 and top 1 training and validation errors are displayed. Top 5 errors are the percentage of classifications where the actual label is not within the top 5 predictions, and top 1 errors are the percentage of classifications where the actual label is not the top prediction. As can be seen, error rates decrease as additional training data is used, until around the 10-15th epoch at which point the error rates stabilize.

Once the network has been trained, new test data can be passed into the network. This is done by reading in the time intervals where a fishcall can be heard (output of Team 1), calculating the midpoint of the data, then creating a 10 second “snapshot” of the data at the midpoint. This is done through the same process as the initial processing and labelling of the training data, minus applying

a label. All of the captured test data points are saved into a .mat file for later use. When classifying the test data, the mean of the training data is subtracted from each test data point, and the resulting data is passed through the network. The top output prediction is saved into a csv file along with the time (in seconds from the beginning of the wav file), and the degree of confidence in the prediction. This data can then be used for further analysis of the trends in fish call types in the area over time.

Team 2 - GUI Process

The GUI was made with Matlab's built-in tool, GUIDE (GUI development environment). This tool allowed us to design the user interface for the existing program. The graphical user interface was designed in mind to allow easy usability for researchers. Therefore, the GUI runs on one single program, with code ported from important files, such as spectrogram.m, sample_generation_main.m, new_parm.m, and loading.m.

The following illustrates the GUI process:

Running the Program

Opening the Intro Menu

Program execution begins at the IntroMenu.m file. Double click on the IntroMenu.m file and select Run in the MATLAB Home tab. A window should appear as illustrated below.

Intro to the Intro Menu

The intro menu is comprised of four buttons which correspond to four different steps in the classification process. Their specific inputs and outputs as well as their functionality will be outlined below

