Monterey Bay Aquarium
Research Institute

Conversion of Julia Code Detecting Clicks in MARS Hydrophone Data to Python and Improving Machine Learning Classification of Sperm Whale Clicks

Tatjana E. Ellis, California Polytechnic University - San Luis Obispo

Mentor: Brent Jones

Summer 2020

Keywords: Sperm Whale Click Detection, Julia, Python, Code conversion, Machine Learning, CNN Image Classification

---

ABSTRACT

This project worked closely with MARS hydrophone data by modifying and working to improve existing code that automatically detects sperm whale clicks within that data. This consisted of two parts: first converting existing Julia code that processes and filters out any sounds from the hydrophone data into Python, and second working on a Machine Learning model to detect sperm whale clicks among those sounds. The details of both parts are described in this paper, as well as the process, implementation, and difficulties along the way.

---

1. INTRODUCTION

This project consisted of two parts: Converting existing code that processes raw hydrophone data and stores any recorded sounds as potential clicks from the Julia programming language into Python, and improving existing Machine Learning Classification that identifies which of these stored sounds are sperm whale clicks. This paper will highlight the conversion process from Julia to Python, including the

motivation to do so, and the difficulties I experienced and observations I made along the way. I will also highlight the results of our current Machine Learning Classification model, and discuss different methods we could try to improve it.

2. BACKGROUND

2.1 WHAT ARE CLICKS?

Clicks is a term used to describe acoustic pings used by marine mammals to locate prey and for other biological processes. Specifically, I am using the term to refer to Odontocete clicks, which refers to any whale of the suborder Odontoceti, including dolphins, killer whales, and sperm whales.[1] The waveform varies for each species, and can thus be identified based on general patterns.

Our focus will be on sperm whale clicks, as seen in **Figure 1**. One ping would be considered as one click, however sperm whales often communicate with a series of consecutive clicks in regular intervals, which are referred to as a sperm whale click train.

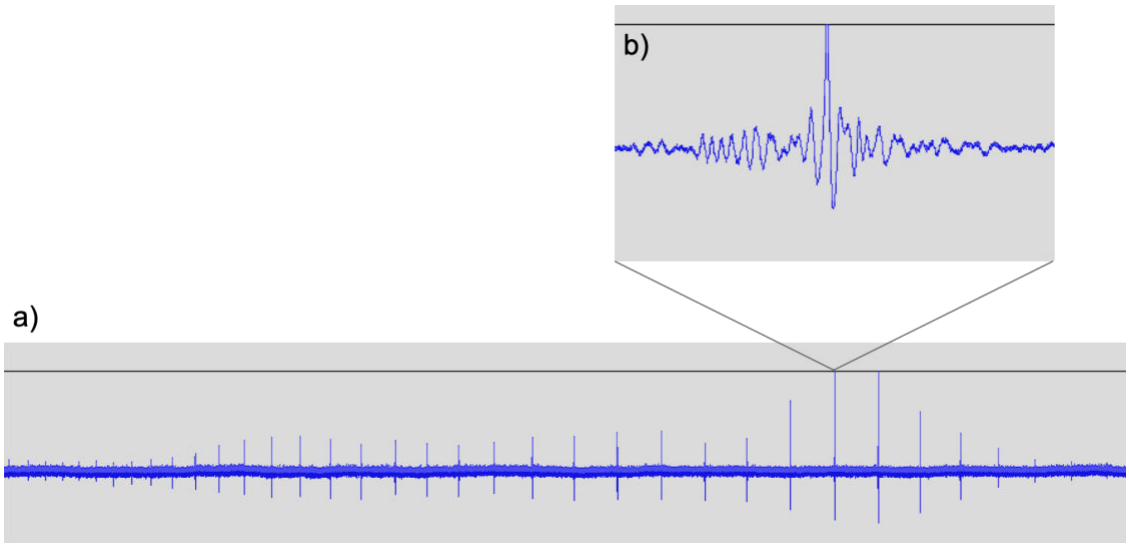

Figure 1. a) Sperm Whale Click Train with a frequency of about 1 click per second and b) zoom into one individual sperm whale click.

