# Adversarial audio synthesis

E6691.2022Spring.WAVE.report.an3078.bmh2168.gs3160

Anh-Vu Nguyen an3078, Brian Hernandez bmh2168, Gokul Srinivasan gs3160
*Columbia University*

## Abstract

*Using GANs to generate audio samples is not as well documented as image GANs. While audio signals differ from images, we can still try to use some techniques and architectures created for image generation for our objective. Audio signals differ from images because of their high sample rate and because they require capturing information on a large range of timescales. In this paper, we try to recreate, validate and analyze the findings of the paper [1] that introduced WaveGAN and SpecGAN as well as the paper [9] that introduced GANSynth. There are two approaches in [1] that were used to generate audio-like samples. WaveGAN is the most straightforward approach, trying to generate raw audio samples, while SpecGAN works with spectrograms that are then inverted to retrieve an audio sample. GANSynth is somewhat similar to SpecGAN, with the difference that the phase information of the audio signal is also incorporated in the audio representation, which leads to some performance improvement. We successfully implemented and tested all three models on various datasets [2]. The metrics that we used to test the generated samples revealed some differences with the paper, notably for the SpecGAN model.*

## 1. Introduction

Audio synthesis can have applications in various fields. Some examples can be generating audio effects, voice commands, or instrument samples for music production for instance. The traditional process of creating samples by recording sounds with microphones is time-consuming, and exploring a wide range of audio samples can also take a lot of time. Using GANs [4] to create audio would not only speed up the creation phase, but also it would allow the user to quickly explore the sonic properties of the generated samples by going through a low dimensional latent space.

An advantage of using a GANs is that it is much quicker than their autoregressive model counterparts [5] as the output does not have to be fed back into the network.

The original paper on WaveGAN and SpecGAN investigated two methods to create audio: a spectrogram-based approach and a direct one working on raw audio directly.

While the spectrogram-based approach allows the use of existing image GANs such as DCGAN [6], it comes with drawbacks because the generated image can only be inverted back to audio using approximation algorithms such as the Griffin & Lim algorithm. SpecGAN uses this method and is largely based on the DCGAN model [6]. One can consider GANSynth to be an improvement over SpecGan, as there is no lossy approximation step such as Griffin-Lim, but a spectrogram representation is used.

The WaveGAN model directly generates 1D raw audio samples by flattening the convolution layers of the DCGAN. Other modifications have been made to address artifacts that can be easily picked up by the discriminator, and the higher dimensionality of audio samples.

## 2. Summary of the Original Papers

## 2.1 Methodology of the Original Papers

### 2.1.1 Working with audio

The WaveGAN / SpecGAN paper had to do some modifications to the DCGAN architecture to account for the high dimensionality of audio. For example, DCGAN [6] only generates 64x64 low-resolution images. If we want to generate a one-second audio sample at 16 kHz sampling rate, then we will need a lot more dimensions.

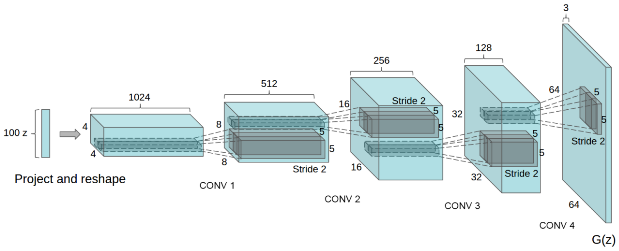


***Fig.*** *DCGAN architecture (generator). The output is of dimension 64x64*

The paper settled for audio formats at a 16kHz sampling rate with 16,384 samples, which allows to generate about one second of audio.

### 2.1.2 SpecGAN

The SpecGAN is a frequency domain generative model. It works by generating spectrograms of size 128x128, which when flattened are of the same dimension of 16,384 as the generated audio. To convert the training data to spectrograms, the audio is processed into spectrograms using a short-time Fourier transform with 16 ms windows and 8 ms stride. This will result in 128 frequency bins. Because the output activation layer of the generator is tanh, the spectrograms are standardized by dividing them by 3 standard deviations and clipped between -1 and 1. The inverse operation has to be done before applying the Griffin Lim algorithm to retrieve the corresponding audio.

| Operation | Kernel size | Output Shape |
|---|---|---|
| Input z ~ Uniform(−1, 1) | | (n, 100) |
| Dense 1 | (100, 256d) | (n, 256d) |
| Reshape | | (n, 4, 4, 16d) |
| ReLU | | (n, 4, 4, 16d) |
| Trans Conv2D (Stride=2) | (5, 5, 16d, 8d) | (n, 8, 8, 8d) |
| ReLU | | (n, 8, 8, 8d) |
| Trans Conv2D (Stride=2) | (5, 5, 8d, 4d) | (n, 16, 16, 4d) |
| ReLU | | (n, 16, 16, 4d) |
| Trans Conv2D (Stride=2) | (5, 5, 4d, 2d) | (n, 32, 32, 2d) |
| ReLU | | (n, 32, 32, 2d) |
| Trans Conv2D (Stride=2) | (5, 5, 2d, d) | (n, 64, 64, d) |
| ReLU | | (n, 64, 64, d) |
| Trans Conv2D (Stride=2) | (5, 5, d, c) | (n, 128, 128, c) |
| Tanh | | (n, 128, 128, c) |

***Fig.*** *SpecGAN architecture (generator). c and d are hyperparameters. c=1 and d=64 in the paper.*

### 2.1.3 WaveGAN and phase shuffling

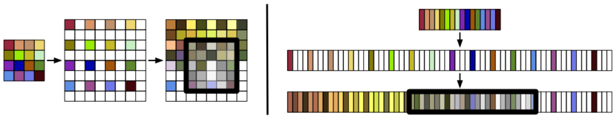


***Fig.*** *Illustration of the transposed convolution operation for SpecGAN (left) and WaveGAN (right).*

WaveGAN is similar to SpecGAN in the sense that it replaces 2D convolutions with 1D convolutions. The strides and kernel size are changed to get the same output dimensions at each layer. As such, the filters are of size 25 and the strides are increased from 2 to 4. While we are working with a low sampling rate of 16kHz, it is sufficient for certain applications such as human speech.

