Juan Dominguez, Yiyun Fan, Daniel Sarafian
CSE 145
Professor Kastner
June 11, 2020

Automated Acoustic Species Identification Project Report

Abstract

To conduct more thorough assessments of regional biodiversity, ecologists are developing new monitoring techniques to complement existing camera traps. To that end, researchers at the San Diego Zoo have deployed audio recorders in the Peruvian Amazon known as AudioMoths. However, the sheer amount of collected data presents unique challenges. We automate the identification of species using acoustic data with Recurrent Neural Networks (RNNs) model, as well as Digital Signal Processing (DSP) techniques such as filtering and time scaling. Automating the identification process allows ecologists to shift their focus to more important monitoring and studying. Currently, our ML pipeline can identify birds with 86.44% validation accuracy, and our DSP package is able to perform filtering, time stretching, and frequency shifting.

Introduction

Conducting non-invasive biodiversity monitoring and surveys can help track creatures identified as belonging to key species, determine the distribution of species over a region, and assess the impact of human interaction.. Historically, camera trapping technology has been one of the primary methods used by ecologists and researchers for these purposes.[1] However, smaller species such as rodents, insects, and aerial birds can be hard to detect using camera traps. Alternative methods for conducting this type of research pose other issues. Track pads, for example, can be ineffective at distinguishing different species. Trapping live animals is often the best solution for small creatures, but this requires a lot of effort and is severely invasive. Additionally, due to flash photography and the varying capabilities of cameras in different environments, camera traps are both more invasive and less effective in all settings than we would want.[2] Thus, researchers would like to develop new forms of technology capable of effectively monitoring biodiversity while maintaining the goal of being non-invasive.

Audio monitoring holds promise in providing effective, complementary solutions to the shortcomings listed above. San Diego Zoo researchers have deployed multiple units of the AudioMoth over a large swath of the Peruvian Amazon. The AudioMoth is a small, embedded audio recording device capable of recording at sample rates ranging from 8000 Hertz to (an experimental) 384 kilohertz,[3] typically beyond frequencies of interest. A single unit is also capable of low power consumption using three AA batteries; for example, Energizer Ultimate Lithium batteries will allow the device to capture over 7000 MB (7GB) of audio data over the span of 35 days.[3] It offers some key advantages over camera traps. It is able to capture the vocalizations of smaller creatures[4]; whereas a camera trap would be incapable of detecting a small rat, the AudioMoth would still be able to record its presence due to ultrasonic vocalizations. It may also be able to more accurately identify birds that can be beyond the field of view of camera traps. The compact size of the AudioMoth makes it a strong contender for the non-invasive gathering of acoustic data.

As mentioned above, the San Diego Zoo researchers have already deployed many AudioMoths in the Peruvian Amazon. About 35 of them have been deployed near Peru's borders with Brazil and Bolivia[5], and have been in deployment as early as 2018. Given the number of units and the length of time for which they have been deployed, researchers have already collected an enormous amount of data (approximately 4.5 TB which amounts to over 1700 hours of data) and are actively collecting more. Unfortunately, while a researcher can easily sort through camera trap recordings and images for the purpose of labeling data, the same cannot be said for audio data. Many animals vocalize at ranges outside of a typical human's hearing range (rats, for example, can vocalize at ultrasonic frequencies up to

50 kilohertz; bats even higher at over 90 kilohertz).[6,7] This makes it difficult for a researcher, or perhaps someone less trained with more time to perform the task, to determine which animals of interest, if any, are present in a given piece of data. The sheer size of the data and difficulty analyzing it has forced researchers to determine the process of manually processing and labeling an unfeasible task. As such, we will apply various Machine Learning and Digital Signal Processing techniques with the goal of automating the analysis of available acoustic data.

