EEE 511 Project

Spectrogram based Neural Audio Synthesis

Team 13
Sreeharsha Raveendra
Swapnil Hattarge
Neerav Naik
Kishan Indravadan Prajapati

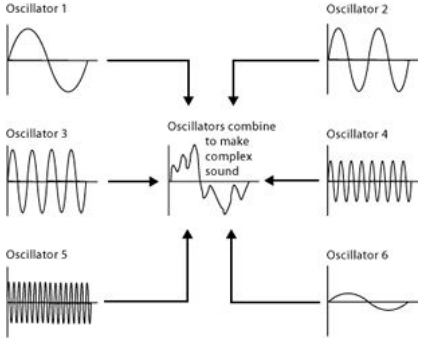
Abstract

1. Traditionally audio synthesis is done using electronic hardware (eg: MIDI board), software (eg: Xfer Serum) called synthesizers, which generates audio from hand-designed components like oscillators and wavetables.

Midi Board - AKAI

Xfer Serum

Hardware Oscillator Synthesizer

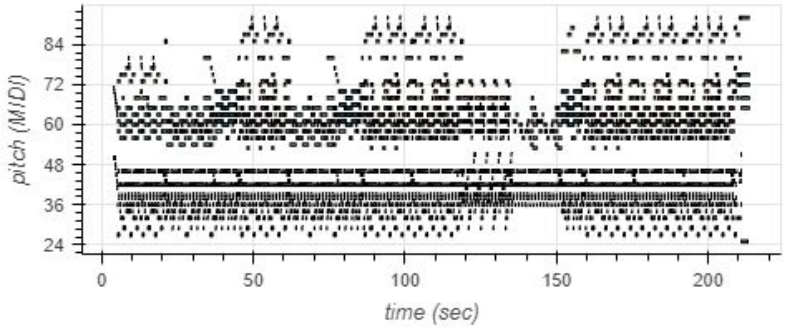


Simplified audio synthesis

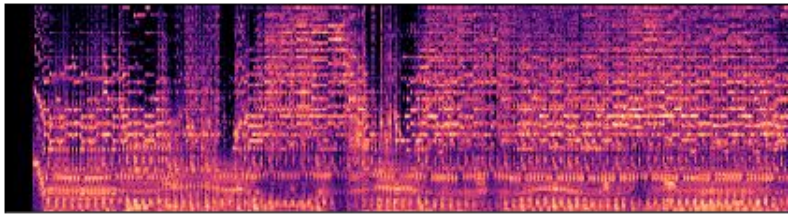
2. In this project, we show that using GANSynth, we can synthesize audio significantly faster than traditional synthesizers and interpolate instrument sounds into audio to generate an audio clip consisting of new sounds.
3. GANSynth uses the NSynth dataset which contains a large collection of annotated musical notes samples from individual instruments across a range of pitches and velocities.

Introduction

- Reference paper - GANSynth: Adversarial Neural Audio Synthesis [1]



