Automatic Detection for Acoustic Monitering of Wild Animals

Jonathan Gomes Selman
Stanford
jgs8@stanford.edu

Nikita Demir
Stanford
ndemir@stanford.edu

Abstract

We achieve, to our knowledge, state of the art results for detecting elephant calls from acoustic data on two different tasks of interest to researchers: 1) A segmentation task where we predict for each time step input whether there is an elephant call present and 2) A trigger word detection task where we predict the completion of an elephant call. The latter is of particular interest because of its potential to allow researchers to run in real time computationally efficient models on low-cost devices. We used data collected by passive acoustic monitoring (PAM) systems in the African jungle provided generously by the Cornell Lab of Ornithology.

1. Introduction

Over the course of history, humans have dramatically shaped and altered the environment in which they live. At an accelerating rate, human influence has taken a toll on the environment and the species it supports. Poaching, illegal logging, infrastructure development, and other human activities present great threats to wildlife and demand increased conservation efforts. In order to properly allocate resources and shape conservation, scientists must accurately model endangered species populations, migratory behaviors, and daily activities. For species that have vast territories or exist in remote areas not easily accessible by humans, monitoring tasks become much more difficult. One promising solution is Passive Acoustic Monitoring (PAM), which involves placing autonomous recording devices throughout a habitat to passively record the vocalizations of species. More generally, acoustic monitoring serves as a cheaper and in many situations more practical data collection mechanism as its range is often considerably larger than direct line of site. However, in many cases, extracting meaningful semantics from large scale audio sources presents unique non-trivial challenges that, to be effective, require automation.

In this paper we focus our attention primarily on the African forest elephant. The African forest elephant represents a keystone species for the African rain forest, as they are responsible for the spreading and dispersal of many different seed species [source]. Tragically, however, it is deeply threatened by human behaviors, such as poaching and deforestation, which have led to dramatic deceases in the population – over 60% in the last 100 years [source]. Through the Cornell Lab of Ornithology, extensive acoustic data has been collected with the goal of better understanding population distributions, behaviors, and threats to survival.

In our work we examine different methods in supervised deep learning to help automatic data analysis in PAM. With the goal of real-time acoustic analysis for both population modeling and threat detection, we pose the problem as a per time step classification. Given raw audio data, we frame the task of elephant call detection in two separate ways: 1) predict for each time-step whether an elephant call is present, similar to a classical segmentation task; 2) treat an elephant call as a trigger word in a "trigger-word" detection task where we predict the completion of an elephant call, similar to the way that the Amazon Echo is awoken by the word "Alexa". Although audio manipulation is traditionally thought of as a one dimensional problem, we leverage the two dimensional information captured in spectograms to re-frame this problem as a visual recognition problem for characteristic patterns in animal vocalizations.

Figure 1. African forest elephants gathering in a clearing.

2. Related Work

The field of bio-acoustics has placed great emphasis on the automatic detection of different species based on auditory vocalizations. Because sound waves attenuate less in water (i.e. sound can travel much farther) a great deal of work has been put into PAM of different marine species as well as birds, due to their distinct "songs"[6][7][4]. Previous work has ranged from "hand crafted" feature learning [2], to representation of rich distributions through Gaussian mixture models [3], to more recent applications of deep models [4]. A very common feature in these past works is the transformation of 1d raw audio into the more expressive 2 dimensional equivalent [4][10]. Moreover, with this feature representation, success has been shown employing different deep learning architectures directly on these spectogram features. For example, work by Leon et al. used convolutions architectures to study the calls of blue and fin whales[4] and Grill et al. utilize convolutional recurrent nets for classification of bird songs [7].

Although a lot of research has been done into marine mammals and bird calls, there has been little research into other large mammals such as elephants. One specific paper that we looked to improve directly upon deals with elephant call detection [1]. Motivated by their success in elephant call segmentation with recurrent networks applied directly to spectogram data we looked to build upon their methods for segmentation as well as data and error analyses. While we leverage many of their general modeling approaches, we explore and build upon the complexity of their models and experiment with more expressive data pre-processing/ normalization. Moreover, we look to their work as impetus for framing the problem in a different light through trigger-word detection, an approach primarily inspired by the coursera project notebook for Andrew Ng's CS230 course [14].

Taking a broader perspective, we see that a great deal of research has been done into the general area of sound event detection. Different research has ranged from speech detection, to animal sound detection, to the detection of various noises that exist within the home [12][13][11]. The DCASE (Detection and Classification of Acoustic Scenes and Events) challenge is a research driven challenge that looks to push the boundaries of acoustic detection and classification through different challenge tasks[15]. We draw particular inspiration from the success of different models in the "Detection of rare sound events" challenge, as it mirrors in many ways the goal of our particular problem for both segmentation and trigger-word detection[16]. Analyzing the top scoring models we see a common trend of combining convolutional and recurrent structure into the modeling of the problem [10][11][12]. In this work we chose to explore closely the work of the winning team that specifically uses a 1D convolutional recurrent architecture over the feature domain[12]. While this model perform the best on the challenge, based on other research and the time invariance plus frequency dependence of sound, we felt that traditional convolutional approaches applied to the spectogram were not as promising[12][8].