2.2 WHY CONVERT THE CODE FROM JULIA TO PYTHON?

Julia is a relatively new high-level language that was released in 2012. It was created by Dr. Jeff Bezanson, Stefan Karpinski, Dr. Viral B. Shah, and Prof. Alan Edelmann with speed and ease of use in mind, and has a lot of great potential, especially for numerical analysis and computational science.[2),3),4),5),6)] However, Julia's dedicated community, though growing, is still relatively small and the language is still relatively unknown, making it more difficult to share and distribute to other scientists and researchers throughout the world.[4),5),6)]

The nature of our code is meant to be open source, so anyone should be able to easily understand and use it. However, using a lesser known language like Julia makes sharing difficult, especially since it would require any potential user to first learn the syntax of Julia, as well as how to run it.

On the other hand, Python, which was released in 1990, has a well-established and large community, especially in science and research.[4),5),6),7),8)] Python also has several well-established libraries, such as the SciPy modules[9)], including NumPy, pandas, and Matplotlib, as well as the tensorflow library[10)] to perform the type of work we require, including large data analysis, parallel processing, and machine learning.[8),11)] This makes Python code very convenient to distribute to communities throughout the world.

3. MATERIALS AND METHODS

3.1 WHERE DOES THE DATA COME FROM?

The data used for our project comes entirely from the Monterey Accelerated Research System (MARS) observatory off the coast of MBARI (**Figure 2**).[12)] This is a Cabled Research Station with several instruments attached, which all record and send back data to MBARI real-time via the cable. Among these instruments is a hydrophone, which is an underwater microphone

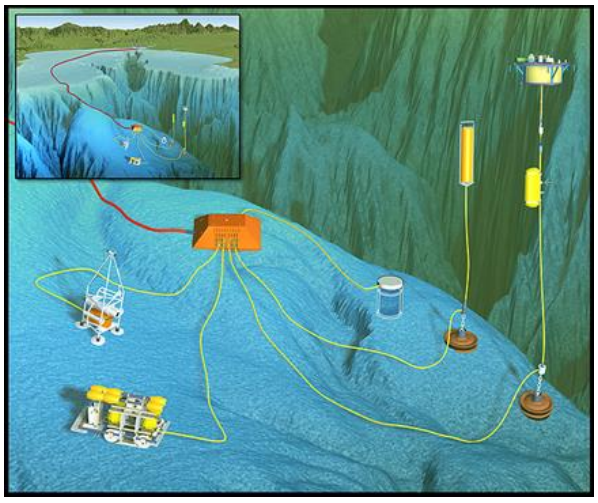

Figure 2. Illustration of the MARS Observatory off the coast of MBARI. Image Credit: MBARI[12)]

in simple terms. The hydrophone records audio data continuously and sends it back to MBARI as 10-minute long .wav files which are stored by year and month for easy access. However, this means we have terabytes of audio data that is impossible to manually sift through. For perspective, we currently have continuous audio data available from July 2015 to this day, and each 10-minute audio file is around 450,000 KB in size, with between 4,000 to 4,500 files stored per month. Because of this massive inventory, code was written to automate the detection process wanted for the audio data.

3.2 CONVERSION OF JULIA TO PYTHON

Atom[13] was used to test and run the original Julia code on the PALAU remote desktop (PALAU.shore.mbari.org) located at MBARI, allowing full access to all the files and data needed. To test and run specific Julia syntax and libraries, I used the default Julia terminal.

The converted Python code was ran using PyCharm[14] on the remote desktop at MBARI once a runnable version was created. However, I primarily used the default Linux terminal built into my Mac laptop to test and run Python functions, syntax, and files, allowing me to easily test equal functionality between both languages.