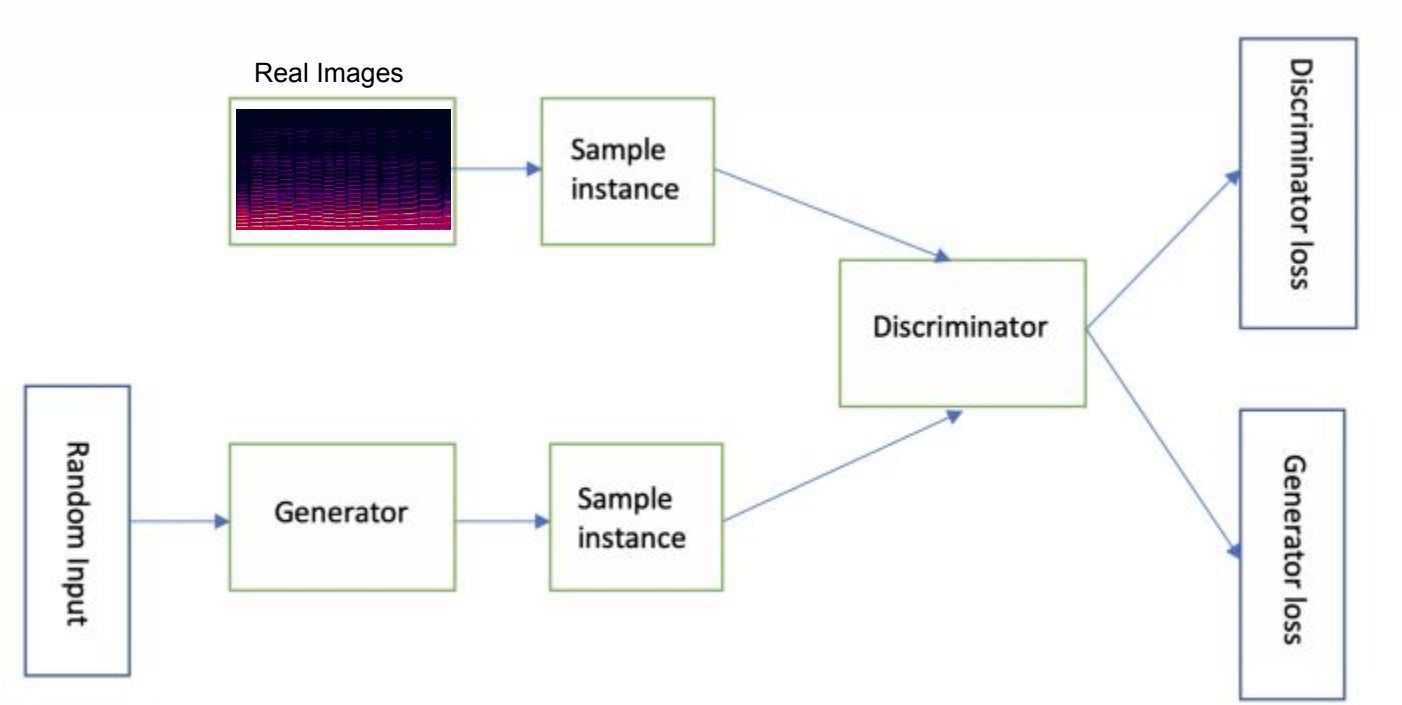
Input (pitch vs time)



Output Spectrogram
(frequency vs time)

- The model is trained on the NSynth audio dataset which contains 305,979 musical notes, each with a unique pitch, timbre, and envelope. There are 3 different instrument sources for the musical notes in the dataset - acoustic, electronic and synthetic . Also, there are 11 different instrument families of the musical notes - bass, brass, flute, guitar, keyboard, mallet, organ, reed, string, synth_lead and vocal.[4]

GAN Architecture

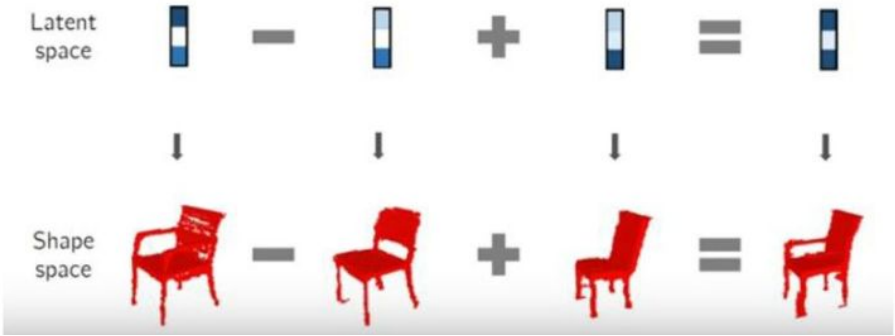


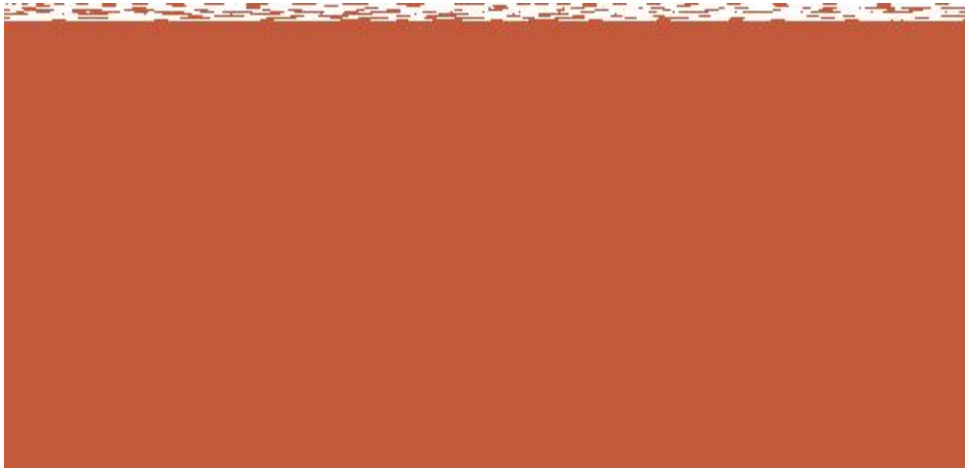
GAN Concepts

- Latent space
- Latent space interpolation
- Z-space

Why GAN?