3. Problem statement

We define the problem of automatic processing for PAM systems in two distinct ways: 1) as a segmentation problem where our goal is to classify each discrete time-step as being part of an elephant call or not and 2) as a trigger-word detection problem where we focus specifically on detecting the occurrence of an elephant call, while being less precise about its exact start and end time. More specifically, in the trigger-word detection setting we look to predict or "wake" our system on the time step directly after a call has occurred.

We consider these separate problem formulations for several reason. On the one hand, there is a great need for tools that can process large, pre-collected datasets and output accurate time level segmentations. On the other hand, these datasets are extremely large, motivating light weight methods to filter out non-elephant noises in real time so that only relevant information gets logged to the device. By framing the problem of elephant call detection as a wake-word detection problem, we propose a simplified definition of detecting an elephant call (i.e. detecting its conclusion) to allow for both a simplified model and real time task. Whereas in the case of pre-collected data segmentation we leverage more complex models that take advantage of both forwards and backwards processing, for real time processing/detection we simplify the problem in order to use light weight more efficient models. In real time, we care most about detecting the presence of a call, from which we can then choose to save a particular data chunk or simply notify a system of the event. For example, one can imagine a setting where many lightweight, cheap devices are deployed to count the occurrences of species.

4. Dataset

4.1. Elephant Listening Project

Established in 2000, the Elephant Listening Project (ELP) uses acoustic methods to study the ecology and behavior of forest elephants in order to improve evidence-based decision making concerning their conservation [18]. The ELP has gathered over 700,000 hours of data from around 150 locations within the African rain-forest, providing a very diverse and complete dataset for developing machine learning models. We are very thankful and lucky to have access to a rich subset of data collected between 2007 and 2012 from three sites in Gabon and one in the Central African Republic[18]. At each sight, a single recording device was placed in a tree 7-10 meters above the ground

near forest clearings where elephants congregate. Thanks to the tireless effort of experts and trained volunteers, hundreds of hours of audio have been labeled , where individual calls are labeled with their temporal duration (i.e. start and end time). As typical with PAM of large range species in a dynamic/chaotic environment (such as the rain-forest), elephant calls appear infrequently and many other sources of noise, both from other animals and human activity (logging, planes, cars), make accurate labeling a challenging task.

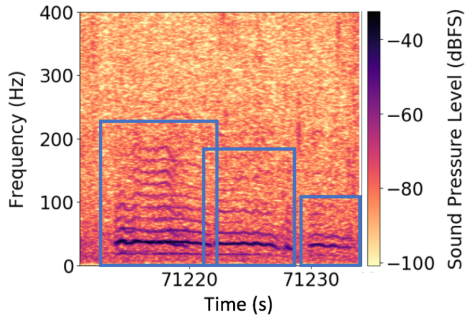

Figure 2. Example of three elephant calls.

4.2. Data Representation

Although the raw audio data is naturally one-dimensional, just as volunteers label/view the calls on a spectogram, we leverage the rich descriptive features within the 2 dimensional representation of sound. For a given raw audio sequence, we perform a short term Fourier transform to transform the data into a 2-dimensional frequency by time representation. This data representation lends itself naturally to this problem because of the distinct acoustic characteristics of the elephant when expressed in this 2D "sound image": namely stacked parallel "eyebrow" shaped lines, representing the harmonics of the fundamental frequency (see Fig 2). When computing the spectrogram representation, we use a window size of 512 and hop-length of 383 for transforming the audio signal into the frequency domain through the FFT. Additionally, we remove all frequency bands above 150Hz, as elephant calls rarely have significant signals above such frequency. Implemented on the down sampled audio data (1000Hz), each interval in the spectogram encodes the frequencies within .512 seconds of raw audio.

Due to the large class imbalance within the data-set (depending on the location elephant calls make up only 0.7% to 20% of the data), we look to carefully facilitate a roughly homogeonous data set to avoid majority class bias in training. Namely, for both of the proposed tasks, we extract fixed 25.5 time-step windows for each of the elephant calls, with the call randomly placed within. Because most calls range from length 5-10 seconds, we chose this window size to both capture significant elephant noise and background/adversarial noise. Based on this definition, we produce data samples that are 64 time steps by 77 frequency band tensors. For the segmentation problem, we define the ground truth labeling of each spectogram time-step with a binary label indicating whether an elephant call occurs in that time-step. In contrast, for the trigger-word detection task we label 5 time-steps immediately after the end of the call with a positive 1 label. Although in trigger word detection we look to predict the end of the call, we soften this requirement (i.e. predict the end of the call within a specific interval) in order to help balance the training set.

