SYNTHESIZING AUDIO USING GENERATIVE ADVERSARIAL NETWORKS

Bowen Zhi **Fan Yang** **Nadee Seneviratne** **Patrick Owen**

ABSTRACT

While generative adversarial networks (GANs) have seen success in generating realistic images, very few studies have explored their application to analogous audio generation tasks. One difficulty in adapting GANs for such tasks is in finding a suitable data domain on which to operate. Although image-like spectrogram features have seen widespread use for discriminative audio tasks, such representations cannot be readily transformed back to audio without adversely affecting audio quality. A recently explored option is to model the raw audio waveform directly. However, such an approach presents other problems, including audio artifacts and long training times. In this paper, we propose an approach to unsupervised audio synthesis using GANs on invertible image-like audio representations, and demonstrate that our method is competitive with the state-of-the-art raw waveform baseline while requiring less than half the training time.

1 INTRODUCTION

Audio synthesis is a challenging problem with applications in music production, text-to-speech, and data augmentation for various audio-related tasks. Manually synthesized audio tends to sound artificial compared to real audio in some domains, while existing data-driven synthesis models typically rely on large amounts of labeled audio [1]. Thus, in this work we wish to explore audio synthesis as a generative modeling problem in an unsupervised setting. Generative adversarial networks (GANs) are one possible solution. However, despite recent successes in generating realistic images with GANs [2, 3, 4], the use of GANs for audio generation has remained relatively unexplored.

As an attempt to adapt image-generating GANs to audio, Donahue et al. [5] proposed two different approaches to generating fixed-length audio segments based on the DCGAN [2] architecture: SpecGAN and WaveGAN. Inspired by the use of convolutional neural networks on spectrograms for audio classification [6], SpecGAN operates on image-like 2D magnitude spectrograms of audio. However, the process of converting audio into such spectrograms lacks an inverse, as phase information is discarded. Hence, the audio samples generated from SpecGAN are derived from approximate inversion methods that incur significant distortion. To circumvent the need for such inversion methods, WaveGAN instead operates on raw audio waveforms. This model converts the 2D convolutions over images present in DCGAN to 1D convolutions over time-domain audio signals. However, due to fundamental differences between images and audio, artifacts caused by convolutional operations become more troublesome to mitigate in the audio domain. Qualitative judgements show that both WaveGAN and SpecGAN generate audio of significantly lower quality relative to real recordings. Moreover, while audio generated from WaveGAN had higher perceived quality than those from SpecGAN, WaveGAN requires far more training time, which suggests a possible shortcoming of the raw waveform representation.

We extend the work of Donahue et al. [5] to use GANs for unsupervised audio synthesis. However, instead of using raw waveform or spectrogram representations, we use time-frequency representations with exact inverses. Since frequency information is more representative of human sound perception, we expect that GANs operating on such representations might yield subjectively better results and be easier to train than those operating on raw waveforms. Moreover, since our representations are invertible, the generated audio does not suffer the distortion introduced by spectrogram inversion techniques.

2 METHODOLOGY

Our approach is similar to that used with SpecGAN from Donahue et al. [5]. As with SpecGAN, we consider the problem of generating audio clips consisting of 16384 samples at 16 kHz. However, rather than discarding phase information and using a spectrogram representation, we instead transform the audio signal in an invertible manner. This involves using the short-time DFT or DCT to create a time-frequency representation with phase information. A

fixed transformation is applied to all input audio data, and its inverse is applied to the generator output to produce audio signals. GAN training then proceeds entirely in the transformed time-frequency domain, using the Wasserstein loss with gradient penalty [7].

2.1 SHORT-TIME FOURIER TRANSFORM

One possible time-frequency representation can be generated with a short-time Fourier transform. We use Hann windows of length 256 with a stride of 128 samples, which allows for constant overlap-add during resynthesis. To ensure the original signal is completely reconstructable, we zero-pad the boundaries. This yields 256 complex values for each of the 129 windows. Since the signal is real, the upper 127 frequencies can be discarded due to Hermitian symmetry, leaving us with 129 frequencies. Treating the time and frequency dimensions as a 2D spatial grid, we have a single-channel complex-valued image of shape (129, 129, 1). Separating the real and imaginary components into individual channels yields an equivalent real image of shape (129, 129, 2).

