Birdsong Generation with Generative Adversarial Networks

Jack Ruder
jack@ruder.town
University of Puget Sound
Tacoma, Washington, USA

Lucas Gover
lucasgover@gmail.com
University of Puget Sound
Tacoma, Washington, USA

Bennett Baynham
bebaynham@gmail.com
University of Puget Sound
Tacoma, Washington, USA

ABSTRACT

This paper explores the use of generative-adversarial neural networks (GANs) to generate artificial birdsong. We analyze two types of GAN architecture, recurrent neural networks (RNNs) and transformers, for their ability to model sequential data in order to generate convincing bird vocalizations. Our GAN model is trained on birdsong scalagrams which leverage the Continuous Wavelet Transform for more precise time and frequency measures. Through experiments, we evaluate the efficacy of both architectures and investigate how GANs can help create realistic birdsong that captures many of the features of real birdsong.

KEYWORDS

gan, neural networks, generative neural network, transformer, wavelet, inverse continuous wavelet transform, audio generation, birdsong

1 INTRODUCTION

Birdsong is an incredible tool for researchers all over the globe to investigate the mysteries of language, vocalization, and auditory processing. Birdsong demonstrates complex and dynamic patterns and a robust ability for vocal learning, which is critical for the evolution of speech [1]. Research on birdsong has enabled us to gain a better understanding of avian vocal behavior and the evolution of language. Birdsong can provide researchers with valuable insight into the subtle intricacies of how we communicate with each other.

This paper explores the use of generative-adversarial neural network models for generating realistic artificial birdsong. Recent advances in artificial intelligence and deep learning technology have made it possible to develop systems that emulate and even surpass what can be achieved by existing approaches for bird generation. We discuss the use of such novel neural network techniques for generating birdsongs . Additionally, we train our model on birdsongs transformed using the morlet wavelet, allowing us to cut against the Heisenberg uncertainty principle. This data is then transformed back into waves after generation. We present a series of experiments highlighting the effectiveness of the proposed neural network methods, demonstrating the potential of artificial intelligence-based approaches for creating viable and naturalistic birdsong recordings.

Birdsongs have been found to contain a highly structured syntax, despite emerging from the complex interactions between the bird's nervous system and the syrinx — a highly non-linear vocal organ [1]. The syrinx is the sound-producing organ located between the lungs and the vocal tract of a songbird. It consists of vibrating membranes called labia which oscillate when the air from the lungs exerts a force on them. The syrinx produces sound waves, which are then filtered through the vocal tract to produce its song [12]. The syrinx can produce two notes at once due to its two bronchial tubes, with each controlled independently [25].

Birdsong is produced for other birds to hear. Most bird species have the greatest sensitivity to sounds around 1 - 4 kHz, though their full range is comparable to the human hearing range [2]. This isn't true for all birds; for example, pigeons can hear infra-sound, with the typical pigeon being able to hear 0.5 kHz, allowing them to detect distant storms and even volcanoes [2]. Birds use birdsong to defend their territory, identify an individual bird, and attract mates. Each species of bird has its own distinctive set of short calls and songs that make up its repertoire. Calls tend to be shorter and used to communicate a threat to a nearby bird [25].

To generate birdsong, we use a Generative Adversarial Network (GAN). A GAN is a powerful machine learning model based on a generative model and a discriminator network trained against each other. The generative model creates samples given a noise vector, while the discriminator network evaluates these samples and tries to guess whether they are real or generated [8]. The models are trained iteratively, where the generative model tries to produce samples that are as realistic as possible, and the discriminator attempts to identify which are real and which are generated. For GANs, both the discriminator and generator use neural networks to learn to optimize their task.

Audio data is recorded as a series of amplitude values over time. This type of data could be difficult for a Generator to learn to replicate, so most researchers tend to use some transformation on the data to make it into a more comprehensible format. The windowed Fourier Transform is most commonly used in the literature. However, the windowed Fourier Transform cannot concurrently give good information about both frequency and time. To offer more accurate results, we use a Continuous Wavelet Transform. By expressing data as a series of wavelets modulated by a scaling factor and shifted in time, the Continuous Wavelet Transform enables more precise time & frequency measures.

Because bird vocalizations have variable lengths, and the frequency a bird produces depends on the sequential placement of the current timestep among other timesteps, we generate bird vocalizations with the types of neural networks that can model sequential data: Recurrent Neural Networks and Transformers. Recurrent Neural Networks (RNNs) are neural networks designed to process sequential data using a combination of feedback loops and traditional neural networks [11]. The network (and its weights) are repeated for each timestep, allowing the network to "remember" what happened previously in the sequence. In our case, this "memory" is used to model birdsong – by having multiple timesteps, the model can improve its understanding of a bird's vocalization and produce an accurate sample.

The transformer architecture was introduced by Vaswani et al. 2017 [28] an improvement to recurrent architectures by making use of a mechanism called attention. Attention attempts to model the relationship between words in a sentence by keeping track of

every combination of token relationships. This allows a transformer network to "focus" on certain parts of the sequence at certain times. As a result, transformers should have the capacity to generate birdsong more accurately than a standard RNN, as they can learn which parts of the sequence are most important through a more precise representation.

We experiment with creating two different GAN models: one using RNNs the other using Transformers. As the models are trained iteratively, they learn more about birdsong, making the samples increasingly sophisticated. This GAN approach is particularly powerful for producing birdsong, as it allows the models to create realistic samples while also providing a high degree of variability. By using this approach, we attempt to generate convincing birdsong that captures many of the features that characterize real birdsong.

2 BACKGROUND

2.1 CWT in Audio Analysis

Within audio analysis, the Continuous Wavelet Transform is increasingly being used because it more accurately describe the frequency at each timestep than the windowed Fourier transform. The Continuous Wavelet Transform is used by Kim et al. 2020 [15] to convert EEG signals into time-frequency images, which can be used as an input to a convolutional neural network (CNN). The CWT takes advantage of the signal's magnitude and phase information to extract features that are important to EEG signal classification. The signal's real part and imaginary part are extracted separately and then combined to create an image input that encodes both magnitude and phase information. This allows the neural network to consider both properties when classifying EEG signals. The use of this technique results in more accurate classification performance, increasing the reliability of the system. Smith and Kristensen 2017 [23] used a Convolutional Neural Network to analyze mouse vocalizations. To achieve this, they employed the Continuous Wavelet Transform to generate a higher-quality spectrogram than the conventional STFT approach, providing them with a way to capture the frequency components of the calls and create informative spectrograms.