When compiling our dataset, because of the size of our dataset, we employ a 95, 5 train/test split, resulting in $(106986, 5539)$ train, test examples for the segmentation dataset and $(105447, 5451)$ for the trigger-word detection set (different sizes are due to subtle differences in handling overlapping calls). When splitting the dataset, we must pay special attention to potential overlap in example windows that could result in data-set pollution. After analyzing the data loading code used by the Cornell research team in [1], we noticed unfortunately that their test set is likely polluted by including windows that overlap with training examples (i.e. when two calls occur near each other). Because of this very real danger, we specifically draw the train and test windows from different time periods.

4.3. Data Pre-Processing/Normalization

As in almost any deep learning/data driven problem, one key step in the modeling pipeline is data pre-processing and normalization. We consider several different data pre-processing techniques, which we later show have a very significant impact on model performance. Through research and experimentation we found that logarithmically scaling each matrix entry in the spectogram helps sharpen the visual contrast within the spectogram, a key benefit when approaching the problem as a vision task.

In addition we consider a collection of different normalization techniques - where normalizing refers to subtracting mean and dividing by standard deviation. Motivated by traditional image processing models, we considered normalizing each spectogram image by the mean and standard deviation across the train set. Additionally, because of the frequency variant nature of audio, we define "feature-norm" as a normalization across each frequency band in the spectogram. Lastly, we consider a very domain specific norm that looks to normalize based on background noise while preserving important call information. Because the signals are very sparse, we consider normalizing by just the frames without elephant calls (i.e. background), again across each frequency.

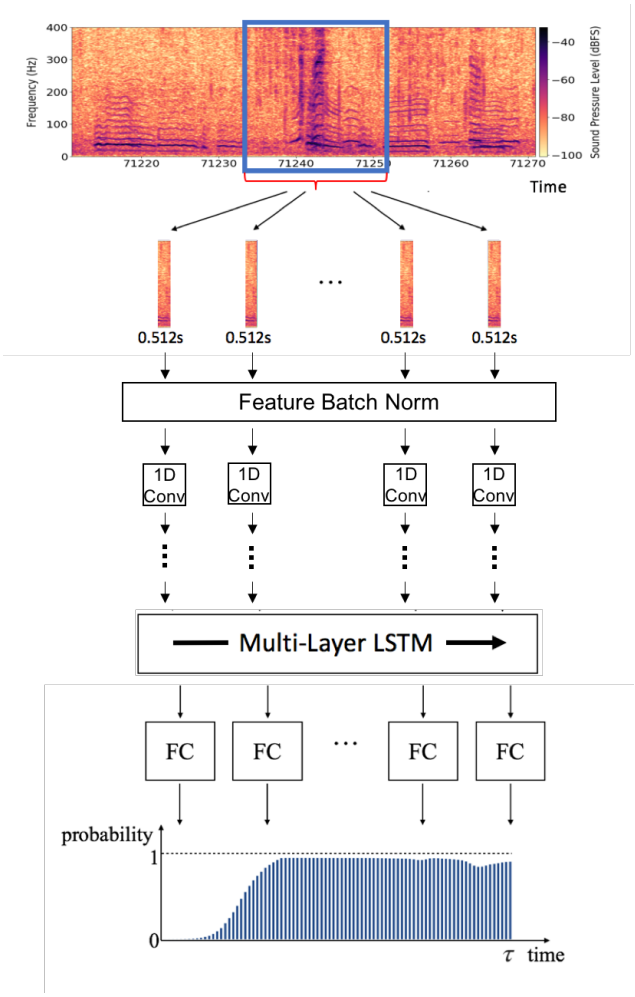

Figure 3. Adapted Model framework. Inspired by [12]

5. Methods

5.1. Unidirectional (Real Time) Models

Given that our task is to produce a label for each time step in real time and our models need to accept variable length inputs and keep track internally of previously seen information we realized the natural candidate model family would be Recurrent Neural Networks. RNN's process time step data sequentially, while keeping a representation of the past to help inform predictions at each state. Traditional RNN's struggle to maintain long term information and so we realized early on that we really want to use Long short-term RNN's which have internal structure specifically built to promote longer term memory as well as to avoid gradient vanishing issues in training.