| Operation | Kernel size | Output Shape |
|---|---|---|
| Input z ~ Uniform(−1, 1) | | (n, 100) |
| Dense 1 | (100, 256d) | (n, 256d) |
| Reshape | | (n, 16, 16d) |
| ReLU | | (n, 16, 16d) |
| Trans Conv1D (Stride=4) | (25, 16d, 8d) | (n, 64, 8d) |
| ReLU | | (n, 64, 8d) |
| Trans Conv1D (Stride=4) | (25, 8d, 4d) | (n, 256, 4d) |
| ReLU | | (n, 256, 4d) |
| Trans Conv1D (Stride=4) | (25, 4d, 2d) | (n, 1024, 2d) |
| ReLU | | (n, 1024, 2d) |
| Trans Conv1D (Stride=4) | (25, 2d, d) | (n, 4096, d) |
| ReLU | | (n, 4096, d) |
| Trans Conv1D (Stride=4) | (25, d, c) | (n, 16384, c) |
| Tanh | | (n, 16384, c) |

***Fig.*** *WaveGAN architecture (generator). c and d are hyperparameters. c=1 and d=64 in the paper.*

The paper noticed that the upsampling layers introduce artifacts resulting in periodic patterns. This would produce undesirable checkerboard anomalies. To prevent the discriminator from learning from these phenomena, the paper introduced phase shuffling layers in the discriminator to shift each layer's activation by a random number between -n and n (n set to 2). This makes it more challenging for the discriminator to distinguish the fake audio samples from the real ones.

Note that here the term phase does not refer to the phase of the generated audio, but to the inner activation layer's output.

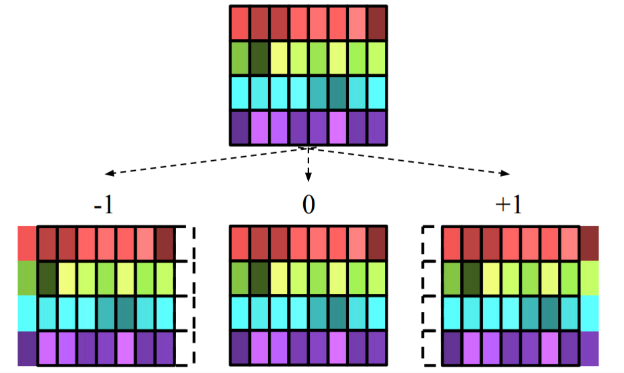


***Fig.*** *Examples of phase-shifting with n between -1 and 1.*

## 2.1.4 Wasserstein loss

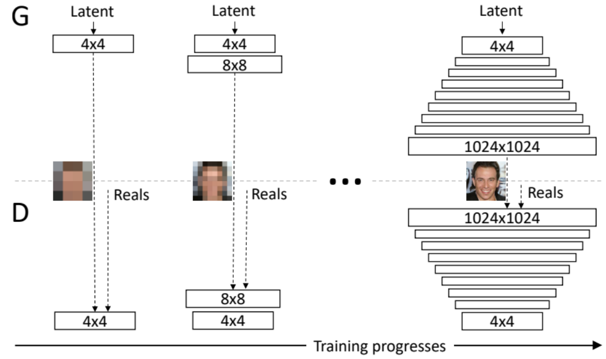
The discriminator of SpecGAN and WaveGAN do not use a cross-entropy loss, but a Wasserstein loss with gradient penalty inspired by WGAN-GP [7]. It approximates the Earth Mover's Distance, and it prevents mode collapse in vanishing gradient problems:

$$V_{\mathrm{WGAN}}(D_w, G) = \mathbb{E}_{\boldsymbol{x}\sim P_X}[D_w(\boldsymbol{x})] - \mathbb{E}_{\boldsymbol{z}\sim P_Z}[D_w(G(\boldsymbol{z}))].$$

***Fig.*** *Wasserstein distance. The discriminator is trained as a function that assists in computing the Wasserstein distance*

## 2.1.5 GANSynth

The second paper that we were aiming to replicate and compare with was the GANSynth paper [9]. For their model, the authors adopt the previous work of Karras et. al. [10] whereby a progressive approach to training GANs is implemented. Essentially, both the generator and discriminator begin training with a low spatial resolution and, over the course of training, new layers are introduced to both networks to allow the model to learn more granular features.



Throughout this entire process, all of the layers are trainable. In their original paper, Karras et. al. show that this leads to more stable GANs [10].

An important difference between the ProgressiveGAN model proposed in [10] and GANSynth is in the latent vector. With the view of achieving controllable pitch, a one-hot encoded vector of musical pitch is appended to the latent vector.

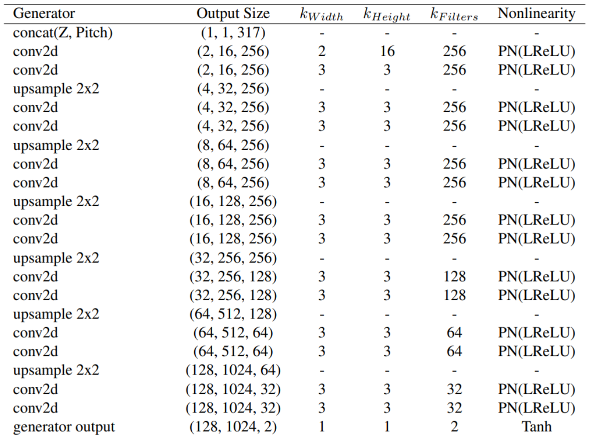
### 2.1.5.1 Audio Representation

Audio representation used by GANSynth is similar to the representation used in SpecGAN. Engel et. al. also utilize a short-time Fourier transform, but with a window size of 1024 and a stride of 256 to obtain a final spectral representation of size (256,512,2). The final dimension is of size 2 as it accommodates the magnitude and phase information for each time-frequency point in the spectrogram. The magnitude is scaled to lie between -1 and 1 to match the output of tanh activation. The phase is then unwrapped. The unwrapped phase is then differentiated using the discrete finite difference operation to obtain the instantaneous frequency. Finally, in an attempt to mimic the human auditory system, both magnitude and phase are transformed into their mel-variants. For the rest of this report, we will refer to this audio representation as the "M-IF-Mel" representation, where 'M' stands for magnitude and 'IF' for instantaneous frequency.