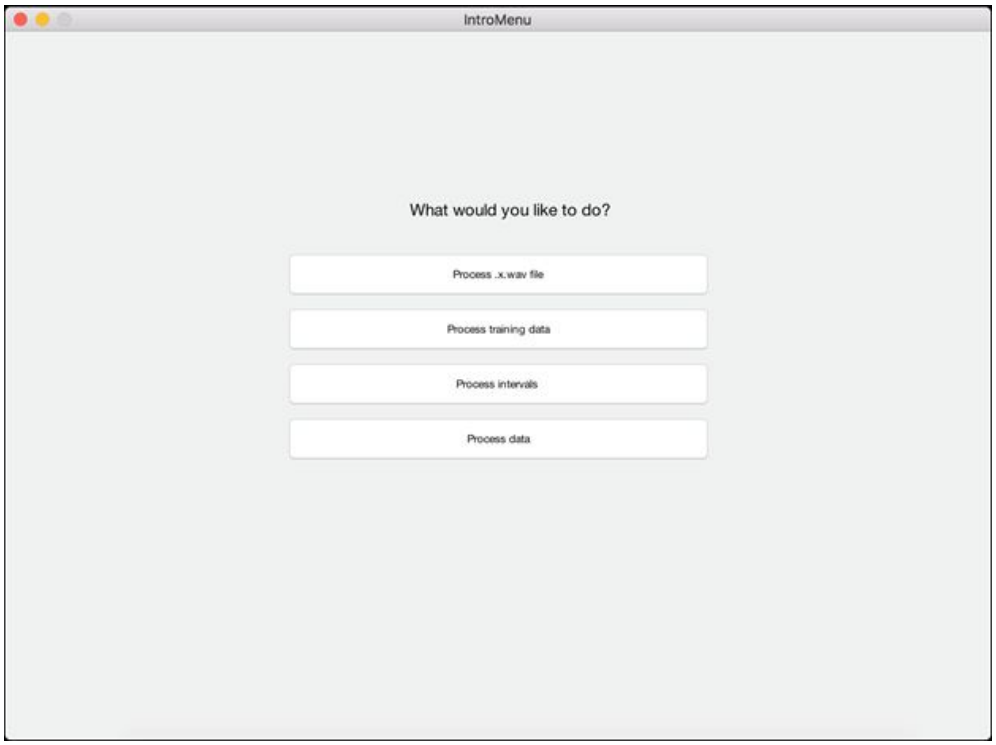


Figure: The IntroMenu window as it appears when running IntroMenu.m

- **Process xwav data**

  Given a .x.wav file, this option will open a new window which will convert the .x.wav into a series of seventy-five second audio wave images, that are themselves, parsed and cut up into seven smaller audio wave images. This portion of the program is designed to prompt the user to manually label these smaller images matching them as closely as possible to images displayed on a legend given on its graphical user interface. An iteration is designated as the labeling of these seven consecutive smaller images. Depending on the size of the .x.wav file passed into the program, this labeling step may require several iterations. Each iteration produces a series of .mat files which are saved into a directory specified by the user at the window's opening. Optionally, the user may skip to a certain iteration at any point in time to continue labeling from a previous stopping point.