As the upscaling methods used in our models are most effective when the spatial dimensions are equivalent (i.e. square) and have prime factorizations consisting solely of small numbers, the Nyquist frequency and the last time window are discarded, reducing both the number of frequencies and time windows from 129 to 128. Thus, our final image has shape (128, 128, 2). Although these changes are irreversible, we believe that they will not perceptibly change the audio: the Nyquist frequency rarely has much energy, and most audio recordings naturally taper off at the end anyways.

2.2 SHORT-TIME DISCRETE COSINE TRANSFORM

An alternative to the discrete Fourier transform is the discrete cosine transform (DCT). Specifically, we use the type-II DCT, which computes the DFT of a signal's even extension. As such, the DCT's outputs are purely real, which may be better suited for conventional deep learning architectures.

As in the DFT representation, we represent the outputs of the short-time DCT as an image along time and frequency axes. We would like to keep the image square, as this allows us to use very similar architectures between DFT and DCT representations. While using rectangular windows of length 128 with a stride of 128 would yield a square image of shape (128, 128, 1), the lack of overlapping regions can yield discontinuities between time windows in the resynthesized audio. Moreover, as such discontinuities are difficult to visually detect from the time-frequency representation, the models will likely also struggle to recognize these issues. Instead, we opt to use Hann windows of length 256 with a stride of 64 samples, which still maintains the constant overlap-add property. As with the DFT representation, we discard the zero-padded windows at the end to yield a real image of shape (256, 256, 1).

2.3 DATA SCALING AND NORMALIZATION

The magnitudes of the DFT and DCT outputs are scaled roughly logarithmically both to simulate human auditory perception and to better bound the range of values. To do so without affecting phase information, we use an invertible function that maps 0 to itself:

$$f(x) = \frac{x}{|x|} \log_{10}(|x| + 1), \quad f(0) = 0,$$

along with its inverse:

$$f^{-1}(x) = \frac{x}{|x|}(10^{|x|} - 1), \quad f^{-1}(0) = 0.$$

Further, the magnitudes of the data are rescaled on a per-frequency basis: individually for each frequency, all values are divided by the maximal magnitude across the entire training dataset. Thus, all values are scaled into the range $[-1, 1]$, matching the range of the $\tanh$ activation function used for the generator. These values are then multiplied by 0.9 to further limit this range, thereby allowing the generator to predict all possible values present in the dataset without saturating its output activations.

2.4 NETWORK ARCHITECTURE

After a brief discussion about the GAN training methods chosen, we introduce four model configurations, which we denote as DFT GAN, DCT GAN, DFT-SP GAN, and DCT-SP GAN. The generator architectures for each of these are shown in Table 1, and the discriminator architectures in Table 2.

2.4.1 GAN TRAINING

We train all of our models using the Wasserstein loss with gradient penalty (WGAN-GP) [7], as it was shown in [5] to outperform other losses [2, 8, 9]. In accordance to WGAN-GP, our generator and discriminator losses L_G, L_D are:

$$L_G = \mathbb{E}_{\boldsymbol{z} \sim \mathbb{P}_I}\big[D(G(\boldsymbol{z}))\big],$$
$$L_D = \mathbb{E}_{\boldsymbol{x} \sim \mathbb{P}_R}\big[D(\boldsymbol{x})\big] - \mathbb{E}_{\boldsymbol{z} \sim \mathbb{P}_I}\big[D(G(\boldsymbol{z}))\big] + \lambda\, \mathbb{E}_{\hat{\boldsymbol{x}} \sim \mathbb{P}_C}\big[(\|\nabla_{\hat{\boldsymbol{x}}} D(\hat{\boldsymbol{x}})\| - 1)^2\big].$$

