

**香港中文大學**

**計算機科學及工程學系**

**Department of Computer Science and Engineering,**

**The Chinese University of Hong Kong**

**Proposal Title**

Audio Synthesis with Generative Adversarial Networks

**Name(s)**

**CHU Chun To 1155127149**

**CHOI Sen Ho 1155125303**

Supervised By

**Prof. LEUNG Kwong-Sak**

©2020 The Chinese University of Hong Kong

The Chinese University of Hong Kong holds the copyright of this proposal.

Any person(s) intending to use a part or whole of the materials in the thesis

in a proposed publication must seek copyright release from the University.

Contents

[1. Introduction 3](#_Toc53093085)

[1.1. Motivation(s) 3](#_Toc53093086)

[1.2. Objective(s) 3](#_Toc53093087)

[2. Background (Survey/System Analysis) 4](#_Toc53093088)

[3. Proposal 10](#_Toc53093089)

[3.1. User Specifications 10](#_Toc53093090)

[3.2. Design Specifications 11](#_Toc53093091)

[4. Milestones 13](#_Toc53093092)

[5. Schedule 13](#_Toc53093093)

1. Introduction
   1. Motivation(s)

Audio synthesizing is important to the development of computer music, electronic music and even AI music generation. Nowadays, synthesizing audio for specific domains has many practical applications in creative sound design for music and film. The demand for audio is high for different purposes, such as musicians finding sound effects for specific scenarios.

One popular approach of audio synthesizing is using Generative adversarial networks (GANs). In current stage, GANs could enable rapid and straightforward sampling of large amounts of audio. Different models such as WaveGan, SpecGan, ParallenGan, GanSynth have their attempts on audio synthesis. They indeed produce some promising result and able to generate a lot of audio using small amount of time. However, the quality of result varies among different models and some of them has a lower matching human perception. Also, since audio is, unlike simple 2D images, time-domain data, the different representations of audio input will cause different restrictions during GANs training. Therefore, although GANs have seen wide success at generating images that are both locally and globally coherent, they have seen little application to audio generation.

Even though today we have some powerful audio representations instead of raw audio waveform, such as Discrete Fourier Transform (DFT), Short-Time Fourier Transform (STFT), aligned linguistic features, to narrow the limitations, we still need to find the best one for different training models, especially the one suiting ours. This is important for us to have a good starting point in order to build the optimal model. Also, in order to generate audio with higher quality and quantity (longer in length), we need to enhance our training model by using latest technology related to GAN combining with promising audio input pre-processing. Hopefully, we can have some new discovery in the audio synthesis technology, and this would help us chasing the current progress of digital music development.

* 1. Objective(s)

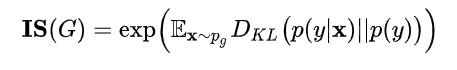
To achieve higher quality and quantity of generated audio, quantity of generated audio refers to a longer length of audio, while quality refers to a better hearing effect from the human perception. To measure how good our model when compared to other models can do, several measurements including both qualitative and quantitative will used in the evaluation.

**Human Evaluation** (if funding provided):

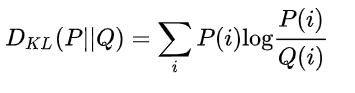
Human perception is important for reviewing the generated audio samples. For the generated examples, we can use Amazon Mechanical Turk to perform a comparison test on examples for different models which putting in for comparison.

**Inception Score (IS) (Salimans, 2016)**:

It reflects both distinction and variety of the group of generated audios. It is defined as the mean KL divergence between the label distribution p(y|x), and the marginal distribution p(y) using the predictions of an Inception classifier.



KL divergence:



The Inception classifier return the probability distribution of the input data. The Inception Nets will be trained on the tasks of instrument and pitch classification. The label distribution of an output represents the probability of output belonging to a certain category, which indicates the distinction of generated outputs, while the marginal distribution is given by summing the label distribution of many generated samples, which indicates the variety of generated outputs. KL divergence can measure how different two probability distributions are. High KL divergence indicates the ideal situation.

**Frechet Audio Distance (FAD) (Kilgour, 2018)**:

It compares the statistics of real and fake data computed from an embedding layer of a pre-trained VGGish model (Kilgour, 2018). Viewing the embedding layer as a continuous multivariate Gaussian,

the mean *µ* and covariance Σ are estimated for real and fake data. The FID between the real images x and generated images g is computed as:

where Tr is sum of all diagonal elements.

Lower FAD means smaller distances between synthetic and real data distributions.

1. Background (Survey/System Analysis)

GANs have seen wide success at generating images that are both locally and globally coherent, but they have seen little application to audio generation due to limitations of audio data. One of the restrictions is the time-domain features of audio representations.

