Final Report

Version: 1.0

Date: May 7, 2020

Team Name: IntelliChirp

Project Sponsors: Colin Quinn and Patrick Burns

Team's Faculty Member: Fabio Santos

Team Members: Steven Enriquez, Michael Ewers, Joshua Kruse, Zhenyu Lei

Table of Contents

Introduction

It is becoming ever more important to track and manage the biodiversity that lives on Earth. More and more animals and plants are becoming extinct everyday which can create a major impact on other parts of the ecosystem. The group that our team is involved with, Soundscapes2Landscapes, is a science-based project looking to further advance biodiversity monitoring in order to save the lives of plant and animal species. Biodiversity is the study that ”refers to the variety of living organisms on Earth, how they relate to each other, their ecological function, and genetic diversity. All aspects of biodiversity are intimately linked to the functioning of ecosystems, where species interact with their physical environment. Biodiversity plays a vital role in many ecosystem functions, such as clean water, clean air, nutrient cycling, food production, and responses to disturbances, such as fires.” [1] It is vitally important to conduct proper biodiversity monitoring, in order to understand the ever changing environments that humans share with plant and animal species.

Our clients Colin Quinn and Patrick Burns are part of the Global Earth Observation Dynamics of Ecosystems Lab (GEODE). Colin Quinn is a PhD student and Patrick Burns is a Research Associate at Northern Arizona University. They work with Soundscapes2Landscapes to help achieve their biodiversity monitoring goals. Our clients use a specific type of monitoring called passive acoustic monitoring (PAM). This process allows more spatially extensive and continuous metrics for biodiversity. In Sonoma County, PAM has the ability to provide land managers and users with a better idea of animal species affected by development and conservation efforts. Our clients have assigned our team with the task of automatic sound identification from a soundscape, as currently sound identification is conducted in a manual time consuming way.

Problem

To understand the problems of the current implementation, the workflow process will be discussed. First, soundscape recording data is collected from low cost audio recording devices that are placed in different landscapes across Sonoma County, California. Once placed, these devices record one minute of every ten minutes for three to five days at each site. This has so far resulted in a total of over 500,000 minutes of gathered audio data. The soundscape recording data then moves onto sound analysis where biodiversity can be identified along with the specific layers of biophony, geophony, and anthrophony. Once the identification and analysis are done, satellite imagery from the International Space Station is used to create visual representations of the surveyed sites. Finally, the satellite data and sound data are put together to create a species distribution model, which can be used to track locations of bird species and potential environmental changes in their ecosystems.

The part of the process we are involved with is the soundscape manual analysis. Currently researchers would manually listen to audio files and draw boxes around various sounds. For a one minute clip, a researcher must listen to the clip, determine what the sounds are, draw boxes, and label each box with the corresponding audio component that is occuring. With noisy files, a researcher may spend over a minute going through a single file. This tool is useful but takes too long for scientists to effectively research the biodiversity in Sonoma County, California. Because of the frequent recording of audio, the researchers have resulted in terabytes of sound data for each individual site. Additionally, our clients would like for volunteers, or citizen scientists, to be able to analyze their own files. For example if a volunteer is working out in the field and records some audio, there is no current way for this volunteer to analyze their file as most of the features of the current identification tool are closed to volunteers.

Overall, the problems include:

- The manual identification process is very time consuming. Terabytes of audio is collected from each site, and requires people to listen to the audio and manually draw boxes around the sound components being searched for.
- Current interface is not easily accessible to volunteers. Soundscapes2Landscapes wants this tool to be able to be used by anyone, and the current interface is not very easy to navigate for non-tech volunteers.

Solution

Our solution to these problems was an application called the Soundscape Noise Analysis Workbench. This solution involves a user-friendly user interface that hosts a machine learning model. The goal of this application is to allow any user to upload their audio files for analysis.

Overall, our solution will consist of:

- User-friendly web application.
- An ability to automatically classify different audio components in the inputted file using Neural Networks.
- Calculated acoustic indices (data statistics used by sound researchers) for each uploaded sound file.
- Visualizations of the analyzed audio components.
- Table of audio components and acoustic indice values.
- A way to export each models' classification and the calculated acoustic indice values.
- A standalone version of all the features of the web application for offline use in the field.

The solution ingests audio files. Researchers from Soundscapes2Landscapes will use audio files that they collect with the low-cost audio devices being used around Sonoma County, California. We are using a machine learning algorithm to automatically classify different types of sounds in these recordings. Our machine learning model can accomplish the task of identification in a fraction of the time than current implementations, classifying an audio file in under a few seconds. The machine learning model requires training to accurately classify audio components. We have trained the model on previously classified audio data. This previously classified audio data consists of many audio files that have been labelled with each category of sound that we are looking for. The sound categories will include birds, cars, rain, wind, and others. Collecting data to train the machine learning model came from multiple sources, including open source data and data from Soundscapes2Landscape's audio recorders. The results of this classification are visualized in a variety of ways. The visualizations include a labeled spectrogram, showing the classified components in the inputted audio, as well as a pie chart of the proportions of each sound category. These categories include geophony, biophony, anthrophony, and a no sound present category. Additionally a table of all the information collected from the analyzed file will be displayed. The solution will also be created as an application for offline use in the field. This application will provide users the ability to classify their audio without an internet connection.

Process Overview

In the beginning of development, our team decided to work within an Agile/Trello development cycle. Our team held weekly meetings to go over issues that surfaced along with tasks which needed to be completed for the coming week. The issues and bugs we encountered would be added to the weeks Github trello style project page. Finally we would assign the tasks to different team members, and were responsible for completing said task by our next team meeting. The main tool which helped our development cycle and version control was GitHub. Through development, our roles became apparent as what we were working on the most through each week. Steven was our team lead, but began to also take control of the overall process for the front-end development. Joshua worked thoroughly on the research and development of the machine learning models. Michael worked on the back-end API of the web application. Our roles were set, but this did not mean we couldn't help each other's sections.

Requirements

From developing our planned solution to the problem our clients presented us with, we have created specific requirements that our envisioned solution needs in order to properly create a working final product. Our team has determined that this project contains these key domain level requirements: users will be able to upload audio files, then analyze the uploaded files, then see the results of the analysis visualized in a timely manner, and then export all results.

From these key domain level user requirements, key functional requirements for the system were created. These requirements detail the specific features the product will provide our users when accessing the envisioned application. The application will ingest audio files, then use machine learning to classify the sound components in the uploaded file, then calculate acoustic indices, then display the results in multiple ways, then export these results, and finally an offline version of the application for fieldwork will be available. In addition to the specific functional requirements our product will need to solve our clients goal, specific non-functional performance requirements were needed for our product to succeed. The performance requirements include: uploading a one minute file should take less than 5 minutes, and should only take at most three seconds to complete a full analysis. As well as specific performance requirements our team realized one important environmental constraint featured in our planned system. Our clients work currently only consists of data from Sonoma County, California. Since the data only consists of Sonoma County data, it is not easy to guarantee a high accuracy from our machine learning model when data from other parts of the world is to be used with our application.

After discussing with our clients the problem they are facing, we worked to find an envisioned solution to solve their problem. A specific design was created in order to finalize how the final product will be built.