Here, G and D are the functions computed by the generator and discriminator, and $\lambda > 0$ is a constant hyperparameter known as the gradient penalty weight. $\mathbb{P}_I$ and $\mathbb{P}_R$ are the latent and data distributions, and $\mathbb{P}_C$ is a distribution comprised of all convex combinations of samples from $G(\mathbb{P}_I)$ and $\mathbb{P}_R$, i.e. $\hat{\boldsymbol{x}} \sim \mathbb{P}_C = \alpha \boldsymbol{x} + (1 - \alpha) G(\boldsymbol{z})$, where $\boldsymbol{x} \sim \mathbb{P}_R, \boldsymbol{z} \sim \mathbb{P}_I, \alpha \sim [0, 1]$.

As the above formulation is intractable in practice, during training we compute L_G and L_D by approximating $\mathbb{P}_I$, $\mathbb{P}_R$, and $\mathbb{P}_C$ with minibatches. For each generator, we use 100 iid uniform random variables on $[-1, 1]$ as $\mathbb{P}_I$.

2.4.2 DFT GAN

DFT GAN operates on the DFT representation proposed in Section 2.1. We adopt the generator and discriminator architectures for SpecGAN [5], and change the generator output and discriminator input to have 2 channels. Kernel weights are initialized using He uniform initialization [10] for operations directly preceding ReLU or leaky ReLU, and Glorot uniform initialization [11] for all other operations. All layers also have biases, which are zero-initialized.

2.4.3 DCT GAN

As the name suggests, DCT GAN operates on the DCT representation proposed in Section 2.2. We also base DCT GAN heavily off of SpecGAN. For the DCT GAN generator, we take the SpecGAN generator architecture and add a transposed convolution layer at the end. Similarly for the discriminator, we take the SpecGAN discriminator architecture and add a convolution layer at the beginning. Adding these layers allows the generator's output and discriminator's input to have the correct dimensions for our DCT representation. Weights and biases are initialized as in DFT GAN.

2.4.4 DFT-SP AND DCT-SP GANS

As transpose convolutions are prone to producing "checkerboard" artifacts [12], we also explore using an alternative to transpose convolutions proposed in [13]: regular convolutions followed by "sub-pixel shuffling". Specifically, to replace a 2D transpose convolution of stride k mapping an input of shape (h, w, c_{in}) to an output of shape (kh, kw, c_{out}), we first perform unstrided 2D convolutions using $k^2 c_{out}$ filters to obtain an intermediate output of shape $(h, w, k^2 c_{out})$. This result is then reshaped via sub-pixel shuffle to (kh, kw, c_{out}), wherein each group of k^2 channels are merged by creating $k \times k$ blocks out of values with the same spatial correspondence, as demonstrated in Figure 1 for $k = 2$. For brevity, we will refer to this operation simply as sub-pixel convolution. As shown in [14], when each group of k^2 kernels are initialized to the same value, sub-pixel convolution is less prone to checkerboard artifacts than transpose convolution.

For the DFT-SP and DCT-SP GANs, we copy the architectures for the DFT and DCT GANs and replace the 5×5 stride-2 transpose convolutions of the generators with 3×3 sub-pixel convolutions. The choice of 3×3 filter sizes allow the sub-pixel convolutions to maintain the same effective receptive field as the replaced transpose convolutions.

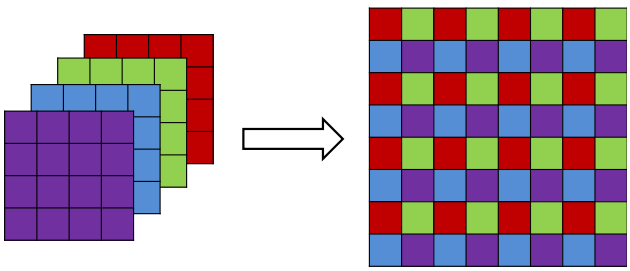

Figure 1: Visualization of the sub-pixel shuffling operation combining 4 channels into 1 upscaled channel.