2.2 Convolutional-Recurrent-Neural-Networks

The use of Convolutional Networks together with Long-Short Term Memory (LSTM) layers has become increasingly popular in the task of time series classification due to its advantages in accuracy, minimal pre-processing requirements, scalability with a larger amount of data, and its ability to calculate features on its own. A 2019 study by Karim et al. [13] used a Convolutional-RNN with LSTM layers and achieved a dramatic increase in accuracy over traditional classification methods and made time series analysis more accessible. The paper's authors build upon the conventional LSTM model used for time series classification by adding a convolutional neural network (CNN) to the LSTM stack. The CNN layer allows the system to look for certain features within a given dataset and use those features to refine its predictions further. CNNs are usually aimed at image processing, but the authors demonstrate that CNNs are effective for time-series data as well.

Gupta et al. 2021 [9] combined Convolutional Neural Network (CNN) with a recurrent neural network (RNN) to develop a hybrid architecture that they used to classify birds more accurately. To implement the hybrid architecture, Gupta et al. [9] first split each audio recording into small equal-length chunks, extracted mel spectrograms (visual representation of sound frequencies) from each chunk, and fed the mel spectrograms as inputs to the CNN model. The CNN model then extracted lexical features from the mel spectrums, i.e., it recognized certain aspects of sound that are recognizable as belonging to a particular bird species. These lexical features were then fed as inputs to the RNN model, which processed them further to recognize longer time-sensitive patterns, such as the presence of multiple calls. The combined architecture then classified the bird recordings accordingly, leading to higher accuracy than that achieved by previous models.

2.3 GANs in Audio Generation

Audio generation and analysis is a growing field of study within artificial intelligence research. Much of the research has been into the study of Text-To-Speech models, where there has been dramatic progress in recent years. The study of artificial music Generation has also grown in popularity in recent years. Many works focus on Defining Input Representation and a set of generative models that can produce "realistic" music from a given input spectrogram. In their 2019 study, Engel et al. [6] introduce an audio synthesis technique that uses Generative Adversarial Networks (GANs) to generate waveforms from spectrograms. Their end result was higher quality, more globally consistent, and more locally coherent sound than previously attainable with WaveNet, the current standard of audio waveform synthesis. They achieved this result by including a second channel in their input spectrograms; a phase-derivative channel. Not only were Engel et al.'s GANs able to outperform existing WaveNet baselines in automated and human evaluation metrics, but they were also able to synthesize audio several orders of magnitude faster than their autoregressive counterparts. This suggests that using GANs and frequency-phase spectrograms for audio generation is effective. These results suggest not only a revolutionary new way of creating high-fidelity audio but also a more efficient and ultimately better way to do it [6]. Some of the most successful models in this space use deep learning methods such as Generative Adversarial Networks and Recurrent Neural Networks. Mogren 2016 [19] uses a Convolutional-Recurrent-Neural-Network GAN to generate variable-length MIDI files. This model generates notes sequentially, with each note informed by the notes that were generated in previous time steps. Mogren [19] found that the C-RNN-GAN model can generate high-quality music samples.

There is only limited bio-acoustic research into generation, possibly because there's no clearly defined metric to determine the quality of the generated sounds [24]. Bhatia 2021 [3] tests various techniques of audio generation to generate birdsong. The first approach uses the *Wavenet vocoder*. The *Wavenet vocoder* is a technique used to generate birdsong audio, which involves training a feature prediction network for frame-level ground truth-aligned predictions and training wavenet with a teacher-forcing method to predict conditioned frame-level sample waveforms. The second approach uses a *wavenet autoencoder* model to generate birdsong

audio. This model learns temporal encodings of audio data, removing the need for external features. It then uses a vanilla WaveNet decoder to upsample the encoded data and generate birdsong. Bhatia's final approach, *Parallel Wavegan*, uses a generative adversarial network to generate realistic birdsong audio by jointly optimizing a multi-resolution spectrogram and an adversarial loss function. All these approaches generate only a fixed-length birdsong.We extend this literature in 2 significant ways. First, we use sequential network architecture, which are able to better represent the fact that sound produced at each time-step is informed by the previous time-steps. And second we use the Continuous Wavelet Transform to get more accurate frequency readings at each timestep.

3 PREPROCESSING

3.1 Scalogram Generation

3.1.1 Spectrograms. Audio classification and generation tasks typically rely on spectrograms, usually produced from the windowed Fourier transform. Spectrograms are a time-frequency representation of a waveform that show the strength of a signal over time for a range of frequencies. In generative tasks, Mel-scaled (logarithmic, targeting frequencies for human hearing) spectrograms are especially popular since popular text-to-speech models are developed with the goal of outputting a Mel-spectrogram. The windowed Fourier transform works by iteratively computing the Fourier transform over a small portion of a waveform. For each window, a representation of the underlying frequencies is obtained; when the representation for each window is viewed in sequential order we obtain a spectrogram. The primary disadvantage of the windowed Fourier transform is that the window has a constant width. Higher frequencies have shorter periods, and a large window will fail to accurately identify when these frequencies are present. Likewise, low frequencies have long periods and therefore do not need a small window to capture their temporal locality. This is known as the Heisenberg uncertainty principle, which tells us that in analyzing a signal we must choose between frequency resolution and time resolution. As a result, any window width chosen for a windowed Fourier transform will be a compromise, especially when we need to capture the high frequencies present in birdsong.

3.1.2 Continuous Wavelet Transform. The Continuous Wavelet Transform (CWT) provides a solution to this conundrum by simultaneously providing frequency and time information. The CWT is based on wavelets, which are oscillating signals typically having zero mean and time-frequency localization. To capture a certain frequency, a mother wavelet is stretched in time. Then the wavelet is shifted over the entirety of the signal, and at each time step a correlation is computed by means of convolution. For a discrete time signal x, the CWT is computed as:

$$w(a,b) = \sum_{t=0}^{N-1} x_t \psi^* \left[\frac{(t-b)}{a} \right], \tag{1}$$

where ψ is a wavelet function (known as the mother wavelet), a is a frequency-associated scaling factor, and b is a time-shift [26]. We refer to $\psi \left[\frac{(t-b)}{a} \right]$ as a daughter wavelet since it is just a scaled and shifted version of the original wavelet function. In practice, the above formula is inefficient, we can reduce the complexity by performing the transform in fourier space. In fourier space, we have:

$$w(a,b) = \sum_{t=0}^{N-1} \hat{x}_t \hat{\psi}^* \left(a\omega_t\right) e^{i\omega_t b}, \tag{2}$$

which is quickly computed with an inverse fourier transform [26]:

$$w(a,b) = \mathrm{IFFT}(\hat{x}_t \hat{\psi}^* \left(a\omega_t\right)). \tag{3}$$