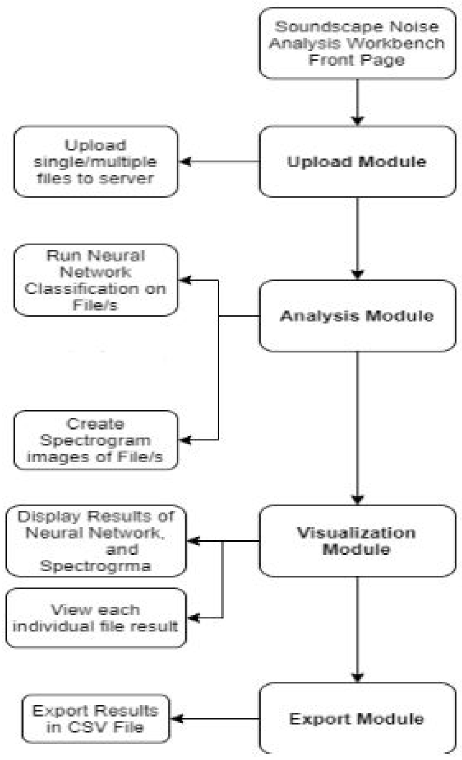
Implementation Overview

Through our requirements acquisition, our clients helped to reveal the major problems which we looked to solve. The main problems which arose through our acquisition consist of the following:

1. The clients currently use manual identification when analyzing their large storage of audio files, which has proven to be very time consuming.
2. The current program, Arbimon, is not easily accessible to the client's volunteers.
3. The current analysis does not run efficiently on an HPC.

The solution we have created for our clients is an application named the Soundscape Noise Analysis Workbench (SNAW). The SNAW consists of the following attributes:

Figure 1: Diagram of the Soundscape Noise Analysis Workbench System Components


Soundscape Noise Analysis Workbench Front Page
Upload Module
Upload single/multiple files to server
Analysis Module
Run Neural Network Classification on File/s
Create Spectrogram images of File/s
Visualization Module
Display Results of Neural Network, and Spectrogrma
View each individual file result
Export Module
Export Results in CSV File

- A user-friendly web application and an offline application.
- Utilizes a machine learning algorithm which can automatically classify specific audio components which are being searched for.
- The online web application returns clear visualizations of the analyzed audio file.

Our solution requires that the inputted audio files are in the WAV format, as the client's audio recording devices collect and store data in WAV format. Once the files are uploaded, the application will then use a machine learning algorithm to start the identification process. The usage of a machine learning algorithm to identify sounds within an audio file solves problem (1) stated above. We plan to create a simplistic design for the web application to ensure that it presents a user friendly experience while maintaining full functionality for a complete analysis, creating a simple design that solves problem (2) stated above. Lastly, the creation of an offline application will directly solve problem (3) listed above, as it will be able to use the same functionality as the web application without the visualizations. The offline application will be better for analyzing bulk files on an HPC.

The technologies we have chosen for the solution consist of the following: React, JavaScript, Flask API, and many Python libraries. Each of the technologies listed contribute and work together to create our product. React was used to create our web application and allow us to create a user-friendly front-end. The Flask API is used as the “glue” to connect our React front-end to our machine learning scripts. The Flask API will also be able to run the React front-end as a server which will receive API calls and present different pages through URL requests sent from the front-end. These technologies will be able to produce the tools we will need to finish the product according to how our clients expect it to be. To view more details on information for the specific Python and React libraries we have chosen to work with, please refer to the Technologies Appendix.

Architectural Overview

In the previous section, we discussed the implementation overview that our team has developed to produce our product. To understand the architecture of our system, we will discuss the high-level detail on how the Soundscape Noise Analysis Workbench will be built. Below are multiple diagrams of our system components [Figures 2,3,4].

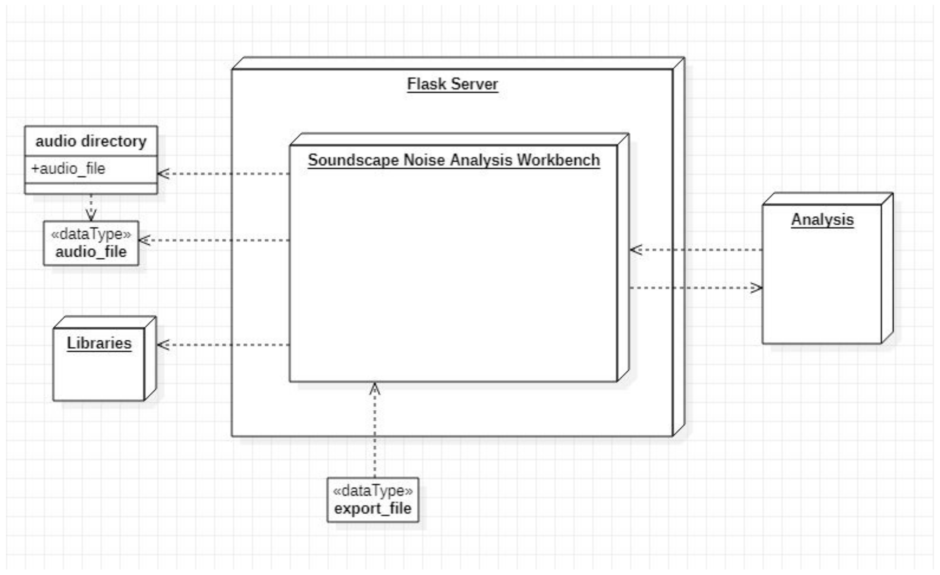


Figure 2: Diagram Overviewing the S.N.A.W Server Architecture

An outline of our Soundscape Noise Analysis Workbench server architecture is shown in Figure 2. A Flask server runs the SNAW module. The server takes in inputs of either a directory of audio files, or a single audio file. The server connects to the libraries needed by each component of SNAW. The server sends data and receives data from our analysis module. Finally the server sends an exported file to the user.