(a) DFT GAN Generator

Operation	Kernel Shape	Output Shape
Input		(100)
Dense	$(100, 256d)$	$(256d)$
Reshape		$(4, 4, 16d)$
ReLU		$(4, 4, 16d)$
Conv2DTr	$(5, 5, 16d, 8d)$	$(8, 8, 8d)$
ReLU		$(8, 8, 8d)$
Conv2DTr	$(5, 5, 8d, 4d)$	$(16, 16, 4d)$
ReLU		$(16, 16, 4d)$
Conv2DTr	$(5, 5, 4d, 2d)$	$(32, 32, 2d)$
ReLU		$(32, 32, 2d)$
Conv2DTr	$(5, 5, 2d, d)$	$(64, 64, d)$
ReLU		$(64, 64, d)$
Conv2DTr	$(5, 5, d, 2)$	$(128, 128, 2)$
Tanh		$(128, 128, 2)$

(b) DCT GAN Generator

Operation	Kernel Shape	Output Shape
Input		(100)
Dense	$(100, 256d)$	$(256d)$
Reshape		$(4, 4, 16d)$
ReLU		$(4, 4, 16d)$
Conv2DTr	$(5, 5, 16d, 8d)$	$(8, 8, 8d)$
ReLU		$(8, 8, 8d)$
Conv2DTr	$(5, 5, 8d, 4d)$	$(16, 16, 4d)$
ReLU		$(16, 16, 4d)$
Conv2DTr	$(5, 5, 4d, 2d)$	$(32, 32, 2d)$
ReLU		$(32, 32, 2d)$
Conv2DTr	$(5, 5, 2d, d)$	$(64, 64, d)$
ReLU		$(64, 64, d)$
Conv2DTr	$(5, 5, d, d)$	$(128, 128, d)$
ReLU		$(128, 128, d)$
Conv2DTr	$(5, 5, d, 1)$	$(256, 256, 1)$
Tanh		$(256, 256, 1)$

(c) DFT-SP GAN Generator

Operation	Kernel Shape	Output Shape
Input		(100)
Dense	$(100, 256d)$	$(256d)$
Reshape		$(4, 4, 16d)$
ReLU		$(4, 4, 16d)$
Conv2D	$(3, 3, 16d, 32d)$	$(4, 4, 32d)$
Shuffle		$(8, 8, 8d)$
ReLU		$(8, 8, 8d)$
Conv2D	$(3, 3, 8d, 16d)$	$(8, 8, 16d)$
Shuffle		$(16, 16, 4d)$
ReLU		$(16, 16, 4d)$
Conv2D	$(3, 3, 4d, 8d)$	$(16, 16, 8d)$
Shuffle		$(32, 32, 2d)$
ReLU		$(32, 32, 2d)$
Conv2D	$(3, 3, 2d, 4d)$	$(32, 32, 4d)$
Shuffle		$(64, 64, d)$
ReLU		$(64, 64, d)$
Conv2D	$(3, 3, d, 8)$	$(64, 64, 8)$
Shuffle		$(128, 128, 2)$
Tanh		$(128, 128, 2)$

(d) DCT-SP GAN Generator

Operation	Kernel Shape	Output Shape
Input		(100)
Dense	$(100, 256d)$	$(256d)$
Reshape		$(4, 4, 16d)$
ReLU		$(4, 4, 16d)$
Conv2D	$(3, 3, 16d, 32d)$	$(4, 4, 32d)$
Shuffle		$(8, 8, 8d)$
ReLU		$(8, 8, 8d)$
Conv2D	$(3, 3, 8d, 16d)$	$(8, 8, 16d)$
Shuffle		$(16, 16, 4d)$
ReLU		$(16, 16, 4d)$
Conv2D	$(3, 3, 4d, 8d)$	$(16, 16, 8d)$
Shuffle		$(32, 32, 2d)$
ReLU		$(32, 32, 2d)$
Conv2D	$(3, 3, 2d, 4d)$	$(32, 32, 4d)$
Shuffle		$(64, 64, d)$
ReLU		$(64, 64, d)$
Conv2D	$(3, 3, d, 4d)$	$(64, 64, 4d)$
Shuffle		$(128, 128, d)$
ReLU		$(128, 128, d)$
Conv2D	$(3, 3, d, 4)$	$(128, 128, 4)$
Shuffle		$(256, 256, 1)$
Tanh		$(256, 256, 1)$

