ChirpGAN: A Generative Adversarial Network (GAN) for Synthesizing Artificial Bird Calls

Woo Jung, Derek Albosta
University of Puget Sound

ABSTRACT

Birds engage in social communication through the use of complex sequences of chirping. Bird linguistics is a speculative topic where studies have formalized language as a combinatorial communication system [8]. This project aims to create a tool ChirpGAN to synthesize bird vocalizations based on a collection of actual bird song recordings. ChirpGAN makes use of generative adversarial networks (GANs) and is delivered with a full data pipeline and user interface designed for simple and quick navigation. ChirpGAN successfully synthesizes scalograms of artificial bird vocalizations. However, unfortunately, ChirpGAN is incomplete as of writing this paper, lacking its final conversion of synthesized vocalizations back to audio.

1 INTRODUCTION

We are two decades into the 21st century and we have still have yet to develop a method by which to understand animal communication. There are several modes of communication that mainly fall into the categories of visual, auditory, tactile, and chemical [21]. In this paper, we will be focusing on the auditory form of communication of a specific type of bird—that is, communication via the transmission of sound by canaries.

There exist studies that have used generative adversarial networks (GANs) to synthesize audio [5, 6, 12, 20]. A few dedicated research teams have used these methods to apply GANs to synthesise bird vocalization data [5]. However, these audio synthesizer GANs have made generalizability its priority. Mainly, GAN-based audio synthesizers are focused on generating better human recordings and musical sounds or pieces [13].

In this paper, we present an attempt at a fully-featured program ChirpGAN to synthesize artificial bird calls using generative adversarial networks. Features of ChirpGAN include a full data pipeline, dataset creation, and bird vocalization synthesis using a GAN. The data pipeline processes the user's collection of bird audio files of choice into scalograms using wavelet transforms and background noise reduction. The dataset creation feature automatically extracts important calls from the set of scalograms processed by the data pipeline. Finally, the GAN trains on the user-created dataset using the progressively growing methodology and generates scalograms of artificial bird vocalizations. Unfortunately, ChirpGAN was not able to implement the final conversion of generated scalograms back into audio in a timely manner and thus lacks the ability to hear the end result.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 BACKGROUND AND RELATED WORK

Auditory communication is an interesting topic that has appealed to the inquisitive nature of humans for centuries. There is a considerable amount of history in the study of animal auditory communication. In fact, dating back to the fourth century BC, Aristotle argues at the beginning of his *Politics* that birds are a species of animal that are capable of the transmission of information but not that of values such as justice and injustice or useful and harmful things. In the first century AD, Pliny the Elder; In the third century AD, Claudius Aelianus; Including modern bird vocalization research, there have been numerous extensive works that write on or refer to bird communication [3].

What is especially attractive of bird communication is the seemingly inexhaustible variations in bird talk. There are in total about 10,000 bird species described worldwide. However, a study estimates that the total number of species may be double that number [1]. There is a difference in vocalization between two different bird species. In fact, some species exhibit within-species variations as well, depending on relocation and other localizing factors.

One framework for studying animal communication and language is that of experimental playback. That is, playing recordings of bird vocalizations to real birds and collecting responses to the playbacks. In fact, a study has found evidence for compositional syntax in birds calls by means of experimental playback [19]. This study has found correlation between different note types (vocalization shapes viewed on a scalogram) and a range of contexts and actions. A limitation of the study is that the combination of notes were produced using slices of recorded audio which significantly reduces the variation and thus the generalizability of the results.

Why scalograms? Studies that have used GANs for audio synthesis as well as studies in signal processing and analysis have predominately used spectrograms for the choice in visual representation of audio signals. Spectrograms are results of a Fourier transform which decomposes its input signal into its constituent frequencies. The resulting spectrogram has time on its x-axis and linearly scaled frequencies on its y-axis. Instead, we chose to use scalograms which are also visual representations of audio signals that have logarithmically scaled frequencies on its y-axis. The main reason for this is that humans, as well as any terrestrial mammal, hear on a log scale [9]. Thus, the log scale gives us a closer representation of how humans would in fact hear the sound depicted. Birds are not mammals. In the search for birds' auditory sensitivity, we have found no clear cut evidence that birds do not hear logarithmically.

Studies such as WaveGAN have used raw waveforms in its 1-dimensional array shape for synthesizing audio [5]. Working directly with raw audio seems like an attractive option as it does not lose any information due to a means of signal decomposition. However, learning from raw waveforms does not improve much, and produces scalogram-like first layers [4, 17]. Thus, we will stick to using scalograms for the visual representations of bird vocalizations in this project.

Understanding auditory bird communication is a truly daunting task. However, there is always a place to start and this project is an attempt to create a tool for that very first step by utilizing generative adversarial networks.

3 METHODOLOGY AND DESIGN

This section covers the biggest aspects of the project's methodology and design. The subsections are organized in chronological order in terms of the full project implementation.

Data collection is covered in subsection 3.1 which goes in depth regarding the first step of the project including audio file format choice and the sources as well as how the data was retrieved. Next, subsection 3.2 explains the entire system of what it took for us to go from the hours of audio data to creating a dataset ready for use in training the GAN. Following, subsection 3.3 describes in depth what a GAN is and the project's implementation of one for the generation of bird vocalizations. The next subsection 3.4 specifies the mathematics behind reconstructing an audio signal from generated scalograms. The final subsection 3.5 deliberates on the design choices as well as implementation of the GUI which ties together the entirety of the project nicely for the client.

3.1 Data Collection

This subsection writes regarding the data collection portion of the project. This subsection will discuss the persistent search for audio data in the WAV file format as well as disclose the origins of all data used throughout this project.