The conversion relied heavily on the official Julia documentation[15] and Python documentations[16],[17] accessible online, which were frequently referenced to ensure correct functionality and syntax when replacing each line of code.

3.3 CREATING AND RUNNING OUR MACHINE LEARNING MODELS

We are using supervised machine learning to train our models to detect sperm whale clicks, by providing labeled images of each recorded sound to the machine for it to determine patterns in each category. The labeled categories are either 0, meaning the image is not a sperm whale click, or 1, meaning the image is a sperm whale click. This is an instance of Binary Classification, which means the model is trained to classify each potential click into one of two categories. Our Machine Learning model was implemented in Python using the tensorflow keras sequential model[18], through which we

implemented a Convolutional Neural Network (CNN), a type of neural network that is primarily used for image classification and object recognition.[19),20)]

The models are run using PyCharm[14)] on the remote desktop at MBARI, and each model is stored after being created and compiled, so that it can be loaded easily if the model proves to be effective.

3.4 CREATING THE LABELED DATA

First, each potential click from the initial code (discussed in section 4.1) is taken and normalized to be on the same time and amplitude scale. This allows the machine learning model to fairly compare each sound by focusing on the waveform only, without having to account for variations in how far away or how loud the sound source was.

Next, a grey-scale image of each normalized sound is created, which is represented by a matrix of either 0 (black) or 255 (white) representing each pixel location of the image. This creates a white image with black data to provide a high contrast for the machine learning model to detect waveform patterns using image classification, as seen in **Figure 3**.

Then, a label is assigned manually to each image, preferably by a biologist who is most familiar with Odontocete click patterns, which are stored in a separate array consisting of only labels. Each label is assigned to each image by their index. A label of 0 is given to those that are not considered sperm whale clicks, and a label of 1 to those that are considered sperm whale clicks.

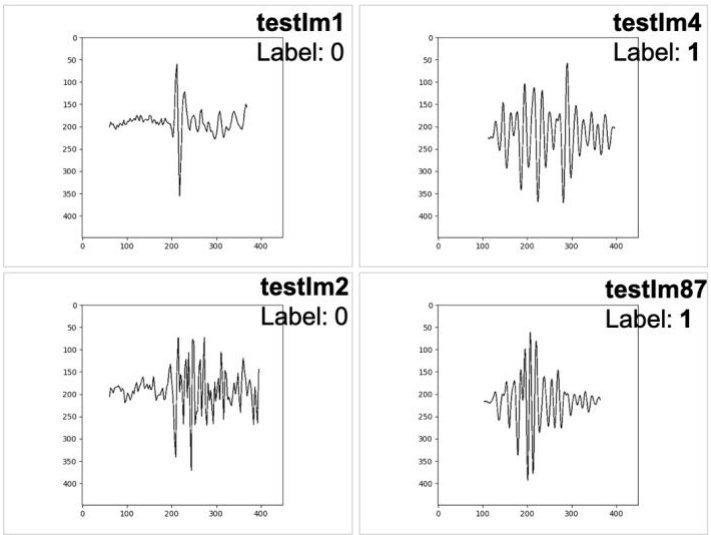

Figure 3. Examples of normalized data images with labels attached. 0 means it is not a sperm whale click, 1 means it is a sperm whale click. Y-axis is amplitude of sound, x-axis is time in seconds.

Lastly, the labeled data is split into a large training set, which is used to train the machine learning model to classify the sperm whale click images, and a smaller testing set, which is used to check the accuracy of the model's predictions once trained.