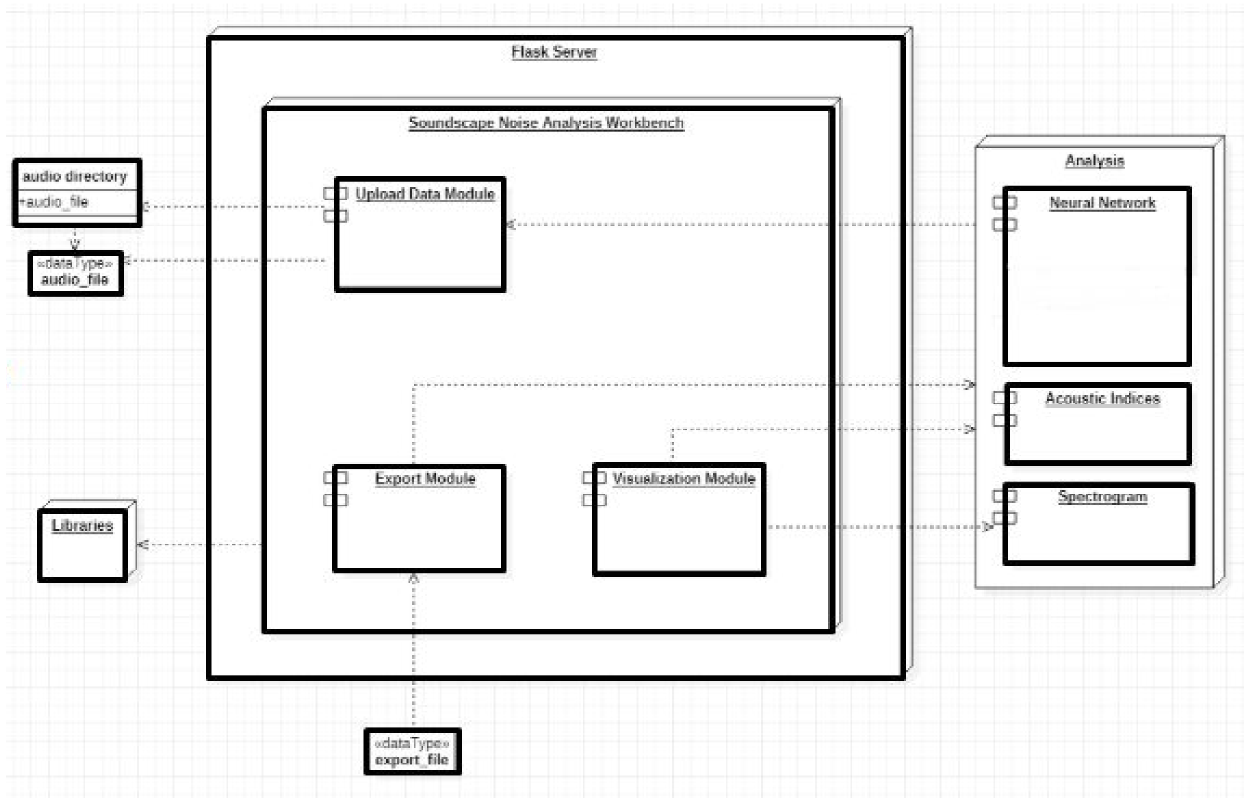


Figure 3: Diagram of the S.N.A.W. Architecture's Modules

An outline of the Soundscape Noise Analysis Workbench's architecture modules is shown in Figure 3. Taking in the inputted directory of audio files or a single audio file, the upload data module will handle the data and send the server locations of each file to the analysis module. The Analysis module uses the uploaded files and runs the Neural Network, Acoustic Indices, and Spectrogram Modules. These modules return the Data Structure as described in "Data Structure Description" to the Export Module and Visualization Module. The Export Module sends an export file to the user.

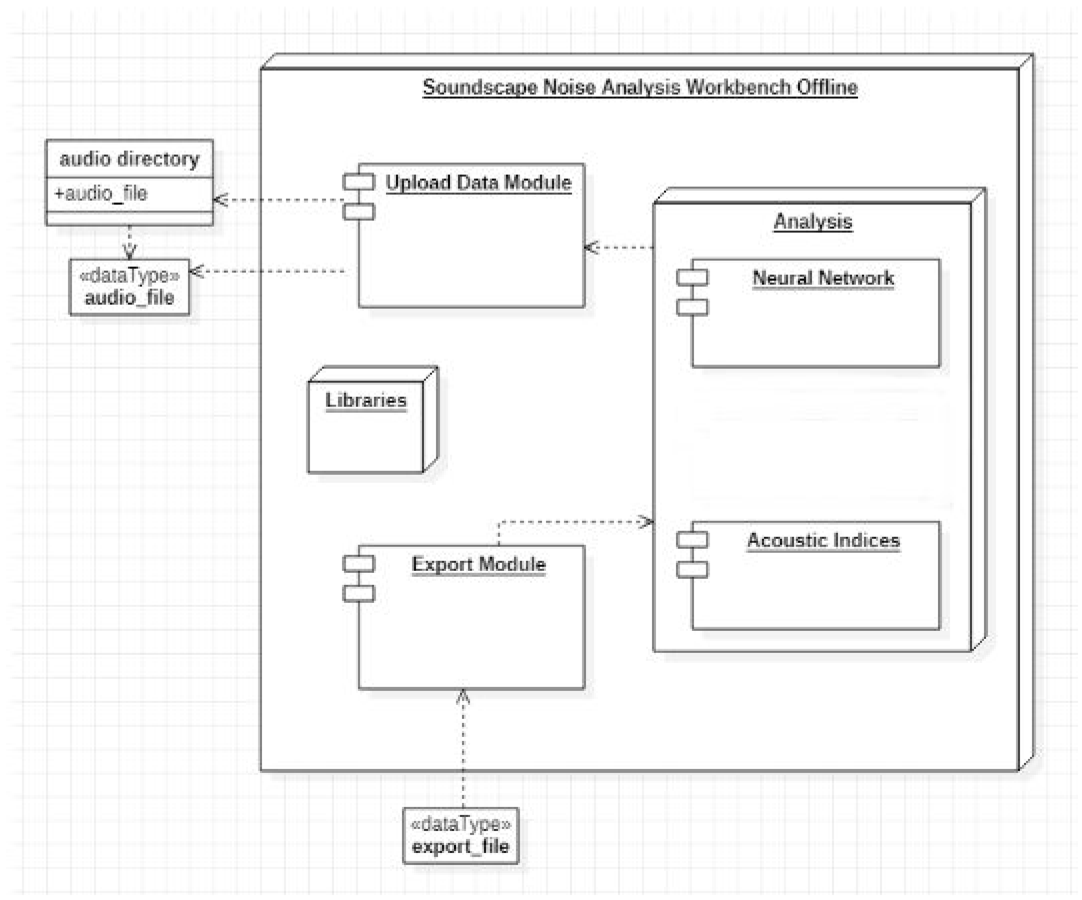


Figure 4: Diagram of the Standalone Offline SNAW Architecture

A diagram of the Standalone Offline SNAW architecture is shown in Figure 4. The component modules of the system work very similarly to the web application with a few differences. Firstly the workbench does not sit in a server but is instead a standalone python file. The libraries needed by each component are included in the file itself instead of being called by the server. Each module found in the analysis module is included in the offline file itself instead of being called by the server. Additionally the Spectrogram Module and Visualization Module are not included in the standalone application as the only output to the user is the exported file.

Our product, the Soundscape Noise Analysis Workbench, will be developed in Python and Javascript. Python will be utilized for the back-end of our system with the microframework Flask. Our front-end will be developed using React, a Javascript library. Our offline version of our application will run as a single python script. Below we will overview the key responsibilities and features of each component of our system, which include:

Upload Data Module

The Upload Data Module is utilized to ingest audio files that will be analysed. The application ingests files in WAV format and stores them for further analysis. The user will be able to choose to analyze a single file or multiple files. This will be done by selecting the file(s) in the web application.

Analysis Module

The Analysis Module is used to take the ingested audio files and run multiple types of classifications on them. This includes running the audio files through a Neural Network to identify sound components, using a neural network to identify sound components, and run acoustic indices calculations.

Export Module

The Export Module will allow a user to export the analyzed results. The user will then be able to keep a log of the results on their local machine. The results will be in the CSV files. There will be one CSV file for the Neural Network classification as well as the acoustic indices calculations.