Technical Material: General Overview

Machine learning can help us detect and identify different features from large datasets. In the scope of this project, we start from using machine learning techniques to detect bird vocalisation. The lack of labeling in our dataset naturally leads us to consider unsupervised machine learning as our first approach. In fact, another project working on the same dataset tried it earlier and successfully split the audio clips into clusters. However, intuitive, unsupervised learning suffers from a few drawbacks for birdsong detection. First, it is difficult to understand the meaning of different clusters. Although clips within the same cluster bear similar characteristics, it still requires manual verification to know which cluster has the presence of bird vocalisation. Second, the audio files generated by different AudioMoths may be slightly different, which may lead to inconsistencies in clustering. With these concerns we considered the deep learning approach instead. Bird vocalisation detection is not a new subject in the field of deep learning. Many researches and implementations have been conducted on birdsong detection. There is even a bird audio detection challenge in 2016/2017, which competed on giving the best detection model based on two given audio datasets: *freefield1010* and *the Warblr.* (reference http://machine-listening.eecs.qmul.ac.uk/bird-audio-detection-challenge/) Our work is largely based on *the microfaune*[8], an open source bird detection model built and trained by the microfaune team. The team trained a model based on *freefield1010* and *Warblr*, and reached an accuracy of 90.18%. They first converted the ten-second audio clips from the training set into mel-spectrograms and used them to train a recurrent neural network, which is most commonly used to recognize the sequential characteristics of data. We will discuss in later sections on how we utilize the microfaune model and make changes to it for our project.

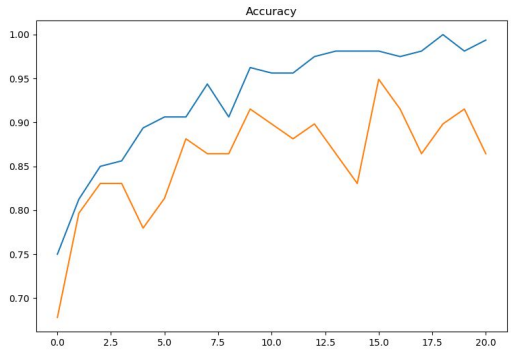
DSP gives us the ability to perform a variety of audio processing tasks without necessarily analyzing the data being processed. DSP will allow us not only to prepare data for analysis by our ML pipeline for comparison to unprocessed data, but it will also help us to mutate data in ways that will make it more accessible for humans to manually analyze. The two main fields of focus are filtering and scaling. Various filter types (bandstop, lowpass, highpass) allow us to isolate areas of focus or attenuate unwanted frequencies. By using infinite impulse response (IIR) filters defined by second-order filter coefficients (second-order sections, SOS), we can filter data to fit an assortment of use cases. More specifically, we might be interested in identifying a particular species individually which may have a specific and established vocalization frequency range. Filtering could potentially help us to narrow our focus of the frequency spectrum to that range. Additionally, since the Peruvian Amazon is a lively rainforest, there may be lots of unwanted noise in gathered recordings, such as that of rain or trees rustling. Filtering can also help to attenuate the noise. The second area of study, scaling, is split into two subcategories - time scaling to facilitate manual labelling and pitch scaling to listen to inaudible frequencies. Time scaling allows users to change the duration of an audio clip while preserving the original pitches. This would be especially helpful for streamlining the data labelling process, as it could potentially take half as long or even shorter. Pitch scaling does the opposite of time scaling - changes the pitch while preserving the clip duration. Our pitch scaling functions allow users to selectively shift one frequency to another, maintaining the harmonic relationships from the original clip. In the following two sections, we describe the progress of our milestones throughout the ten-week project, highlighting accomplishments and how we achieved them, shortcomings and challenges we faced, and revisions to milestones due to aforementioned challenges and shifting project focus.

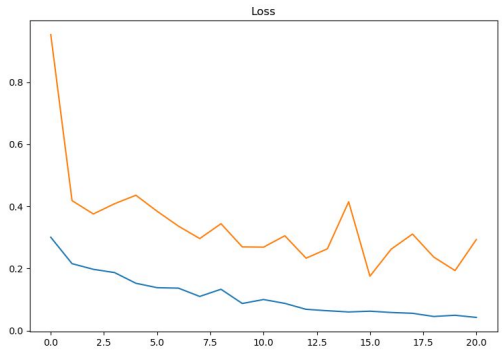
Technical Material: Implementation/Accomplishments

At a high level, we implemented all the originally planned milestones, but the purpose of the accomplishments shifted as we progressed through the quarter. These revisions are discussed in detail in the “Challenges & Plan Revisions” section.