ω_t is known as an angular freqency, computed by

$$\omega_t = \begin{cases} \frac{2\pi t \cdot \mathrm{hz}}{N} & t \leq \frac{N}{2} \\ -\frac{2\pi t \cdot \mathrm{hz}}{N} & t \geq \frac{N}{2}. \end{cases} \tag{4}$$

When we compute the wavelet transform for a range of scales over an entire signal, we obtain a scalogram, like in figure 1. A scalogram is visually very similar to a spectrogram, but it cheats the Heisenberg uncertainty principle by providing the right amount of time localization for all frequencies present. Since we are working

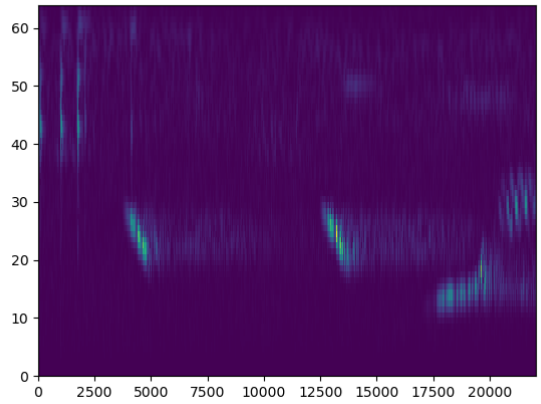

Figure 1: A CWT-generated Scalogram depicting 3 seconds of birdsong. Y axis is representative of increasing frequencies. X axis is time. Warmer colors indicate presence of a signal.

in the domain of animal sounds we follow Smith and Kristensen [23] and use the complex morlet wavelet, which is defined in fourier space as

$$\hat{\psi}(a\omega) = \begin{cases} \pi^{-1/4} \exp\left(\frac{-(a\omega - 2\pi)^2}{2}\right) & \text{if } \omega > 0, \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

Usage of a complex wavelet improves the recovery of information when the phase of a signal's frequency does not align with the wavelet's phase. The output of a CWT with the complex morlet wavelet has the primary advantage of providing an imaginary channel in a scalogram. For the purposes of audio generation, obtaining phase in a generated scalogram will allow a more realistic reconstruction with fewer artifacts. We feed a tight range of scales for our wavelets, representing 64 different frequencies logarithmically spaced.

3.1.3 Inverse CWT with a Morlet Wavelet. A lack of use of the complex morlet wavelet in vocoders or other audio processing techniques can be attributed to the difficulty of inverting the CWT. While there is a well-known inverse for the CWT, the inversion requires any wavelet used to satisfy the admissibility condition. The morlet wavelet does not satisfy this condition, and as a result is avoided in scenarios where inversion is needed. Postnikov et al. [20] proposes an analytical reconstruction method for the classic complex morlet wavelet. We use a slightly modified version of the morlet wavelet, but our inversion formula only differs by a multiplication of π. Our (slightly) modified inversion is

$$f(t) = \frac{\pi}{\sqrt{2\pi}} \text{Im}\left[\int_0^\infty \frac{\partial w(a,b)}{\partial b} da\right], \tag{6}$$

where $w(a,b)$ are coefficients from the wavelet transform with the complex morlet wavelet as previously described. Ideally, we would normalize our wavelets in a transform to be able to produce scalograms that are normalized across different scales. Unfortunately this reconstruction does not work when wavelets are normalized to have unit energy, so for current experiments we leave our wavelets un-normalized.

Practically, this inversion does very well, see figure 2. We notice no loss in perceptual audio quality after feeding a waveform through our CWT and then inverting it. This allows us to recover audio from our generated scalograms, a crucial step in our process. The downside of trying to generate these scalograms is their size.

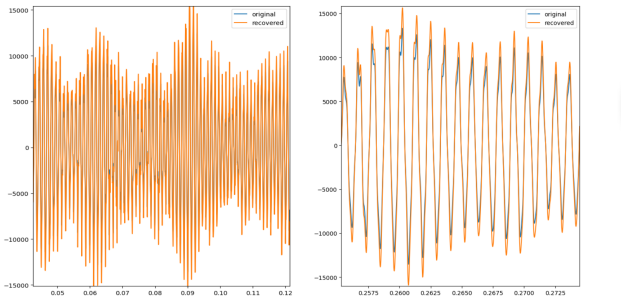

Figure 2: Here we have a bird chirp which passed through our CWT, and then was fed through the inverse. Though the peaks here are not perfect, they are almost exact. The operations producing this reconstruction used 64-bit precision.

For any given signal, only frequencies that are less than half the sample rate of the original signal can be detected. The highest frequency possible to detect is known as the Nyquist frequency. With our inverse transform, we experimentally observed that amplitude recovery is poor when there are frequencies present that surpass 1/3 of the sample rate. As birds produce and hear 8k+ frequencies, we cannot easily dip below a standard 22kHz sampling rate. This makes the generation of scalograms very expensive and difficult, likely a reason we do not see this task attempted often in literature.

3.2 Augmentation

Finding clean training data for neural networks can be difficult. Even when data is obtained, a small number of samples can easily be memorized by large networks such as ours. To reduce overfitting and increase the diversity of the Generator's output we apply a random set of augmentations to each of our waveforms before they are passed through a CWT. The waveforms used should not fundamentally change the audio; we employ frequency-preserving time stretching, pitch shifting, noise reduction. The effects of these transforms should produce affine effects on our scalogram images. Many classification tasks add input noise, we avoid this in favor of providing our generator clean samples (though noise is employed in the training process).

3.3 Polar Transformation

After audio samples are passed through augmentation and a CWT transform, the resulting scalograms are in cartesian complex form. These are converted into magnitude and phase, which are more representative of the underlying signal. In fact, single channel spectrograms/scalograms are interpreted by viewing the coefficients as amplitude. For each complex $w(a,b)$ in our scalogram,

$$A = ||w(a,b)||, \tag{7}$$

$$\theta = \text{atan2}(w(a,b)), \tag{8}$$

where atan2 is the 4-quadrant arctangent. It is crucial to note that the domain of $atan2$ is $[-\pi, \pi]$. The phase of a signal is not restrained in any way: this means that phase discontinuities (2π jumps) are present in our resulting signal as an artifact of $atan2$. We rely on simple phase unwrapping from [27], applying the unwrapping process across each frequency band indiviudally.

Engel et al. 2019 [6]suggests for their classification techniques to use the derivative of phase, instantaneous angular frequency. Unwrappped phase is difficult to normalize since it is unbounded and periodic–this makes the generator's task difficult. Using instantaneous angular frequencies as a non-periodic value will hopefully help the generator predict a realistic combination of amplitude and phase.