The WAV File The Waveform Audio File Format, widely known as the `WAV` due to its file extension `<filename>.wav`, is the audio file format of choice for the data used in this project. It is the standard digital audio format used by the Library of Congress, Association for Recorded Sound Collections (ARSC), the British Library, and other institutions that are committed to preservation of audio. `WAV` is an uncompressed audio file format that maintains the accuracy of a recording for conservation and research.

One may ask, "why use the WAV file instead of a more prolific file format the likes of MPEGs (i.e. MP3, M4A)?". For the same audio recording, an MP3 or M4A will always witness a smaller file size in contrast with a WAV. This means the use of WAVs is a clear disadvantage to MPEGs in terms of disk space consumption. The smaller file size on MPEGs is attributed to the lossy data-compression encoding scheme. This means that some amount audio data is lost as a result of the compression process that reduces the size of these files. Thus for research purposes, under which this project is classified, the full frequency spectrum of uncompressed data that the `WAV` format delivers far outweighs its larger file size.

Pre-dataset As the bird vocalization dataset from the client was not delivered until several weeks into the project timeline, the earlier versions of the data preparation pipeline was built using data collected from the Cornell Lab of Ornithology's Macaulay Library. The Macaulay Library is an outstanding scientific archive of natural history audio, video, and photographs, rooted in and mostly consisting of birds.

Although we did not receive the client dataset at this point, we learned that the it would be comprised of vocalizations of canaries (*Serinus canaria*). Thus the audio files used for the initial stages of project development were canary vocalizations acquired from the Macaulay Library.

Client dataset The main dataset was sourced from our client Professor Melvin Rouse of the Psychology Department at the University of Puget Sound. His research lies in the analysis of the interaction of hormones, the brain, and reproductive behavior. His lab uses songbirds as the model system for his studies. The dataset dates back to his paper from previous research in analyzing canaries song behavior [14]. According to the paper, the recordings were sampled at approximately 22 kHz which produces a frequency range of up to 11 kHz.

The data was shared via a Google Drive folder containing 286 WAV files, the majority of which are half an hour long. This amounts to more than 30 GB of data. After several failed attempts at manually downloading the files, we turned to the Google Drive API. We wrote a script to crawl through and download just the WAV files from the folders. Using this script, we were able to download all of the client's data to our allocated server that was ready to begin the data processing.

3.2 Data Processing

This section writes regarding the data processing portion of the project. This is likely the largest section of the Methodology/Design section. This portion includes every step in the data processing pipeline, including its history (our attempts and how that lead up to the final pipeline we have). This section (and the methodology/design section as a whole) tries to limit its overlap with the background/related work section which covers the majority of the theoretical aspects of the project.

Datasplitting Datasplitting is the first step in our data processing pipeline which splits the original `.wav` files into several `.wav` files of shorter duration. This is done to reduce the file size to accommodate for the limited heap size required for the scalogram transform as well as the bucket fill.

The original recordings used in this project were about 30-minutes long. Due to challenges faced with physical memory limitation when running the scalogram transform and bucket fill algorithms, it is imperative to cut down the duration of each file fed to the pipeline. There are several advantages of shorter-duration files, of which the most important is a higher throughput rate. Although not commonplace, there are cases in which the scalogram transform and bucket fill operations will encounter an error due to poor rounding of values in a division operation. Thus, by splitting the

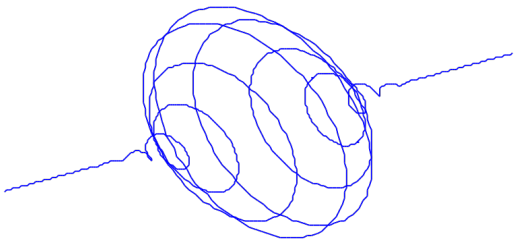


Figure 1: Plot of the complex-valued Morlet wavelet. Reproduced from Wikipedia.

recordings into shorter files in a mini-batch sense, we lose less on an error. In our case, we split each 30-minute recording into approximately 130 second recordings.

Wavelet transform : The *wavelet transform* is responsible for the conversion of `.wav` audio files into `.scal` scalograms in our project.

By the Fourier theory, any given signal can be expressed as the sum of, possibly an infinite, series of sinusoids of varying frequencies. Built on this theory, the Fourier transform is still one of the most well-widely used methods for signal decomposition into the joint frequency and time domain. A critical limiting factor of the Fourier transform is the phenomenon known as the Heisenberg uncertainty principle: functions localized in the time domain are spread out in the frequency domain, and vice versa. Simply put, it is impossible to know the exact frequency and the exact temporal location of this frequency in a signal. This results in the blurring of measurements in either or both domains.

The wavelet transform is the latest advancement to overcome the shortcomings of the Fourier transform. The main difference is in the wavelet transform's use of scales to target specific frequency ranges. Thus, instead of shifting the Fourier window over the signal's entire frequency range, the wavelet transform slides a window over each selected frequency.

A wavelet $\psi(t)$ is a mathematical function that can be thought of as a concise wave that satisfies the *admissibility condition*:

$$\int \frac{|\Psi(\omega)|^2}{|\omega|} d\omega < +\infty, \tag{1}$$

where $\Psi(\omega)$ is the Fourier transform of the wavelet function $\psi(t)$. This admissibility condition implies that the average value of wavelet function in the time domain is zero:

$$\int \psi(t) dt = 0, \tag{2}$$

which implies that the wavelet function must be oscillatory and thus a wave. An example is the complex Morlet wavelet (*a.k.a.* Gabor wavelet),

$$\psi_0(t) = \pi^{-\frac{1}{4}} e^{i\omega_0 t} e^{-\frac{1}{2}t^2}, \tag{3}$$

where ω_0 is a constant that defines the trade between time and frequency resolutions. Evident from the function definition, the complex Morlet is composed of a sinusoid multiplied by a Gaussian. The complex Morlet has an imaginary axis as can be seen in Figure 1.