**2.1 Data representation**

Modelling raw audio is a particularly challenging problem because of the high temporal resolution of the data (usually at least 16,000 samples per second) and the presence of time-domain structure at different timescales with short and long-term dependencies.

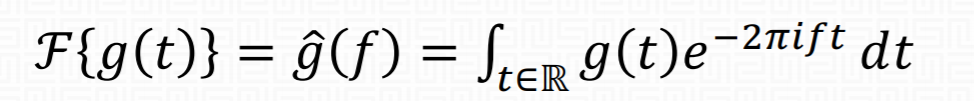
Thus, instead of modelling the raw temporal audio (waveform) directly, most approaches simplify the problem by modelling a lower-resolution representation that can be efficiently computed from the raw temporal signal (Kumar, 2019). Also, since deep learning for audio has shifted from using hand-crafted features requiring prior knowledge, to features learned from raw audio data or mid-level representations, this has allowed us to build models requiring less prior knowledge, yet at the expense of data, computational power, and training time (Nistal, 2020).

There are several representations of audial data in a deep-learning approach, include aligned linguistic features and different types of spectrograms. The former is usually the intermediate representation given text as input, which is more likely processing speeches. The latter transforms the intermediate representation back to audio, which is more applicable in general cases of audio synthesis. Therefore, in our project, we will mainly focus on spectrograms as the intermediate representation. We will justify the selection of data representation by testing different representation in our model.

Spectrogram is a frequency domain (magnitude vs frequency) representation of the original time domain (amplitude vs time) audio data. It can be obtained by Fourier analysis such as Discrete Fourier Transform (DFT), Fast Fourier Transform (FFT), Short-Time Fourier Transform (STFT), etc.. They are commonly used intermediate representations suiting different audio data.

**2.1.1 Discrete Fourier Transform**

The basic idea of Fourier Analysis is to extract waves of different frequencies and magnitudes consisted in an audio soundwave. Here is a mathematical equation of how it works:



Notice that , so the equation can be viewed as counting the occurrence of frequencies in the waveform by spinning the signal around a circle at the particular frequency. However, in a digital system, the input values (audio samples) are equally spaced. This function is continuous for and , which cannot be applied to digital signals. Therefore, we have to deal with the sample in a discrete approach, to find the sum of finite series of sinusoidal waves. For a sequence of complex samples , we can turn it into a sequence of complex numbers:

Here, the series are called the DFT coefficients of frequency bins. We can easily find the magnitudes (), phase () and bin frequency with some easy mathematical approaches.

DFT is popular in digital signal processing because of its simplicity. DFT is usually applied simply as a black box in different programming libraries, without understanding the math behind. Also, it is usually implemented as Fast Fourier Transform (FFT) with complexity , where traditional DFT is . However, performing the DFT on a time series will give us the overall frequency components for the entire time series. It can only show the general “histogram” of frequencies, which cannot tell us how the frequency appears to be changing over time.

**2.1.2 STFT**

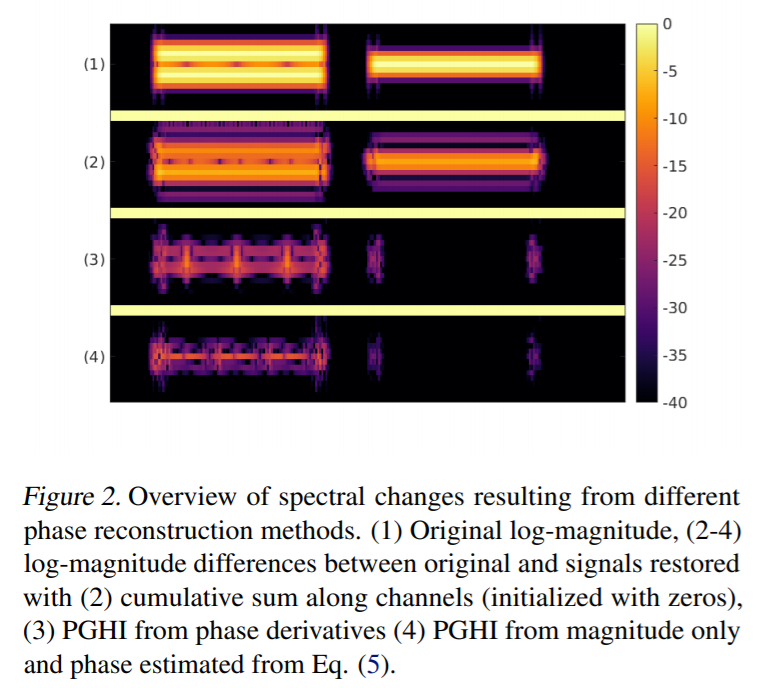
By Short-Time Fourier Transform (STFT), we can break the process into multiple DFT (as well as FFT) in time segments to analyze the frames. STFT decomposes a signal as a weighted sum of complex sinusoidal basis vectors with linearly spaced center frequencies, unveiling the time-frequency structure of an audio signal.