4. JULIA TO PYTHON CODE CONVERSION

4.1 WHAT DOES THE CODE DO?

The Julia program to be converted first goes through all the already processed audio files and creates a list of files that have not been processed yet. Next the code goes through this list of unprocessed files and filters out any sounds in each audio file using the Teager-Kaiser energy operator, which allows the machine to better distinguish background noise, such as the regular sound of the ocean, from any desired sound.[21),22),23)] Finally, it determines the time interval of each sound, or here referred to as a potential click, and stored it as its own entry in a dataframe, which is then saved as its own file for later use. This means each audio file has its own dataframe of potential clicks, which are stored one per audio file in a separate folder. In Julia, each dataframe is stored as a .jld2 file, which "saves and loads Julia data structures in a format comprising a subset of HDF5."[24)] An HDF5 format is desirable for our project as it is easily optimizable for quick access time and low storage size when dealing with large datasets.[25)]

One big feature of this code is its implementation of parallel processing, allowing for multiple files to be processed simultaneously on separate processors of the computer, instead of having to process one file at a time. This massively reduces the time needed to process the massive amounts of audio data that has been recorded, making the code much more useful and efficient to use.

However, one issue is that this portion of the project does not distinguish what type of sound was captured. This means that any sound, including ships, seal bombs, explosives, or any other marine organism's sounds, will be filtered out and stored as potential clicks. For this reason, we develop and use Machine Learning models to further classify what type of sound it is.

One machine learning model is created for each Odontocete species, and has so far been implemented for dolphins and sperm whales. These models are trained to classify what is a click of that species vs. what is not using supervised machine learning, which is discussed in more detail section 5 of this paper.

4.2 CONVERSION PROCESS AND DIFFICULTIES

ClickDetection.jl

Overall, five files were used in Julia to run the code, of which one could be completely removed in Python. **ClickDetection.jl** is a Julia module which allows a file to import and use functions from a different file by importing the module. In this case, the ClickDetection module was created so that the main function in detect_clicks.jl could call smaller functions and structures defined in detector.jl and io.jl. However, such a module is not required in Python, as Python allows any files to easily be imported and called from other files directly. As a result, ClickDetection.jl was not converted to Python, but instead the detector.py file and functions from io.jl are directly imported into the detect_clicks.py file.

list_files.jl

list_files is the file that runs through all the raw data and processed data folders to determine which audio files have already been processed, and then stores the names of the unprocessed audio files as a list. This file was very simple and easy to convert, so it did not bring any difficulties. Additionally, the new Python code has been modified from the original Julia code so that the time period over which files are listed can be specified by the user. This means that the user sets the start year and month and the end year and month, so that only the unprocessed files within that time frame will be listed. Originally, the function would go through every year and month of raw and processed data and check what has been processed every time it is called, which required much more time to run. With this additional functionality, list_files can be run much faster on selected data, making it much more practical for frequent use.

detector.jl

The **detector** file contains important structures and functions used by the main file, **detect_clicks**. This file contains the Click structure and the ClickPointer structure, as well as several small functions belonging to each structure, allowing for easy access to components within them. It also contains very important functions, such as the *teager* function, the *merge_intervals* function, and the *detect_clicks* function.

The *teager* function calculates the Teager-Kaiser energy operator (TKEO) for the audio data. The TKEO is generally used to aid a machine in distinguishing background noise from the desired sounds, which can be identified through large spikes or changes in energy.[21),22)] Energy in our case refers to "the concept of energy in the generation of acoustic waves"[23)] created by sperm whale clicks and other underwater sounds. Specifically, the TKEO has been shown very effective in detecting sperm whale clicks in a 2006 study conducted by two Researchers in Greece, V. Kandia and Y. Stylianou.[23)] For this reason, the TK energy operator is used in our program to detect high energy indicating a potential click is occurring.

The *merge_intervals* function simply merges together any overlapping intervals that had been detected as potential clicks, and combines them into one interval. This prevents any continuous sound from being split into and stored as multiple smaller sounds, which would cause many issues during further detection processes.

The *detect_clicks* function contains the code that detects potential clicks in a sampled signal by calling the *teager* function on the data and filtering out all indexes where the teager energy is above a given detection threshold. Then, for each index a closed interval is created and stored in an array of intervals, which are then given to the *merge_intervals* function that checks for any overlaps and merges those that do overlap to be stored as one sound. Once the intervals have been properly merged, each interval is stores as its own ClickPointer, and finally an array of all the ClickPointers is returned. This function is called from the main detect_clicks.py file.