With a wavelet that satisfies the admissibility condition, we can proceed to the wavelet transforms which are classified into

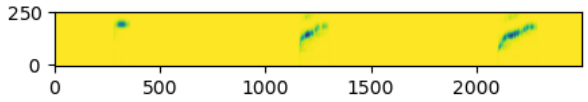


Figure 2: A 2500 px crop of the PNG export of a scalogram file. Less color contrast between the actual call signals (blue/green) and the background (yellow) is due to closer values which signifies high background noise.

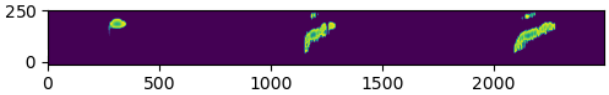


Figure 3: A 2500 px crop of the PNG export of a scalogram file after it has been floodfilled. There is a clearer distinction between the actuall call signals and the background after the background noise has been flood filled.

two general types: continuous and discrete. For this project, we employed the continuous wavelet transform. Let x_n be a time series, discretized by equal time spacing δt, such that $n = 0 \dots N-1$. Then, the continuous wavelet transform of x_n is defined as,

$$W_n(s) = \sum_{n'=0}^{N-1} x_{n'} \overline{\psi}\left[\frac{(n'-n)\delta t}{s}\right], \tag{4}$$

where $\overline{\psi}$ is the complex conjugate of ψ, n is the localized time index, and s is the wavelet scale. To approximate the continuous wavelet transform, we can run convolution (4) N times per scale. Thus, the resulting matrix is is the complex scalogram of the original sound file.

Note that when we save to a scalogram, we are resampling the transform such that each pixel in the scalogram is the mean of a selected number of horizontal neighbors in the wavelet transform. This is due to the high 44.1 kHz sampling rate of the original `.wav` file. Without any resampling measures, the resulting scalogram would be an array of shape $(J, 44100)$ per 1 second of audio, where J is the number of scales.

PNG export : Once the scalogram transform of the sound file is created and saved as a `.scal` file, the next step is to export this scalogram data as a `.png` image. This helps us visually confirm the success of the scalogram transform.

Note that this export from `.scal` to `.png` is another form of compression as it squeezes the range of values to the domain $[-255, 255]$.

Floodfill : Take a closer look at the exported scalogram Figure 2. About the central frequency range 4 kHz, one can easily observe the transcribed bird vocalizations. However, there is still a noticeable blurring in both time (x-axis) and frequency (y-axis) domains, due to the trade-off in accuracy between the frequency and temporal resolutions (Gabor limit).

The specks of darker hued pixels detected beyond the central frequency range and distant from the most call transcriptions are due to background noise (air conditioning, motors, etc) from when the audio files were recorded. As background noise can always be introduced to the reconstructed vocalization audio files, we implemented the floodfill algorithm to distinguish the important bits of data from noise as seen in Figure 3. The floodfill algorithm may

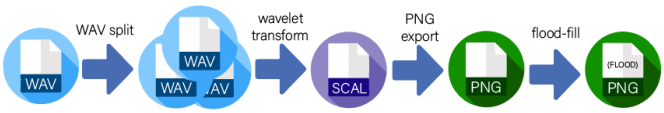


Figure 4: Diagram depicting the organization of our data pipeline

sound unfamiliar but it is the algorithm that powers the "bucket" fill tool found in many graphics editors used to fill in an enclosed area of similarly-colored pixels.

Dataset Prep : This stage of data processing deals with the building of the dataset using the vocalization images that have been flooded to mask over the silent frames as well as background noise pixels. At this point, we have hundreds of $256 \times N$ long bucket-filled vocalization images. One of the most important aspects of the dataset preparation is choosing the right width for the images slices. The reason being that the convolutional layers in the GAN are excellent in localizing object detection. That is, it shines at detecting an object disregarding where the object is located in the image. We tried to find some literature on how to select the most appropriate width for each of the call slices, but we did not find one that was intriguing enough. Although we desire most of the dataset to be comprised of individual calls whose start and end are captured in the single image, we want the dataset to also contain images of vocalizations that are also cut off on either end. This would theoretically train our GAN for one of our future ideas of continuous synthesis vocalizations.

Thus, we concluded on a width of 600 pixels after regarding for the average and median lengths of individual calls as well as an appropriate amount (10%) of white space besides the call in each slice.

Pipelining : The data pipeline brings together and streamlines all parts of the data preparation. The data processing pipeline simply consists of the datasplitting, wavelet transform, .png export, and the bucketfill. Our initial attempt at the pipeline was a breadth-first approach. This turned out a mistake as the resulting scalograms from the wavelet transform have considerably large file sizes. Thus the pipeline ran out of disk space before processing scalograms for all audio files. Therefore, we adopted a depth-first approach where each audio file is processed through the entire process before moving onto the next file.

Figure 4 details the steps of our data pipeline in order. A given WAV file is first split into 30 second WAV file segments. These split WAV files undergo a wavelet transform, converting them into their scalogram representations. The scalograms are then exported to a PNG file format and then run under the flood-fill process. This entire process allows the initial WAV file to be formatted to create a dataset for the GAN.