Table 1: The generator architectures for DFT (a), DCT (b), DFT-SP (c), and DCT-SP (d) GANs, where d is a hyperparameter, Conv2DTr refers to transpose 2D convolution, and Shuffle refers to sub-pixel shuffling. The batch dimension is excluded in the output shape. All Conv2DTr layers have a stride of 2, and all layers with kernels also have biases.

3 EVALUATION

We implement our models in Keras [15] using TensorFlow as the backend, and evaluate our models alongside the best WaveGAN and SpecGAN models presented in [5] as baselines.

3.1 DATASET

For direct comparison to the baseline models, we choose to evaluate our models on the same dataset used in [5]: SC09. This dataset is comprised of roughly 21000 single-second utterances of spoken digits "zero" through "nine" from a

(a) DFT(-SP) GAN Discriminator

Operation	Kernel Size	Output Shape
Input		$(128, 128, 2)$
Conv2D	$(5, 5, 2, d)$	$(64, 64, d)$
LReLU		$(64, 64, d)$
Conv2D	$(5, 5, d, 2d)$	$(32, 32, 2d)$
LReLU		$(32, 32, 2d)$
Conv2D	$(5, 5, 2d, 4d)$	$(16, 16, 4d)$
LReLU		$(16, 16, 4d)$
Conv2D	$(5, 5, 4d, 8d)$	$(8, 8, 8d)$
LReLU		$(8, 8, 8d)$
Conv2D	$(5, 5, 8d, 16d)$	$(4, 4, 16d)$
LReLU		$(4, 4, 16d)$
Reshape		$(256d)$
Dense	$(256d, 1)$	(1)

(a) DCT(-SP) GAN Discriminator

Operation	Kernel Size	Output Shape
Input		$(256, 256, 1)$
Conv2D	$(5, 5, 1, d)$	$(128, 128, d)$
LReLU		$(128, 128, d)$
Conv2D	$(5, 5, d, d)$	$(64, 64, d)$
LReLU		$(64, 64, d)$
Conv2D	$(5, 5, d, 2d)$	$(32, 32, 2d)$
LReLU		$(32, 32, 2d)$
Conv2D	$(5, 5, 2d, 4d)$	$(16, 16, 4d)$
LReLU		$(16, 16, 4d)$
Conv2D	$(5, 5, 4d, 8d)$	$(8, 8, 8d)$
LReLU		$(8, 8, 8d)$
Conv2D	$(5, 5, 8d, 16d)$	$(4, 4, 16d)$
LReLU		$(4, 4, 16d)$
Reshape		$(256d)$
Dense	$(256d, 1)$	(1)

Table 2: The discriminator architectures for DFT and DFT-SP GANs (a), and DCT and DCT-SP GANs (b), where d is a hyperparameter, LReLU refers to leaky ReLU with $\alpha = 0.2$, and all Conv2D layers have a stride of 2. The batch dimension is excluded in the output shape.

variety of speakers. The recordings were made at 16 kHz under inconsistent recording conditions, and the resulting audio clips are not time-aligned. The dataset is split into training and test sets consisting of roughly 18500 and 2500 of these recordings, respectively.

3.2 Training configuration

For all models, we train both the generator and discriminator with model size $d = 64$ using Adam [16] with $\alpha = 1\text{e}{-4}$, $\beta_1 = 0.5$, $\beta_2 = 0.9$ on the WGAN-GP losses with gradient penalty weight $\lambda = 10$. Using a batch size of 64, we train the discriminator for 5 iterations for every generator update.