One major difficulty encountered in this file is one line of unusual Julia syntax, which I have not been able to find proper documentation on and have not been able to run successfully in Julia. This line of code is a return statement in a smaller function called *samples*, which takes as input a ClickPointer. The original line of code is:

```
return Array(series(c)[left(c)*s..right(c)*s])
```

The ClickPointer Object is here denoted by c, and calling *series(c)* returns an array belonging to the ClickPointer structure. *left(c)* and *right(c)* simply return float values representing the left-most and right-most time indicator of a Click. What I have also deciphered is that the syntax [n..m] creates a closed Interval in Julia. The mystery here is that the array seems to be indexed by a closed interval, which itself is not a valid operation in any programming language. More confusingly, the left and right values are multiplied by s, which is a variable that has not been declared anywhere within this file. For these reasons, I have not been able to decipher what type of structure the original code returns, and have replaced it with a temporary placeholder by indexing into the desired array from *left(c)* to *right(c)*, both converted from floats to integers, in intervals of 1. This allows the code to run, however further testing is required to ensure the output is not too different from the original and still performs its desired task properly.

detect_clicks.jl

detect_clicks is the main file, which primarily calls functions from **detector**. In simple terms, this file contains the main portion of code which reads each audio file from the list created in list_files and passes each to a function called *process*_file. In this function, a highpass butterworth zero-pole gain filter is created and applied in the forward and reverse directions on the audio data, and then the teager and detect_clicks functions from the detector file are called on the data to find each potential click. It then creates a dataframe of all potential clicks within each audio file and saves them to an output folder. **detect_clicks** is also where parallel processing in implemented, so that the main function can be called on multiple audio files simultaneously by having them run parallel on separate processors of the computer it is running on.

This portion of the code was a bit more difficult to convert. First, the syntax to run parallel processing is very different in Julia than Python. In Julia, modules and several functions must be declared in an @everywhere section to make them accessible by and run them on all available processors, and the desired number of processes to use are declared and started before the code runs using addprocs(n).[26] This starts a total of n+1 processes to use. Then pmap is used to call the *process_file* function, which maps the function to all the audio files wanted and runs multiple at the same time. Parallel processing was easily implemented in Python using the multiprocessing (mp) library, by first creating a pool by calling mp.Pool(), and finally calling pool.map() on the *process_file* function and the list of files to be processed. The functionality is essentially the same as pmap in Julia, allowing for easy conversion. The difficulty here was determining equal functionality of the @everywhere sections and addprocs function call, and if there even exists a replacement in Python. However, my ultimate decision was to remove those parts, as it did not seem to have any obvious replacement or use in the Python language. Therefore, the converted code simply relies on the multiprocessing pool and map functions to implement parallel processing, which seems to work well during testing so far. Additionally, the Pool function has an option to declare how many processes should be used by declaring mp.Pool(processes=n), which could replace the addprocs functionality from Julia.

Another difficulty was found when creating and applying the highpass butterworth filter to the audio data. Python offers the same functions for creating and applying the filters as Julia in a library called signal, allowing for an equal filter to be created easily. However, the equivalent filtfilt function, which applies the filter to the data in the forward and reverse directions, has some differences when dealing with zero-pole gain filters. The filtfilt function in Python is primarily structured to take a polynomial filter as input, whereas the Julia filtfilt easily allowed a zero-pole gain (zpk) filter to be applied. Although it seems possible to apply a zpk filter using filtfilt in Python as well, it requires more research and thought to ensure the variables given to the function are correct. Because of this difference, I am temporarily using a polynomial filter as a placeholder until additional calculations can be made to properly implement the zpk filter.