3.3 GAN Building

This section is the heart of our project where the actual bird vocalization synthesis is happening. Here, the paper will introduce the reader to the concept of a GAN before diving into our implementation of the progressive GAN as well as some tweaks in response to challenges we faced.

GAN A Generative Adversarial Network (GAN) is a machine learning framework developed by Ian Goodfellow and his colleagues in 2014 [7]. The framework adopts a game theoretic model in which two neural networks are put in competition against each other. Specifically, one neural network is a generative model which learns to generate new data sampled from the distribution of the training set. The other neural network is a discriminative model which learns to classify given images as either real or fake.

The generative network learns to map a latent space to the distribution of the data of interest. Thus, the input to the generative network is a set-size sample from a pre-defined latent space characterized by a multivariate statistical distribution. The generator synthesizes candidates based on the parameters it learned from training. These candidates are then passed on to the discriminator network for evaluation.

The discrimination network is a simple binary classification neural network. It is initially trained on the training dataset until it achieves sufficient accuracy. It learns to tell whether a given image is part of the true data distribution or not. Thus, once the generator synthesized candidates are evaluated by the discriminator and the resulting loss is backpropagated through both networks. This results in both neural networks learning to be better at each of its tasks. That is, the discriminator becomes more skilled at catching synthesized images and the generator produces images that are less distinguishable from images sampled from the true data distribution.

Progressive GAN There are some evident limitations of GANs in synthesizing images. For instance, GANs have trouble producing sharp images in higher resolutions. This is because it is much easier for the discriminator network to tell apart the synthesized images from the training images at higher resolutions and results in large gradients [15]. Tero Karras and his colleagues at NVIDIA have developed the training methodology of progressively growing GANs [10]. As the name suggests, the key point is growing the generator and discriminator progressively—that is, training starts at a low resolution and progressively builds up to higher resolutions. This training method sees significant improvements in training stability, increased variation, and most importantly, finer details in higher resolution images. As one can see in Figure 5, the full-resolution 1024×1024 images found in the CELEBA-HQ dataset as downscaled by factors of 2 down to the lowest resolution 4×4. The lower resolution stages excel in capturing the more general features of the data distribution whereas the higher resolution stages are better tuned for the finer details that are filled in.

Our Implementation The GAN employed by this project closely follows the progressive GAN framework. This is to make use of the excellent detail-oriented nature of the progressive training scheme. The aim was to utilize the general pattern recognition of the lower-resolution stages to pick up on the broad structure or shape of the vocalizations. Then, the higher-resolution stages will fill in the finer complexities that are more difficult to catch.

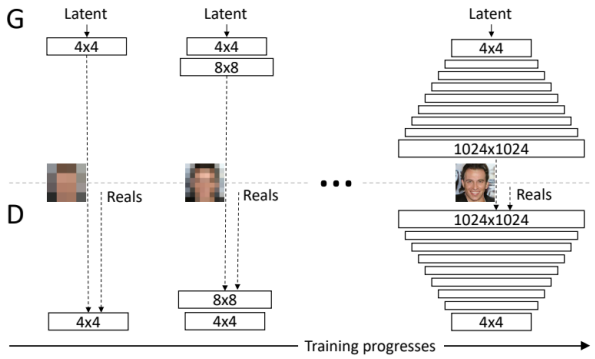


Figure 5: Progressively Growing of GANs methodology, illustrated. The top half is the generator network and the bottom half the discriminator network. As these networks progressively grow in size, the generator learns to synthesize progressively more detailed images.

Again, the "original", full resolution of the vocalization images that will be used to train the GAN is 256×600 pixels. Our training starts with both the generator and the discriminator having a lower spatial resolution of 32×75 pixels. As the training advances, we incrementally add layers to both networks, increasing the spatial resolutions to 64×150, 128×300, and finally 256×600 pixels. These downscaled versions of an individual bird call can be seen in Figure 6.

The generative network first builds off of a latent space of dimension 100. From there, it learns to map the latent space to its lowest resolution image of 32×75 pixels. Then, it progressively maps one resolution image up to its next higher-resolution image until it reaches the its full resolution. Each of these upscaling operations is a neural network model of its own which builds on its previous model's output. Hence, by building up one "block" at a time, it is effectively able to train from the latent space to the full 256×600 pixel final result.

The discriminative network works similarly, but in reverse at each stage. In the initial stage, it takes as input a 32×75 pixel image and outputs its guess of whether that input image was real or fake. In the next stage, it grows to take inputs of size 64×150 pixels and this pattern continues until it takes full size images as input and returns its classification guess.

training time GANs are notorious for extraordinarily long training times. This is another factor in which the progressively growing methodology shines. Not only does this methodology generate finer detailed images, it also takes less time to train in comparison to the traditional method. In fact, with the progressively growing methodology most of the iterations are done at lower resolutions and comparable output image quality is often obtained up to 6 times faster depending on the final output resolution and the number of growth stages.

optimizations The GAN implementation witnesses several optimizations. One such is the equalized learning rate. A not so uncommon occurrence with convolutional neural networks is with some parameters having a larger dynamic range than others, causing the parameter to take longer to reach a local minimum. This can result in a set learning rate becoming too small for parameters with a larger dynamic range while too big for parameters with a smaller dynamic range. Thus, the implementation of an equalized learning rate ensures that the dynamic range is the same for all weights, ensuring the same learning rate for all parameters in the network and thus its training stability.