Visualization Module

The Visualization Module is used to visualize the results in a user-friendly manner. The user will be able to get an intuitive visual of what sound components were present in the file, where they were found, as well as how big of a proportion of the audio file was identifiable sound components.

Standalone Offline Script Module

The Standalone Offline Script Module is used to have a standalone version of the Soundscape Noise Analysis Workbench. This will be in the form of a script that can be run through a terminal. Using a standalone version of the application will be useful for anyone looking to analyze audio without a connection to the internet. The user will need to provide a path to a directory of audio files to be analyzed.

These modules make up all of the functionality that SNAW provides. Now we will look into the communication mechanisms and information flows of our architecture. The web application connects the Upload Data Module, the Analysis Module, the Export Module, and the Visualization Module. The Upload Module will ingest the audio files and store them in a location that the Analysis Module will pull from. The Export Module will input the data from the Analysis Module and allow for the results to be downloaded. The Visualization Module also uses the Analysis Module’s output to create user-friendly visualizations for the end-user. The Standalone Offline Script Module is separated from all of the other modules. The control flow will be in a script that runs each classification offline.

With the communication mechanisms and information flows discussed, we will discuss the influences from our architectural style embodied by our architecture. For our React App, we have modularized the application into many components. This allows for updating the code base in the future and debugging the application much easier.

Module and Interface Descriptions

In the previous section, we discussed the high-level details on how the Soundscape Noise Analysis Workbench will be built. To understand the lower-level details of our system's architecture, we will explain each individual module in detail.

Upload Data Module

This module is used to ingest WAV files into the system. Files will be selected from a file chooser on the web application. An error will be shown if the user attempts to upload a file that is not accepted by the web application. The ingested files will be uploaded to a server to then be utilized by the Analysis Module. The Upload Data Module sits at the very beginning of our products architecture, as files are needed to move forward with the execution of the product. Below is a UML diagram of the Upload Data Module [Figure 5].

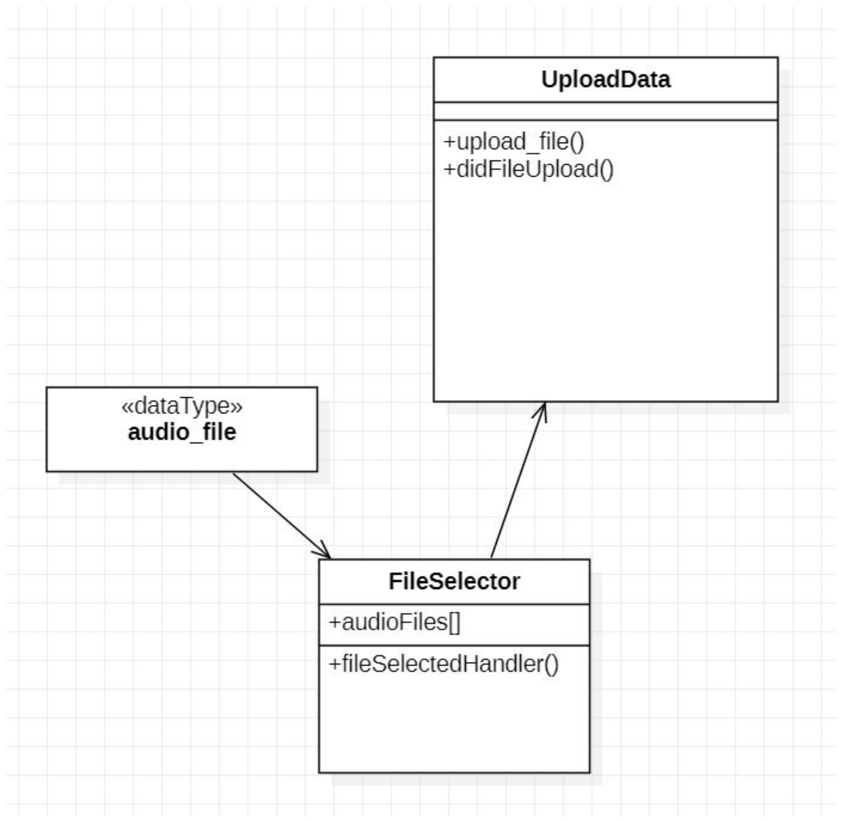


Figure 5: UML diagram of the Upload Data Module

Input = A WAV file or multiple WAV files (unless we choose to accept other popular file formats).
Output = An array of files uploaded to the server with a confirmation message to the end-user.

Analysis Module

This module will be used to run Neural Network classifications and acoustic indice classifications on the uploaded audio file(s). Once the Upload Module has completed, the user will be able to press the “Analyze Audio” button which will run 3 seperate classes and their methods to analyze the audio. The NeuralNetworkClassification classes will run our machine learning models on the uploaded audio file/s and return a JSON dictionary populated with the results from each of the classes. The AcousticIndices class will run many different methods to calculate the specific acoustic indices within the file, and then return a JSON dictionary populated with the result data. Upon completion of all three classes running successfully, the Analysis Module will then send a populated JSON dictionary to the Visualization Module to be processed and displayed on the front-end of the product. The results from the Analysis Module will also be sent forward to the Export Module. The Analysis Module is an important part for the overall product, as this is where the main portion of calculations on files is done for the product. Once completed, the product is then accessible through the Visualization and Export Modules. Below is a UML diagram of the Analysis Module [Figure 6].