The final issue that needs to be addressed is the dataframe storage type. As mentioned previously, Julia uses the .jld2 file type, which is a subset of HDF5[24]. HDF5 is a desirable format that offers optimizable storage and access time[25], and therefore the storage format will be converted from .jld2, a subset of HDF5, to HDF5 directly using the .h5 extension. This change was easy to implement, however the storage size for the .h5 output files I have already created are very large, having a size of 1,041KB each, while the original .jld2 output files are only 10KB each. This could create a big storage problem when dealing with the massive amounts of data we are processing, and needs to be reduced before full implementation. However, this change in size is more likely a result of the placeholders and change in data structures rather than the file type itself, so further testing and debugging is required to determine the cause and to fix any changes that resulted in this.

Besides these difficulties that were encountered, the conversion was successful, and a change in functionality was implemented as well. Originally, the detect_clicks and list_files programs needed to be run entirely separately, so one change that I implemented is that detect_clicks now automatically calls list_files for the time interval specified. This means that only one function needs to be run now, which both creates the list of unprocessed files for a given time interval automatically and then processes only those files. This can reduce the time needed to run the program, as well as provide more flexibility to the user who can chose what files to process. Currently, this has been implemented to call list_files for a file only if it does not already exist for the desired time interval, which cuts down time in case the file has already been created. However, further consideration must be put into this, since the listed files need to be updated once additional audio files are processed. Under the current circumstances, an update will only occur if the file list for that time interval is manually deleted, so changes can be implemented to automate the update process as well in future work.

io.jl

The last small file is **io.jl**, which contains the as_dataframe function. This function originally takes as input a vector of potential clicks, and can be called with two additional optional variables, being the start time of the first potential click and the name of the audio file they belong to. This function simplifies the process of creating a dataframe of the potential clicks found in each file, by automatically formatting the data as desired when called. I have not converted this file to Python yet, as I directly formatted the dataframe using the pandas library when called in detect_clicks.py. However, upon closer inspection, this function is very useful for simplifying that formatting process, so I plan to convert and implement it in my future work.

4.3 WHAT NEEDS TO BE DONE

Besides io.jl, which was decided late to be converted for additional functionality, and the two placeholders described above, which are the filter type in detect_clicks.py and the odd line of code in detector.py, all of the files have been fully converted and the Python program is able to run without error. However, these two placeholders seem to significantly change the output of the function, and must be addressed before full implementation of the Python code.

Additionally, the code needs to be tested more thoroughly to ensure equal output to the original Julia code, and more improvements can be made to optimize the code in terms of speed, such as by implementing more NumPy vectorization in place of loops, and storage as well.

Once the Python program is ready for implementation, the final change to fully transition to the Python program is to modify list_files.py to detect the new file type as the determinant for which files have been processed, so it can replace the .jld2 files. However, the original .jld2 files will be kept, as they will provide useful references to compare the new output once generated.

5. SPERM WHALE CLICK DETECTION USING MACHINE LEARNING

5.1 HOW DOES IT WORK?

As mentioned previously, Machine Learning has already been implemented for dolphins and sperm whale clicks, however my project is only concerned with improving the sperm whale click detector, as it currently requires more improvement than the dolphin detector. The reason for this difference is discussed in section 5.3.

This sperm whale click detector is implemented using a supervised CNN model, by creating black-on-white images of each potential click with normalized data, as described in the methods section 3.3. Using the tensorflow keras library, we create a new sequential model and add several layers to it, before compiling the model and finally training it using the labeled training set on 2 epochs. Given the data size, 2 epochs were determined as the best option, since any increase in epochs quickly resulted in overfitting, thus reducing the accuracy of the model predictions. Finally, we use the model to predict the testing set, while reporting the model accuracy, and then store the model so that it can easily be loaded and used for further testing without having to be recompiled and trained.

5.2 INITIAL RESULTS

Although the model's predictions are not extremely confident, there is still a clear distinction between what is and is not a sperm whale click, as seen in **Figure 4**. Here, each dot is the predicted value of the machine learning model, and the color represents the actual label of that image. This makes the model somewhat successful, as we can set a distinct threshold separating what is and is not a sperm whale click. However, the model's accuracy can definitely be improved, and will be addressed in future work. An increase in accuracy would make the model more reliable and allow it to more confidently predict unlabeled data for future use.