### 2.1.5.2 Model Architecture

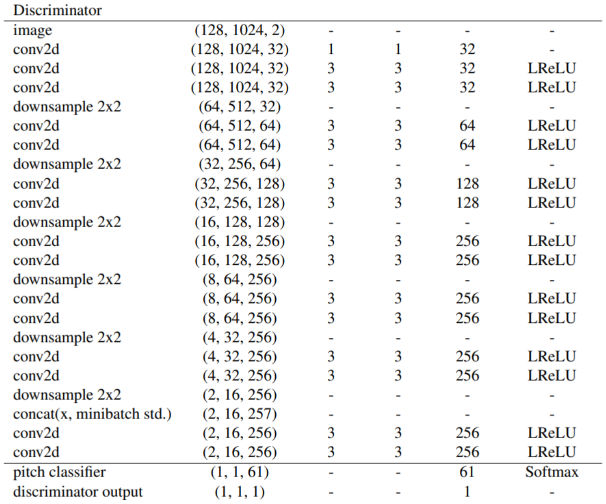
The model architectures for the generator and discriminator of GANSynth are shown in the figures below. Note the size of the latent representation in the architecture of the generator. The latent vector is of size

317, where 256 bins are from the gaussian unit normal distribution and 61 bins are used for pitch encoding.



| Generator | Output Size | $k_{Width}$ | $k_{Height}$ | $k_{Filters}$ | Nonlinearity |
|---|---|---|---|---|---|
| concat(Z, Pitch) | (1, 1, 317) | - | - | - | - |
| conv2d | (2, 16, 256) | 2 | 16 | 256 | PN(LReLU) |
| conv2d | (2, 16, 256) | 3 | 3 | 256 | PN(LReLU) |
| upsample 2x2 | (4, 32, 256) | - | - | - | - |
| conv2d | (4, 32, 256) | 3 | 3 | 256 | PN(LReLU) |
| conv2d | (4, 32, 256) | 3 | 3 | 256 | PN(LReLU) |
| upsample 2x2 | (8, 64, 256) | - | - | - | - |
| conv2d | (8, 64, 256) | 3 | 3 | 256 | PN(LReLU) |
| conv2d | (8, 64, 256) | 3 | 3 | 256 | PN(LReLU) |
| upsample 2x2 | (16, 128, 256) | - | - | - | - |
| conv2d | (16, 128, 256) | 3 | 3 | 256 | PN(LReLU) |
| conv2d | (16, 128, 256) | 3 | 3 | 256 | PN(LReLU) |
| upsample 2x2 | (32, 256, 256) | - | - | - | - |
| conv2d | (32, 256, 128) | 3 | 3 | 128 | PN(LReLU) |
| conv2d | (32, 256, 128) | 3 | 3 | 128 | PN(LReLU) |
| upsample 2x2 | (64, 512, 128) | - | - | - | - |
| conv2d | (64, 512, 64) | 3 | 3 | 64 | PN(LReLU) |
| conv2d | (64, 512, 64) | 3 | 3 | 64 | PN(LReLU) |
| upsample 2x2 | (128, 1024, 64) | - | - | - | - |
| conv2d | (128, 1024, 32) | 3 | 3 | 32 | PN(LReLU) |
| conv2d | (128, 1024, 32) | 3 | 3 | 32 | PN(LReLU) |
| generator output | (128, 1024, 2) | 1 | 1 | 2 | Tanh |

***Fig.*** *GANSynth architecture (Generator).*



| Discriminator | | | | | |
|---|---|---|---|---|---|
| image | (128, 1024, 2) | - | - | - | - |
| conv2d | (128, 1024, 32) | 1 | 1 | 32 | - |
| conv2d | (128, 1024, 32) | 3 | 3 | 32 | LReLU |
| conv2d | (128, 1024, 32) | 3 | 3 | 32 | LReLU |
| downsample 2x2 | (64, 512, 32) | - | - | - | - |
| conv2d | (64, 512, 64) | 3 | 3 | 64 | LReLU |
| conv2d | (64, 512, 64) | 3 | 3 | 64 | LReLU |
| downsample 2x2 | (32, 256, 64) | - | - | - | - |
| conv2d | (32, 256, 128) | 3 | 3 | 128 | LReLU |
| conv2d | (32, 256, 128) | 3 | 3 | 128 | LReLU |
| downsample 2x2 | (16, 128, 128) | - | - | - | - |
| conv2d | (16, 128, 256) | 3 | 3 | 256 | LReLU |
| conv2d | (16, 128, 256) | 3 | 3 | 256 | LReLU |
| downsample 2x2 | (8, 64, 256) | - | - | - | - |
| conv2d | (8, 64, 256) | 3 | 3 | 256 | LReLU |
| conv2d | (8, 64, 256) | 3 | 3 | 256 | LReLU |
| downsample 2x2 | (4, 32, 256) | - | - | - | - |
| conv2d | (4, 32, 256) | 3 | 3 | 256 | LReLU |
| conv2d | (4, 32, 256) | 3 | 3 | 256 | LReLU |
| downsample 2x2 | (2, 16, 256) | - | - | - | - |
| concat(x, minibatch std.) | (2, 16, 257) | - | - | - | - |
| conv2d | (2, 16, 256) | 3 | 3 | 256 | LReLU |
| conv2d | (2, 16, 256) | 3 | 3 | 256 | LReLU |
| pitch classifier | (1, 1, 61) | - | - | 61 | Softmax |
| discriminator output | (1, 1, 1) | - | - | 1 | - |

***Fig.*** *GANSynth architecture (Discriminator)*

## 2.2 Key Results of the Original Papers

### 2.2.1 Training and Datasets

In this section, we provide details about training and the datasets used to train WaveGAN/SpecGAN and GANSynth in the original papers.

#### 2.2.1.1 WaveGAN and SpecGAN

The paper trained the models on various datasets containing a mix of human speeches, and instruments sounds:

1. Drum sound effects (0.7 hours): Drum samples for kicks, snares, toms, and cymbals
2. Bird vocalizations (12.2 hours): In-the-wild recordings of many species (Boesman, 2018)
3. Piano (0.3 hours): Professional performer playing a variety of Bach compositions
4. Large vocab speech (TIMIT) (2.4 hours): Multiple speakers, clean (Garofolo et al., 1993)

Training was done with a batch size of 64, the model capacity d set to 64, and the number of channels to 1. The paper mentioned the use of an Nvidia P100 GPU. The WaveGAN model converged within four days with 200k iterations (3500 epochs), and the SpecGAN model took two days to complete the training phase with 1750 epochs.