The Machine Learning part of our project started from literature review on existing bird vocalisation detection algorithms and models. In the early stage of our project, we ran the model trained by *microfaune* directly on the Peru rainforest dataset captured by AudioMoth to test viability. Although we managed to get it up-and-running, the results were not good - around 60% accuracy and there were many false negatives. We speculated that a few factors might have caused this issue. First, different recording devices might have caused different features on output audio files, and thus influenced their respective mel-spectrograms which were then fed to the model. Second, our audio data captured by AudioMoth has a very high sampling rate of 192kHz, while everything from *freefield1010* and *Warblr* is 44.1kHz. Even though we downsampled our dataset before fitting it to the model, it is very likely that the downsampling process left special features to the audio files which confused the original model.

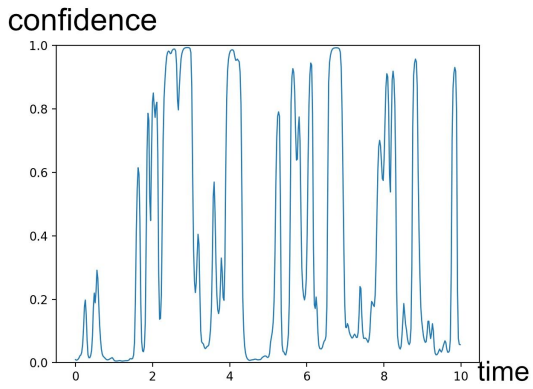
To solve this problem, we decided to finetune our model with audio files from the Peru rainforest dataset as additional data points. We hand labeled 305 ten-second audio clips based on the presence of bird vocalisation. 36 clips are then picked out and completely hidden from the model. The remaining 269 clips are split in 4 to 1 ratio, where ⅘ are used as training set for fine tuning and ⅕ used as validation set. We then load the trained model from the original microfaune project, and fine tune it with Adam optimization, where the learning rate is set as 1/10 of the original to achieve a more fine-grained result. We train the model over 100 epochs, and each epoch consists of 5 gradient steps. However after a few trials we found out that this method alone may cause the issue of overfitting - while training accuracy continues to increase over epochs, the validation accuracy increases to a point and drops afterwards. We therefore introduce early stopping to prevent model overfit, which halts the training process when the model doesn’t improve on the validation loss function value in 5 epochs.

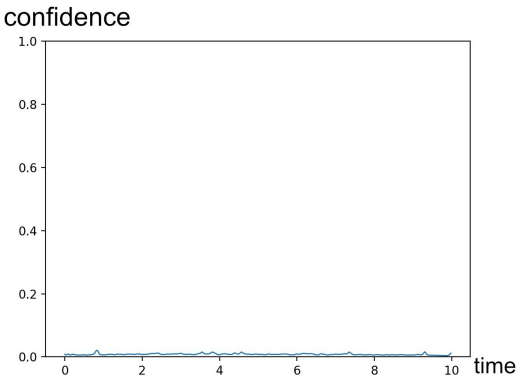




The figures above illustrate our training process, where the x-axis is the number of epochs, the y-axis is accuracy value and loss function value respectively, the orange line represents the validation set, and the blue line represents the training set. Training stops at 20 epochs due to early stopping.

Our fine tuned model shows some promising results. At the end of the training, our accuracy on the validation set reaches 86%. On the hidden test set, the model gives correct predictions to 34 out of 36 clips. For each clip, the model would output an array of local confidence, where the maximum local confidence value would be the global confidence value of the clip. If a clip has a global confidence over 0.5, it is marked as a positive detection. Below are two diagrams visualising the local confidence result. The one on the left is the result of a positive detection, where the spikes represent the high confidence on birdsong presence at some time. The one on the right is a negative detection, with low confidence throughout the clip.





Although the validation set and the training set are both relatively small due to the limited size of our labeled dataset, the result shows viability in terms of bird vocalisation detection with deep learning techniques and suggests potentials of application to larger audio files. Another potential usage is combining the model with unsupervised learning. With its help we may easily and efficiently find out which clusters are collections of birdsong and which are not. Moreover, since the model works on birdsong detection, the RNN model may work on another animal voice detection as well by changing the training set.