5.1.1 LSTM Baseline

We decided to use a standard unidirectional LSTM as our baseline algorithm since it was the most obvious candidate for a model that could make real time predictions accurately and efficiently by leveraging its internal state to store information about previously seen data rather than needing to store and reprocess that data at each time. At every time step an LSTM outputs a hidden state that we passed through a single linear layer to get our class prediction as well as a cell state that is used to keep track of information memorized over a long time. This cell state is the key contributor to the model's longer term memory. Initially the hidden and cell states can be set to zero; however, we decided to make their initialization learnable parameters so that there would not be a drop in accuracy at the beginning of every batch while the model warmed up.

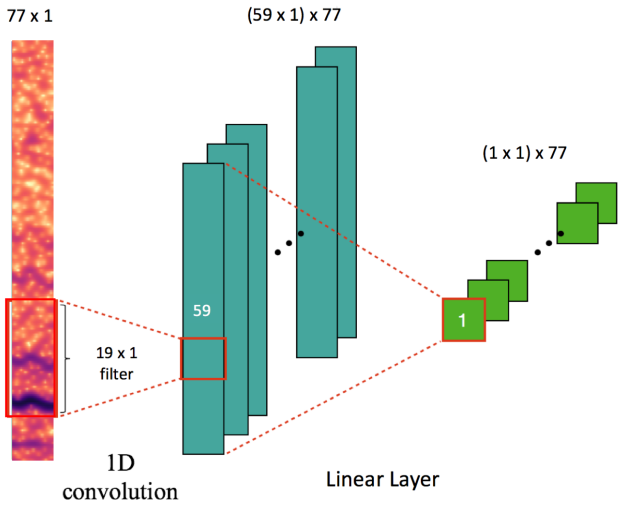

Figure 4. The characteristic shape of a Right Whale call.

5.2. Conv1D Layers With RNN-LSTM

Motivated by [12]'s work we looked into increasing our model's representational power and accuracy by passing a time step slice first through 1 dimensional convolution layers over the *frequency* domain, interspersed with maxpool, average pool, or a linear layer before inputting to the LSTM. While 2 dimensional convolutions are commonly used in the literature for looking at temporal and spectral features, their usage is predicated on having access to the entire dataset chunk which at least in our unidirectional problem statement is not possible. In addition, our research highlighted that 1 dimensional convolutions may be sufficient to achieve state of the art results [12]. The 1 dimensional convolution works by sliding multiple 1 dimensional windows of weights and a bias over a single time step input. Each 1 dimensional window weight, or filter, leads to a single out-

put channel and these are combined to produce an output of shape (num-filters by output of sliding weights operation). Since the 1DConv layer increases the input's size and even converts it from 1D (1 channel by num input frequencies) to 2D then we follow the convolutional layer with some form of a pooling or dimensionality reduction[12]. We experimented with a **1)** *flatten operation* which compresses the 2 dimensions into one long vector, a **2)** *maxpool layer*, an **3)** *average pool layer*, and a **4)** *linear layer*. These last 3 layers were run over the frequency dimension. Standard 1D maxpool reduces the size according to the filter size, and while we experimented with this we were motivated by [12] to use a maxpool with filter size equal to conv output to effectively squash an entire dimension. An average pool employs a similar operation but rather than selecting the max element selects the average of the elements. And, lastly, applying a linear layer as a pooling operation we found was a very effective way to reduce dimension since its learnable parameters allow for a smarter averaging operation.

5.3. Linear Layers With RNN-LSTM

We found during our experiments with the previous model that the linear layer worked surprisingly well as a dimensionality reduction operation. We also came across research that indicated a key difference in audio spectrograms from regular 2d images. While images in machine vision tasks are assumed to be translationally invariant, as in the distribution of values of a pixel do not depend on the pixel's location in the input image but rather its relative location to other important pixels, spectrograms are translationally invariant in the time direction but not the frequency dimension [8]. This is because the meaning of an activation can drastically change depending on the frequency band it was produced in. For example, elephant calls are characterized by the low frequency rumbles discussed earlier. Our model is invariant to where a call appears in the time dimension since we process each time step feature separately and the LSTM is translationally invariant in its input features. However, by taking a 1d convolution and then maxpooling over the entire sequence as described in [12] we realized that we were key frequency spatial information and our accuracies as discussed in results suffered because of it. For this reason, we decided to also experiment with more linear layers both as the previously described pooling operation and as entire replacements of the convolutional operation.

5.4. Multilayer RNN-LSTM

To increase our models' representational power we experimented with multi-layer LSTMs in which the hidden states of one LSTM inputs to the next LSTM. Theoretically, this allows higher LSTM layers to learn more general features and approx. more complex functions.