#### 2.2.1.2 GANSynth

The paper trains on the NSynth dataset. The NSynth dataset consists of 305,979 musical notes. From commercial sample libraries of 1006 instruments, monophonic audio snippets, each of 4 seconds and 16KHz, were generated which make up the dataset. As the authors also wanted to include human evaluations of audio quality, they constrain the model to train on the subset of acoustic instruments and fundamental pitches ranging from 32-1000Hz, as those timbres are most likely to sound natural to an average listener. The resulting dataset thus contains 70,379 samples.

For the authors, the GAN training took roughly 4 days on a V-100 GPU. They use a batch size of 8, and progressively train higher layers of the network, as has been noted in the previous section. They use 7 progressive training stages, each of which trains on roughly 1.5 million examples.

### 2.2.2 Evaluation and Metrics

It is hard to come up with meaningful metrics to measure the quality of the generated samples. It has been shown [8] that quantitative measures of sample quality are not always correlated with human judgment. However one metric used by the original paper is the Inception score. It consists of training an Inception-V3 model on the dataset. The model then labels the generated samples. From there, the distribution of labels is used to calculate the Inception score according to the following KL-divergence formula:

$$\text{IS}(G) = \exp\left(\,\mathbb{E}_{\mathbf{x}\sim p_g}\, D_{KL}(\,p(y|\mathbf{x}) \parallel p(y)\,)\,\right),$$

Essentially, this score measures the diversity and discriminability of samples by measuring the difference between the overall distribution of classes p(y) versus the distribution with actual data p(y|x). If a model produces diverse, distinguishable samples, then these two distributions would necessarily diverge since an Inception-V3 model would learn the distinguishable features and skew and or alter the original distribution p(y). The Inception Score (IS) metric is used to measure the efficacy of all the three generative models considered in this report.

In the original papers, an inception score of 6.03 was achieved for the SpecGan model on the sc09 dataset, while the WaveGAN model achieved a lower 4.67 inception score. One thing to note is that the inception score is computed from the generated audio samples by converting them into spectrograms. Thus the frequency approach of SpecGAN might artificially boost its score.

Indeed, the paper found that WaveGAN produced better quality samples when heard by humans. GANSynth was able to produce an inception score of 38.1 when trained on the NSynth dataset.

While the inceptions scores of WaveGAN and SpecGAN seem to be far lesser in the paragraph above, it is important to note that GANSynth was trained on a different dataset as compared to WaveGAN and SpecGAN, and we believe that this is the main reason for largely differing inception scores. It has been found in [9] that when WaveGAN is trained on the same dataset as GANSynth, the authors obtained an inception score of 13.7. This is still significantly lower than GANSynth, thus indicating that GANSynth may be the superior model.

## 3. Methodology (of the Students' Project)

With respect to our methodology, our primary aim was to reproduce the results of the original papers. From there, we hoped to gain some insight into how our results might differ from the original paper, as well as how we might be able to extend and improve them.

## 3.1. WaveGAN and SpecGAN

We initially trained our models on the drum sound effect dataset from the original paper. This dataset was chosen primarily because of the short lengths of the audio clips which would lead to shorter training times. The intention behind this was to reduce the time spent training our models, which would afford us more time to experiment with and adjust our models based on results. This was especially important as we hoped to improve upon the models later on in the project after reproducing some results from the original paper. Other datasets were also used, such as the sc09 as it was more suitable for the inception score analysis.

### 3.1.1 Objectives and Technical Challenges

The objectives of this report are to reproduce the architecture of the SpecGAN and WaveGAN models and train them on datasets that are as close as possible to the ones used by the paper. We also perform performance analysis on the inference time, training time, inception score, and human-based analysis.

Because of limited time and resources available we did not reproduce every possible setup addressed in the paper. We did however experiment with different shuffling hyperparameters.

Our training setup used a mix of personal computing grade hardware, and google cloud instances. Graphics cards that were used for training included the Nvidia T4 GPU giving access to 16GB of VRAM and 8.1 TFLOPS of single-precision FP32 performance, and an RTX 3060 with 6GB of VRAM and 12.8 TFLOPS of single-precision FP32 performance. The paper mentioned the use of an Nvidia P100 GPU which is similar in performance to the GPUs that we used, however differences in implementations could also explain some differences in training time.

The small amount of memory and relatively slow training time means that we often reduced the batch size to half of what the paper used, which is 64.

Implementation is done on PyTorch. While the paper used Tensorflow, the framework should not make a difference, and in both cases, models have to be defined with custom layers such as for the phase shuffling layers for example. Careful attention had to be given to the parameters of the trans convolution and convolution so that the output dimension of each layer matches the architecture described by the paper.

### 3.1.2. Problem Formulation and Design Description

Because the models do not weigh a lot, we did not have to make any modifications to the models to make them run on our hardware. As such, the generator is the same as described in part 2, and the discriminators have the following architecture:

| Operation | Kernel size | Output Shape |
|---|---|---|
| Input x or G(z) | | (n, 128, 128, c) |
| Conv2D (Stride=2) | (5, 5, c, d) | (n, 64, 64, d) |
| LReLU ($\alpha$=0.2) | | (n, 64, 64, d) |
| Conv2D | (5, 5, d, 2d) | (n, 32, 32, 2d) |

| | | |
|---|---|---|
| (Stride=2) | | |
| LReLU ($\alpha$=0.2) | | (n, 32, 32, 2d) |
| Conv2D (Stride=2) | (5, 5, 2d, 4d) | (n, 16, 16, 4d) |
| LReLU ($\alpha$=0.2) | | (n, 16, 16, 4d) |
| Conv2D (Stride=2) | (5, 5, 4d, 8d) | (n, 8, 8, 8d) |
| LReLU ($\alpha$=0.2) | (5, 5, 8d, 16d) | (n, 4, 4, 16d) |
| Conv2D (Stride=2) | | (n, 4, 4, 16d) |
| Reshape | | (n, 256d) |
| Dense | (256d, 1) | (n,1) |