3.4 Normalization

We choose a global normalization scheme for our neural network inputs. Based on the global mean and variance of scalograms generated by our data, for each scalogram we normalize amplitude and instantaneous angular frequencies separately to a range of -1 to 1, matching the range of a Tanh activation function.

4 TIME-TRANSFORMER GAN

Our proposed architecture makes use of transformers by feeding time series vectors of frequencies in temporal-order to a transformer to capture long-range context of audio signals. This is similar to work done with the Audio Spectrogram Transformer (AST), which feeds flattened square patches of a spectrogram as transformer input [7]. In Gong et al. 2021 [7], the use of patches is meant to help capture spatial structure of the spectrogram as a whole, but they observed that feeding rectangular, temporal-sequenced patches leads to higher performance in classification. A popular architecture for images, the Vision Transformer described in Lee et al. 2021 [16],

has a very similar structure and achieves state-of-the-art results compared to traditional CNN architectures.

Within GANs, the use of transformers has been somewhat limited: We know of no architectures that use transformers for audio generation. In image generation there are many techniques that have emerged in the past few years, our approach is most similar to ViTGAN, which uses a ViT in both the generatior and the discriminator. ViTGAN proposes several improvements to the transformer architecture that we will make use of in our model. Notably, to improve lipschitz continuity in the discriminator, ViTGAN introduces L2 normalized attention for the transformer-encoder, and an improved spectral normalization scheme [16, 18].

An additional modification to the transformer is needed to generate audio. The traditional layer normalization used in transformer architectures normalizes inputs across multiple channels. With images, this is not an issue because the separate channels each are representative of a different color; they have similar underlying distributions. In this case there is one channel for amplitude and another for instantaneous angular frequency, normalizing both together is problematic since they are distributed differently. To solve this issue, a flattened patch needs to be reshaped to recover channel as a dimension. Then, layer normalization is applied across each channel individually before being re-flattened. We will call this Channeled Layer Normalization (CLN). Following Lee et al. 2021 [16], we additionally make use of self-modulation as described in Chen et al. 2019 [5] to improve the generator's function and to control labels in both the generator and discriminator. This results in the transformer-encoder architecture seen in figure 3.

For this architecture, we expect an input/output scalogram of size $\mathbb{R}^{B\times 2\times W\times H}$, where B is batch size, $W = T \cdot hz$ is the number of time samples, and H is the number of frequencies. Again, one channel contains CWT amplitudes, the other contains instantaneous angular frequencies. Due to the high sample rate of audio, it is impractical to use a sequence length of W within a transformer architecture, since the memory required is $O(W^2)$. In the discriminator, this means we downsample our scalograms before feeding them to a transformer-encoder. In the generator, we perform latent operations to produce a downsampled scalogram before upsampling to the final size. Likely this is not a big issue; the number of features in time of an audio signal should not be as high as the sample rate required for bird audio. By downsampling the time dimension in the discriminator by a factor of 128, a sample rate of 22050 is representative of 172 sets of frequencies per second. In the AST architecture, spectrograms contained 100 samples per second so we should be confident that we have enough information to distinguish images as real or fake. To give the generator more strength and avoid computationally expensive upsampling, we limit the time downsampling of the generator to a factor of 32.

4.1 Generator

First, our generator samples a noise vector $\mathbf{z} \in \mathcal{N}(0, I_{1024}) \in \mathbb{R}^{1024}$. We take $\mathbf{z}$, feed it through a fully connected layer, then reshape to obtain $\mathbf{w} \in \mathbb{R}^{H\times W\times H\cdot C}$. Similar to Lee et al. 2021 [16], we replace the layer normalization layers of the traditional transformer-encoder with self-modulating CLN, and use $\mathbf{w}$ to control the self-modulation.

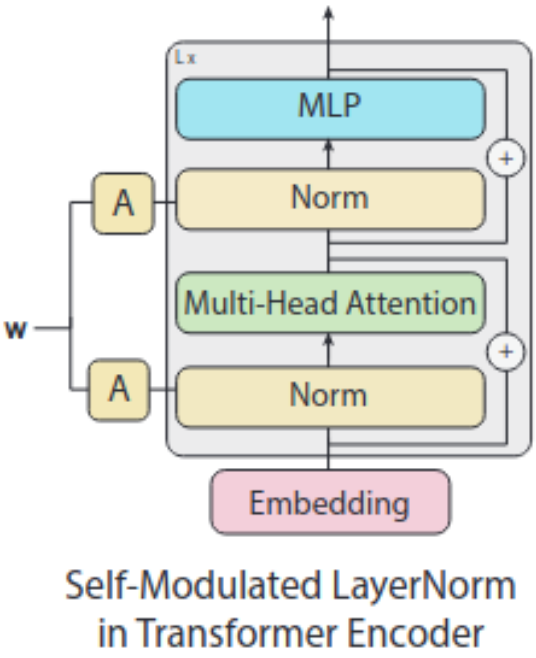

Figure 3: Transformer architecture using self-modulation as suggested by [16]

We expect our transformer-encoder to output tokens in temporal order, so we use the original positional encoding presented in Vaswani et al. 2017 [28] as the transformer's input. Next, to extract the information encoded by the transformer layers we feed the transformer's encoded output through a MLP, and next a sin activation function [16, 22]. The result at this stage is esentially a downsampled scalogram. The scalogram is upsamlped to the final size using stride 2 deconvolutions, passed through a stride 1 convolution layer, and finally output through a Tanh activation.

4.2 Discriminator

The discriminator used closely follows the AST architecture. An input scalogram of size $\mathbb{R}^{B\times 2\times W\times H}$ is passed through a series of stride 2 convolutions to a size of $\mathbb{R}^{B\times 2\times W'\times H}$, where $W' = \lfloor \frac{W}{128} \rfloor$ is the downsampled time dimension. The resulting tensor gets reshaped to $\mathbb{R}^{B\times W'\times 2H}$. We add positional encoding and concatentate a learnable class token to the sequence of frequency vectors before feeding to a transformer-encoder. To incorporate labels, we again replace the layer normalization with self-modulating CLN, and use the label vector to control self-modulation. Just as in AST, we use a MLP layer on the class token for a final classification, outputting a prediction via a Tanh activation. We apply the improved spectral normalization in all applicable layers, and use L2-normalized attention to constrain the lipschitz constant of the discriminator.

4.3 Training

Our network as described does not train easily; this is where there is still much progress to be made. The following training tricks have made significant contributions to improving traning stability.