5.5. Bidirectional RNN-LSTM

For the segmentation problem, in which we had access to the entire data chunk at test time, we decided to implement the described above models but with bidirectional LSTMs. A bidirectional LSTM processes the sequential input features in both forwards and backwards directions separately and then concatenates the two produced output hidden states at every time step. These concatenated hidden states can then be used in the same way as before with additional linear layers to produce an output logit. This bidirectionality is benifitial for the task of segmentation because in a single direction predicting the presence of an elephant call takes "some time". Namely, the uni-directional model must see some frames of the call before being able to confidently say that it is in fact a call. By running in both directions the model is able to process the call from both ends, allowing the model to make a more informed decision about the early time steps of the call as well as the end steps.

5.6. BatchNorm 1d

We used a 1 directional batchnorm on the inputs over the frequency dimension to allow our model to be more tolerant to differences in background noise across regions as well as to allow the model to separate out interesting anomalies in a learnable loss driven pre-processing step. 1 dimensional batchnorm operates by using the per feature mean and standard deviation over an entire input batch to perform an initial scaling and then performing a second scaling using learned means and standard deviations learned with gradient descent from the entire training dataset. At test time we use our training time learned values. Effectively this extra learnable scaling allows the model to learn to either completely undo itself or discover the most useful way to scale the training data. Motivated by the succes in [13], we chose different parameters for each feature in hopes of preserving the semantic differences between different frequencies as opposed to a normal image pre-processing operation in which we normalize by the mean and std over all pixels.

5.7. Softmax and Binary Cross Entropy Loss

To produce a binary output for each time step we passed the output logits from our models through a sigmoid function. The sigmoid function maps a real valued input to the range 0 to 1 according to the following function:

$$Sigmoid = \frac{1}{1 + e^{-x}}$$

To train our models we use the standard for classification tasks binary cross entropy loss. Essentially minimizing every parameter's contribution to the BCE loss equates to minimizing the difference in the distribution of our predicted outputs for a given input to that input's true value.

$$BCE = -(y \log(p) + (1 - y) \log(1 - p))$$

6. Experiments

6.1. Segmentation Task and Trigger Word Task

We ran all of our unidirectional models on both the segmentation and wake word tasks. The bidirectional model variants were only applicable to the segmentation task since those at test time looked at time windows. We trained and tested on window chunks for both tasks for batching, and thus efficiency purposes, as well as to control the balance of labels our model saw so that we wouldn't be training mostly on background noise. However, to more accurately evaluate our final models we ran the final models on the full uncut test dataset.

6.2. Normalizations

As described in section **4.3**, we experimented with the log scaling of the input features, as well as on three different normalization methods: 1) A traditional mean and standard deviation normalization across the whole spectogram (Standard); 2) Feature/per frequency independent normalization (Feature); 3) Background noise feature normalization where we only normalize based on regions of the data without elephant calls (Background).

6.3. Models

We experimented with many variants of the model architectures described in methods. Primarily our variations were motivated from analysis of our results for bias and variance issues to achieve higher accuracy, but some of them were also as part of ablation tests to show that theoretically important elements were in fact important. In terms of ablation tests we compared models with and without 1d batchnorm on top of the norm discussed above. We then compared the linear LSTM variants with the Conv LSTM variants. For the Conv LSTM variants we compared max-pooling, average pooling, and linear pooling after convolutional layers along with different filter sizes experimenting even with filters of size equal to the entire input dimension motivated by [13]. On the segmentation task we compared bidirectional models with unidirectional ones. To increase performance we first addressed high bias indicated by low training rate accuracy by increasing the representational power of our models. We added more trainable layers to our model both before and after the LSTM layer as well as stacking multiple LSTMs. We also tried increasing the complexity of individual layers by increasing their sizes.

6.4. Optimization

We trained our models on Tesla P100 GPUs for anywhere between 5-50 epochs. For our optimization algorithm we chose Adam since it typically requires less hyperparameter tuning and is resistant to noisy or sparse gradients. For most of our models the default initial learning rate of 1e-3 worked quite well. We experimented with learning rate scheduling but found that any large decay rates only slowed down training and decreased performance. We originally ran large batch sizes of 512 and even 1024 and 2048 since our GPU's could handle it and the speed increases were significant. However, we saw that larger batch sizes even if faster led to lower performance and that small batch sizes of size 32 with many iterations and thus updates per epoch worked better. We experimented with different values for l2 weight regularization and found that a value of 1e-4 worked well to minimize our model's variance. To further combat variance we early stopped once we saw our validation loss plateau for a few epochs or even diverge. We tuned the optimization hyperparameters for our top models but found that besides the ones mentioned here the rest worked well at their default values.

7. Metrics

7.1. Accuracy, Precision, Recall, and F-score

In line with standard procedure for classification tasks we used accuracy, precision, recall, and f-score. Accuracy is defined as the fraction of target labels correctly labeled. Precision is defined as the fraction of for example elephant call labels we predicted that were in fact elephant calls. Whereas recall is defined as the fraction of real elephant calls that we labeled as elephant calls. As a holistic metric we weighted and combined precision and recall to get a single metric called f-score. We considered using intersection over union as a possible metric but found that our precision and recall metrics captured that information enough and our following trigger word detection precision and recall were ultimately the most important.