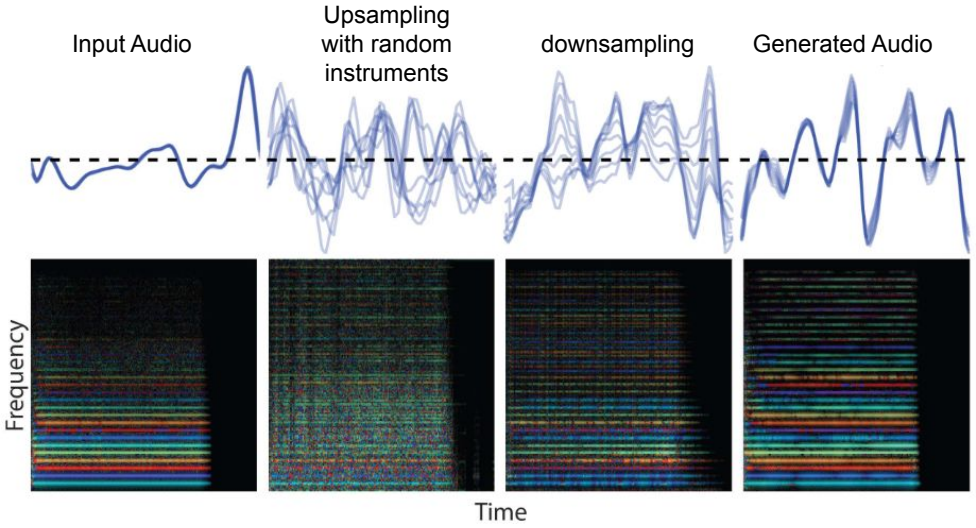
- High speed
- High fidelity
- Global structure





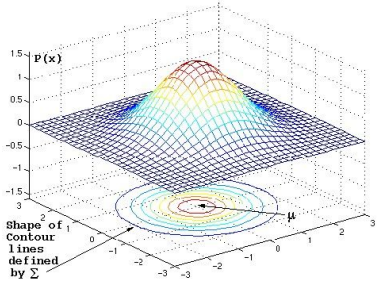
Methods - Overview

- The generator samples a random vector from a *spherical gaussian*, that represents instruments from the dataset
- The vector is run through several layers of convolution and upsampling to generate an output spectrogram [1]
- The discriminator structure is a mirror image of the generator where it uses several layers of convolution and downsampling layers [1]
- The discriminator estimates the divergence measure between real and generated distributions [1]
- With pitch provided as a conditional attribute, the generator learns to use its latent space to represent different instrument timbres. This allows us to synthesize performances from MIDI files, either keeping the timbre constant, or interpolating between instruments over time. [6]



Pitch - represents the frequency of sound on a musical scale

Timbre - represents the quality of sound/tone, denoted by shape of wave (smooth, jagged etc.). Distinguishes different types of sounds.



← *spherical gaussian*

Methods - Model Architecture

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

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	-

Simulation

Setup: google colab to implement the algorithm and python for the code

Source:

a. **magenta** - development platform for utilizing machine learning in art and music [8]
b. **gansynth_generate.py** - model that generates audio clips based on training checkpoints and user input midi files [9]

Pretrained checkpoints:

1. **acoustic_only** - gansynth model parameters trained on only the acoustic instruments of the NSynth dataset [9]
2. **all_instruments** - gansynth model parameters trained on all instruments of the NSynth dataset [9]

Developed custom code on google colab that utilizes libraries, programs as defined in **source** and provides user with the ability to [7]:

a. Provide custom midi files
b. Generate audio from pretrained checkpoints
c. Customize time intervals between instrument changes
d. Customize choice of instruments in generating synthesized audio

Follow the steps given in notebook which are based on implementing model from open source checkpoints the authors from original paper have shared [7]:

Step 1: install necessary libraries using pip (tensorflow-gan, magenta, etc)

Step 2: clone magenta repository from github (https://github.com/tensorflow/magenta.git)

Step 3: create directories for storing files to the model and download the pretrained checkpoints

Step 4: run defined helper functions and import midi files from (http://www.midiworld.com/files/, https://bitmidi.com/)