4.3.1 Equalized Learning Rate. As with all GANs, training is an arduous process where convergence is not guaranteed. A big issue is that the job of the transformer layers is much more difficult that the upsample/downsample layers. This means that for any given learning rate, the rate might be too high for one part of the architecture while too low for another. To mitigate this impact we make use of an equalized learning rate, presented in Karras et al. 2018 [14]. The equalized learning rate rescales weights at runtime by dividing the weights by the constant from He's Initializer [10]. With this adjustment, weights can be initialized from a $\mathcal{N}(0,1)$ distribution, and we can ensure that each layer trains at the same rate. We found that weight normalization has been crucial to improving training stability, and the equalized learning rate is an effective improvement over dividing the weights by their standard deviation at runtime.

4.3.2 Cost Function. For our cost function, we choose to use Relativistic Averaging Least Squares GAN. In Zhang et al. 2020 [29]relativistic GAN is proposed as a cost function that meaningfully improves training stability over other alternative cost functions, for a wide variety of normalization techniques and a wide variety of tasks. Relativistic GAN edits the discriminator loss to measure whether a given real image is "more real" than a given fake image. The generator aims to produce fake images that are more real than a given real image. If f is a non-saturating loss function, then for discriminator classification C, real samples $\mathbb{P}$ and generated samples $\mathbb{Q}$, the relativistic GAN formulation is

$$\mathcal{L}_D^{RGAN} = \mathbb{E}_{(x_r,x_f)\sim(\mathbb{P},\mathbb{Q})}\left[f(C(x_r) - C(x_f))\right], \tag{9}$$

$$\mathcal{L}_G^{RGAN} = \mathbb{E}_{(x_r,x_f)\sim(\mathbb{P},\mathbb{Q})}\left[f(C(x_f) - C(x_r))\right]. \tag{10}$$

$$\tag{11}$$

With a smaller batch size, it is not too expensive to use Relativistic Averaging GAN [29]. Relativistic Averaging GAN makes the modification that the discriminator should estimate the probability that a real image is more real than the average realness of a batch's fake images. This is most clearly stated mathematically: With a least squares cost function, our loss becomes

$$\mathcal{L}_D^{RaLSGAN} = \mathbb{E}_{x_r\sim\mathbb{P}}\left[\left(C(x_r) - \mathbb{E}_{x_f\sim\mathbb{Q}}C(x_f) - 1\right)^2\right] \tag{12}$$

$$+ \mathbb{E}_{x_f\sim\mathbb{Q}}\left[\left(C(x_f) - \mathbb{E}_{x_r\sim\mathbb{P}}C(x_r) + 1\right)^2\right], \tag{13}$$

$$\mathcal{L}_G^{RaLSGAN} = \mathbb{E}_{x_f\sim\mathbb{Q}}\left[\left(C(x_f) - \mathbb{E}_{x_r\sim\mathbb{P}}C(x_r) - 1\right)^2\right] \tag{14}$$

$$+ \mathbb{E}_{x_r\sim\mathbb{P}}\left[\left(C(x_r) - \mathbb{E}_{x_f\sim\mathbb{Q}}C(x_f) + 1\right)^2\right]. \tag{15}$$

$$\tag{16}$$

Note that the complexity of this loss function is $O(B)$ since each image in a batch is required to pass through the discriminator. With a batch size of $B = 1$, the above cost functions reduce to RLSGAN. In testing different hyperparameters for our model we favor the use of non-averaging relativistic GAN with a higher batch size to avoid the computational penalty of RaLSGAN. RaLSGAN shows notably higher training stability with our model, but we avoid have avoided it thus far, since we are still evolving our architecture and training routine. Once we see better results from this architecture we can consider swapping in RaLSGAN for increased overall quality of generation.

The choice of a least squares cost function helps to avoid the issue of vanishing gradients and increase stability [17]. Though Karras et al. 2018 [14] suggests that LSGAN is less stable than Wasserstein GAN with a gradient penalty, Lee et al. 2021 [16] states that introducing a gradient penalty halts their training with the transformer architecture. We find similar results here, with immediate mode collapse of the generator when introducing a gradient penalty.

To help LSGAN, we draw another trick from Karras et al. 2018 [14] that penalizes the discriminator when it's output approaches 1. We can add multiplicative noise to each input layer of the discriminator, where the magnitude of the noise is determined as an exponential moving average of the discriminator's output. This technique functions similarly to label smoothing (for example, changing the discriminator's target to 0.9), but is a much more adaptive technique. At the moment we use both label smoothing and multiplicative noise, though it is unclear if mixing these techniques is detrimental. The moving average here is the same as described in Karras et al. 2018 [14],

$$\sigma_{\text{noise}} = 0.2 \cdot \max(0, \hat{d}_t - 0.5)^2, \text{ where } \hat{d}_t = 0.1C(x_{\text{prev}}) + 0.9\hat{d}_{t-1}. \tag{17}$$

4.3.3 Consistency Regularization. Again taking inspiration from Lee et al. we choose to implement Consistency Regularization [29]. Consistency Regularization relies on the use of an augmentation transform to introduce an additional loss term. The loss term is computed as

$$\mathcal{L}_{cr} = ||C(x) - C(T(x))||^2, \tag{18}$$

where $T(x)$ is an augmentation transform. In this case we compute an augmented scalogram by first applying additional augmentations (pitch shift and time shift) to the original waveform. Next, the resulting waveform is fed through the same processing steps to produce a scalogram as described in section 3.2. This loss term is added to the discriminator's loss, so that $\mathcal{L}_D^{cr} = \mathcal{L}_D + \lambda\mathcal{L}_{cr}$. In early stages of training, our consistency regularization is extremely high due to how difficult our GAN's task is. Instead of the conventional constant of $\lambda = 10$, we instead choose to start at $\lambda = 1$ and linearly grow λ to 10. Scheduling λ in this way reduces the severity of the penalty in the initial training epochs, but ramps up the penalty as the discriminator begins to understand its task.

4.4 Results

This architecture is still a work in progress. Training is generally slow, with the a model producing 1 second of song requiring roughly 15 minutes per epoch using RaLSGAN and a batch size of 8 on an RTX 4090. A good metric for birdsong scalograms should be developed to quantitatively evaluate any results, but we rely on visual scalogram patterns and perceptual audio quality. No audio

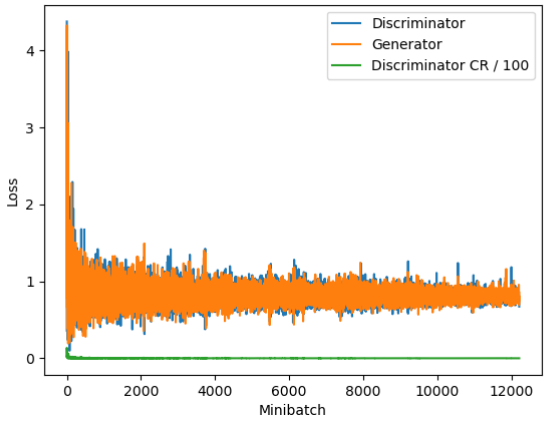

Figure 4: Loss from our most recent training run, stopped at 9 epochs. Similar loss values indicate discriminator underperformance. Notice however, that the networks still converge towards an equilibrium.

samples have been produced that sound like birdsong, though a few scalograms during training started to show convincing results until training collapsed. Examples of common collapse in our generator can be seen in Appendix A. Figure 4 shows that our training techniques are able to produce converging networks, this is a big improvement over our initial iterations. With moderately recent changes and developments in the model, it is possible that a long training run will produce good results.