It is commonly decomposed into magnitude and phase components. The latter is typically noisy, which makes it difficult for neural networks to model. This problem is mitigated by using the Instantaneous Frequency (IF), providing a measure of the rate of change of the phase information over time. The STFT of a finite, real signal , with the analysis window , time step and frequency channels is given by:

一張含有 文字 的圖片

自動產生的描述

We can get a spectrogram by using STFT, which shows the magnitude and frequency changes within a time period (magnitude vs frequency vs time). Sometimes, spectrogram with Mel scale (log scale) is applied due to the features of changing in sound frequencies. STFT is cheap to compute and perfectly invertible, which makes it popular for audio synthesis.



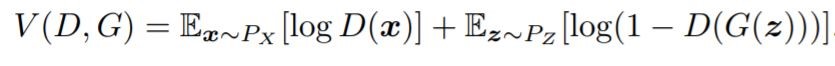
From the above figure, we can see the difference forms of spectrogram after STFT reconstruction of audio. We will test different kinds of spectrogram to figure out the easiest and fastest one to be applied before training the GNA models.

**2.2 GAN models**

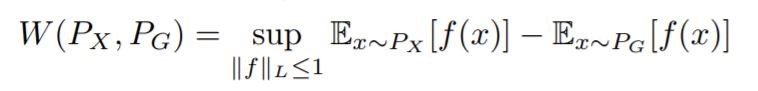
It is hard to look for the best representation in general cases, but the options can be narrowed down after discovering the GANs training modal.

**2.2.1 WGAN-GP (Arjovsky, 2017)**

Generative adversarial network is an unsupervised learning that given a certain data set, they can produce new data that resemble the given data. Those generated data can reach a high level of realism. The network consists of two parts: a generator G and a discriminator D. The generator learns to generate plausible data to deceive the discriminator, while the discriminator learns to distinguish the generator's fake data from real data. Goodfellow et al. (2014) uses the following objective function where G is trained to minimize it and D is trained to maximize it.



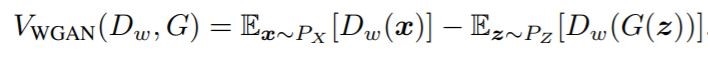
Px represents the real data distribution and Pz represents the random data with Gaussian distribution. The generator will be trained to map those random data to data like real data. However, this training model is difficult to train, the unstableness brought by the serious vanishing gradient due to Jensen-Shannon divergence, the low variety of generated samples brought by the model collapse due to Kullback–Leibler divergence (Arjovsky, 2017). Therefore, a recent improvement on GAN is using Wasserstein-1 distance between generated and data distributions. The Wasserstein distance is the minimum cost of transporting mass in converting the data distribution Px to the data distribution Pg.



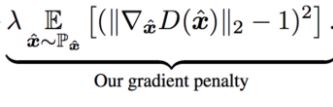
where ||f||L ≤ 1 is the family of functions that are 1-Lipschitz where it follows Lipschitz constraint.

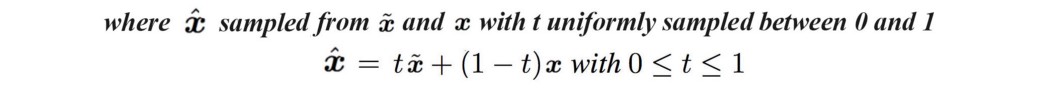


To minimize Wasserstein distance, the objective function is changed as below.



This represents Wasserstein Generative adversarial network. To maintain Lipschitz continuity, weight clipping is used as enforcing that Dw is 1-Lipschitz. The incorrect tuned hyperparameter in weight clipping will produce poor result. Proven in paper of Improved Training of Wasserstein GANs (Gulrajani, 2017), a differentiable function f is 1-Lipschitz if and only if it has gradients with norm at most 1 everywhere. Gradient penalty make use of this feature and it replaces weight clipping. It will allow the model to penalize the model if the gradient norm moves away from its target norm value 1. The additional term of the objective function will be the following.