***Fig.*** *SpecGAN architecture (Discriminator). c and d are hyperparameters. c=1 and d=64 in the paper*

| Operation | Kernel size | Output Shape |
|---|---|---|
| Input x or G(z) | | (n, 16384, c) |
| Conv1D (Stride=4) | (25, c, d) | (n, 4096, d) |
| LReLU (α = 0.2) | | (n, 4096, d) |
| Phase Shuffle (n = 2) | | (n, 4096, d) |
| Conv1D (Stride=4) | (25, d, 2d) | (n, 1024, 2d) |
| LReLU (α = 0.2) | | (n, 1024, 2d) |
| Phase Shuffle (n = 2) | | (n, 1024, 2d) |
| Conv1D (Stride=4) | (25, 2d, 4d) | (n, 256, 4d) |
| LReLU (α = 0.2) | | (n, 256, 4d) |
| Phase Shuffle (n = 2) | | (n, 256, 4d) |
| Conv1D (Stride=4) | (25, 4d, 8d) | (n, 64, 8d) |
| LReLU (α = 0.2) | | (n, 64, 8d) |
| Phase Shuffle (n = 2) | | (n, 64, 8d) |
| Conv1D (Stride=4) | (25, 8d, 16d) | (n, 16, 16d) |
| LReLU (α = 0.2) | | (n, 16, 16d) |
| Reshape | | (n, 256d) |
| Dense | (256d, 1) | (n, 1) |

***Fig.*** *WaveGAN architecture (Discriminator). c and d are hyperparameters. c=1 and d=64 in the paper*

Since the Discriminator takes more time to learn the dataset, it makes five steps for every step the generator takes.

Because the datasets can have various formats that are different from the desired input format, there is some audio processing to do when loading the datasets. The audio samples have to be split and resampled to 16 kHz clips of length 16,384.

When the audio sample is short, we pad the end with silence i.e. zero pad. For audio samples that are really long such as the piano tracks, we would split them into chunks of the same length to generate multiple samples from the original one.

An additional step has to be done for the SpecGAN data loading step: the 16,384 long audio sample is then converted into a spectrogram using a window size of 16 ms, and 8 ms of stride to generate 128x128 spectrogram images.

The generated images can be inverted back into audio using the Griffin-Lim algorithm with 16 iterations. We have to use such an algorithm as the generator does not generate a real spectrogram.

## 3.2. GANSynth

In our project, we train the GANSynth model on the same dataset that was used in the original paper – NSynth. While the original paper trains various models for different audio representations such as magnitude-only spectrogram, M-IF representation (refer to section 2.1.5.1), M-IF-Mel representation, and another representation of M-IF-Mel with higher spectrogram resolution, we train the M-IF-Mel model in the interest of time as each model takes roughly 4 days to train on the NSynth dataset. The implementation is done using the TensorFlow framework, which is the same as the one used by the authors of the original paper [11].

The architecture used by us to implement GANSynth is exactly the same as the one used in the paper, which has been detailed in the figures in section 2.1.5.2.

# 4. Implementation

## 4.1. SpecGAN and WaveGAN

In this section, we describe the methods we used to reproduce the SpecGAN and WaveGAN models. We will provide specific details on the datasets used and the settings that were chosen for training.

### 4.1.1 Data

| Dataset name | # samples | # classes | # size |
|---|---|---|---|
| drum | 3,002 | 8 | 105MB |
| Piano | 19 | - | 439 MB |
| sc09 | 18,620 | 10 | 576 MB |

***Fig.*** *Datasets used. The piano dataset only has 19 tracks, but they are split resulting into about 1400 samples.*

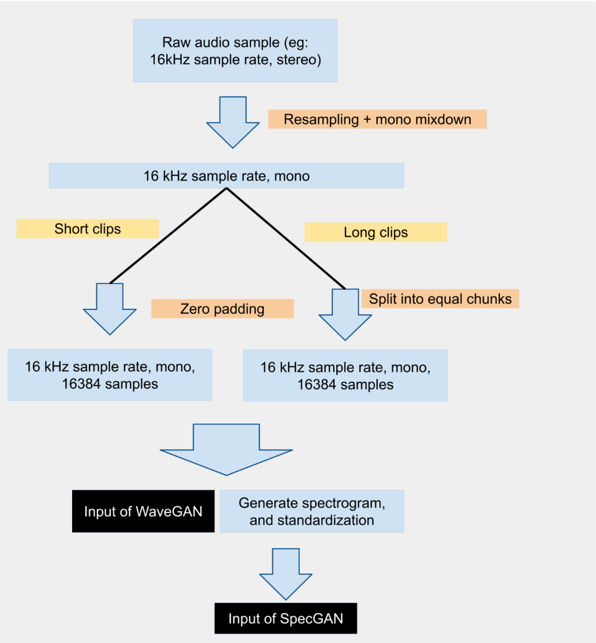

Raw audio sample (eg: 16kHz sample rate, stereo)
Resampling + mono mixdown
16 kHz sample rate, mono
Short clips
Long clips
Zero padding
Split into equal chunks
16 kHz sample rate, mono, 16384 samples
16 kHz sample rate, mono, 16384 samples
Input of WaveGAN
Generate spectrogram, and standardization
Input of SpecGAN

***Fig.*** *Data processing steps*

All chosen datasets were also used by the paper. The sc09 dataset contains human speeches while the drum and piano datasets have instrument sounds.

Both the sc09 and the drum datasets have very short samples close to one second in duration. This means that we only had to cut or pad them to the 16,384 length. The piano dataset contains very long musical clips, so they were cut into multiple chunks to generate data samples.

**Data processing**: all data processing follows the above ***Fig.*** steps. The Librosa library and Numpy were used to load the audio clips and convert them into the appropriate format. Torchaudio was used to compute the spectrograms and to convert the images back to audio clips with the Griffin-Lim algorithm.

**Data loading**: our Pytorch Dataset class implementation allows us to load the clips in multiple ways. They can be loaded from the disk every time a clip is fed into the model, or they can all be loaded directly into the system memory or the VRAM memory, assuming there is enough space available. The relatively lightweight nature of the datasets means that if possible, it is advantageous to load it into the system memory or the GPU memory as the instance hard drive access speed can quickly become a performance bottleneck, even if multithreading is enabled. The more powerful the GPU, the more important it is to make sure that its computing power is not held back by the data loading step.