5 CRNN-GAN

For our other model, we primarily used recurrent layers and convolutional layers to form the Generative Adversarial Network (GAN). A primary motivation for the approaches taken with this model is its ability to produce variable length signals without padding, with the output length able to be determined by the user. This model is largely inspired by a paper from Tampere University of Technology, Cakir et al. 2017 [4], which outlines their construction of a convolutional recurrent neural network used to classify bird vocalizations. Their classifier takes log-magnitude spectrogram input which is first passed through a series of convolutional and max pooling layers, eventually shrinking the frequency dimension to one. The output of this process is then passed through several recurrent layers which preserve dimensionality. A temporal max pooling layer is then used to prepare the input for a final, fully connected layer which gives the output classification. At the time of its release, their model achieved an impressive second place standing in the Bird Audio Detection challenge, scoring only half a percent behind the first place winner by an area under the curve (AUC) metric.

Of course, Cakir et al. [4] describe the implementation of only a classifier, designed to discriminate whether fixed-length audio recordings contain bird vocalization. Our model, on the other hand, is designed to solve the not related problem of producing artificial bird vocalizations. Since classification is half the process of training a GAN, we have in large part used their design for our discriminator. Further, and in traditional fashion, we used the building blocks that their paper lays out to design our generator, which resembles our discriminator with its steps in reverse. Our selection of their classifier as a template is partly due to its success in the Bird Audio Detection (BAD) challenge, but also due to its structural flexibility when it comes to signal length. Convolution, upsampling, and pooling layers all have the inherent ability to perform on inputs of various sizes, and recurrent layers work iteratively along some axis. It is likely that the authors of the paper were aware of this functionality, given their use of a max pooling layer, but no mention of it is made in their paper, possibly because the BAD dataset uses signals of only a single length. Other authors like Karim et al. [13] also use CRNNs, and find them to be highly effective, suggesting they are a good candidate for our model. We have extended these ideas to a convolutional, recurrent generator, which we have not found examples of in the literature, but may still exist.

5.1 Discriminator

This model's discriminator takes the real and imaginary output of the continuous wavelet transform as two channels. This input is of size HxWxD. H is the height of the input, which is the number of frequencies convoluted in the CWT. W is the width of the input, or the number of samples in the signal. W does not have a determined size. D is the depth of the signal, which starts as two, the real and imaginary values, and grows over time through the convolution layers much like traditional convolutional RGB image discriminators. Alternating ReLU activation convolutional layers and non-overlapping max pooling layers take this input, with the pooling layers shrinking the H dimension, and the convolutional layers increasing the D dimension. After the last convolutional layer, the dimensionality is 1xWxD. This output is reshaped as DxW, where each channel of depth is stacked over the width dimension as a two dimensional tensor. This tensor is then fed into several GRU layers, keeping each iteration of predicted output and preserving the shape DxW. The resulting DxW tensor is globally max pooled over the variable size W dimension, resulting in a Dx1 shaped tensor which will be the same size regardless of the input signal length, Therefore, a fully connected layer with a TanH activation function can be used to compute the classification of the signal as either real or generated.

5.2 Generator

The generator resembles the discriminator in reverse, but there are complications. The input to this neural network is of shape 1xW, which is variable and will be the desired length of the output signal. This noise is put through a gated recurrent unit layer, which upscales the tensor to the shape DxW. From here several more dimension preserving GRU layers are used, each of which is bidirectional, and merges the two iterative directions through addition. The DxW vector is then reshaped as HxDxW, where H is 1, and from here the process resembles more traditional convolutional generator models. The HxDxW tensor passed to alternating upsampling and convolutional layers. The upsampling steps are all performed by all non-overlapping, k-nearest neighbor layers, increasing the H dimension, while convolution decreases the depth

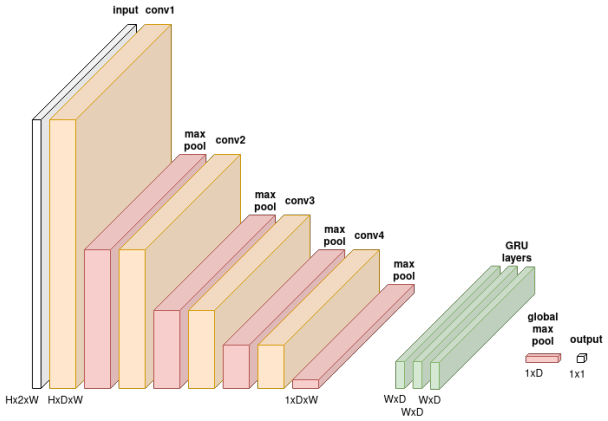

Figure 5: CRNN-GAN Discriminator

of the tensor. After the final convolutional layer, the tensor reshapes to HxWxD, the shape of each datum of the dataset. Throughout the internal processes of both the discriminator and the generator, the size of the W dimension is not altered. The use of bidirectional GRU layers in the generator comes out of a unidirectional RNN's inherent inability to know when iteration will end for variable length input.

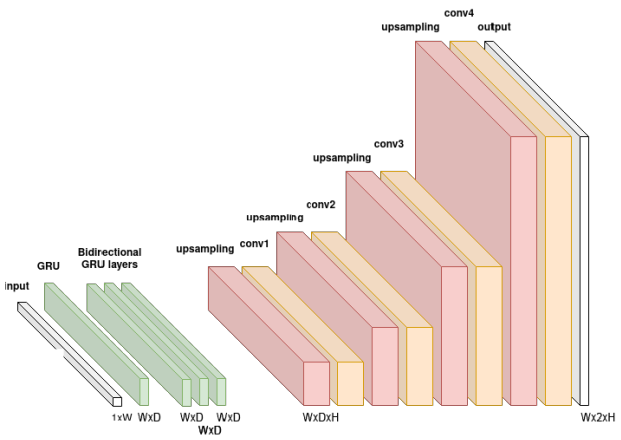

Figure 6: CRNN-GAN Generator

5.3 Data

5.3.1 Data. The dataset for this GAN consists of more than 38,000 audio signals stored as Numpy arrays at 44100 samples per second from zebra finch cages. The data needed to be parsed from a much larger dataset supplied to us by Dr. Melvin Rouse, which included the large stretches of silence between the bird vocalizations. Our method of parsing this data is through a relatively simple algorithm which clips part of the longer recordings based on a minimum volume threshold, and the amount of time between meeting the said threshold and meeting it again. The parsed dataset contains samples ranging from slightly less than half a second to over six seconds. It is important to note that longer signals longer bird vocalizations, which should result in a GAN where the user can control the length of the bird vocalizations produced, not just the signal, which should help reduce mode collapse.