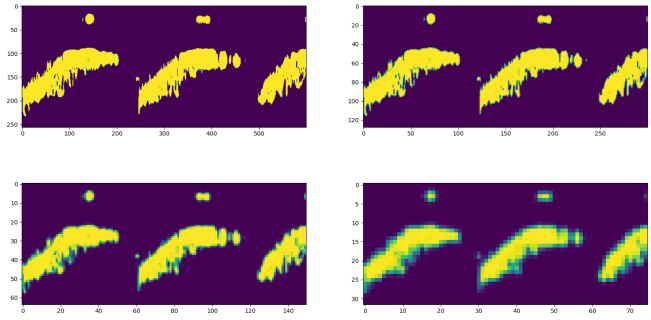


Figure 6: Plot of the full resolution image as well as its downscaled images. The top-left image is the full size image at 256×600; top-right is downscaled once to 128×300; bottom-left is downscaled twice to 64×150; bottom-right is downscaled three times to 32×75.

Hyperparameter tuning is a tricky yet necessary step to bring out the best performance from any neural network. Even with the use of neat techniques like equalized learning rate, pixel normalization, and minibatch standard deviation, it will take many training attemps to find the set of hyperparameters that work best for the GAN and dataset pair. This takes time and requires training of the GAN to some extent. The initial GAN code was written to save the model state after each resolution model has completed training. Since it would be a significant waste of time to ignore previously trained submodels, we wrote a load and training scheme to load saved models and build the missing higher-resolution models on top for training. This effectively resumes training from the saved models instead of having to restart from the lowest resolution and build up.

3.4 Audio reconstruction

The next step of our project involves developing an algorithm to convert the generated vocalization scalograms back into audio files. The wavelet transform from audio to scalogram is a form of deconstruction of the full composite signal by extraction and separation into a set of frequencies. By reconstructing the composite signal from the set of frequencies in the scalogram, we can find the "sound" of the scalogram.

As the wavelet transform is a bandpass filter, the reconstruction can be done using deconvolution or inverse filter. The reconstructed signal x_n' is the sum of the real part of the wavelet transform over all scales:

$$x_n = \frac{\delta j \delta t^{1/2}}{C_\delta \psi_0(0)} \sum_{j=0}^{J} \frac{\mathbb{R}(W_n(s_j))}{s_j^{1/2}}, \tag{5}$$

where C_δ is a constant associated with a given wavelet function ($C_\delta = 0.776$ for Morlet) and $\psi_0(0)$ is the zero Morlet wavelet.

Unfortunately, due to difficulties faced in implementation along with a time limitation, the audio reconstruction step is left incomplete as of writing this paper.

3.5 GUI

To allow our users to be able interact with our codebase, we have implemented a simple user interface using the GUI toolkit PySimpleGUI [16]. Our design focused on allowing our users to: process their data through our pipeline to create scalogram representations of selected bird vocalizations, create a dataset from the processed data for training the GAN, and the ability to train the GAN itself. Figure 7 details the layout of our GUI's organization with their individual window elements respectively.

The main menu acts as the hub for each part of our algorithm's process. It consists of a load data, create dataset, and train GAN option and includes a settings tab. Each of these buttons opens another GUI element with respect to the type of operation the user wants to run. This gives the user flexibility in being able to circumvent some steps of the process if they already have properly processed data.

In general, a user would first open the Load Data interface, where they would proceed to select their desired folder containing bird audio they would like to be processed. Once selected, the user can run the data pipeline process. From there, the user can then select the desired scalogram files they would like to package into a dataset under the Create Dataset interface .The last step would be to open the Train GAN interface, where the user can select their prepared dataset to train in the GAN.

The organization of the Load Data interface and the Create Dataset interface are the same. Users are prompted to select a folder containing the relevant data of the process they want to executed. Once selected, the user can then run the process, which will then display a window with a progress bar informing the user of how many files are left to process, how many steps in the process are left to run, and an estimated finishing runtime. Once the progress bar notifies the user that the process has been finished, the progress bar window will close and the processed data will appear in a folder in the same directory as the program.

The Train GAN interface contains the same features as the Create Dataset and Load Data interface but include an additional option to either train the GAN with a pre-trained model, or completly from scratch. This option gives the user the ability to either create a quick and dirty model or train a model more specialized to the data they are training on.

The settings window has options to change the appearance of the interface but most importantly has an option for a developer mode and a notify_run option. The developer mode enables a console window to appear in each of three process's interfaces and allows the user to debug the process if something were to go wrong. The notify_run option creates a server instance that notifies the user remotely about the progress of a given process.

4 RESULTS

This section presents the results of the project. It will first describe the results obtained as of writing this paper and explain evaluations of said results.

The project is not *complete*. That is, we have not fully processed the audio files provided by our client Professor Rouse, the GAN has not finished training at the time of writing this paper, and we have not been able to implement the inverse wavelet transform to convert the generated images back to audio for playback validation. Thus, the results we have are preliminary results.

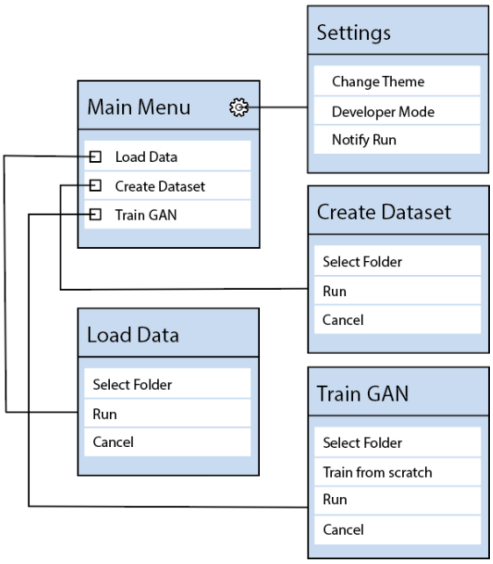


Figure 7: Diagram depicting the layout of ChirpGAN's GUI organization

Currently, out of the 286 original half-hour wav files, we have processed just over half of the pool with 160. This has taken a surprisingly long time, with the wavelet transform on the split 130-second wav files each taking approximately 15 minutes. Additionally, the flood-fill algorithm takes approximately 10 minutes to process for each exported png file.