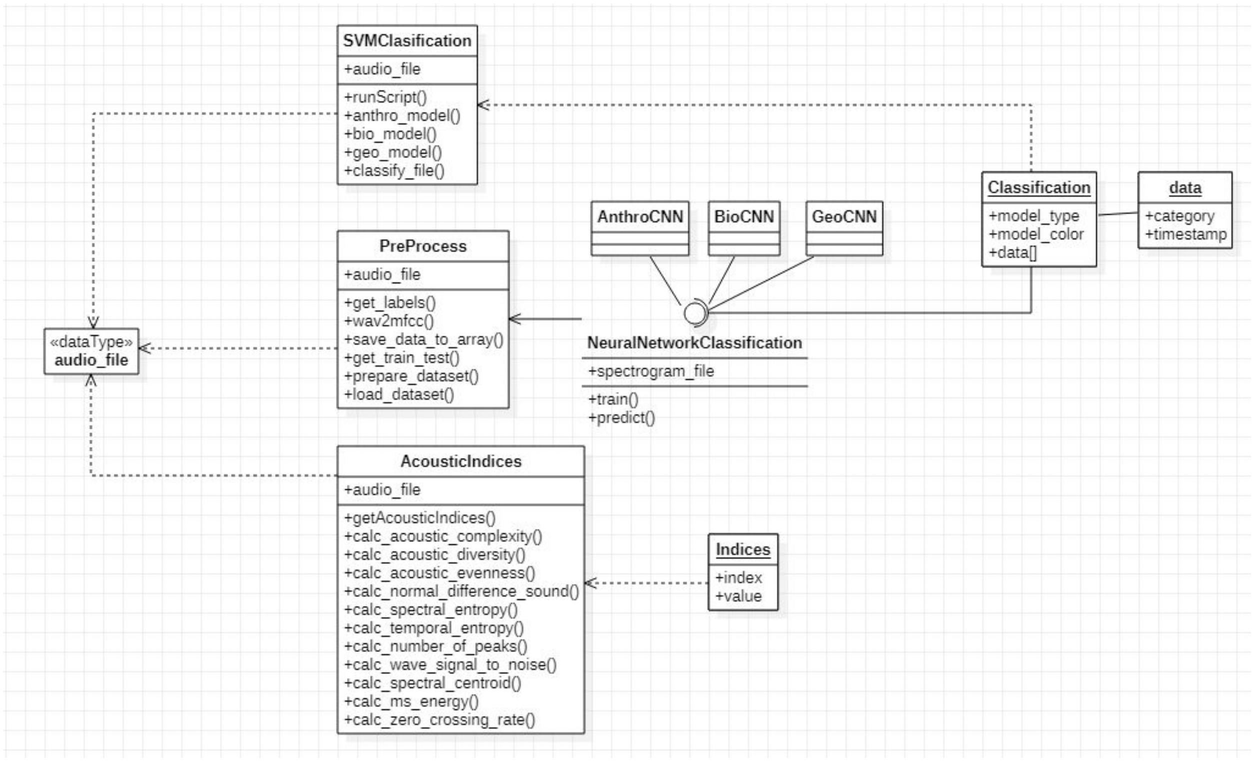


Figure 6: UML diagram of the Analysis Module

Input = Audio File(s).
Output = JSON Dictionaries from each classification containing the results.

The Analysis Module's output references a data structure known as a JSON Dictionary, which will be one of our main data structures organizing our resulting data. JSON dictionaries utilize Key-Value pairs, in which a unique "Key" (string, integer, etc) is stored and will be attached to specific data or "Value" that we choose. We pass a single JSON object to the front end. This overall object contains a JSON object for each uploaded file. Associated with each uploaded file is a JSON object with the results of each of the four different back-end analysis functions. The UML Diagram below showcases the design that we have decided on for our JSON dictionary [Figure 7].

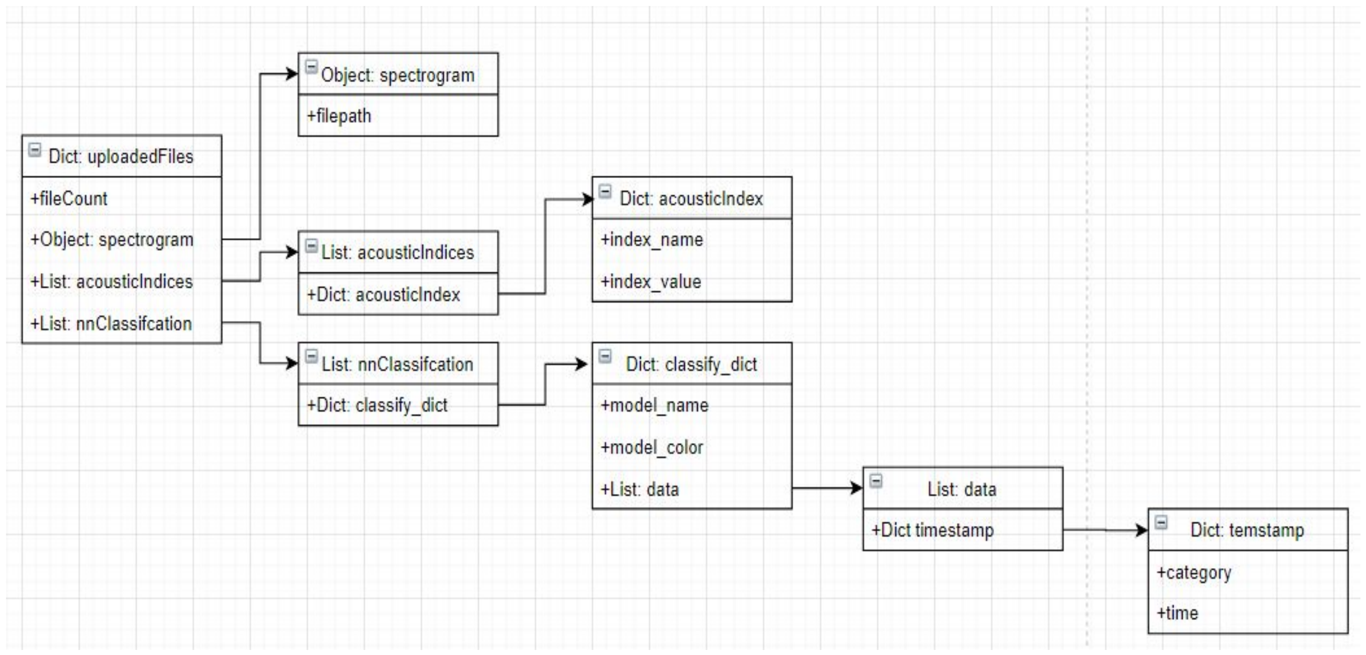


Figure 7: Diagram of the JSON Structure

Visualization Module

This module is used to visualize the data needed by users in a user-friendly way. It accepts the results generated by the neural network and acoustic indice classifications from the back end. The Visualization Module will generate a certain number of extension panels imported from material UI, depending on the number of files uploaded by users. We have the ReceiveAndCalculate sub-module, which receives the dictionary that is passed by the Analysis Module running neural network and acoustic indices calculation. The dictionary contains three categories: anthrophony, biophony and geophony. The visualization module also has another GenerateGraph sub-module, which will calculate the number seconds each file contains each of the three categories respectively. This data will be placed in pie charts, one for each category. They all include the charts imported from the rechart package. This sub-module also takes the spectrogram from the results folder and displays it in cardmedia. The pie chart is a scalable vector graph. All the data in the table are from our calculation of classification dictionary and acoustic index dictionary. Here is the UML diagram of the visualization module [Figure 8].