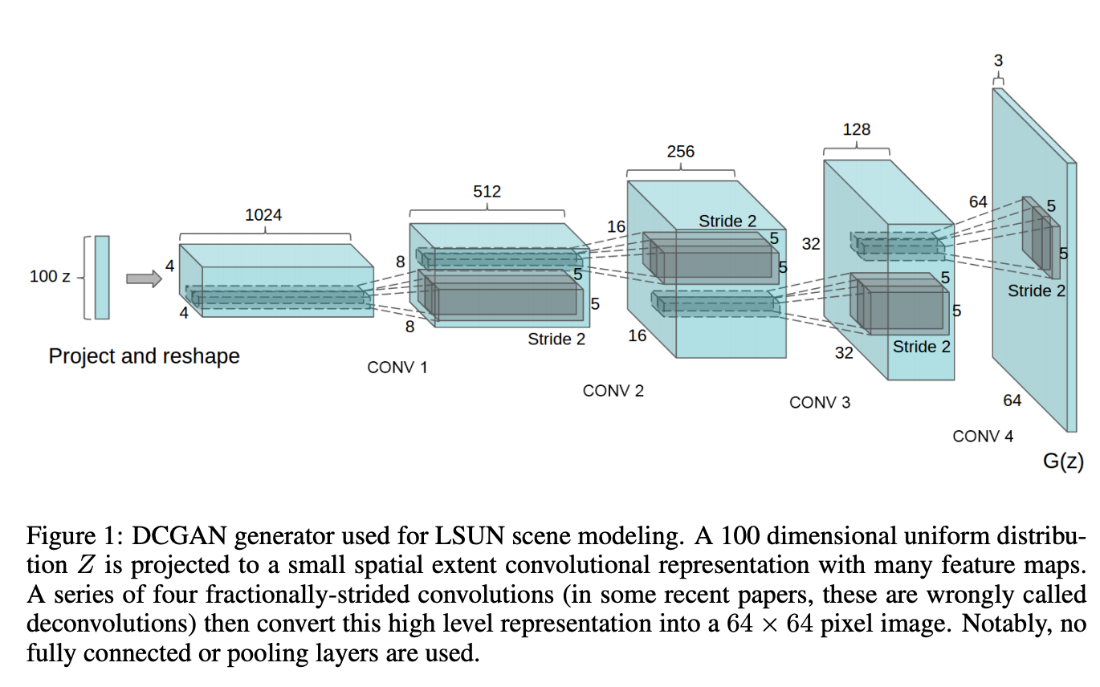




This will be the whole picture of WGAN-GP. The training will be based on WGAN-GP and the detailed implementation will be shown in later part.

**2.2.2 DCGAN (Radford, 2015)**

The audio is constructed with multiple periodic sine wave. To extract the periodic bias in audio, convolution layers is used to extract those features. Deep Convolutional Generative Adversarial Network (DCGAN) will include inside the model, it explicitly uses convolutional and convolutional-transpose layers in the discriminator and generator (Radford, 2015).



(Radford, 2015)

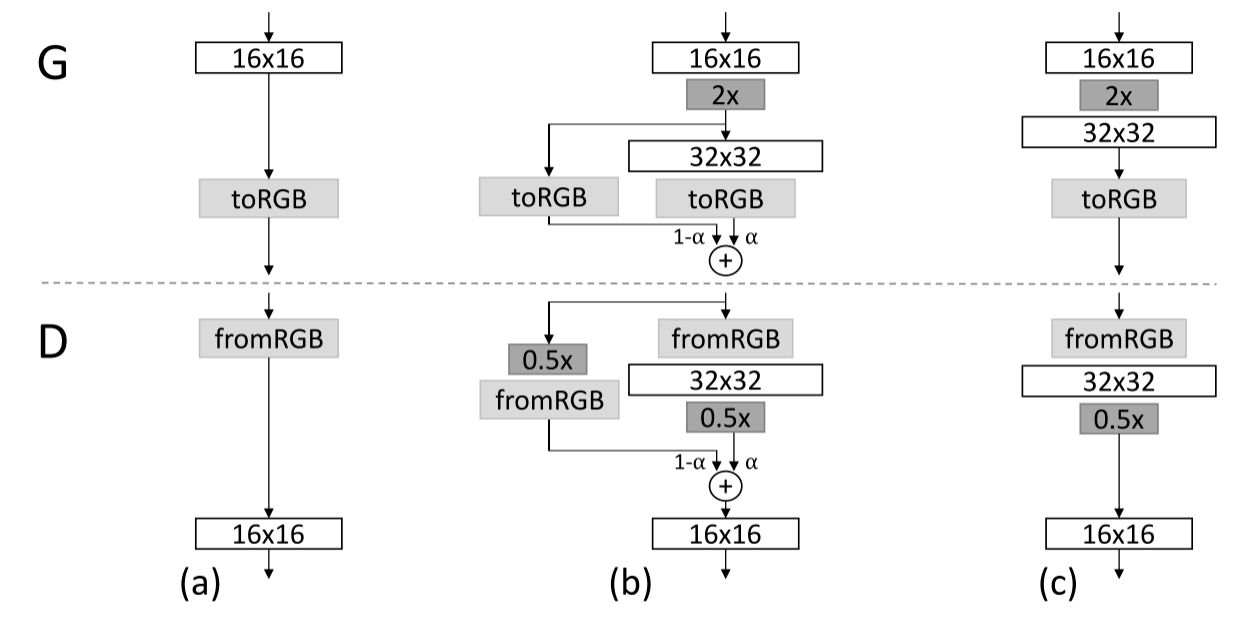
As shown in the paper, the convolution layers will allow both the generator and discriminator to become strong feature extractor. The convolution layer in the generator can achieve up-sampling from the low dimension input into the high dimension of desired output, while the convolution layer in the discriminator can achieve down-sampling to determine whether a desired output is true or fake. The discriminator can learn features for determining the output, just as supervised CNN on object classification, while the generator learns specific representations for major components. For example, the unsupervised models can learn to convincingly model object attributes like scale, rotation, and position.