The GAN completed training models at the resolution levels 32×75, 64×150, and 128×300. However, the model is still training and has yet to have completed training at the full resolution 256×600. In Figure 9 in the Appendix A, we show samples of generated scalogram images from the set of generator models after training. The evaluation of these generated images will be dealt in the next section 4.1.

Regrettably, we have not been able to implement the inverse wavelet transform which is responsible for the conversion of the generated scalogram images back to audio. Thus, our final results consist of the generator models that produce synthesized scalogram images.

4.1 Evaluation

A generative adversarial network is the combination of a generator model and a discriminator model in which the generator model is trained by the performance of the discriminator in differentiating between the generator's output and real images. Thus, there exists no concrete objective function or measure for evaluating model performance. This lack of an objective function makes it difficult to compare performances across different GAN models [18]. It naturally follows that there is no generally agreed upon method of evaluating GAN generator models and there is still much research in this open problem of GAN evaluation methods. Early studies on GAN models have relied on mostly qualitative evaluation [2].

Despite seemingly unscientific, one of the most common means of qualitative evaluation is via the manual assessment of the images synthesized by the generator model. This involves creating batches of synthesized images and judging the quality and diversity of the images relative to the target domain. Naturally, this method of evaluation has many limitations. Human reviewers are subjective. They must be trained to gain knowledge in the target domain, carry inherent biases, and are limited by individual review load capacity. This makes human vision evaluation an expensive task. However, this makes for a good starting point for comparison with more quantitative evaluation methods.

5 DISCUSSION

5.1 On results

Overall, the results of the trained GAN models look promising to the human reviewer. Examining the results shown in Figure 9, there is a clear increase in detail as the resolution increases. The generated images of lowest 32×75 resolution seem to feature prominently patches of calls whereas the higher resolution images fill in the missing details. It is interesting to note that there are some generated images that are faded or completely blank. These are likely due to the presence of blank or near-blank training images which account for the time frames between calls that are nothing but background noise. These will in fact come in handy when we extend this project to continuous generation of longer-duration vocalization files. That is, the GAN will learn to fill in empty spaces between distinct bird calls with background noise.

Although we did not have the time to run some metrics for quantitative evaluation of the produced images, we have included a plot of real images from the training dataset downscaled to 128×300 px in Figure 10 in Appendix A for comparison with the highest-resolution generated images produced by our GAN. We will leave the qualitative judgement up to the reader of this report.

Next, we discuss the lack of reconstruction of generated scalogram images to audio. Although the mathematics and theory behind the signal reconstruction was covered and ready, there was trouble in its implementation. We believe the problem to lie in the process of exporting the reconstructed signal into a wav file. Had there been a few more weeks to solely focus on this last step, the audio reconstruction may have been completed and we could have covered the playback and analyses of our generated bird vocalizations. Thus, the audio reconstruction is presented as the first step in section 6 in which we cover future work.

5.2 Challenges

GUI We faced a lot of trouble over the course of the GUI development process. As specified in the use-cases documentation at the beginning of the project cycle, the GUI was designed to feature a training progress bar. Although we were able to meet the requirements of a begin training button as well as a training completion notification system using `notify-run`, the overall progress monitoring system could use some help. Currently, the progress bar window becomes unresponsive for extended periods of time while a background process (any prolonged data processing function calls) is actively running. This means the GUI is not clickable and thus the user is not able to scroll up the debug output console during the entire data processing pipeline. This prompted the experimentation of implementing a functional, responsive progress bar. We ended up using threads for a multi-threaded system such that the GUI would be run in the main thread and it would spawn a worker thread in the background when the button to start the data processing pipeline is pressed. This would allow the GUI to be fully responsive while the data processing functions were actively running in the background. This would also make the message queues to be read and the display of the progress to the debug output update as the process was running. However, it quickly became clear that our GUI toolkit of choice PySimpleGUI did not support multi-threading. PySimpleGUI is a higher-level package that builds on the more prominent GUI toolkit tkinter which finds itself problematic in its lower-level interaction with tcl/tk and sends a crash signal to the main thread once the worker thread has completed and exited.

GPUs	1024×1024	512×512	256×256
1	41 days 4 hours	24 days 21 hours	14 days 22 hours
2	21 days 22 hours	13 days 7 hours	9 days 5 hours
4	11 days 8 hours	7 days 0 hours	4 days 21 hours
8	6 days 14 hours	4 days 10 hours	3 days 8 hours

Figure 8: Expected training times for StyleGAN model using the default configuration using Tesla V100 GPUs.

GAN Another source of challenges in our project involves training of the GAN models. Besides the architecture of the GAN model, there are hyperparameters that affect the performance and training rate. The effectiveness of the choice of hyperparameters cannot be evaluated until the model has completed several epochs of its training schedule and saved the model weights. Each model takes a significant duration to complete a single epoch of training and thus for each resolution of our progressive GAN, we had to train the same model with tweaked hyperparameters several times before qualitatively deciding that the chosen set of arguments satisfies our needs. Furthermore, it takes a surprising amount of time to train and the time it takes only scales up with the resolution. Figure 8 is from the StyleGAN paper which lists the expected training times for the StyleGAN model to train for a varied number of Tesla V100 GPUs [11]. Tesla V100 GPUs are each currently listed for $8,000 USD online and is designed specifically for tensor operations which accelerates training rate. In contrast, our GAN training relied on a single CPU system.