On SC09, we train DFT GAN, DFT-SP GAN, DCT GAN, DCT-SP GAN, and SpecGAN to convergence within 58000 generator iterations (roughly equivalent to 200 epochs). Using a single NVIDIA P6000 GPU, all of these models finish training within 60 hours. Due to the long training time of WaveGAN (which converges in 700 epochs), we use the pre-trained WaveGAN model given by [5].

3.3 Metrics

We evaluate the audio generated by the models with respect to several quantitative and qualitative metrics, as described below. As the quantitative metrics used may not correlate with human judgement [17], we use these metrics simply to sanity check our models, and opt to use qualitative metrics as the main point of comparison.

3.3.1 Inception score

We compute the inception score as an approximate measure of the diversity and semantic intelligibility of the generated audio samples. To do so, we use the classifier trained by [5] on SC09. This classifier, which we call C, takes as input a 128×128 Mel-scale spectrogram. C consists of four 2D convolution and pooling layers followed by a 10-class softmax layer that outputs probabilities of each digit. Given labels $\boldsymbol{y}$, the inception score IS for a generator G is defined as:

$$\text{IS}(G) = \exp\left(\mathbb{E}_{\boldsymbol{z} \sim \mathbb{P}_I}\left[D_{\text{KL}}\left(p_C(\boldsymbol{y} \mid G(\boldsymbol{z})) \parallel p_C(\boldsymbol{y})\right)\right]\right),$$

where D_{KL} is the Kullback-Leibler divergence, $p_C(\boldsymbol{y} \mid G(\boldsymbol{z}))$ is the output of $C(G(\boldsymbol{z}))$ interpreted as a probability mass function over the labels $\boldsymbol{y}$, and $p_C(\boldsymbol{y}) = \int_{\boldsymbol{z}} p_C(\boldsymbol{y} \mid G(\boldsymbol{z}))p(\boldsymbol{z})$ is the marginal probability. $\mathbb{P}_I$ is the latent distribution used by the generator, as defined in Section 2.4.1. For the 10-class SC09 dataset, $\text{IS}(G)$ ranges from 1 to 10: the easier C can classify data generated by G, the higher the score.

$\mathrm{IS}(G)$ is approximated using 10 sets of 5000 generated samples to obtain a mean score and standard error. For comparison, the inception scores of the training and test sets are also computed: for these, the score is computed using 10 roughly equal-size subsets.

3.3.2 NEAREST NEIGHBOR COMPARISON

There are two degenerate, undesirable cases in which a generative model can achieve a high inception score on SC09. In the first case, the model only outputs 10 samples, one for each of the 10 labels. In the second case, the model heavily overfits to the training data and exclusively outputs samples from the training data. As is done in [5], we use two metrics to monitor the occurrence of these cases: one metric for each case.

For both metrics, we operate in the same data space used by classifier C. Both metrics take as input a set S of 1000 generated audio examples.

The first metric, $|D|_{\mathrm{self}}$, computes the average L^2 distance between each example $\boldsymbol{x} \in S$ and its nearest neighbor in $S \setminus \{\boldsymbol{x}\}$. Thus, if the first degenerate case mentioned above occurs, $|D|_{\mathrm{self}}$ will become small.

The second metric, $|D|_{\mathrm{train}}$, computes the average L^2 distance between each example $\boldsymbol{x} \in S$ and its nearest neighbor in the training set. Hence, if the second degenerate case occurs, $|D|_{\mathrm{train}}$ will be small.

As was done for inception score, we also compute these two metrics on the SC09 training and test sets. All values for $|D|_{\mathrm{self}}$ and $|D|_{\mathrm{train}}$ are then rescaled relative to those of the test set (so that $|D|_{\mathrm{self}}$ and $|D|_{\mathrm{train}}$ for the test set are 1).