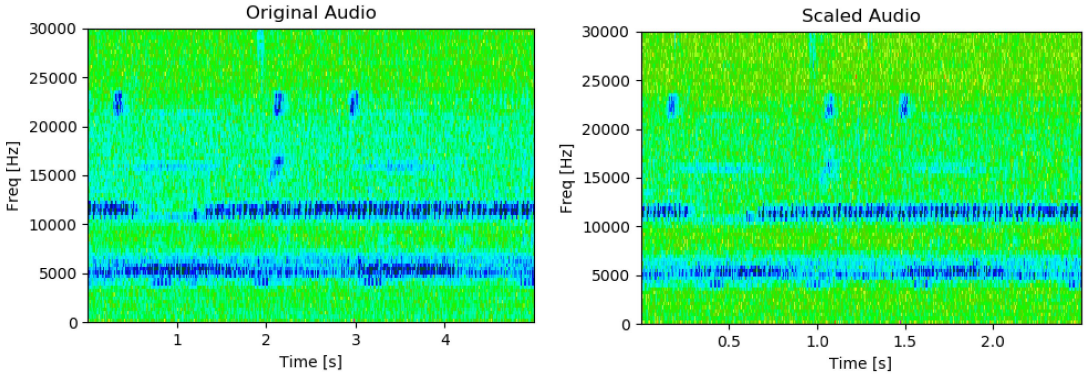
Apart from our main implementation on the deep learning model, we have also created some utility tools for our project and hopefully future work. Some of them include slicing, downsampling, and creating mel-spectrograms for audio files in large bulks. Since our Peru dataset mostly comprises audio clips of 1 minute, we also write a script that takes in an audio of any length or sampling rate, convert it to 44.1kHz, slice it into 10 sec clips, and finally runs it against our detection model and outputs the clips where the presence of birdsong is detected.

With regards to filtering, we completed a majority of the goals laid out in the milestone report. Since our focus on filtering shifted away from integration with ML, we decided to build an intuitive, easy-to-use Python library module that did not force its users to have more than a very basic understanding of DSP. Called *aasi_filters,* the library does just that. Clients need only to specify the desired cutoff or center frequencies of filters and whether the signal consists of stereo or mono, while the module takes care of things such as computing the normalized frequencies. The low-level filter-design functions provide clients with a little more control of the filtering process. They can specify the type of filter to be computed, between a filter defined by its polynomial coefficients or its second-order sections (SOS). SOS filters produced by the module are the default filter type as they produce fewer numerical problems and are thus more accurate. After a filter is designed, the module becomes even more intuitive. An audio file can be filtered as easily as calling a function on the original signal and a filter. However, at the core of the library is a particular function designed to make the filtering process as easy as possible. Given that we often deal with large audio datasets, the module is capable of filtering entire directories. The client is only required to specify as function parameters the name of the directory containing the dataset and the frequency or frequencies to use for the filters. Optional parameters include the order of the filter, the type of filter(s) to apply (among highpass, lowpass, and bandstop), and bandwidth (for bandstop). Apart from returning filtered data, the most beneficial optional parameter allows clients to specify a directory name to which the filtered data will be written. This function automates a majority of the process. Apart from automatically detecting the sampling rates of the data and detecting stereo vs. mono, a dataset can contain mixed files. I.e, it will work even if the directory contains files that are stereo and mono AND of varying sampling rates. With the library complete, and our goal of an intuitive, “plug and play” library met, we hope researchers and our own team will find effective use cases. Any good library consists of good documentation and an even better tutorial. As such another one of our goals, which we saw to

completion, was the creation of a Jupyter Notebook tutorial to accompany the library. Viewable on our branch in the team's Github repository, the tutorial briefly but thoroughly and effectively demonstrates by example how to filter individual audio files using the lower-level design and filtering functions, as well as how to filter entire datasets. To improve versatility of the library, we sought to implement more filters than what we had presented before the milestone report. We began by implementing a band-stop filter whose bandwidth and center frequency clients could specify. Now, the library also has the capability to compute and apply high-pass and low-pass filters. Additionally, there are functions that allow a more refined computation of filters, one that computes a finite-impulse response filter, and some that require convolution. However, these additional functions have not been fully implemented and may still have some problems. We also greatly improved the visualization of data, as was another one of our goals. We can plot the spectrograms of an original audio file and its filtered counterpart juxtaposed. The colorbar that displays next to the graphs is normalized according to both graphs in order to better understand the effects that filtering had on a file. We also had a goal of optimizing the performance of the library functions. This was a partial success. Thanks to caching of computed filters, we were able to improve the runtime of filtering a small data set of 228 ten second clips from 2510 milliseconds to 2136 milliseconds; a modest, but noticeable improvement. While filtering large datasets is relatively effortless and quick, this is not the case for filtering a file and plotting its spectrograms. In one benchmark, filtering a single file using a bandstop filter and subsequently plotting its spectrograms took an enormous 161525 milliseconds. Lastly, although far from obtaining desirable results, we did manage to integrate the DSP and ML pipelines.Thus, filtering tasks were mostly accomplished, some to a greater degree than others. We now have a capable, versatile, and mostly effective filtering mechanism.