Loading the data into the DRAM of the VRAM is similar in terms of performance. While data on the DRAM has to be copied into the GPU, this method can be easily parallelized with PyTorch's DataLoader class, which is not possible on the GPU resulting in crashes for more than 1 worker.

### 4.1.2 Deep Learning Network

Training was done with parameters as close to the paper as possible. However, some details have been changed, mainly to reduce the training time and memory footprint of our models.

To compare SpecGAN and WaveGAN, all parameters are kept the same for both models for a given dataset when possible.

| Name | Value - paper | Value - our implementation |
|---|---|---|
| Num channels (c) | 1 | 1 |
| Batch size (b) | 64 | 32,25,64 |
| Model dimensionality (d) | 64 | 64 |
| Phase shuffling (WaveGAN) | 2 | 2 |
| Phase shuffling | 0 | 0 |

|  |  |  |
|---|---|---|
| (SpecGAN) |  |  |
| Loss | WGAN-GP | WGAN-GP |
| WGAN-GP λ | 10 | 10 |
| D updates per G update | 5 | 5 |
| Optimizer | Adam<br>($\alpha$=1e-4,<br>β1=0.5,<br>β2=0.9 ) | Adam<br>($\alpha$=1e-4,<br>β1=0.5,<br>β2=0.9 ) |

***Fig.*** *Training hyperparameters*

It took from one day to two days to train on a given dataset. Generating samples with the drum dataset was the quickest to achieve, probably because of the very short audio clips. The piano dataset is also able to converge quickly with good piano-like samples after 15k iterations.

The sc09 takes twice as long to generate human-like speeches and as such has been trained for 70k iterations.

To keep track of the progress, the weights of both discriminator and generator are saved periodically every k epochs (k depending on the dataset). At the same time, some generated samples are also saved to monitor how the generator evolves.

The models are randomly initialized with weights having a mean of 0 and a standard deviation of 0.02 similarly to how a DCGAN is initialized.

| **Value** | **Model** | **Drum** | **Piano** | **sc09** |
|---|---|---|---|---|
| Batch size | SpecGAN + WaveGAN | 32 | 32 | 64 |
| Iterations | SpecGAN + WaveGAN | 30k | 30k | 70k |
| Training time | SpecGAN | 1 day | 1 day | 2 days |
|  | WaveGAN | 1 day | 1 day | 2 days |

***Fig.*** *Training time on the T4 GPU*

### 4.1.3 Software Design

We built the SpecGAN and WaveGAN models using PyTorch and its implementation of trans convolution and convolution layers. The phase shuffling layer has to be implemented in a custom layer, and the WGAN-GP loss function had to be implemented from scratch.

Most of the implemented functions allow working with different sampling rates, and clip lengths, allowing for potential exploration/extension of our work.

```
Phase shuffling

Input: n, tensor X of shape (batch_size, channels, x_len)

let r be a random int between [-n,n]

if n=0:
   return X

else:
   for each batch_size b and channel c:
       shift X[b,c] by r indexes (circular index)

   return X
```

***Fig.*** *Pseudo-code for the phase shuffling layer*

```
Wasserstein loss

Input: eps, lambda, tensor real, tensor generated
// generated = Generator(noise)

interpolated_sound=(1 - eps) * real + (eps) * generated

mixed_score = discriminator(interpolated_sound)

gradients=$\partial mixedscore / \partial interpolatedsound$

grad_penalty = (lambda*||gradients||^2)

// normal Wasserstein loss
loss_GP = discriminator(generated).mean() - discriminator(real).mean()

// adding gradient penalty

 loss_GP += grad_penalty

 return loss_GP
```

***Fig.*** *Pseudo-code for the Wasserstein loss with gradient penalty.*

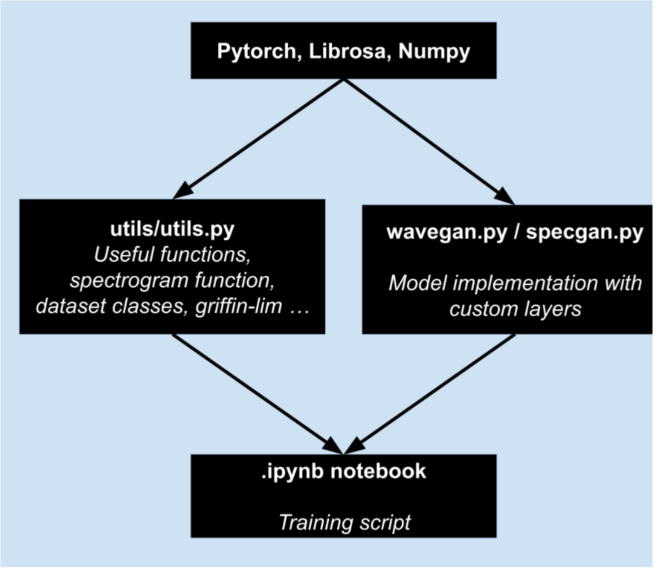

Pytorch, Librosa, Numpy
utils/utils.py
Useful functions, spectrogram function, dataset classes, griffin-lim …
wavegan.py / specgan.py
Model implementation with custom layers
.ipynb notebook
Training script

***Fig.*** *Simplified code dependency diagram*

## 4.2. GANSynth

### 4.2.1 Data

The data used to train our GAnSynth model is the NSynth dataset, which is the same as the one used in the original paper. The original NSynth dataset consists of roughly 300,000 examples from 1000 different instruments. However, after only using examples of acoustic instruments with fundamental frequencies in the range of 32-1000 Hz, the total number of examples is then 70,379.

### 4.2.2 Data Processing

Since the audio representation that we are considering is the M-IF-Mel representation, the appropriate processing needs to be completed on the time-domain audio signal. As a recap, the processing quired to generate the M-IF-Mel representation consists of 3 main steps – (i) generating spectrogram using STFT, (ii) generating Instantaneous Frequencies (IF) of the phases by taking the finite difference, and (iii) applying the Mel Transform to both magnitude and phase components.

All the required helper functions to complete the audio processing are provided in [11], which we utilize. No additional frameworks are required for this processing, and the processing is done using simple mathematical operations on TensorFlow tensors.