3.3.3 QUALITATIVE RATINGS

As human perception of audio quality tends to be subjective, we conduct a small, informal listening survey wherein subjects are asked to assess the audio generated by our models with respect to three metrics: sound quality, ease of intelligibility, and speaker diversity. For each of the evaluated models, each subject rates 10 randomly-generated audio samples according to these metrics on a 0-10 scale. As a control, subjects are also asked to rate 10 randomly-selected audio samples from the SC09 test set. In attempt to remove potential biases, subjects are not told how each set of 10 audio samples were created.

4 RESULTS AND DISCUSSION

Table 3 shows the quantitative metrics for samples produced by the tested models, along with the training data and test data. None of the tested models seem to heavily overfit or have low diversity, as indicated by $|D|_{\mathrm{self}}$ and $|D|_{\mathrm{train}}$. All four proposed models achieve inception scores around that of the WaveGAN baseline.

The qualitative results are shown in Figure 2. While the small sample size makes it difficult to draw strong conclusions, all four proposed models seem to outperform SpecGAN in speaker diversity and are comparable to WaveGAN with respect to all three qualitative metrics.

Our proposed models seem to perform at levels comparable to those of WaveGAN in both qualitative and quantitative metrics, whilst converging in much fewer training iterations (200 epochs for ours vs 700 epochs for WaveGAN). This suggests that time-frequency representations may be better suited for learning with standard convolutional GAN architectures compared to raw waveforms.

5 RELATED WORK

More recently, [18] also explore using GANs on invertible time-frequency representations of audio to synthesize musical notes conditioned on pitch. Unlike our approach, the time-frequency representations used all involve representing amplitude and phase separately, which introduces a problematic branch cut in the phase. They address this branch cut with assumptions about the phases' time-evolution, which is valid for their experiments on unmodulated signals of constant pitch, but not valid for general audio signals. In contrast, our work investigates synthesis of more general audio signals using time-frequency representations that are free of branch cuts.

Audio Source	Inception Score	$\lvert D\rvert_{\text{self}}$	$\lvert D\rvert_{\text{train}}$
SC09 (train)	9.18 ± 0.04	1.0	0.0
SC09 (test)	8.01 ± 0.24	1.0	1.0
WaveGAN	4.68 ± 0.04	0.8	2.3
SpecGAN	6.02 ± 0.04	1.0	1.4
DFT GAN	4.71 ± 0.03	1.4	2.3
DFT-SP GAN	4.37 ± 0.04	1.0	2.3
DCT GAN	4.47 ± 0.03	1.2	2.3
DCT-SP GAN	4.85 ± 0.04	0.9	2.4

Table 3: Quantitative results for the SC09 dataset. For the Inception Score column, given are the mean scores followed by the standard error. A high inception score implies that the generative model's outputs are easily classified. Values for $\lvert D\rvert_{\text{train}}$ and $\lvert D\rvert_{\text{self}}$ that are $\ll 1$ are indicative of overfitting and low diversity, respectively.