Therefore, deep convolution layers will be included in the model for better training result. The detailed implementation will be shown in later part.

**2.2.3 PGAN (Tero, 2017)**

Last, Progressive Growing GAN approach will appear inside the model. The GAN training process will allow for the stable training of generator models that can output with high-quality (Tero, 2017). It involves starting with a very small image and incrementally adding blocks of layers that increase the output size of the generator model and the input size of the discriminator model until the desired output size is reached. During training, new blocks of convolutional layers are systematically added to both the generator model and the discriminator models.

The phasing in of a new block of layers involves using a skip connection to connect the new block to the input of the discriminator or output of the generator, adding it to the existing input or output layer with a weighting. The weighting controls the influence of the new block and is achieved using a parameter alpha that starts at zero or a very small number and linearly increases to 1.0 over training iterations.



(Tero, 2017)

The incremental addition of the layers allows the models to effectively learn. This incremental nature allows the training to first discover large-scale structure of the distribution and then shift attention to increasingly finer scale detail, instead of having to learn all scales simultaneously.

1. Proposal
   1. User Specifications

Our system is an open source GANs model using GitHub as the platform. The overall process will be run under Python environment. Our system has the following functions: input audio pre-processing, training the GANs model, monitoring the training process, generating sounds and evaluation.

**3.1.1 Input audio pre-processing**

Audio input are from NSynth dataset. The input audio needs to be pre-processed by STFT (converted into spectrogram) for further analyzing and training, instead of its raw waveform. Besides, the standard audio file (wav, mp3, aiff) is capable to be preprocessed with audio features automatically extracted or self-input. Also, the time consideration is required in order to speed up the training process. For short sound effects, we can just extract each slices of audio sample during specific time periods. For more shorter sounds, we may get zero padded to fill the slice.

**3.1.2 Model training**

Training the GANs model after data processing. You can check out the trained sample after a time period. If noises are created, a post-processing filter is provided. Increasing either the model size or filter length may improve results but will increase training time.

**3.1.3 Model monitoring**

We can monitor the training process via tensorboard. The latent vectors can be fixed at each checkpoint on the CPU. Also, since GAN training may occasionally collapse, back up checkpoints every specific time period (usually an hour) is provided to prevent data loss.

**3.1.4 Generating sounds**

Eventually we will convert the trained spectrogram-based data into waveform in order to render the audio file. The spectrogram of the processed audio data can be used as the reference (do comparison or features extracting) or further training.

**3.1.5 Evaluation**

Quantitative and qualitative evaluation can be performed. For speech and music, we provide audio examples online. For speech, we performed listening tests and evaluated the inception score (IS) (Salimans et al., 2016) and Frechet inception distance (FID) (Heusel et al., 2017), using the pre-trained classifier provided with (Donahue et al., 2019).