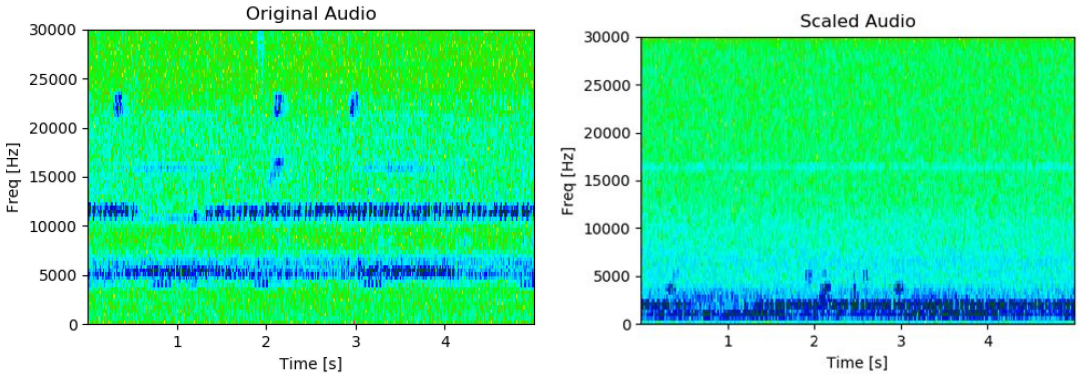
Originally planned as our first DSP venture, scaling was backlogged but revisited later in the quarter. Like filtering, we completed all our planned tasks and focused our attention on building a simple Python library for users to easily apply scaling techniques to audio files. The *dsp_scaling* module on GitHub contains both the time and pitch scaling functions. For each type of scaling, there are three variants of the scaling function - one built for inline use, where a previously read audio array is scaled, another for scaling a single file, and the last for scaling an entire directory. These variants provide great versatility of use.

Built with simplicity in mind, the user only needs to specify a positive factor to stretch or compress the provided audio sample. For the single file variant, a filename must also be specified, and optionally, the folder where the file is, the output filename, and boolean whether the file should be written. For the directory variant, the input directory must be specified, and optionally, the output directory. As an example, we can consider the two spectrograms below, showing how a one minute clip can be scaled by a factor of 2 to half its original duration while preserving the original pitch.



The pitch scaling functions work very similarly to the time scaling ones, with the main difference being the specification of a target frequency and an original frequency rather than a factor. In the pitch scaled output, the "original frequency" specified will end up at the specified target frequency. The pitch scaling function group also contains an extra function that applies a bandpass filter over the desired frequency range before shifting it into the audible range. This is especially useful for those studying ultrasonic

vocalizations such as bats and some mice. The two spectrograms below show an example of a pitch scaled audio sample, bringing 25kHz down to 4kHz.



To maintain its ease-of-use for new users, the module is heavily documented, both within the code and with a Jupyter notebook tutorial. The code documentation delves into each parameter and what each function achieves, and the tutorial provides a thorough walkthrough of how to use each function, accompanied with input and output spectrograms and audio samples.