6 FUTURE WORK

This project admits many straightforward extensions:

1. *Audio reconstruction* from the synthesized scalogram images. This was the primary goal of this project which we were unable to achieve in the given timeframe. Thus the implementation of the inverse wavelet transform to convert the generated images to audio is the very next step of extending this work.
2. A *Multi-threaded GUI* is next step as the current GUI lacks a responsive progress bar window. This can be implemented

using a carefully chosen GUI toolkit which must support multi-threading without concerns for the type of operating system.

3. *Cloud-based computation* can be useful in taking the load off of the client's own computer resources and doing the computations necessary in a cloud server instance. Currently, the GUI and backend processes consume the client's computer resources which can prevent the user from using other programs during data processing or GAN trainig.
4. *StyleGAN* is the successor to the progressive GAN, also developed by the team of researchers at NVIDIA. StyleGAN makes use of latent space exploration to extract and specify which exact features of the resulting image are desired. This means a StyleGAN based project will allow the user to generate scalogram images as well as resulting bird vocalizations that satisfy their exact needs.
5. *Continuous generation* of bird vocalization would consist of a GAN that could generate a specified duration of bird calls. That is, the user would provide a desired duration and the GAN would produce a bird vocalization file that is of that duration with distinct calls as well as blank timeframes between said calls to mimic a real set of bird vocalization.
6. *Variance reduction* by means of training individual GANs on each test group can help with the training rate as well as quality of the resulting scalogram images.

REFERENCES

[1] George F. Barrowclough, Joel Cracraft, John Klicka, and Robert M. Zink. 2016. How Many Kinds of Birds Are There and Why Does It Matter? *PLOS ONE* 11, 11 (11 2016), 1–15. https://doi.org/10.1371/journal.pone.0166307

[2] Ali Borji. 2018. Pros and Cons of GAN Evaluation Measures. (2018). arXiv:cs.CV/1802.03446

[3] Gordon Lindsay Campbell and Thorsten Fögen. 2014. Animal Communication. (04 2014). https://www.oxfordhandbooks.com/view/10.1093/oxfordhb/9780199589425.001.0001/oxfordhb-9780199589425-e-013

[4] Monika Doerfler, Thomas Grill, Roswitha Bammer, and Arthur Flexer. 2017. Basic Filters for Convolutional Neural Networks: Training or Design? *Neural Computing and Applications* (09 2017). https://doi.org/10.1007/s00521-018-3704-x

[5] Chris Donahue, Julian McAuley, and Miller Puckette. 2018. Adversarial Audio Synthesis. (2018). arXiv:cs.SD/1802.04208

[6] Jesse Engel, Kumar Krishna Agrawal, Shuo Chen, Ishaan Gulrajani, Chris Donahue, and Adam Roberts. 2019. GANSynth: Adversarial Neural Audio Synthesis. (2019). arXiv:cs.SD/1902.08710

[7] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative Adversarial Nets. In *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 2672–2680. http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf

[8] Jack Parker Hailman and Millicent Sigler Ficken. 1986. Combinatorial animal communication with computable syntax: Chick-a-dee calling qualifies as 'Language' by structural linguistics. *Animal Behaviour* 34 (1986), 1899–1901.

[9] Karlis Kanders, Tom Lorimer, Florian Gomez, and Ruedi Stoop. 2017. Frequency sensitivity in mammalian hearing from a fundamental nonlinear physics model of the inner ear. *Scientific Reports* 7 (12 2017). https://doi.org/10.1038/s41598-017-09854-2

[10] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. 2018. Progressive Growing of GANs for Improved Quality, Stability, and Variation. In *International Conference on Learning Representations*. https://openreview.net/forum?id=Hk99zCeAb

[11] Tero Karras, Samuli Laine, and Timo Aila. 2018. A Style-Based Generator Architecture for Generative Adversarial Networks. *CoRR* abs/1812.04948 (2018). http://dblp.uni-trier.de/db/journals/corr/corr1812.html#abs-1812-04948

[12] Kundan Kumar, Rithesh Kumar, Thibault de Boissiere, Lucas Gestin, Wei Zhen Teoh, Jose Sotelo, Alexandre de Brebisson, Yoshua Bengio, and Aaron Courville. 2019. MelGAN: Generative Adversarial Networks for Conditional Waveform Synthesis. (2019). arXiv:eess.AS/1910.06711

[13] Yandong Li, Yu Cheng, Zhe Gan, Licheng Yu, Liqiang Wang, and Jingjing Liu. 2020. BachGAN: High-Resolution Image Synthesis from Salient Object Layout. (2020). arXiv:cs.CV/2003.11690

[14] Farrah N Madison, Melvin L Rouse, Jacques Balthazart, and Gregory F Ball. 2015. Reversing song behavior phenotype: Testosterone driven induction of singing and measures of song quality in adult male and female canaries (Serinus canaria). *General and comparative endocrinology* 215 (May 2015), 61—75. https://doi.org/10.1016/j.ygcen.2014.09.008

[15] Augustus Odena, Christopher Olah, and Jonathon Shlens. 2017. Conditional Image Synthesis with Auxiliary Classifier GANs. In *Proceedings of the 34th International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Doina Precup and Yee Whye Teh (Eds.), Vol. 70. PMLR, International Convention Centre, Sydney, Australia, 2642–2651. http://proceedings.mlr.press/v70/odena17a.html

[16] PySimpleGUI. 2018. PySimpleGUI. (2018). https://github.com/PySimpleGUI/PySimpleGUI

[17] Tara N. Sainath, Ron J. Weiss, Andrew W. Senior, Kevin W. Wilson, and Oriol Vinyals. 2015. Learning the speech front-end with raw waveform CLDNNs. In *INTERSPEECH*.