* 1. Design Specifications

(Specify your design principles and planning on this project to accomplish the functional specifications. Should contain an overall design of your system/algorithms, i.e. system architecture)

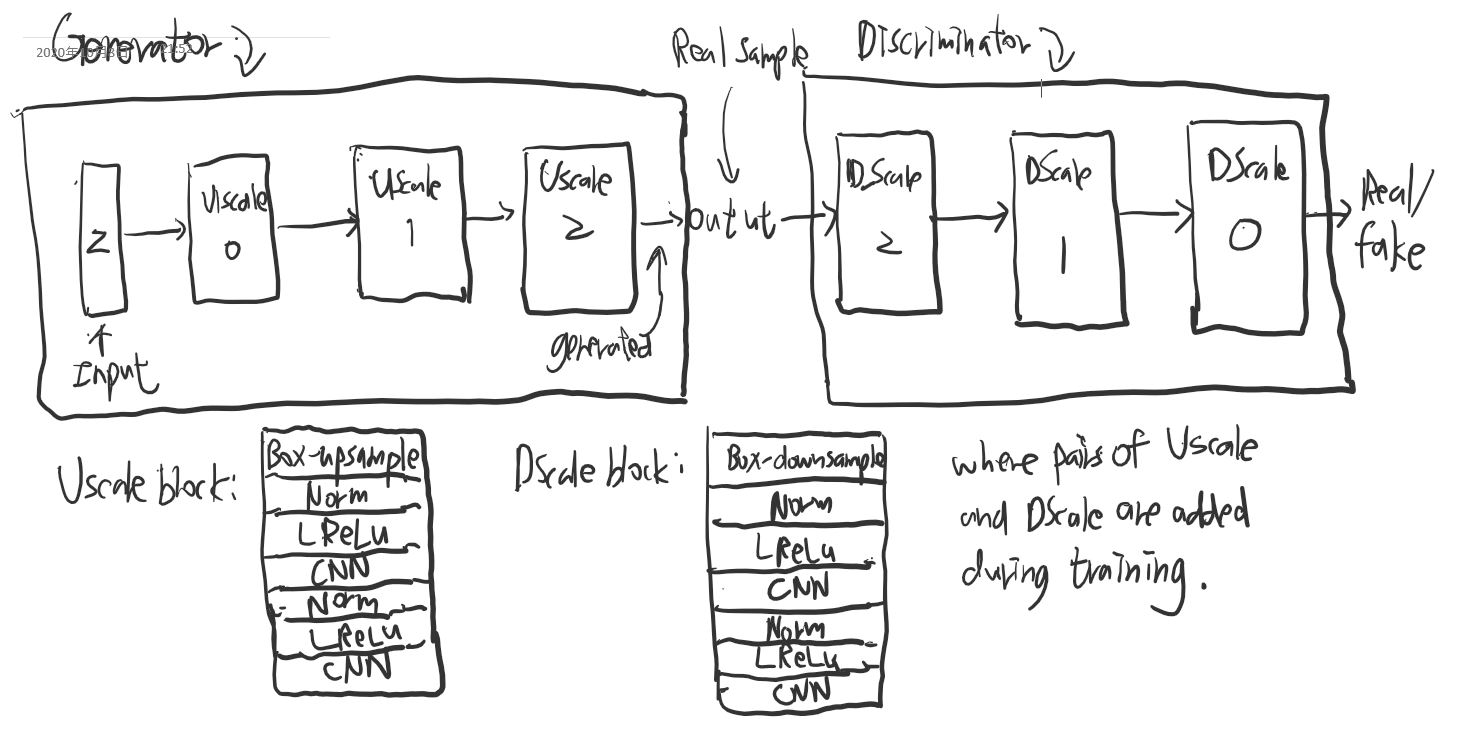
**Input**:

Audio sample from NSynth dataset. The dataset contains 300,000 musical notes from 1,000 different instruments aligned and recorded in isolation (Engel, 2019). NSynth is a difficult dataset composed of highly diverse timbres and pitches, but it is also highly structured with labels for pitch, velocity, instrument, and acoustic qualities. Each sample is four seconds long, and sampled at 16kHz, giving 64,000 dimensions. Under STFT transformation, audio data will transfer into the format like Spectrogram and further processing using the magnitude of frequency with time will become the sample data for training.

**Model**:

The model consists of the generator and the discriminator. All model weights for all layers shall be randomly initialized from a Normal distribution. The generator G samples a random vector z with 128 components from a spherical Gaussian. The random vector runs through a stack of transposed convolutions to up-sample and generate output data, and then the output data will go to the discriminator with downscaling convolution layers with only stripe convolution for down sampling to classify whether the data is real or fake. At each convolution layer, pixel normalization is performed (Sane, 2017). The error between predicted output and actual output is computed by Wasserstein objective function and gradient penalty which enforce the Lipschitz continuity. Those error will prorogate back to check the weights of the model. Moreover, training is divided into phases, wherein each phase a new layer, generating a higher-resolution output, is added to the existing stack in both generator and the discriminator. A blending parameter α progressively fades in the gradient derived from the new layers.