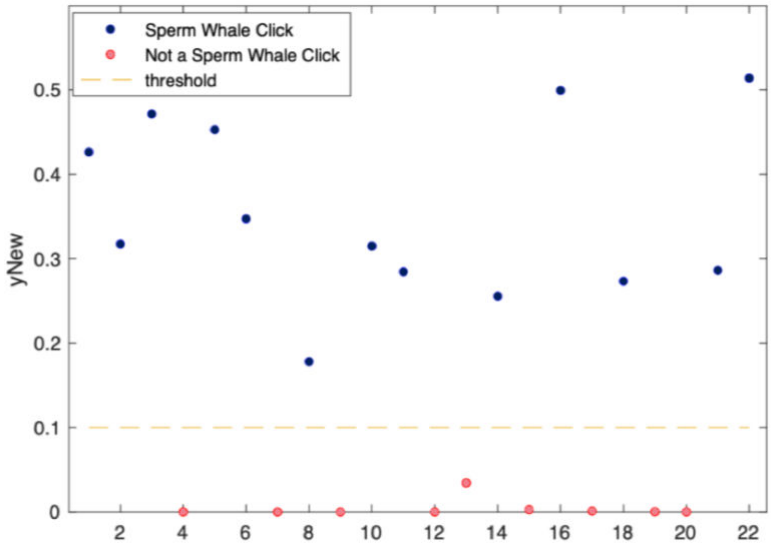

Figure 4. Initial results of sperm whale click detector using image detection. Y-axis is the model's predictions for each image, x-axis indicates image number, color indicates the correct label of each image. Distinguishing threshold placed at 0.1.

5.3 DIFFICULTIES

The main issue reducing the accuracy of our sperm whale click detection model is the deficiency of labeled sperm whale clicks, meaning we do not have enough identified sperm whale clicks to train the machine learning model as accurately as we would like. Looking at our training and testing sets, the training set has 1000 total images, of which only 10%, or 100 images, are sperm whale clicks. Likewise, our testing set has 500 total images, of which only 27 are sperm whale clicks. In contrast, our dolphin click detector, which works relatively well, has abundant data available to train the model.

Multiple factors resulted in this deficiency. Most notably, sperm whale clicks are much less abundant than dolphin clicks, especially in the location where the sound is being captured. This makes it much more time consuming and difficult to identify sperm whale clicks manually in order to build a reliable training set. Creating such a set requires a scientist, preferably a biologist, to go through each image of potential clicks and manually identify and label them as sperm whale clicks or not. However, the lower occurrence rate of sperm whale clicks makes it even more difficult and time consuming to manually undergo this process than for dolphins, since many more images must be viewed to find a sufficient amount of sperm whale clicks for training purposes.

6. FUTURE WORK

For the following year I will be continuing to work on this project with Brent Jones through a collaboration with MBARI. During this time, I plan to further improve the converted Python code, specifically focusing on optimizing speed, storage, and output accuracy. Once it is optimized, I will work to fully implement the code so it can be used in place of the Julia code and freely accessed by whoever may need it.

I also plan to further improve the current image-based machine learning model, and would like to investigate other machine learning methods as well. For example, I would like to train a model on the normalized data instead of images of the data, or try using natural language processing to detect distinct sound patterns, and investigate if these provide higher accuracy in the model's predictions.

Besides this, a great way to ensure increased accuracy no matter what model is used would be to have someone, preferably a biologist, manually go through potential clicks and label them accordingly to provide a larger labeled dataset to train the model. The only negative aspect is that this would be very time consuming. However, even if a larger labeled dataset is eventually created, it would be beneficial to test different machine learning models to determine the effectiveness of different models for Odontocete click detection.