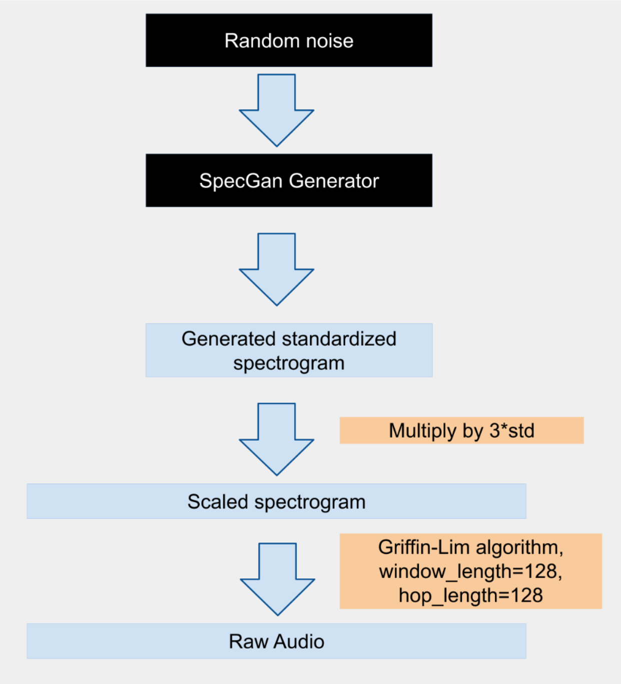

Random noise
SpecGan Generator
Generated standardized spectrogram
Multiply by 3*std
Scaled spectrogram
Griffin-Lim algorithm, window_length=128, hop_length=128
Raw Audio

***Fig.*** *Transformations from noise to generate audio clip for SpecGAN.*

**Github link:** https://github.com/ecbme6040/e6691-2022spring-project-WAVE-an3078-bmh2168-gs3160

### 4.2.3 Training Details

The model used by us to train GANSynth is exactly the same as the model used by the authors in [9], and training is almost entirely the same as [11]. After doing a sweep over learning rates, the authors in [9] found that a learning rate of 0.0008 was best, and that is the value that we use. Further, we use the ADAM optimizer, as is done in [9].

As in the original progressive GAN paper, both discriminator and generator networks use box upscaling/downscaling and the generators use pixel normalization, which is given as

$$x = x_{nhwc}/(\frac{1}{C}\sum_{c} x_{nhwc}^2)^{0.5}$$

where n, h, w, and c refer to the batch, height, width, and channel dimensions respectively, x is the activations, and C is the total number of channels.

In the interest of trying to speed up the training process, we use a batch size of 16 instead of 8, which is used in the paper. The network was progressively trained on a V-100 GPU, and took approximately 5 days to train (350 epochs).

It is important to note here that, unlike SpecGAN, GANSynth is essentially an end-to-end method. What we mean by that is, that the representations fed to the discriminator and the representations generated by the generator have a closed-form mapping to the time-domain audio. In SpecGAN, the Griffin-Lim algorithm must be used to estimate the time-domain signal from the magnitude spectrogram as the phase information is lost. This additional step of performance degradation is not present in GANSynth.

## 5. Results

### 5.1 Project Results

Some examples of generated audio clips, along with comparisons with the paper's examples are included on the demo site https://ecbme6040.github.io/e6691-2022spring-project-WAVE-an3078-bmh2168-gs3160/.

### 5.1.1 Human evaluation

It is interesting to note that the speed at which the model is able to generate clips that resemble the dataset depends on the dataset itself, meaning that some datasets are easier to train than others. The drum dataset was the fastest one to generate satisfactory samples, with WaveGAN being able to produce short drum-like samples after less than 10k iterations while the sc09 was the harder dataset to train with human-like clips being produced after 30k iterations.

Those observations are based on human evaluation, meaning that the human perception might not reflect the state of convergence of a model, or that the Wasserstein distance might not indicate discrepancies on the same scale as human perception.

It also seems that the phase shuffling parameter degrades the quality of the generated sample with noticeable artifacts when compared to training without phase shuffling.

### 5.1.2 Inception score

We tested the quality of generated clips with the sc09 dataset as it had clear labels for each number from 0 to 9, allowing us to implement the inception score. The inception score allows us to test the diversity of the generated sample across all the classes.

To measure the inception score, the audio clips have to be transformed into spectrogram images in order to be fed into the inception-V3 model.

The trained inception model using transfer-learning is used to test the generated samples. The score is maxed when all predictions have strong confidence, and when they are spread out across all classes.

| | SpecGAN | WaveGAN |
|---|---|---|
| Our result | 1.48 | 3.77 |

***Fig.*** *Inception score on the sc09 dataset using 1000 generated samples.*

We will discuss the differences with the paper in the next section.

What is not expected is that the SpecGAN score is considerably lower than the WaveGAN score. We should expect the opposite, as the SpecGAN already generates spectrograms, and as such should have an advantage in the image-based inception score, as the paper suggested. From our observation, this can be explained by the fact that the quality of the generated samples from the SpecGAN model varies significantly depending on the rescaling parameters and the Griffin-Lim parameter. It is possible that with some tuning a higher quality score can be achieved.

The rescaling of the spectrograms proposed by the paper is to make most of the information sit between -1 and 1, as the activation function of the output layer is tanh. We also tried to use a ReLU output activation function with no rescaling to see if that would be better, but that method resulted in worse audio quality on the drum dataset. With more time some other transformations, and activation functions can be explored.

The GANSynth model produced an inception score of 24.62. The methods to compute the inception score for GANSynth have been directly used from [11].

### 5.1.2.1 Discussion

One important factor that could explain the disparity in inception scores between GANSynth and WaveGAN/SpecGAN is that they are trained on different datasets, with the NSynth dataset being a much larger dataset.

Secondly, GANSynth is designed in such a way that there is conditioning on the pitch which tells them what pitch to generate. Therefore, these generated audio clips are usually classified correctly. This is also an indicator that the inception score may not be the best metric to judge the performance of GANSynth versus SpecGAN or WaveGAN. The authors in [9] consider some other metrics such as "Number of Statistically-Different Bins" (NDB) and "Frechet Inception Distance" which we have not focused on in this project, and it is an avenue for future work.

Finally, GANSynth does not employ a phase-estimation algorithm like Griffin-Lim to reconstruct time-domain audio, thus eliminating a lossy step as compared to, say, SpecGAN.