5.3.2 Data Augmentation. The only data augmentation being performed for this model is the temporal trimming of the audio in the dataset-to-memory function. Each signal in the dataset is padded by retaining .2 seconds of audio before and after the volume threshold is met. The trimming operates on both ends of the signal, holding back a random chunk of the padding from the signal sent to memory. It should be noted that this augmentation has more of an effect on the RNN than the CNN portion of both the discriminator and generator, due to the partial shift invariance of the CNNs used.

6 DISCUSSION

The primary takeaway is that producing a deep-faked scalogram is a much more challenging task than producing a good looking image. While state-of-the-art image generation means producing realistic images of 1024x1024 = 1048576 pixels, a 1 second sample of audio at 22.5kHz has a representation of 64x22050 = 1411200 "pixels", a 34% increase. Plus, the relationship between phase and instantaneous angular frequency is more complex than the relationship between colors. The sparsity of the scalogram inputs also requires more training compared to typical images. A network created for this specific task as a result needs to have similar capacity to state-of-the-art image generation networks. Evaluating these techniques on better hardware could prove to make all the difference.

Since significant training did not begin on the CRNN-GAN, some of the methods used should not be presumed to work well. For instance, the use of bidirectional GRU layers bring awareness of the number of iterations left in a signal should work in theory, but it may be too complicated a task to learn. In place of bidirecionality, adding a normalized countdown as an input to each iteration of the GRU layers may also work, or even be easier for the network to learn. Also, parsing the zebra finch recordings supplied to us will likely result in some parsed samples which do not contain bird vocalization, which harms the training process. Manually checking all of the recordings is not plausible, so the use of a pre-trained bird audio classifier neural network to prune the dataset should be considered.

There are some issues present in the transformer. At the moment, generated scalograms look incorrect across frequencies, suggesting that our network is not capturing the relationships between frequencies very well. One solution would be to use the typical patching scheme as demonstrated in Lee et al. 2021 [16] and Gong et al. 2021 [7], opposed to feeding the Transformer a time-series. However, we think that adjusting the sizes of MLP hidden layers and increasing the network's capacity should prove sufficient; any issues learning likely are due to the huge image size or not training long enough. Again, memory here is the primary restriction. Ideally, we would also remove all convolutions from our network to both simplify the architecture and reduce the number of floating-point-operations needed, though removing convolutions increases the needed memory in the transformer layers.

One solution for reducing the size of scalograms is to get rid of the inverse CWT in reconstruction, instead relying on a neural

vocoder trained on birdsong. This has several benefits: we cheat the Nyquist frequency by downsampling our scalograms before feeding them to our GAN; we are not penalized by the sensitivity of the inverse CWT; and a vocoder would be able to reconstruct the loss in perceptual audio quality as shown in Engel et al. 2019 [6] and Shiba 2017 [21] .

Lastly, more experimentation is needed on the inclusion of phase in the network. With how difficult our task is, including a second channel for the generator to learn might make the task too challenging, even though it could lead to better results. Future work will experiment with different input combinations of complex cartesian input, magnitude and phase.

7 CONCLUSION

Singing birds are a powerful part of our natural ecosystem, yet current methods for reproducing their singing are inept and lack realism. In this paper, we have proposed a new paradigm for producing photorealistic bird calls using generative adversarial networks (GANs). Our GAN model is trained on birdsong scalagrams which leverage the Continuous Wavelet Transform for more precise time and frequency measures. The system we have proposed offers a vast improvement over previous bird call synthesis models, and we believe it will eventually lead to high quality bird vocalization generations.

ACKNOWLEDGMENTS

We would like to express our gratitude to Dr. Adam Smith, whose guidance and support have enabled us to make enormous leaps in our research this semester. His insightful suggestions and regular check-ins were invaluable to our success, and we thank him for his dedication and patience throughout the project. Our appreciation is also extended to Dr. Melvin Rouse for the contribution of his zebra finch data. We are grateful for the information and support he provided in our research endeavor.

REFERENCES

[1] Ana Amador and Gabriel B. Mindlin. 2021. Synthetic Birdsongs as a Tool to Induce, and listen to, Replay Activity in Sleeping Birds. *Frontiers in Neuroscience*, 15, (July 2021), 647978. DOI: 10.3389/fnins.2021.647978.

[2] Robert C Beason. [n. d.] What Can Birds Hear? en.

[3] Rhythm Bhatia. 2021. An Initial study on Birdsong Re-synthesis Using Neural Vocoders. arXiv:2209.10479 [cs, eess]. (Sept. 2021). Retrieved May 13, 2023 from http://arxiv.org/abs/2209.10479.

[4] Emre Cakir, Sharath Adavanne, Giambattista Parascandolo, Konstantinos Drossos, and Tuomas Virtanen. 2017. Convolutional recurrent neural networks for bird audio detection. en. In *2017 25th European Signal Processing Conference (EUSIPCO)*. IEEE, Kos, Greece, (Aug. 2017), 1744–1748. ISBN: 978-0-9928626-7-1. DOI: 10.23919/EUSIPCO.2017.8081508.

[5] Ting Chen, Mario Lucic, Neil Houlsby, and Sylvain Gelly. 2019. On self modulation for generative adversarial networks. (2019). arXiv: 1810.01365 [cs.LG].

[6] Jesse Engel, Kumar Krishna Agrawal, Shuo Chen, Ishaan Gulrajani, Chris Donahue, and Adam Roberts. 2019. Gansynth: adversarial neural audio synthesis. (2019). arXiv: 1902.08710 [cs.SD].

[7] Yuan Gong, Yu-An Chung, and James Glass. 2021. Ast: audio spectrogram transformer. (2021). arXiv: 2104.01778 [cs.SD].

[8] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative Adversarial Networks. arXiv:1406.2661 [cs, stat]. (June 2014). Retrieved May 12, 2023 from http://arxiv.org/abs/1406.2661.