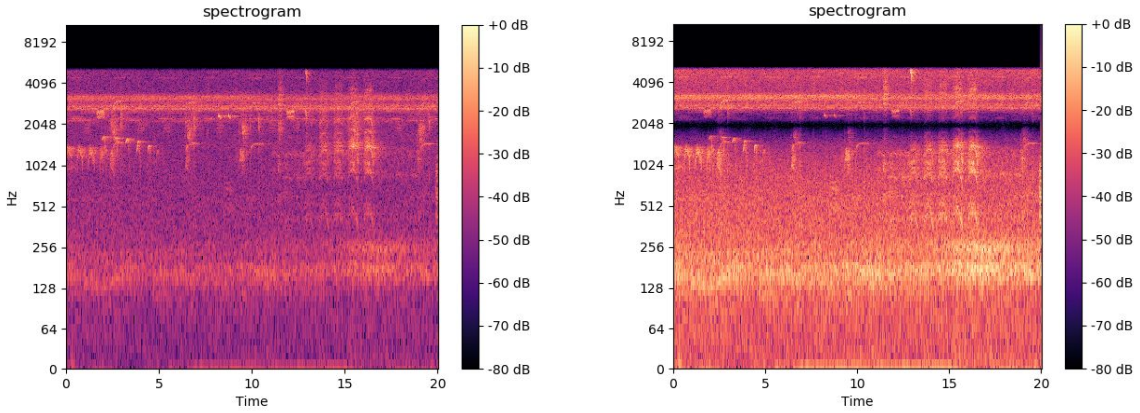
Although we plan to eventually continue the task of integrating the DSP and ML pipelines, at least to truly determine the viability of such an endeavour, the two diverged from a common goal of integration, into separate but equally important goals. We have delivered two, unintegrated pipelines. The ML subtask has produced a bird vocalisation detection model with satisfactory accuracy on our dataset. On the other hand, the DSP subtask has produced a toolkit consisting of a Python library with various filtering, time stretching, and frequency shifting capabilities.

Technical Material: Challenges & Plan Revisions

Originally, we had two deliverables planned, each consisting of a milestone for ML and a milestone for DSP. Given our limited knowledge and experience in DSP and ML, for deliverable 1 the DSP and ML milestones consisted almost entirely of research into the techniques we would use in the project. For DSP, these were techniques such as filtering and frequency shifting. For ML, it was research into existing, trained models for the detection of bird vocalizations. Following the milestones, we hoped to deliver a working, albeit possible incomplete DSP pipeline and partially trained ML model. Deliverable 2 comprised the bulk of our technical implementation. Our milestones for DSP included the completed implementation of a DSP pipeline for filtering, noise reduction, and pitch shifting, as well as its integration into the team's own ML pipeline. The ML milestones included the completion and refinement of the ML model, as well as improving its accuracy and using the integrated DSP pipeline to analyze data before and after pre-processing. In the end, we would deliver a trained model with high accuracy in detecting bird vocalizations and an integrated DSP pipeline capable of performing various processing techniques on acoustic data. While the beginning of the project saw DSP and ML as distinct but tightly coupled subtasks, throughout the course of the project, the milestone and deliverable goals changed drastically. Below we describe the unforeseen circumstances and challenges that resulted in the diverging goals.

In the early stage of our project, we considered using DSP techniques, such as filtering out a specific frequency range from input audio, to improve prediction accuracy. For instance, cricket noise, which ranges around 2000Hz[9,10], is rather pervasive in our dataset. By filtering out sound ranging near 2000Hz, we are able to get "cricket-free" audio clips, which naturally leads to the hypothesis that bird vocalisation will be more distinguishable post-filtering. However, the result does not turn out to be how we expect. When running filtered audio clips against the model, we only reach 68.9% accuracy. Although we have not fully understood its cause, there are a few speculations. First, the filtering process could leave some unwanted distinct features on the resulting spectrogram. The two figures below feature the

spectrograms of the same audio before and after filtering respectively. The figure on the right shows a huge dark line around 2000 Hz. This is a significant feature but has never appeared in the training set since none of the training set audio is filtered, which may confuse the model and lead to inconclusive predictions.



Second, bird vocalisation ranges from 1000 Hz to 8000 Hz[11]. Therefore by directly filtering out sound at 2000Hz, we may have also lost part of bird vocalisation, leading to inaccurate prediction. Although in our earlier milestone report we aimed to integrate DSP filtering with the ML prediction model, we eventually decided not to implement it since filtering does not turn out to improve our model performance. Instead, we turn our efforts to creating utility tools that help ease the labeling process and preprocess training data.

Another big challenge that has been pervasive on the machine learning side of our project is the lack of labeling in our dataset. As we mention above, direct use of the *microfaune* model failed to give accurate prediction on our dataset, and the difference in recording device might have caused this issue. Hence we deemed it necessary to finetune the model with training data from our dataset. To achieve this goal, we labeled audio clips ourselves, which posed a few challenges in the process. First, we are inexperienced in bird vocalisation, which may lead to false labels. To avoid filling the training set with too many false data points, we decided not to be overambitious and labeled 300 audio clips only. Second, though the amount of clips we processed was not enormous, it was still considerable since we were not able to label it in one pass. We overcame this problem by creating the spectrogram for each clip first and eliminating the ones that were almost completely silent or had too much noise.