7.2. Trigger Word Detection Precision and Recall

We also came up with our own metrics that would help us optimize better for the researchers' indicated problem and eventual usage of our models. We called these metrics trigger word detection precision and recall (*Trig R*, *Trig P*). We defined them respectively as the fraction of predicted elephant calls, not fraction of elephant call class labels, correctly labeled and the fraction of actual elephant calls that we predicted. By focusing on whether or not we indicated the existence of the call rather than how closely we captured the start and end times in the segmentation task or the number of arbitrary post wake word labels we were able to evaluate our models better on the actual final task.

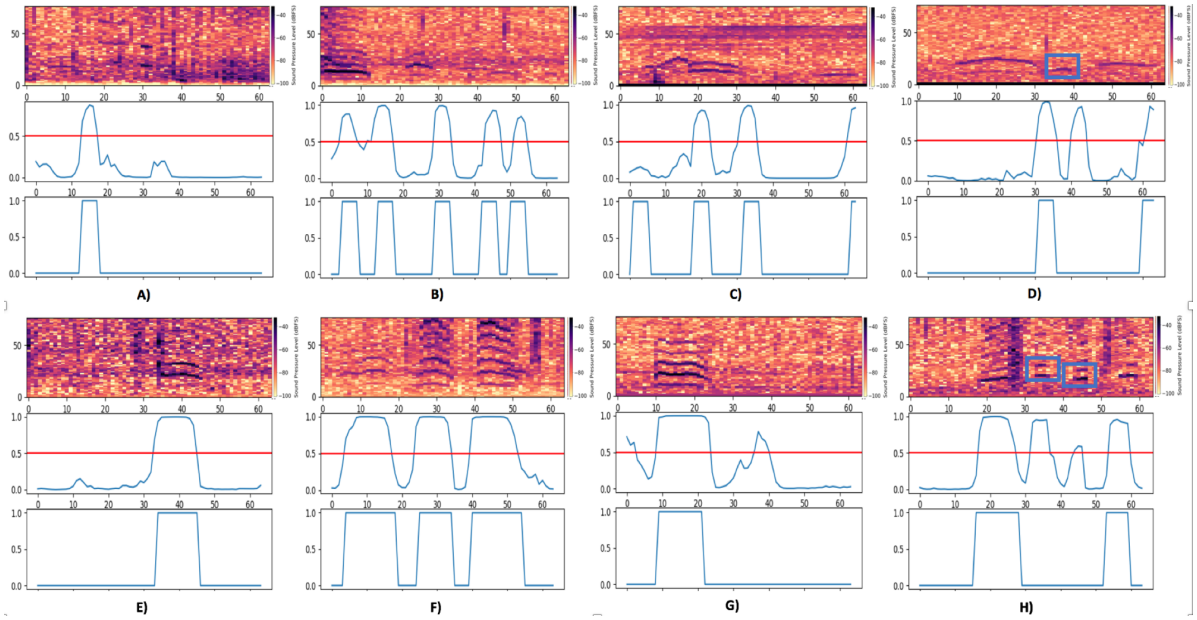

Figure 5. Visual representation of model performance. Top row - trigger-word detection; bottom - segmentation. **A)** Detect single tricky trigger. **B)** Detect many triggers, some obvious some not. **C)** Tricky missed trigger, one cannot see the call in the spectogram. **D)** Potentially miss-labeling in ground-truth identified in blue-box. **E)** Correct segmentation of a single call. **F)** Segmentation of several calls in a window. **G)** Incorrect segmenation of non-call (Note that it only makes a small mistake). **H)** Potential miss-labeling identified by model.

8. Results

We highlight model results that illuminated our understanding of model features as well as showcased the progression of performance on our task. In Tabel 1, we perform a preliminary analysis of the effect the different normalization techniques discussed in section **4.3** on the Baseline LSTM model described in section **5.1.1**.

Table 2, highlights the performance comparison between baseline and best achieving models on the Trigger-word and Segmentation problems. Based Table 1 results and continued experimentation, we use (Standard Norm, Log Scaling) for Trigger-word detection and (Feature Norm, Log Scaling) for segmentation. The different model architectures used are indicated by the numbers $\{0, 7, 1, 9\}$ where we have: Model 0 - Baseline LSTM from section **5.1.1**; Model 7 - BatchNorm 1d **Section(5.6)** followed by Conv1D Layer with *linear layer pooling* and 2-layer LSTM (see Figure 3 and section **5.2**)); Model 1 - Conv1D layer with *flatten operation* and single layer LSTM (see section **5.2**); Model 9 - BatchNorm 1d followed by Conv1D layer with *flatten operation* and two layer Bidirectional LSTM (see Section **5.5**). All models have a final time distributed linear layer.