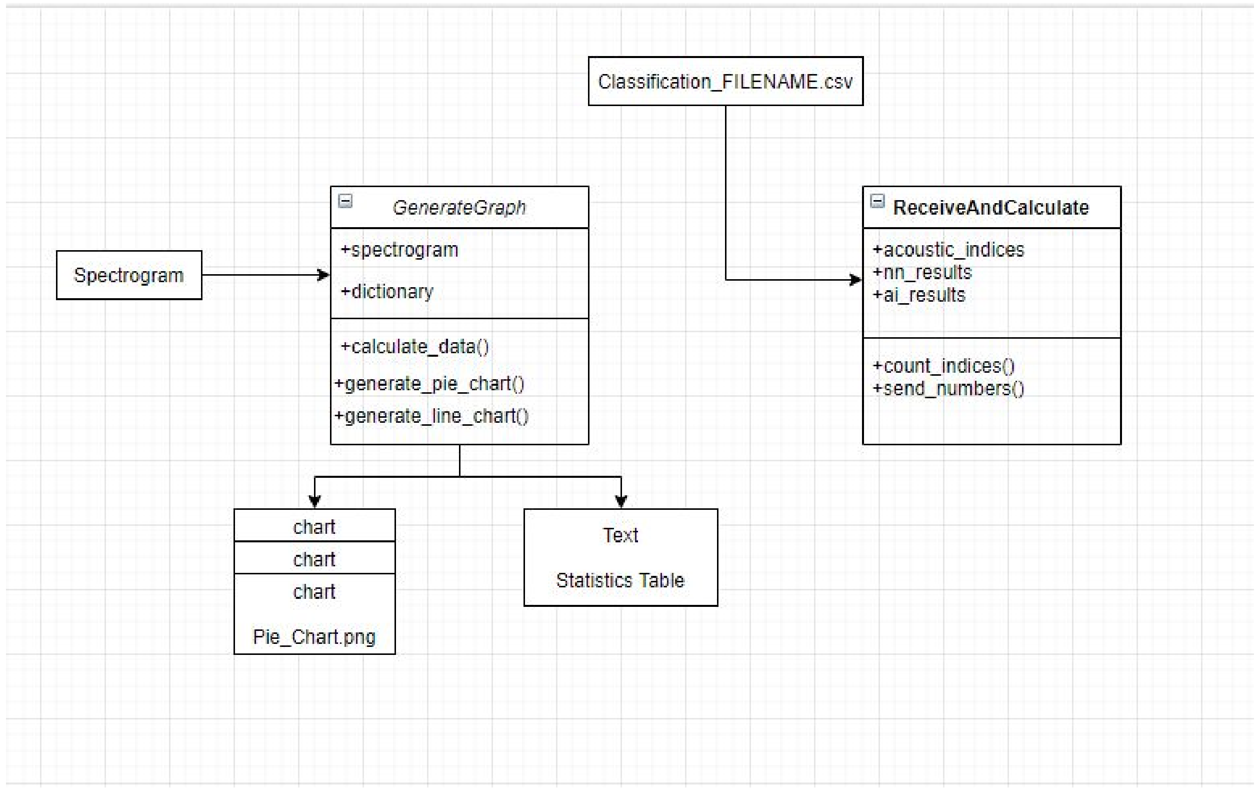


Figure 8: UML diagram of the Visualization Module

Input = Dictionary.
Output = Spectrogram, Charts, Table of values of each of the acoustic index calculations.

Export Module

This module is used to export a CSV file that has been populated with result data from the backend. The Export Module will receive a JSON dictionary which is passed through by the Analysis Module which runs the Neural Network and acoustic indice calculations. The Export Module will be run by the ExportResults class, which contains methods for the Neural Network Classification and Acoustic Indices calculations. Using the data received from the JSON dictionary, the CSV that is created will contain the three different neural network classifications listed under Biophony, Geophony and Anthrophony. The CSV will also include the acoustic indices that were calculated from the file. The Export Module is found at the end of our products architecture as its main objective is to allow the user to retrieve the results of the product after the Analysis Module has been completed. Below is a UML diagram of the Export Module [Figure 9].

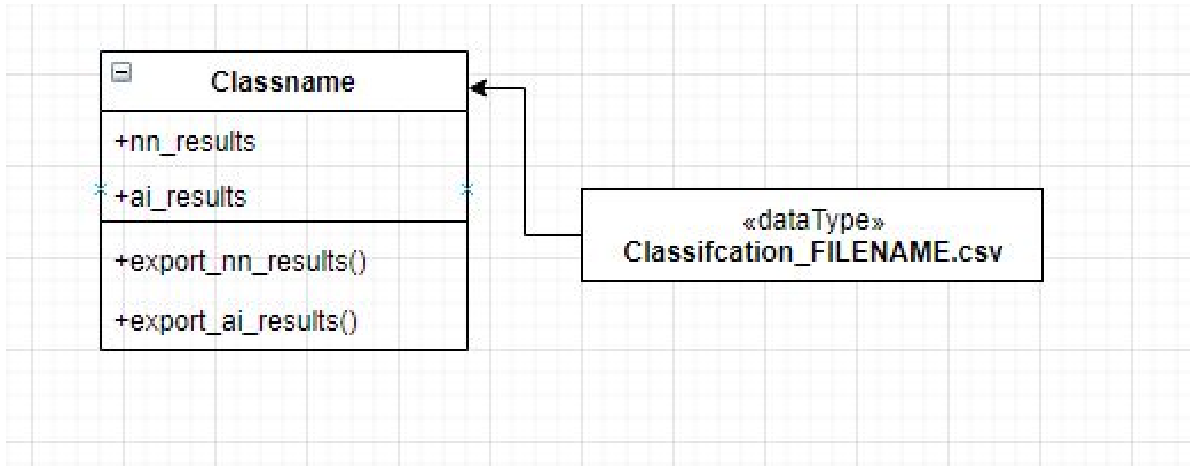


Figure 9: UML diagram of the Export Module

Input = JSON Dictionaries from the Neural Network Classification, and the Acoustic Indices calculations.
Output = Data populated CSV files for each of the classifications and calculations.

Standalone Offline Script Module

This module will be used to run the classifications and acoustic indice calculations on the inputted files through a command line interface (CLI). The classifications will include the Neural Network models. The models are pre trained by the team and are ready to classify audio components. This offline script will allow users to use this model without a connection to the internet. This offline script will also allow users to classify audio files through a high performance computing (HPC) cluster. Our clients are looking to classify audio files on Northern Arizona University's HPC cluster Monsoon. Below is a UML diagram of the Standalone Offline Script Module [Figure 10].