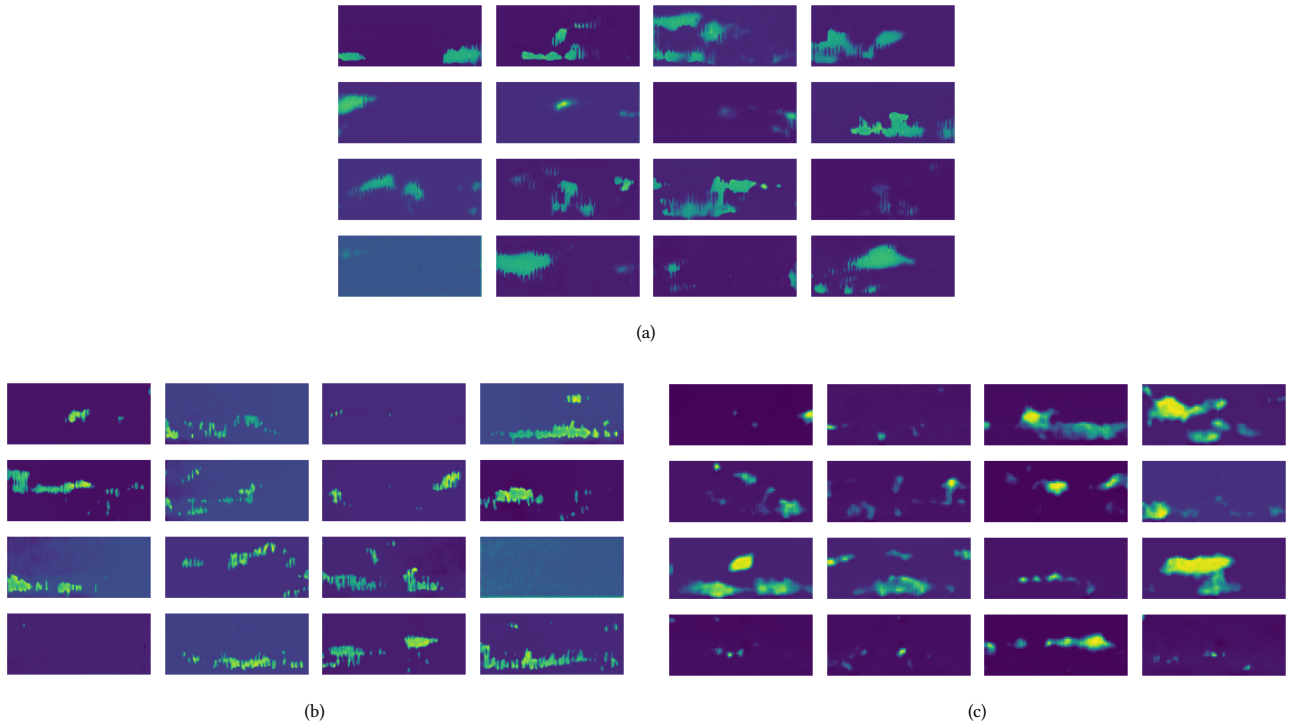
[18] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. 2016. Improved Techniques for Training GANs. (2016). arXiv:cs.LG/1606.03498

[19] Toshitaka N. Suzuki, David Wheatcroft, and Michael Griesser. 2018. Call combinations in birds and the evolution of compositional syntax. *PLOS Biology* 16, 8 (08 2018), 1–5. https://doi.org/10.1371/journal.pbio.2006532

[20] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. 2016. WaveNet: A Generative Model for Raw Audio. (2016). arXiv:cs.SD/1609.03499

[21] EDWARD O. WILSON. 1978. Animal Communication: A Summing Up. *Science* 199, 4333 (1978), 1058–1059. https://doi.org/10.1126/science.199.4333.1058 arXiv:https://science.sciencemag.org/content/199/4333/1058.full.pdf

A APPENDIX



(a)

(b)

(c)

Figure 9: Visualization of samples from the GAN models. Samples are fair random draws and not cherry-picked. (a) are sample generated scalograms from the 128×300 resolution model; (b) are from the 64×150 model; (c) are from the 32×75 model.

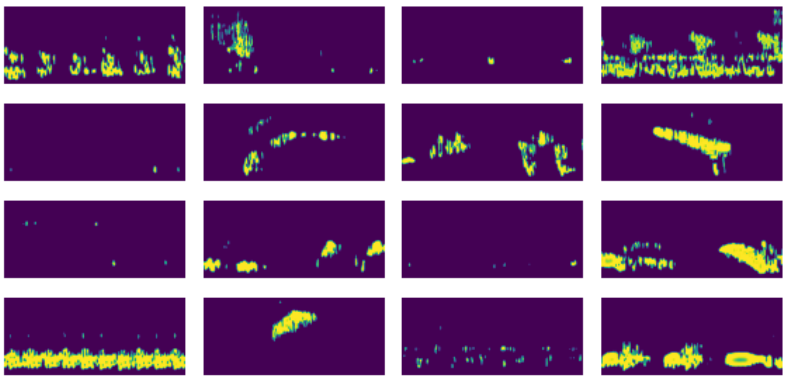


Figure 10: Plot of a sample of real scalograms of bird vocalization files that were part of the training dataset. These scalograms have been downscaled to the resolution 128×300 for comparison with the highest-resolution images that were produced by the GAN