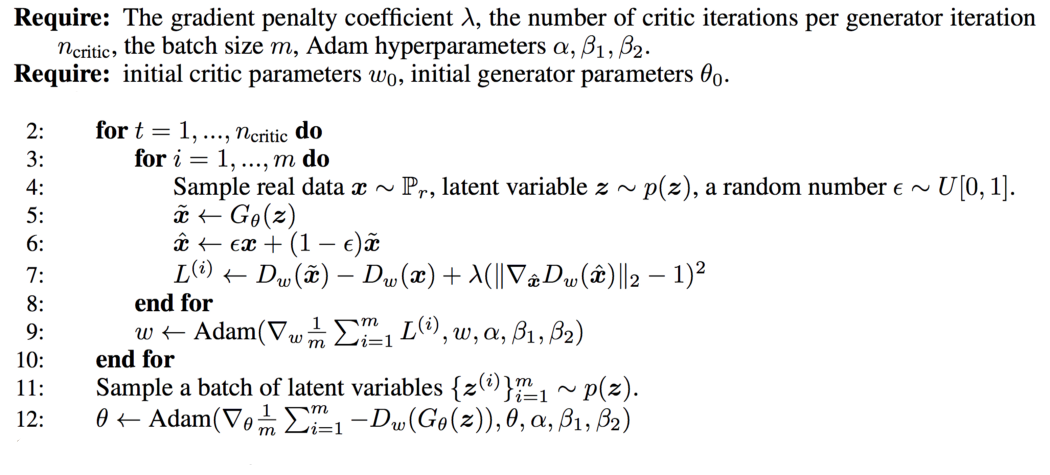
Overview of the system:



**Algorithm**:

For the model training, Adam is used to change the weight of the model. It is an adaptive learning rate optimization algorithm that’s been designed specifically for training deep neural networks. Also, the model architectural is updating because of increasing convolution layers brought by progressive training. During each phase, the model will train for the fade-in model where new model is injected and its output for combining both old and new models is controlled by the blending variable and the normal model.

Train\_epochs():



Train():

//g: generator, d: discriminator, gan: whole model, g\_models: all g models in phase change, d\_models: all d models in phase change, gan\_models: all gan models in phase change

g\_normal, d\_normal, gan\_normal = g\_models[0][0], d\_models[0][0], gan\_models[0][0]

//train normal or straight-through models

train\_epochs(g\_normal, d\_normal, gan\_normal, e\_norm, n\_batch)

//process each level of growth

for i in range(1, len(g\_models)):

//retrieve models for this level of growth

[g\_normal, g\_fadein] = g\_models[i]

[d\_normal, d\_fadein] = d\_models[i]

[gan\_normal, gan\_fadein] = gan\_models[i]

//train fade-in models for next level of growth

train\_epochs(g\_fadein, d\_fadein, gan\_fadein, e\_fadein, n\_batch)

//train normal or straight-through models

train\_epochs(g\_normal, d\_normal, gan\_normal, e\_norm, n\_batch)

Train will represent the whole training process and it completes the whole model.

**Evaluation**:

Inception classifier will be built by making use of online resource to complete. The evaluation will be carried out based on the implementation.

1. Milestones

4.1 Audio Preprocessing

Turning the audio data for desired form (Spectrogram) for training.

4.2 Building the GAN model for audio synthesis

4.1.1 DCGAN

4.1.2 WGAN-GP

4.1.3 PGAN

4.1.4 Combine those results

4.3 Training the GAN model until successful

4.4 Improve the GAN model

Hyperparameter tuning.

4.5 Testing the model with different audio input

Try different combinations audio representations as input of both generator and discriminator and find the better one. Can be evaluated be training time, sampling, and inversion times associated with each representation.