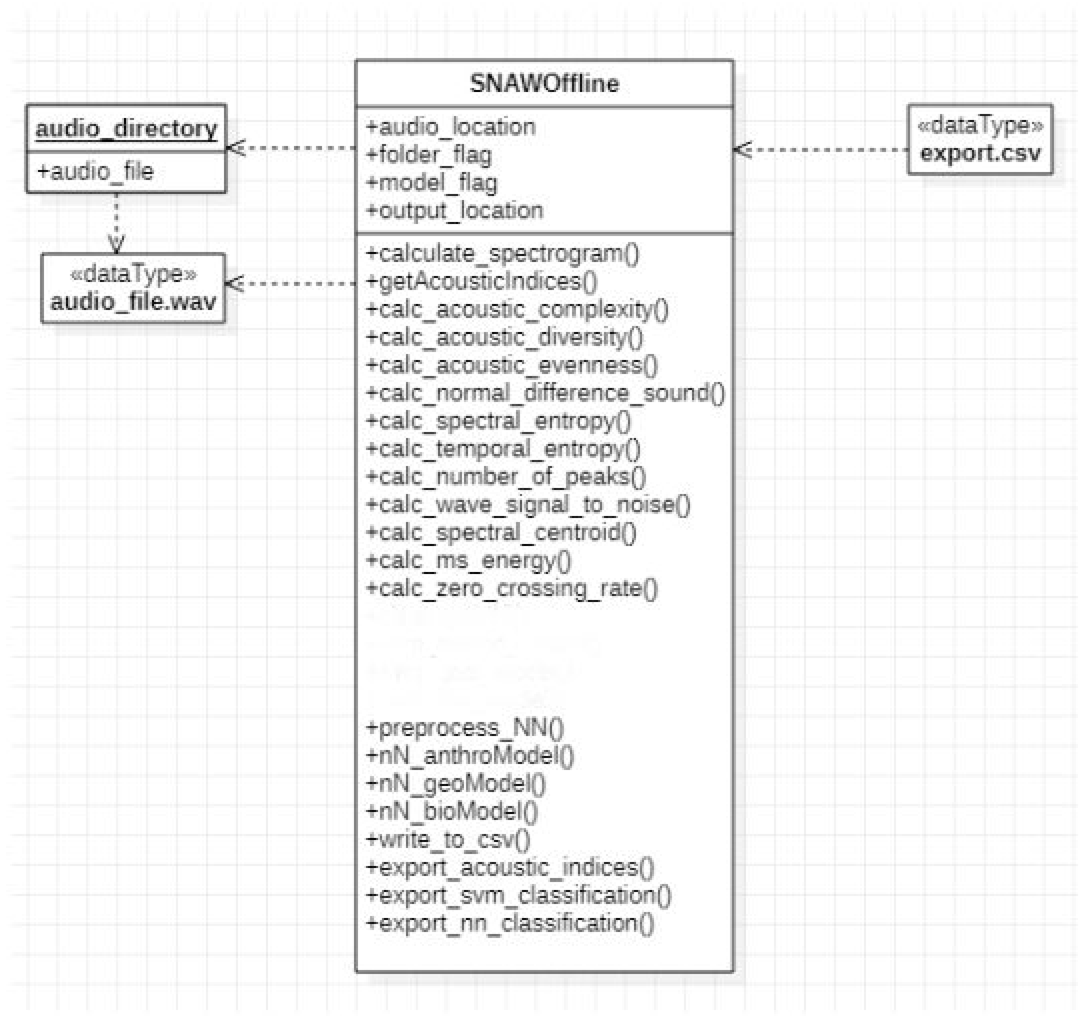


Figure 10: UML diagram of the Standalone Offline Script Module

Input = File path to a single or multiple WAV files to be analysed.
Output = The results of the Neural Network classification, and Acoustic Indices calculations as individual CSV files.

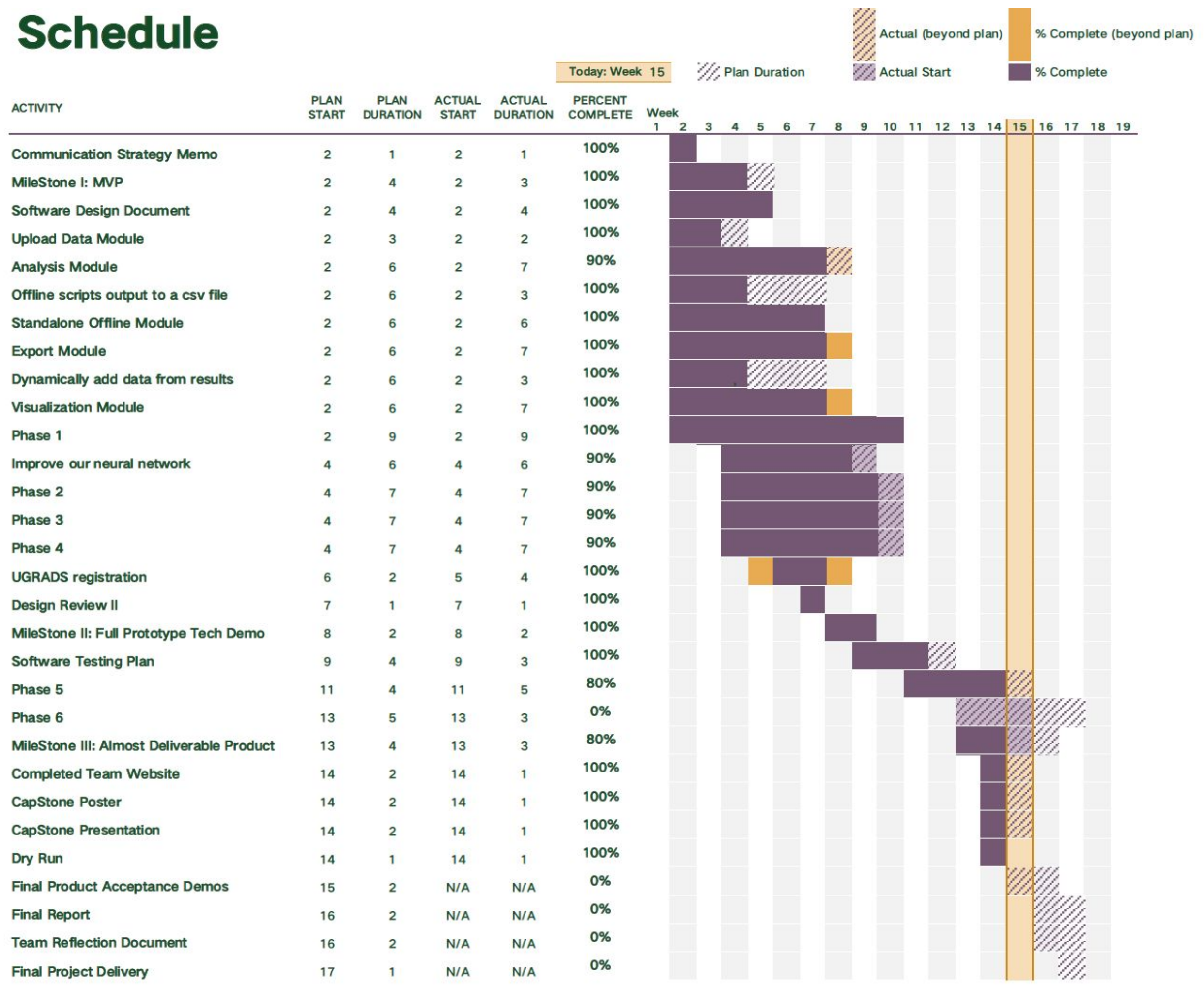
Testing

When designing a software system, many components tend to be developed independently with the intention to later be combined with other components to properly function. In order to ensure functionality of the entire system, a good measure is to test the system thoroughly. A great method for testing is known as Unit Testing, and is primarily used today in software design practices. Unit testing is the practice of testing each component individually. The components: upload, analyze, visualization, and export were isolated and tested separately from all other components. To execute unit testing for our software which consists of components written in both Python and Javascript; we used the unit testing frameworks unittest for Python, and Mocha for Javascript. From the unit testing completed we noticed unseen problems in the acoustic indices module. We added more error handling for broken files and files in the incorrect format so the website would not crash if uploaded.

To verify that our softwares individual, independent components work as intended when they are connected to each other, integration testing is needed. There are many different modules in our application that interact with each other and depend on other modules to function correctly and display the right result to the user. It is prudent to test and make sure that all these different functions inside our modules correctly work with the other functions present, and ensure that each module functions in the expected manner. We effectively managed our integration testing by creating testing ensuring that each of the inputs and outputs are handled in the correct way. In result of this testing we added error handling between components so that if one module failed, the other modules would continue to work as intended.