One of the first tasks that we began work on early in the quarter was frequency/pitch shifting and time scaling. We were able to make advancements rather quickly here. We had already researched various ways to shift frequencies to bring them into the human hearing and had even made some basic implementations in Python. However, we decided to postpone work in this area. There was no clear goal with regards to bringing ultrasonic frequencies into human hearing. We realized that although a human might benefit from this, our primary goal, an ML pipeline, wouldn't. Thus, we shifted our focus to filtering. Some of the members of the project, not from the course, had been working on clustering data and wanted a reliable method of filtering, potentially to reduce or completely attenuate the background noise of crickets. As the quarter progressed and work on the ML pipeline continued, our attempts to integrate the pipelines proved uneventful as mentioned above. We determined that, with the current direction of the ML subtask, we needed to develop new goals for the DSP subtask. With clearer, more refined goals in mind, as well as a better understanding of DSP techniques (after thorough research) we decided to continue work on filtering and returned to the goals of frequency shifting and time scaling.

Conclusion

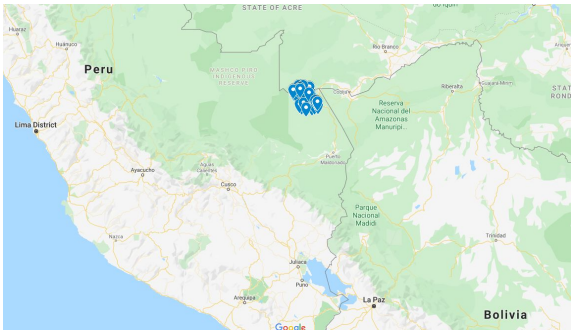
We approached this project recognizing the need for automated processing of massive amounts of audio data. In our initial plan, we aimed to create a pipeline where incoming audio would first be preprocessed with various DSP techniques, then passed through a machine learning model, resulting in accurate bird vocalization identification. As we progressed through the quarter, we realized that these two subsections of the project would function better as separate entities, and decided to create a DSP toolkit consisting of filtering, time stretching, and frequency shifting. Albeit with different end purposes, we met all our goals and can successfully identify bird vocalizations with 86.44% accuracy.

We believe there are a few potential applications of our project. On the Machine Learning side, the most straightforward application would be to directly mark the presence of birdsong throughout large audio files. In this way researchers are able to skip hundreds of hours of meaningless background noises and listen to the interesting parts directly. Another application is to combine it with unsupervised learning approaches to identify entire clusters of birdsongs. It is also useful for future projects of species identification based on the Peru rainforest audio dataset - if researchers want to train a deep learning model to identify specific bird species, it would be a handy tool to filter out all bird vocalisations for ornithologists to label and study.

In the future, our work can be extended by training the model with more data points from different AudioMoth devices. Since our training set data all comes from 2 particular AudioMoths, the model may require more training data to give accurate prediction to different AudioMoth data. Also, we can train a similar model with the same methodology for another kind of vocalisation, such as mammal calls, in the future. With success in species identification, the AudioMoth or similar audio technologies could potentially be deployed in other wildernesses around the world.

References

[1] : https://www.zsl.org/conservation/how-we-work/monitoring-our-planet/camera-trapping
[2] : https://www.researchgate.net/publication/264087573_A_novel_method_for_camera-trapping_small_mammals
[3] : https://www.openacousticdevices.info/open-source - AudioMoth Manual
[4] : https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4598429/
[5] : Audiomoth_Array_Map.png



[6] : https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4598429/
[7] : https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2933263/
[8] : https://microfaune.github.io/microfaune/
[9] : https://doi.org/10.1523/JNEUROSCI.19-04-01508.1999
[10] : https://www.jneurosci.org/content/19/4/1508
[11] : https://www.allaboutbirds.org/news/do-bird-songs-have-frequencies-higher-than-humans-can-hear/#:~:text=Many%20bird%20songs%20have%20frequency,reach%208%2C000%20Hz%20and%20beyond.