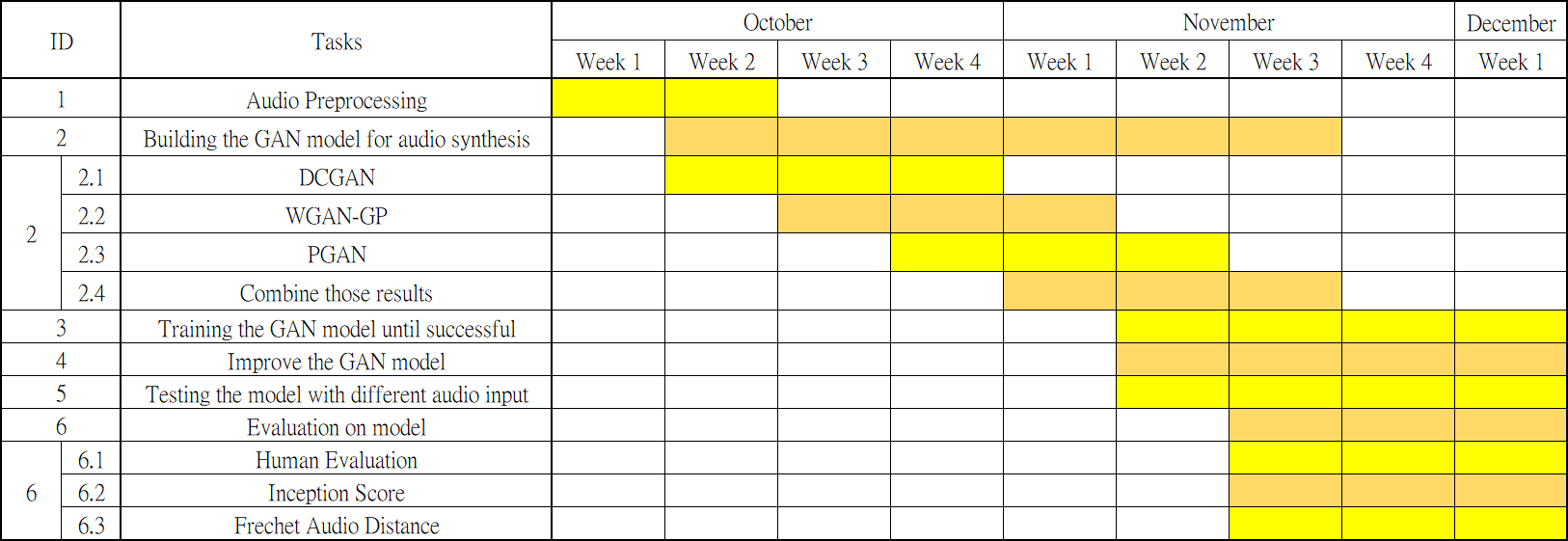
4.6 Evaluation on model

4.6.1 Human Evaluation

4.6.2 Inception Score

4.6.3 Frechet Audio Distance

1. Schedule



References

Donahue, Chris, McAuley, Julian, & Puckette, Miller. (2018). Adversarial Audio Synthesis.

Nistal, Javier, Lattner, Stefan, & Richard, Gaël. (2020). Comparing Representations for Audio Synthesis Using Generative Adversarial Networks.

Engel, Jesse, Agrawal, Kumar Krishna, Chen, Shuo, Gulrajani, Ishaan, Donahue, Chris, & Roberts, Adam. (2019). GANSynth: Adversarial Neural Audio Synthesis.

Marafioti, Andrés, Holighaus, Nicki, Perraudin, Nathanaël, & Majdak, Piotr. (2019). Adversarial Generation of Time-Frequency Features with application in audio synthesis.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. (2014). Generative adversarial networks.

Arjovsky, Martin, Chintala, Soumith, & Bottou, Léon. (2017). Wasserstein GAN.

Gulrajani, Ishaan, Ahmed, Faruk, Arjovsky, Martin, Dumoulin, Vincent, & Courville, Aaron. (2017). Improved Training of Wasserstein GANs.

Lucic, Mario, Kurach, Karol, Michalski, Marcin, Gelly, Sylvain, & Bousquet, Olivier. (2017). Are GANs Created Equal? A Large-Scale Study

Radford, Alec, Metz, Luke, & Chintala, Soumith. (2015). Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks..

Tero Karras, Timo Aila, Samuli Laine, Jaakko Lehtinen. (2017). Progressive Growing of GANs for Improved Quality, Stability, and Variation

Salimans, Tim, Goodfellow, Ian, Zaremba, Wojciech, Cheung, Vicki, Radford, Alec, & Chen, Xi. (2016). Improved Techniques for Training GANs.

K. Kilgour, M. Zuluaga, D. Roblek, and M. Sharifi. (2018). Frechet Audio ´Distance: A metric for evaluating music enhancement algorithms

Kumar, Kundan, Kumar, Rithesh, De Boissiere, Thibault, Gestin, Lucas, Teoh, Wei Zhen, Sotelo, Jose, . . . Courville, Aaron. (2019). MelGAN: Generative Adversarial Networks for Conditional Waveform Synthesis.

Sane, Parth, & Agrawal, Ravindra. (2017). Pixel Normalization from Numeric Data as Input to Neural Networks.

Kingma, Diederik P, & Ba, Jimmy. (2014). Adam: A Method for Stochastic Optimization.