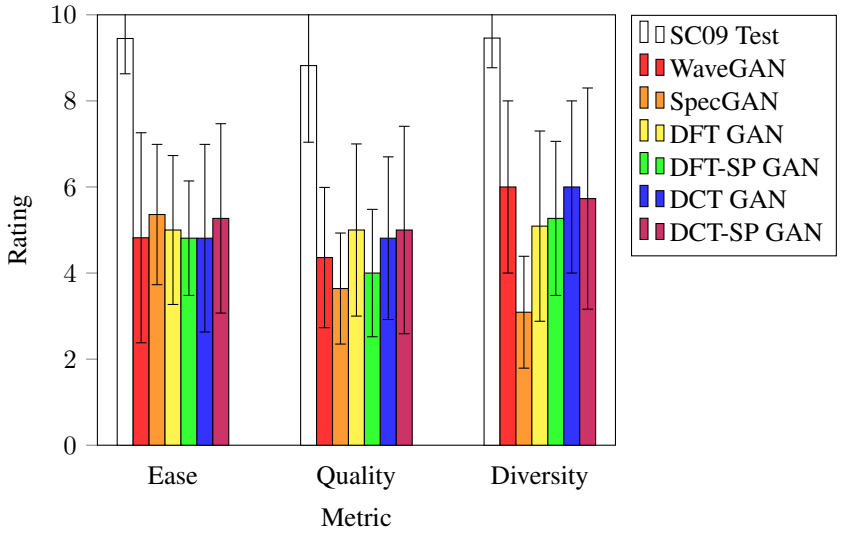

Figure 2: Qualitative ratings for *ease* of intelligibility, sound *quality*, and speaker *diversity* from 11 subjects. Plotted are the mean ratings for each of the three metrics, alongside the standard error.

6 Conclusion and Future Work

We have proposed several time-frequency GANs for unsupervised audio synthesis that are competitive with the raw waveform baseline and outperform the spectrogram baseline in qualitative assessments. Moreover, the proposed models converge in far fewer iterations than the raw waveform baseline, suggesting that our time-frequency representations are effective preconditioners to GAN training. Possible avenues of future work include fine-tuning of the proposed representations and models, evaluating these models on other speech and non-speech datasets, adapting ideas used for high-resolution image generation [3, 4] to improve the fidelity of synthesized audio, and using more advanced network layers to better model the nature of audio signals, such as complex neural networks [19] to model the complex Fourier domain.

REFERENCES

[1] Yuxuan Wang, RJ Skerry-Ryan, Daisy Stanton, Yonghui Wu, Ron J. Weiss, Navdeep Jaitly, Zongheng Yang, Ying Xiao, Zhifeng Chen, Samy Bengio, Quoc Le, Yannis Agiomyrgiannakis, Rob Clark, and Rif A. Saurous. Tacotron: Towards end-to-end speech synthesis. 2017.

[2] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2015.

[3] Emily L Denton, Soumith Chintala, Arthur Szlam, and Rob Fergus. Deep generative image models using a laplacian pyramid of adversarial networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 1486–1494. Curran Associates, Inc., 2015.

[4] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. 10 2017.

[5] Chris Donahue, Julian McAuley, and Miller Puckette. Adversarial audio synthesis. 2018.

[6] Muhammad Huzaifah. Comparison of time-frequency representations for environmental sound classification using convolutional neural networks, 2017.

[7] Ishaan Gulrajani, Faruk Ahmed, Martín Arjovsky, Vincent Dumoulin, and Aaron C. Courville. Improved training of wasserstein gans. *CoRR*, abs/1704.00028, 2017.

[8] Xudong Mao, Qing Li, Haoran Xie, Raymond Y. K. Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks. *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2813–2821, 2017.

[9] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 214–223, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015.

[11] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[12] Augustus Odena, Vincent Dumoulin, and Chris Olah. Deconvolution and checkerboard artifacts. *Distill*, 2016.

[13] Wenzhe Shi, Jose Caballero, Ferenc Huszar, Johannes Totz, Andrew P. Aitken, Rob Bishop, Daniel Rueckert, and Zehan Wang. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[14] Andrew Aitken, Christian Ledig, Lucas Theis, Jose Caballero, Zehan Wang, and Wenzhe Shi. Checkerboard artifact free sub-pixel convolution: A note on sub-pixel convolution, resize convolution and convolution resize, 2017.

[15] François Chollet et al. Keras. `keras.io`, 2015.

[16] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[17] Shane Barratt and Rishi Kant Sharma. A note on the inception score. *CoRR*, abs/1801.01973, 2018.

[18] Anonymous. Gansynth: Adversarial neural audio synthesis. In *Submitted to International Conference on Learning Representations*, 2019. under review.

[19] Chiheb Trabelsi, Olexa Bilaniuk, Ying Zhang, Dmitriy Serdyuk, Sandeep Subramanian, Joao Felipe Santos, Soroush Mehri, Negar Rostamzadeh, Yoshua Bengio, and Christopher J Pal. Deep complex networks. In *International Conference on Learning Representations*, 2018.