In Table 3 we provide the results when running our models models on the full uncut audio files. We see that in the Trigger-word detection problem this represents the process of real time detection; similarly for segmentation, this task models analysis of full uncut recording files.

Norm	Scale	Accuracy
None	None	62.02
None	Log	80.33
Standard	None	76.76
Standard	Log	86.57
Feature	None	82.42
Feature	Log	86.67
Background	None	82.67
Background	**Log**	**86.83**

Table 1. Different Normalization techniques for LSTM Baseline

Model	Acc.	F1 (0)	F1 (1)	Trig R	Trig P
Trig-0	87.3	92.8	72.1	86.5	82.1
Trig-7	**89.3**	**93.0**	**72.7**	**85.73**	**86.4**
Seg-1	86.25	89.2	81.5	96.7	71.2
Seg-9	**90.16**	**92.0**	**86.9**	**94.2**	**86.1**

Table 2. Comparison of Baseline and Best model Performance for Trigger-word detection (Trig-0, Trig-7) and segmentation (Seg-1, Seg-9). See Section **7.2** for def. of Trig R/P.

9. Discussion

Overall we were super impressed with our results on both tasks, and to our knowledge these represent the state of the art. After overcoming initial bias issues by properly normalizing our dataset, using the scaling trick, and adding

Model	Acc.	F1 (0)	F1 (1)	Trig R	Trig P
Trig-7	89.3	99.0	69.0	87.3	65.5
Seg-9	90.16	98.4	71.2	94.6	57

Table 3. Best model performances on uncut Test data.

feature-wise 1d batchnorm we found that we reached a level of accuracy that was very hard to surpass. In trigger-word detection all of our unidirectional LSTM models regardless of number of linear, convolutional layers (with linear pooling), or filter sizes, after hyperparameter tuning, performed pretty similarly and pretty well. For both tasks, our models originally had a variance issue. We addressed this by early stopping and increasing L2 regularization. In the end, we seemed to still have a bias problem, since while our models were able to overfit (generally by 3% higher train accuracy than valid accuracy) the highest train accuracy was still not quite above 94%. From qualitatively analyzing our results we concluded that this model bias may be inherent in the dataset. For example, in examples D and H from figure 5 there seem to be not labeled elephant calls that our model correctly picked up on. In addition, on the correctly labeled elephant calls for example A, B, E, and F in figure 5, while our predictions visually look very similar to the ground-truth, the accuracy and F1 metrics are potentially overly harsh to small insignificant deviations. Overall, we believe our models achieved high domain accuracy, precision, and recall for both problem definitions.

We were impressed by the importance of the logarithmic scaling trick as well as normalization of the dataset. The particular norm we chose didn't seem to matter much overall, although models without an initial batchnorm layer were much more sensitive to this choice. This was somewhat surprising since visually to us the different normalization techniques had a large impact on how easy the task would be. Overall, however, batchnorm seems to help our model tolerate more variable inputs which makes sense and it gave us a slight performance increase.

We also saw very interesting results using the different types of pooling after Conv1D layers. As we hypothesized, but in contradiction to the results of [12], max pooling with a filter size equal to the entire output in the frequency dimension performed poorly. Average pooling with the same filter size was slightly better but still poor. As mentioned previously, we hypothesize that this is because maxpooling over the frequency dimension treats the frequencies as translationally invariant, which may not be a good choice for a task like elephant call detection where the exact pitches of calls are very informative of their source. Ultimately, the best results came from either flattening the convolutional layer outputs and suffering a significant memory and computation cost or using a learnable linear layer to weigh frequency features differently. On the trigger word task we decided to stick with linear pooling for efficiency benefits.

As predicted the best performing model on the segmentation task was the bidirectional model that was able to take advantage of sequentially processing the data in both directions. We theorize that on the segmentation task it is difficult for a unidirectional LSTM to know immediately when to start labelling a time step since it requires a couple moments for call confirmation, whereas the LSTM is able to avoid this by processing the calls in both directions and combining results.

We also want to mention that although we avoided polluting our train and test dataset like we believe we uncovered in the work done by [1], we did train our best models on their datasets and achieved a higher accuracy of 93.32 % compared to their 91.71 % on the segmentation task with our bidirectional LSTM.

Lastly, looking at the performance of our best models on the uncut raw test audio (table 3), we are quite impressed with our models' ability to generalize beyond the training setting. As expected, when applied to the raw test audio, the heavy class imbalance skews the performance of our model compared to table 2. We see that our model has a tendency to over-predict elephant calls, leading to low F1(1) and Trig P metrics; however, this is not necessarily bad as our models both do a good job of identifying a large percentage of the true calls (Trig R). From a research perspective, there is an incentive to over predict in order to have high call recall, at least in the early stages, as models increase in performance and small scale post processing can be done.