Step 5: plot midi file note sequence, observe the original sound clip

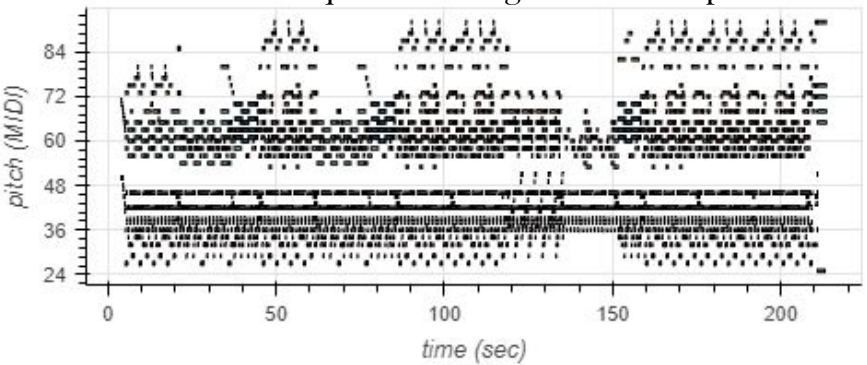
Step 6: generate synthesized audio clips using model, download generated clips and compare with original clip

Step 7: generate synthesized audio with custom instrument interval, custom instrument selection and download generated clips

Step 8: compare produced spectrograms and compare the audio of generated clips

Results

Note Sequence of original audio clip



Clip 1 - Never Gonna Give You Up

Original audio clip:
https://drive.google.com/file/d/1nR1LWhVtULVLNQovdeU6XdP9J8cX-p64/view?usp=share_link

Generated clip (model trained on only acoustic instruments):
https://drive.google.com/file/d/18i99Kue5EBLC4EbKq36YPY-1zVqN9j9N/view?usp=share_link

Generated clip (model trained on all instruments):
https://drive.google.com/file/d/1YASdpBtRdPCjuldY4axNR8uztHd9s4xY/view?usp=share_link

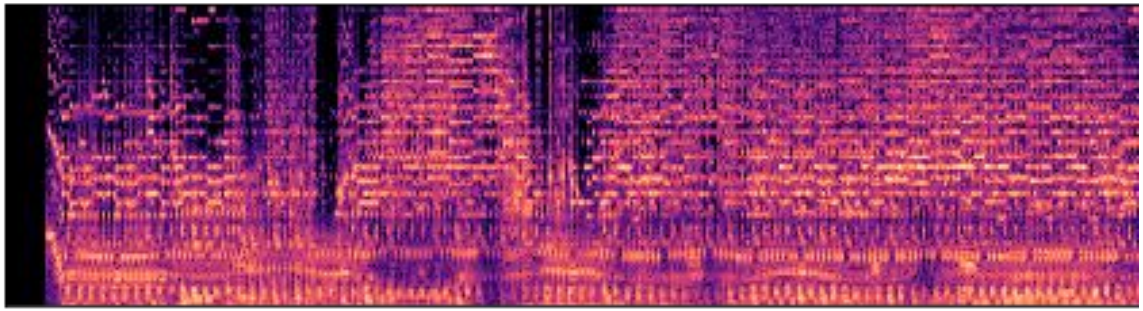
Generated clip with 4 sec interval b/w random instruments:
https://drive.google.com/file/d/17Z8OvuxgCcuyNheuA1ae8VtElE8Cz5XE/view?usp=share_link

Generated clip with custom instruments:
https://drive.google.com/file/d/1m7P43GhmDH72a9uUePiZfbMOUx7zsRuf/view?usp=share_link

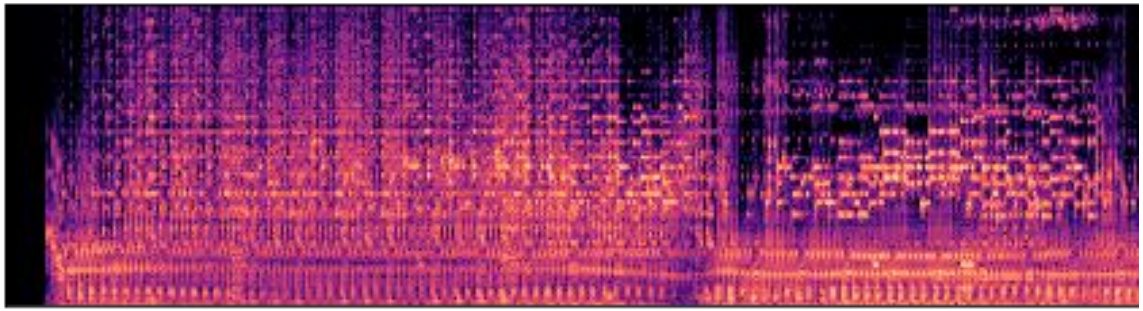
Results

Visualization of notes in spectrogram gives evidence in showing the difference between frequencies at identical time intervals