ACKNOWLEDGEMENTS

I would like to acknowledge my advisor, Brent Jones, for allowing me to work on this project and for providing valuable help and information throughout the process. I am also very grateful to MBARI and the MBARI staff for providing me access to necessary data and networks needed to complete this project, as well as for planning and organizing the virtual summer internship despite the current pandemic. I also acknowledge the David and Lucile Packard foundation, as well as the Dean and Helen Witter Family Fund and Rentschler Family Fund in memory of former MBARI board member Frank Roberts (1920-2019) for funding this internship, providing all the interns amazing experience working in marine research. I am very grateful to all participants.

References:

1) Toothed whale | suborder of mammals. *Encyclopaedia Britannica* (Updated 2019). https://www.britannica.com/animal/toothed-whale
2) Julia (2020). https://julialang.org
3) Bezanson, J., Karpinski, S., Shah, V.B., Edelman, A. (2012). Why We Created Julia. https://julialang.org/blog/2012/02/why-we-created-julia/
4) Yegulalp, S. (2020). Julia vs. Python: Which is best for data science? https://www.infoworld.com/article/3241107/julia-vs-python-which-is-best-for-data-science.html
5) Sinha, D. (2019). Julia vs Python: which programming language to choose? https://www.techaheadcorp.com/blog/julia-vs-python/
6) Taylor, K. (2020). Python vs. Julia: What's the Difference Between the Two? https://www.hitechnectar.com/blogs/python-vs-julia/
7) Python (2020). https://www.python.org
8) Browne-Anderson, H. (2017). The Case For Python in Scientific Computing. https://www.datacamp.com/community/blog/python-scientific-computing-case
9) SciPy (2020). https://www.scipy.org
10) TensorFlow (2020). https://www.tensorflow.org
11) Geer, Z. (2019). The Best Python Libraries for Data Science and Machine Learning. https://dzone.com/articles/the-best-python-libraries-for-data-science-and-mac
12) Monterey Accelerated Research System (MARS) Cabled Observatory. *MBARI* (2020). https://www.mbari.org/at-sea/cabled-observatory/
13) Atom (2020). https://atom.io
14) PyCharm (2020). https://www.jetbrains.com/pycharm/
15) Julia 1.5 Documentation (2020). https://docs.julialang.org/en/v1/
16) Python 3.8.5 documentation (2020). https://docs.python.org/3/
17) Numpy and Scipy Documentation (2020). https://docs.scipy.org/doc/
18) The Sequential model (2020). www.tensorflow.org/guide/keras/sequential_model
19) Bansari, S. (2019). Introduction to how CNNs Work. https://medium.com/datadriveninvestor/introduction-to-how-cnns-work-77e0e4cde99b

20) Saha, S. (2018). A Comprehensive Guide to Convolutional Neural Networks – the ELI5 way. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53
21) Mike X Cohen (2018). Denoising EMG signals via TKEO (Teager-Kaiser energy operator). https://www.youtube.com/watch?v=_I5kYTcjwfI
22) Kvedalen, E. (2003). Signal processing using the Teager Energy Operator and other nonlinear operators. *Candidatus Scientiarum Thesis, University of Oslo, Norway, Department of Informatics*. http://folk.uio.no/eivindkv/ek-thesis-2003-05-12-final-2.pdf
23) Kandia, V., Stylianou, Y. (2006). Detection of sperm whale clicks based on the Teager-Kaiser energy operator. *Applied Acoustics,* 67 (2006): 1144-1163. https://doi.org/10.1016/j.apacoust.2006.05.007
24) JuliaIO/JLD2.jl (2020). *GitHub*. https://github.com/JuliaIO/JLD2.jl
25) HDF5 (2020). *The HDF Group*. https://portal.hdfgroup.org/display/HDF5/HDF5
26) Julia 1.5 Documentation (2020). Multi-processing and Distributed Computing. https://docs.julialang.org/en/v1/manual/distributed-computing/