10. Future Work

We were very impressed with our performance on both the segmentation and trigger-word detection tasks. To our knowledge these results are state of the art for elephant call detection on both tasks. We were especially impressed by the high performance on the trigger word detection task using models with relatively few weights. It suggests a potential future for real time elephant call detection on low resource and inexpensive devices that could majorly improve scientists' ability to monitor these beautiful animals.

Given more time, we believe a promising direction entails better data-selection during training. For example, developing methods for more accurately capturing true class imbalances and more directed approachs to train on adversarial examples (such as chain-saw noises, car sounds, and other animal calls similar to that of an elephant).

We would love to apply our models to similar tasks across other animal call datasets and further iterate on them. If successful, it might be possible one day to have world wide automatic monitoring of animal and insect counts with unprecedented ease and scale. This might open up a number of biological and environmental research directions and help conservation efforts.

11. Contributions & Acknowledgements

A link to our public github repo can be found here [17]. We wrote all of our models from scratch using pytorch and refactored a significant amount of data preprocessing code from [1]. We are very thankful to the Elephant Listening Project for their gracious contribution of the data and the Institute for Computational Sustainability at Cornell for their generous donation of GPU and cluster usage. In terms of contributions, the work was split very evenly between Nikita and Jonathan. Nikita spent significant work on developing the different model architectures and running different experiments. Jonathan worked to process and represent the data effectively through data normalization techniques and scaling. Together, both team members helped analyze the results and brainstorm next steps to take at all steps of the process. And we equally contributed to the final report.

References

[1]Bjorck, Johan, et al. "Automatic Detection and Compression for Passive Acoustic Monitoring of the African Forest Elephant." arXiv preprint arXiv:1902.09069 (2019).

[2]Sahidullah, M., and Saha, G. 2012. Design, analysis and experimental evaluation of block based transformation in mfcc computation for speaker recognition. Speech Communication.

[3]Juang, B.-H.; Levinson, S.; and Sondhi, M. 1986. Maximum likelihood estimation for multivariate mixture observations of markov chains (corresp.). IEEE Transactions on Information Theory.

[4]Leon, D., Danieal, and College, College. Passive Acoustic Monitoring of Blue and Fin Whales through. MBARI publication.

[5]https://www.kaggle.com/c/whale-detection-challenge/data

[6]Kahl, Stefan, et al. "Large-Scale Bird Sound Classification using Convolutional Neural Networks." CLEF (Working Notes). 2017.

[7]Grill, Thomas, and Jan Schlter. "Two convolutional neural networks for bird detection in audio signals." 2017 25th European Signal Processing Conference (EUSIPCO). IEEE, 2017.

[8]Parthasarathy, Dhruv. A Brief History of CNNs in Image Segmentation: From R-CNN to Mask R-CNN. Medium, Athelas, 22 Apr. 2017, blog.athelas.com/a-brief-history-of-cnns-in-image-segmentation-from-r-cnn-to-mask-r-cnn-34ea83205de4.

[9]Messner, Elmar, Matthias Zhrer, and Franz Pernkopf. "Heart Sound SegmentationAn Event Detection Approach Using Deep Recurrent Neural Networks." IEEE transactions on biomedical engineering 65.9 (2018): 1964-1974.

[10]Cakr, Emre, and Tuomas Virtanen. "Convolutional recurrent neural networks for rare sound event detection." Detection and Classification of Acoustic Scenes and Events (DCASE) (2017).

[11]Kao, Chieh-Chi, et al. "R-CRNN: Region-based convolutional recurrent neural network for audio event detection." arXiv preprint arXiv:1808.06627 (2018).

[12]Lim, Hyungui, Jeongsoo Park, and Y. Han. "Rare sound event detection using 1D convolutional recurrent neural networks." Proceedings of the Detection and Classification of Acoustic Scenes and Events 2017 Workshop. 2017.

[13]L. Wyse. 2017. Audio Spectrogram Representations for Processing with Convolutional Neural Networks. Proceedings of the First International Workshop on Deep Learning and Music joint with IJCNN. Anchorage, US. May, 2017. 1(1). pp 3741.

[14]https://www.youtube.com/watch?v=ZqxhbTmN6A

[15]http://dcase.community/challenge2019/index

[16]http://www.cs.tut.fi/sgn/arg/dcase2017/challenge/task-sound-event-detection-in-real-life-audio-results

[17]https://github.com/N-Demir/ElephantCallAI

[18] http://elephantlisteningproject.org/