CQT Spectrogram of generated clip with 4 sec interval b/w random instruments



CQT Spectrogram of generated clip with custom instruments



Link to folder: https://drive.google.com/drive/folders/1gCZGv7I7XkpRDUYzD6mG3KNr8fbZZCgM?usp=share_link

Results (further samples)

Clip 2: Sweet Child O Mine
(Folder link: https://drive.google.com/drive/folders/1kIbqa2P6Vs9bBom2Aw9TSOOOgqXuo-zZ?usp=share_link)
Audio clips:
Input: https://drive.google.com/file/d/1fmtDoi8wpejXwQWzaxEVFu_VOv2j99DJ/view?usp=share_link
Output (acoustic): https://drive.google.com/file/d/1D13Taa2E58WxEoqoDZVSrr30rAb_LxuS/view?usp=share_link
Output (all instruments): https://drive.google.com/file/d/1KoCJryyAwHXIS6iTbAuAg812Bah17USx/view?usp=share_link

Clip 3: Billie Jean
(Folder link: https://drive.google.com/drive/folders/1sp8j1ByoFvLvEidg7xyGqrof5974GZo2?usp=share_link)
Audio clips:
Input: https://drive.google.com/file/d/1Ps9JBcF1ticLUx1BUQMPbvMLke2CUQNe/view?usp=share_link
Output (custom instruments): https://drive.google.com/file/d/1Sp-6Y2Iu6zMsH7aoXOztKHjkgMKeEPMs/view?usp=share_link

Clip 4: Bach Prelude (Default)
Input: https://drive.google.com/file/d/1fiHcVmpu7BkptvBjbruevklyfUwV16jg/view?usp=share_link
Output (acoustic only): https://drive.google.com/file/d/18_v4M7wpJqhHiNrzEb1nbYixy54A8J3a/view?usp=sharing
Output (all instruments): https://drive.google.com/file/d/1gYdAo-2rQJGc9OjCC6MA_jAEqadwOb8b/view?usp=sharing

Conclusion

Implementation hurdles:

1. Storage, memory issues with training model on entire NSynth Dataset (96gb - 100gb)
2. Outdated functions present in the code, i.e. solved by uninstalling and reinstalling new libraries and changing syntax of functions
3. Limited knowledge of tensorflow i.e., ran into computational errors with tensorflow, dealing with tensorflow datasets
4. Outdated libraries on producing rainbowgrams (rainbow spectrograms), i.e. demonstrated in the paper to show differences in phase and magnitude of the audio clips

Inferences:

1. Able to produce synthesized audio clips in order of few seconds to minutes, significantly faster than traditional synthesizers
2. Able to customize the interval between instrument changes and selection of instruments for the synthesized audio
3. Synthesized audio is coherent and in tune with original audio clip

Future work:

1. Ability to further customize audio synthesis with algorithms producing instruments based on music genre of user interest
2. Implementation of GAN model into hardware interfaces, example: NSynth Super (https://nsynthsuper.withgoogle.com/)

References

1. Papers
   a. GANSynth: Adversarial Neural Audio Synthesis (https://arxiv.org/abs/1902.08710)[1]
   b. Progressive Growing of GANs for Improved Quality, Stability, and Variation (https://arxiv.org/abs/1710.10196) [2]
   c. Neural Audio Synthesis of musical notes with WaveNet Autoencoders (https://arxiv.org/abs/1704.01279)[3]
2. Dataset
   a. The NSynth Dataset (https://magenta.tensorflow.org/datasets/nsynth)[4]
3. Online information sources
   a. https://magenta.tensorflow.org/nsynth
   b. https://magenta.tensorflow.org/nsynth-instrument
   c. https://magenta.tensorflow.org/gansynth [5]
   d. https://storage.googleapis.com/magentadata/papers/gansynth/index.html [6]
4. System environment to run implementation
   a. Google Colab: https://colab.research.google.com/drive/1a0YAthotCS7eOget9-R9ZrfqkdFN4AYK?usp=sharing [7]
5. Code repositories
   a. https://github.com/magenta/magenta [8]
   b. https://github.com/magenta/magenta/tree/main/magenta/models/gansynth [9]