Usability testing is one of the testing methods that makes our products better suited for end-users. The process is to allow our clients or users to try our products and interact with the functions of our products in various situations. Our goal is to see if there are any imperfections in our products or any opportunities to improve the overall user experience. Usability testing allows users or clients to understand how our website and standalone application work. Our main usability testing was done with our Clients interacting with the web and standalone application. The changes that were made had to do with the visualization module. The results of the classification were at first not easily readable to our clients. An update adding overlays to the spectrogram visualizations was added to increase the readability of the CNN classification.

Project Timeline



ACTIVITY	PLAN START	PLAN DURATION	ACTUAL START	ACTUAL DURATION	PERCENT COMPLETE
Communication Strategy Memo	2	1	2	1	100%
MileStone I: MVP	2	4	2	3	100%
Software Design Document	2	4	2	4	100%
Upload Data Module	2	3	2	2	100%
Analysis Module	2	6	2	7	90%
Offline scripts output to a csv file	2	6	2	3	100%
Standalone Offline Module	2	6	2	6	100%
Export Module	2	6	2	7	100%
Dynamically add data from results	2	6	2	3	100%
Visualization Module	2	6	2	7	100%
Phase 1	2	9	2	9	100%
Improve our neural network	4	6	4	6	90%
Phase 2	4	7	4	7	90%
Phase 3	4	7	4	7	90%
Phase 4	4	7	4	7	90%
UGRADS registration	6	2	5	4	100%
Design Review II	7	1	7	1	100%
MileStone II: Full Prototype Tech Demo	8	2	8	2	100%
Software Testing Plan	9	4	9	3	100%
Phase 5	11	4	11	5	80%
Phase 6	13	5	13	3	0%
MileStone III: Almost Deliverable Product	13	4	13	3	80%
Completed Team Website	14	2	14	1	100%
CapStone Poster	14	2	14	1	100%
CapStone Presentation	14	2	14	1	100%
Dry Run	14	1	14	1	100%
Final Product Acceptance Demos	15	2	N/A	N/A	0%
Final Report	16	2	N/A	N/A	0%
Team Reflection Document	16	2	N/A	N/A	0%
Final Project Delivery	17	1	N/A	N/A	0%

As shown in the gantt chart above, this is our entire schedule. In the beginning of our development process, some of our major milestones to complete were the MVP and Software Design Document. The MVP and Software Design Document were important to us as we wanted a good start on our product's code base along with a great plan to be in place for how our product would be built. Once the MVP was finished, we continued to work on small features within each of the modules. We then came to the Phases 1-6, which entailed Development, training and improving our neural network, improving the front-end, implementing all modules together, and lastly testing and major bug fixes. During the phases, we hit our second major milestone, which was the prototype tech demo, which allowed us to show our overall progress on our product. Finally as we pass the phases, we hit the last milestone being our product almost ready for deliverance. After our milestones had all been passed, we just had to stride ahead and finish our presentations, capstone poster, and final capstone documents.

Future Work

One of the ongoing desired features that our team has been looking to achieve is a Neural Network with proper accuracy to ensure the user's are being given valuable results according to what may be present within their audio files. In order to achieve a Neural Network with high accuracy ratings, our team recommends more research be done to collect further understanding of the inner workings of a Neural Network along with proper audio data identification and collection to be used in training a Neural Network model. With proper resources, an accurate Neural Network model may be produced which would further the quality of the web application and standalone software.

Conclusion

The impact of human involvement on ecosystems has major consequences. More than one million plant and animal species are going extinct, with the majority happening within the last few decades (IPBES). There is an ever growing need to properly monitor the biodiversity in ecosystems, as well as the factors that impact biodiversity. Our clients Colin Quinn and Patrick Burns work with the science-based group Soundscapes2Landscapes to provide a more effective and efficient way to monitor biodiversity. They have tasked our team with building an automatic way to identify specific, individual sounds present in a soundscape recording.

The problem that we solved is the time-consuming manual identification process of audio components in recordings from various sites in Sonoma County, California. Our solution was to develop a user-friendly web application that hosts a machine learning model to automatically classify these audio components. This solution allows researchers, as well as volunteers, to efficiently classify the components of their recorded soundscape files. As a stretch goal, we implemented an offline field-work application for use on a laptop in order for researchers in the field to also use our application.

This document's goal is to outline the specifics of how our team developed our envisioned solution. Using the requirements developed in our Requirements Document, our team has developed a software workbench that accomplishes all the requirements our solution must meet in a user friendly and timely manner. Our team was confident that we can provide our clients with a user friendly solution that solves all aspects of the problem outlined in our introduction.

Glossary

Anthrophony: Sounds produced by humans or human creations such as tools, vehicles, or people talking.

Geophony: Sounds produced by landscapes on earth such as rain or wind.

Biophony: Sounds produced by animals.

Spectrogram: A visual representation of an audio file.

Acoustic Indice: A statistic that summarizes the distribution of acoustic energies in sound recordings.

CSV: Comma-separated values, commonly used with Excel Spreadsheets.

WAV: Waveform Audio file format.

Standalone Application: Software to be installed on a computer for use without an internet connection.

UML: Unified Modeling Language, used to create a standard way in visualizing the design of a software system.

Neural Network: A machine learning model designed to mimic the human brain for predictions.

Appendix A: Development Environment and Toolchain

- **Hardware:**
  This application was developed on Windows. There are no minimum hardware requirements for effective development of this software.

- **Toolchain:**
  Development environment was a python virtual environment
  We used JetBrains Webstorm and Visual Studio Code as IDEs.
  We used Node JS as the Javascript package manager for the web application.

  All needed python packages are located in the requirements.txt file in the Installation_Requirements repository on GitHub at:
  https://github.com/intelliChirp/Installation_Requirements

  All needed Javascript packages are easily attainable through Node JS. When setting up the web application locally, steps will be provided with NPM commands to gather all needed Javascript packages.

- **Setup:**
  Each GitHub repository at our GitHub Organization IntelliChirp at (https://github.com/intelliChirp) contains a README that defines a step by step guide to install everything needed.

  The User Manual located on IntelliChirp's website (https://www.cefns.nau.edu/capstone/projects/CS/2020/IntelliChirp-S20/documents.html) will contain in-depth step-by-step instructions for installing a Python Virtual Environment, installing Node JS, and all other setup instructions.

  Running Web Application Locally: https://github.com/intelliChirp/SNAW
  Standalone Application: https://github.com/intelliChirp/SNAW-Offline
  Neural Network Models: https://github.com/intelliChirp/Soundscape-Neural-Network

- **Production Cycle:**
  Web Application: Start by activating your Python virtual environment. After editing some code in the front-end of the web application, run the command 'npm run build' in the \snaw-frontend\ directory. This will build a new version of the application with your changes. To run the application, navigate to the \snaw-backend\ directory and run the

command ‘py api.py’. Once this finishes running, you will be given a URL to paste in the address bar of a browser to view the web application.

<u>Standalone Application</u>: After making changes to the snaw.py file, save the file. After saving the file, re-run the terminal command:
‘py snaw.py -i [AUDIO-DIRECTORY] -o [OUTPUT-DIRECTORY].
Replace [AUDIO-DIRECTORY] with the directory of the audio files you would like to analyze, and replace [OUTPUT-DIRECTORY] with the name of the directory that you would like the exported CSVs to go.