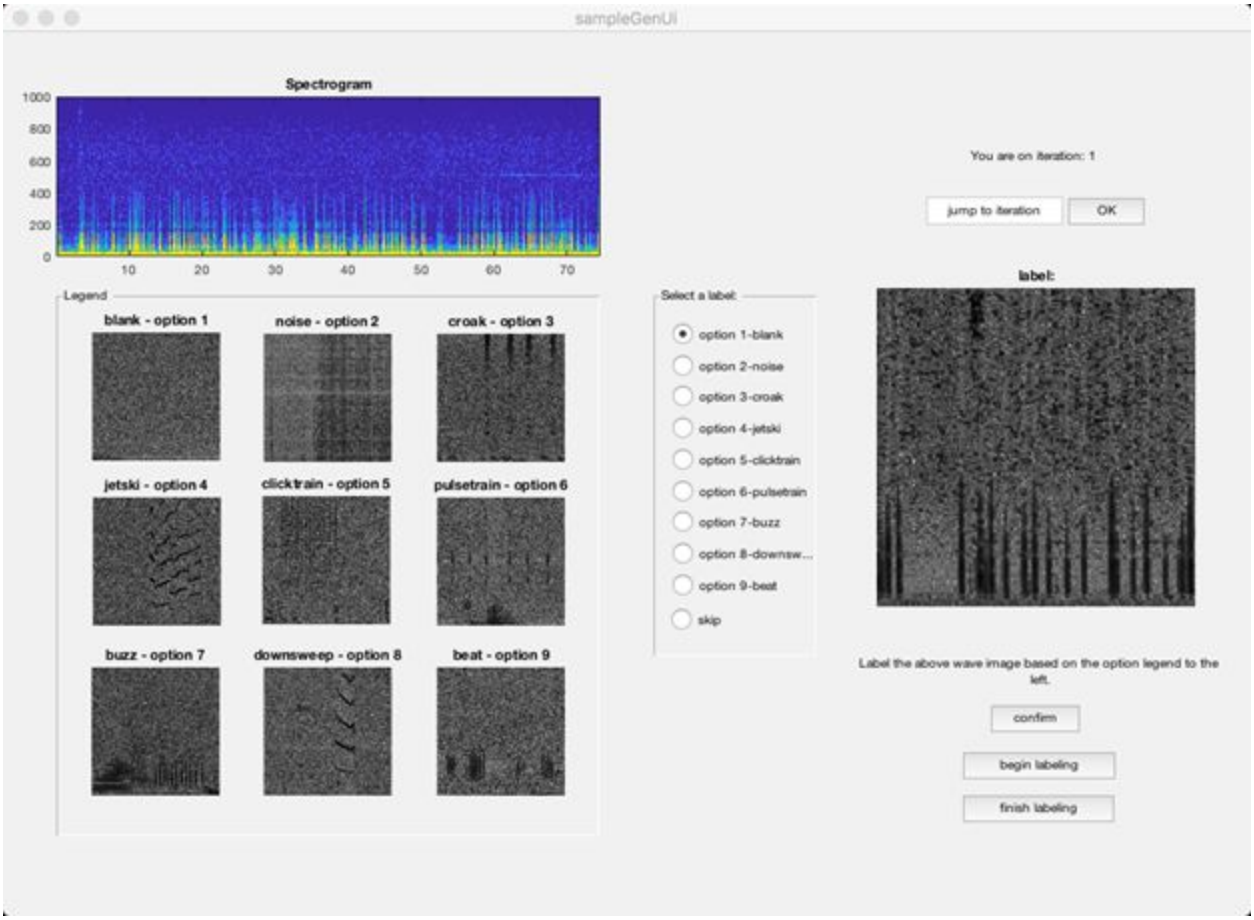


Figure: The SampleGenUI window as it appears following initial user prompt

- **Process training data**

  Once clicked, the program will prompt the user to select a directory containing the .mat files produced by the step prior. Additionally, the user will select a destination directory to store a resulting .imdb file. This .imdb file is the trained Convolutional Neural Network. Aside from these two prompts, no other interface appears.

- **Process intervals**

  This process prompts the user for a .txt file containing time intervals in units of seconds. These intervals have been identified as slots of time during which fish calls might have occurred. Using these intervals, the program will parse audiowave data and produce several unlabeled image_data.mat files of the same format as those produced in the 'Process xwav data' step. These will be input to the subsequent 'Process data' step for automated labeling by the trained convolutional neural network produced in 'Process training data.' The parsing process is fairly lengthy, MATLAB's waitbar is used to display current progress of the process to the user.

- **Process data**

  After having the user pass in the directory containing the unlabeled image_data.mat files, this process will use the .imdb file which should be produced in the 'Process training data' step and stored in the project's '/data/fishcalls-bnorm/' directory, as a trained network for automatically labeling all of the .mat files in the passed in directory.

Milestones

Team 1:
Our major milestones are as follows:

1. Familiarize ourselves with the topic of acoustic detection, through a large set of papers.
    a. Demonstrate this knowledge by intelligently choosing and optimizing a useful algorithm for fish detection.
2. Implement a basic fish detection algorithm using DFT and average amplitudes.
    a. Run this on testing data.
    b. Compare this data against expected values to generate detection error rate.
3. Implement Teager-Kaiser energy operator analysis.
    a. Test and compare this with plain DFT method
4. Compare and contrast both methods on algorithms as improved by given data, and optimize for hardware integration.
5. (NEW) Create a new algorithm to detect sounds, as Teager-Kaiser with DFT synthesized very large and ran very slowly.
6. Test and implement this method on hardware.
7. Verify results after integrating onto ZedBoard, in order to ensure algorithms work as expected.
    a. False positives must not increase more than 10% versus initial software implementation.
    b. Must not filter out known classified fish noise.
8. Generate documentation so that our experiment may be repeated in the future.

We managed to accomplish our milestones, however several unexpected things happened.

- *Incomplete/Incorrect manual signal detection* - Many of the input files that had already been classified were incomplete. This meant that several of our detected sounds had to be verified by hand.
- *Input file size*- The input files that needed detection were extremely large. This meant that we had to either parse this entire file into a table on a file somewhere, or (what we did) we had to cut these files up into much smaller, more manageable files.

- *Overly strict detection* - The detection method used by Ana in her paper was very strict in detection of input noise, so it tended to miss even many of the sounds that were found in hand detection. This is why we chose to use a different implementation of detection.

Team 2:

The following outlines our milestones for the quarter:

1. Demonstrate a thorough understanding of existing legacy base by being able to consistently run current version of CNN and display resulting data.
    - Obtain necessary libraries.
    - Update instances of deprecated MATLAB functions.
    - Restore modified code to original state.
2. Organize a currently scattered legacy code into a singular program that can be run simply and intuitively.
3. Design and implement a graphical user interface to make the CNN simple and intuitive to use. Ideally, our team would deliver a completed application by the end of the quarter that could be used by Dr. Sirovic at a scheduled presentation.
4. Familiarize ourselves on the topic of CNNs by writing a simple report outlining our knowledge and determining proper metrics for measuring performance of the CNN we will be developing as the development process continues.
    - Draw parallels to existing code.
    - Propose improvements to existing code to improve performance.