[9] Gaurav Gupta, Meghana Kshirsagar, Ming Zhong, Shahrzad Gholami, and Juan Lavista Ferres. 2021. Comparing recurrent convolutional neural networks for large scale bird species classification. en. *Scientific Reports*, 11, 1, (Aug. 2021), 17085. Number: 1 Publisher: Nature Publishing Group. DOI: 10.1038/s41598-021-96446-w.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving deep into rectifiers: surpassing human-level performance on imagenet classification. (2015). arXiv: 1502.01852 [cs.CV].

[11] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation*, 9, 8, (Nov. 1997), 1735–1780. DOI: 10.1162/neco.1997.9.8.1735.

[12] Ilknur Icke. 2021. Central Pattern Generators to Synthesize Birdsongs. en. (Mar. 2021). Retrieved Apr. 26, 2023 from https://ickeilknur.medium.com/central-pattern-generators-to-synthesize-birdsongs-f0d09d6936c0.

[13] Fazle Karim, Somshubra Majumdar, and Houshang Darabi. 2019. Insights into LSTM Fully Convolutional Networks for Time Series Classification. *IEEE Access*, 7, 67718–67725. arXiv:1902.10756 [cs, stat]. DOI: 10.1109/ACCESS.2019.2916828.

[14] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. 2018. Progressive growing of gans for improved quality, stability, and variation. (2018). arXiv: 1710.10196 [cs.NE].

[15] Jeonghyun Kim, Yongkoo Park, and Wonzoo Chung. 2020. Transform based feature construction utilizing magnitude and phase for convolutional neural network in eeg signal classification. In *2020 8th International Winter Conference on Brain-Computer Interface (BCI)*, 1–4. DOI: 10.1109/BCI48061.2020.9061635.

[16] Kwonjoon Lee, Huiwen Chang, Lu Jiang, Han Zhang, Zhuowen Tu, and Ce Liu. 2021. Vitgan: training gans with vision transformers. (2021). arXiv: 2107.04589 [cs.CV].

[17] Xudong Mao, Qing Li, Haoran Xie, Raymond Y. K. Lau, Zhen Wang, and Stephen Paul Smolley. 2017. Least squares generative adversarial networks. (2017). arXiv: 1611.04076 [cs.CV].

[18] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. 2018. Spectral normalization for generative adversarial networks. (2018). arXiv: 1802.05957 [cs.LG].

[19] Olof Mogren. 2016. C-RNN-GAN: Continuous recurrent neural networks with adversarial training. arXiv:1611.09904 [cs]. (Nov. 2016). DOI: 10.48550/arXiv.1611.09904.

[20] Eugene B. Postnikov, Elena A. Lebedeva, and Anastasia I. Lavrova. 2015. Computational implementation of the inverse continuous wavelet transform without a requirement of the admissibility condition. arXiv:1507.04971 [math, q-bio]. (July 2015). Retrieved May 13, 2023 from http://arxiv.org/abs/1507.04971.

[21] Shintaro Shiba. 2017. Birdsong generation project. en-US. (2017). Retrieved Apr. 21, 2023 from http://shibashintaro.com/birdsong-generation-project/.

[22] Vincent Sitzmann, Julien N. P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. 2020. Implicit neural representations with periodic activation functions. (2020). arXiv: 2006.09661 [cs.CV].

[23] Adam A. Smith and Drew Kristensen. 2017. Deep learning to extract laboratory mouse ultrasonic vocalizations from scalograms. *2017 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, (Nov. 2017), 1972–1979. Conference Name: 2017 IEEE International Conference on Bioinformatics and Biomedicine (BIBM) ISBN: 9781509030507 Place: Kansas City, MO Publisher: IEEE. DOI: 10.1109/BIBM.2017.8217964.

[24] Dan Stowell. 2022. Computational bioacoustics with deep learning: a review and roadmap. *PeerJ*, 10, (Mar. 2022), e13152. DOI: 10.7717/peerj.13152.

[25] Mya Thompson and Annalyse Moskeland. 2014. How and Why Birds Sing. en-US. (Aug. 2014). Retrieved Apr. 26, 2023 from https://academy.allaboutbirds.org/birdsong/.

[26] Christopher Torrence and Gilbert P. Compo. 1998. A practical guide to wavelet analysis. *Bulletin of the American Meteorological Society*, 79, 61–78.

[27] Stéfan van der Walt, Johannes L. Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D. Warner, Neil Yager, Emmanuelle Gouillart, Tony Yu, and the scikit-image contributors. 2014. Scikit-image: image processing in Python. *PeerJ*, 2, (June 2014), e453. DOI: 10.7717/peerj.453.

[28] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. (2017). arXiv: 1706.03762 [cs.CL].

[29] Han Zhang, Zizhao Zhang, Augustus Odena, and Honglak Lee. 2020. Consistency regularization for generative adversarial networks. (2020). arXiv: 1910.12027 [cs.LG].

A GENERATED SCALOGRAMS

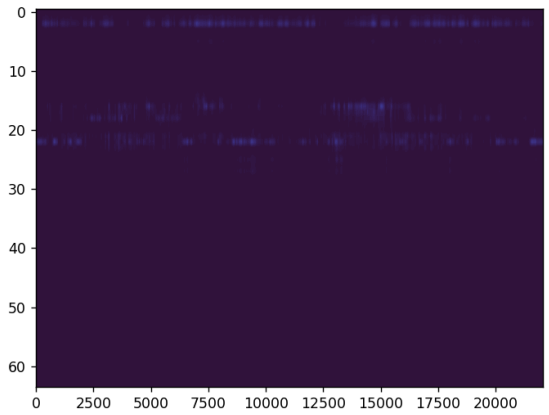

Figure 7: Run stopped after 21 epochs, sample from epoch 15

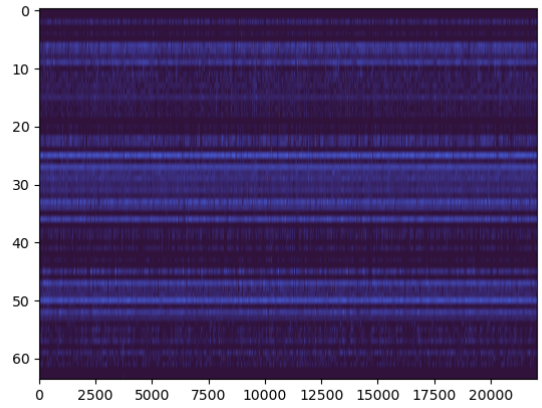

Figure 8: Run stopped after 9 epochs, from training run in 4. This (the last epoch) produces a pulsing noise, a very common mode collapse for our network.

B AN IMPORTANT NOTE

Figure 9: Adam Smith with Birds