### 5.1.3 Other observations

Our models are relatively lightweight and can easily be loaded into a consumer grader GPU such as the Nvidia RTX 3060 with 6GB of VRAM, with room to spare to load data into memory.

| model | # parameters | weight size on disk |
|---|---|---|
| SpecGAN Generator | 19.0 M | 74 MB |
| SpecGAN Discriminator | 17.4 M | 68 MB |
| WaveGAN Generator | 19.1 M | 74 MB |
| WaveGAN Discriminator | 17.4 M | 68 MB |

***Fig.*** *Model sizes.*

**Inference speed**: our WaveGAN and SpecGAN have similar performance and can generate a sample in 17 ms on a consumer mobile AMD 5900HS CPU, and in 0.015 ms on a mobile Nvidia RTX 3060 GPU. Thus inference on a GPU yields an improvement in speed of 1000.

## 5.2 Comparison of the Results Between the Original Paper and Students' Project

The inception score that we have is lower than what is reported on the paper. This is however not surprising given the lower number of training iterations that probably result in worse generated audio clips.

|  | SpecGAN | WaveGAN | GANSynth |
|---|---|---|---|
| Our result | 1.48 | 3.77 | 24.62 |
| Paper | 6.03 | 4.67 | 38.1 |

***Fig.*** *Inception score on the sc09 dataset using 1000 generated samples. Comparison with the paper*

With a human evaluation one can notice that the quality of the generated sample on WaveGAN is quite similar to the examples of the paper, if not slightly worse. However, the examples given by the researchers may have been cherry-picked, and as such, it is hard to make fair comparisons on a small number of available audio clips, even for a human-based evaluation.

For SpecGAN, we noticed that even the paper's examples can contain noticeable artifacts that make the audio clip hard to understand/use, notably for the drum dataset. This is something that we also faced, which might be a clue that the inception score is not ideal, and does not correlate well with human-based evaluations.

Once again, we note that the disparity in inception scores between GANSynth and the other two models has been discussed in section 5.1.2.1.

## 5.3 Discussion / Insights Gained

We learned that existing image GANs can be repurposed to audio synthesis applications by using two possible approaches, a frequency-based approach and a direct one that uses one-dimensional flattened layers.

While we successfully implemented the model and used similar hyperparameters overall, we had difficulties generating high-quality audio clips with the SpecGAN model. We discovered that the time-frequency approach using spectrograms and the Griffin-Lim algorithm is very sensitive to certain parameters such as the rescaling factor, and activation layer.

# 6. Future Work

In the future, we would like to tune the scaling of the spectrograms before they are used to train the SpecGAN discriminator to improve the quality of the generated samples. A nonlinear rescaling that takes into account the distribution of the information, and the way the human ear interprets the frequencies could be used to achieve better results. In a similar fashion, a different activation function can be tested for the output of the SpecGAN model.

It would also be interesting to experiment with different values of the capacity hyper-parameter d, and to adapt the network to generate longer audio sequences.

To measure the quality of the generated audio clips, it would be relevant to replace the inception score with a similar score that is not computed with the help of a CNN classifier but using a classifier that works directly with a one-dimensional audio sequence. As such, the score would not favor models that are based on spectrogram images.

Finally, a broader expansion of our work that seems natural would be to adapt other types of GANs to our application such as CycleGAN or a conditional GAN.

## 7. Conclusion

This report summarizes the papers on which SpecGAN, WaveGAN, and GANSynth are based. We successfully implemented the SpecGAN and WaveGAN models in PyTorch and were able to generate different sounds based on various datasets. We were also able to experiment with GANSynth.

We were able to compare both the spectrogram approach and the direct one. While the metrics suggest that our results differ from the papers, a human evaluation actually finds that there are a lot of similarities in the generated audio clips. Similar to the paper we noticed that SpecGAN generally produced worse samples.

Experimenting with different datasets and parameters allowed us to come up with ideas to further improve the models and explore other ways to generate audio clips.

## 8. Acknowledgement

The implementation of the models was possible thanks to the work of the authors of the respective papers [3][9], and the teaching and supervision of **Prof. Z. Kostic** and **Feroz Ahmad** (TA**).**

The documentation of PyTorch, Torchaudio, and TensorFlow were useful for the training functions, data loading, and processing code.

## 9. References

[1] Anh-Vu Nguyen, Brian Hernandez, Gokul Srinivasan (2022). Adversarial Audio Synthesis [code] https://github.com/ecbme6040/e6691-2022spring-project-WAVE-an3078-bmh2168-gs3160

[2] Anh-Vu Nguyen, Brian Hernandez, Gokul Srinivasan (2022). Adversarial Audio Synthesis [Slides] https://docs.google.com/presentation/d/1yDbBOl-8S3AutIVb8D06H0aAWWslzokgVKZWV6CMTSU/edit?usp=sharing

[3] Donahue, C., McAuley, J. and Puckette, M., 2018. Adversarial Audio Synthesis.

[4] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. In NIPS, 2014.

[5] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, ¨ Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. WaveNet: A generative model for raw audio. arXiv:1609.03499, 2016

[6] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In ICLR, 2016

[7] Martin Arjovsky, Soumith Chintala, and Leon Bottou. Wasserstein GAN. In ´ ICML, 2017.

[8] Lucas Theis, Aaron van den Oord, and Matthias Bethge. A note on the evaluation of generative ¨ models. In ICLR, 2016.

[9] Engel J, Agrawal KK, Chen S, Gulrajani I, Donahue C, Roberts A. Gansynth: Adversarial neural audio synthesis. arXiv preprint arXiv:1902.08710. 2019

[10] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of GANs for improved quality, stability, and variation. In ICLR, 2018a.

[11]https://github.com/magenta/magenta/tree/main/magenta/models/gansynth/lib

## 10. Appendix

### 10.1 Individual Student Contributions in Fractions

| | an3078 | bmh2168 | gs3160 |
|---|---|---|---|
| Last Name | Nguyen | Hernandez | Srinivasan |
| Fraction of (useful) total contribution | 1/3 | 1/3 | 1/3 |
| What I did 1 | Worked on WaveGAN and SpecGAN | Worked on WaveGAN and SpecGAN | Worked on GANSynth |
| What I did 2 | Wrote the report | Wrote the report | Wrote the report |