Here is the link to the report: CNN report

5. Provide documentation on our program for future users of our program to be used as reference guide for clarification.

While we were able to achieve our milestones, it did not happen without encountering several problems along the way:

- *Lack of knowledge in CNNs* - None of us have had any experience with CNNs, so we found it difficult to understand what was going on in the existing code, and to improve its performance. To overcome this, we aimed to write a report on the topic of CNNs, revolving more around the existing code and helping users understand it by introducing neural networks while drawing parallels between a classical neural network and our code. We aim to leave the project in a state that is ready for improvement by proposing possible avenues for exploration by future students and researchers.
- *Working with a scattered legacy code* - The code we were given relied on several unexpected libraries, one of which was developed by the Scripps Institute. We had been expecting the code to be in a “ready to run” state initially and did not anticipate requiring increased correspondence to get everything up and running. We also did not realize that the code would need to be fixed and updated due to the period of time in which it had gone unused. Lastly, neither we nor Ana were aware that the code had been modified beyond its original purpose, and that we would have to attempt to restore it back to its former state to continue with out tasks.

Conclusion

Team 1:
We were able to generate a working acoustic detection scheme, as well as synthesize it to hardware. This detector dramatically reduces the search space for the classification, and helps to guarantee that classification can begin to catch up to the existing database (allowing for further study of new and old information).

Having very little knowledge of acoustic detection at the onset of this project, we were able to generate not only the initial goal (detection), but also the extended goal of hardware integration. This demonstrates that in a future iteration of this project, detection will be able to be integrated with the passive recording so that the output of the recording scheme will only need to be valid fish noises.

In the future, if both steps in our system are able to synthesize onto a board similar to the one currently being used with the recorder, we are optimistic about the results. We believe that both detection and classification will be able to happen in at least near-real time. Further, this work may be improved by modifying the current detection scheme to run on a sliding window using averages in order to better detect multiple simultaneous calls.

Team 2:
We accomplished our baseline goals/milestones stated from the beginning of this project. Coming from a background with no knowledge of CNNs and little familiarity with Matlab, our team was able to work through it and present a working product requested by Dr. Sirovic. Our goals for this project were the following:

- Detector/Classifier: To properly train our CNN in order for it to correctly classify the fish call inputs. Our process includes parsing through a .wav file according to the time intervals given by Team 1. The audio segments that our classifier extracts from our .wav file are segments that could potentially be identified to be a fish call. These extracted audio segments are then converted into a imdb.mat file that are used in the training process of the CNN. The CNN training process later outputs the prediction (of what type of fish call the CNN predicted the audio segment to be), the degree of confidence, and the time interval of the audio that was analyzed. The training process is then incorporated into our GUI.
- Functional Graphical Interface between the detector and classifier that allows easy usability for researchers to train and classify fish calls

The GUI will be used by Dr. Sirovic in a conference presentation at the end of the month, therefore we designed the GUI that enhances user experience for researchers to help communicate their work easily. Gradual improvements are strongly welcomed, so continuation on this project as future work poses a possibility for some of our team members.

References

1. Roch, M.A., A. Sirovic, S. Baumann-Pickering. (2013) Detection, classification and localization of cetaceans by groups at the Scripps Institution of Oceanography and San Diego State University (2003-2013). Detection, Classification, Localization of Marine Mammals using passive acoustics (eds: O. Adam, F. Samaran). Dirac NGO, Paris, France.
2. "In layman’s terms, what is batch normalisation, what does it do, and why does it work so well?" Quora. N.p., n.d. Web. 29 May 2017. <https://www.quora.com/In-layman%E2%80%99s-terms-what-is-batch-normalisation-what-does-it-do-and-why-does-it-work-so-well>.