The sound of AI

Introduction

Music producers and sound engineers have always enjoyed analog music production gear for the uniqueness of the “grain” it colors the sound with, whatever it means. On the contrary, digital audio technology is often regarded as sounding “cold” and kind of boring. Is there a way to put life back into digital audio using famous AI model architecture or smart signal processing?

Goals

The project is a mixture of research and development. It can be divided into two steps.

Part 1

This part studies various audio representations and their interaction with some AI model architectures (image generation models), hoping to find representations that add interesting “character” to an audio input.

First, ignore the neural network part. Spend some time reading about digital representation of audio and images; it will give you ideas on how they can interact.

Part 2

When you have interesting image representations, you can try to train a neural network with them.

Eventually, it would be nice to experience some of this project's results in a simple audio plugin that music producers could use.

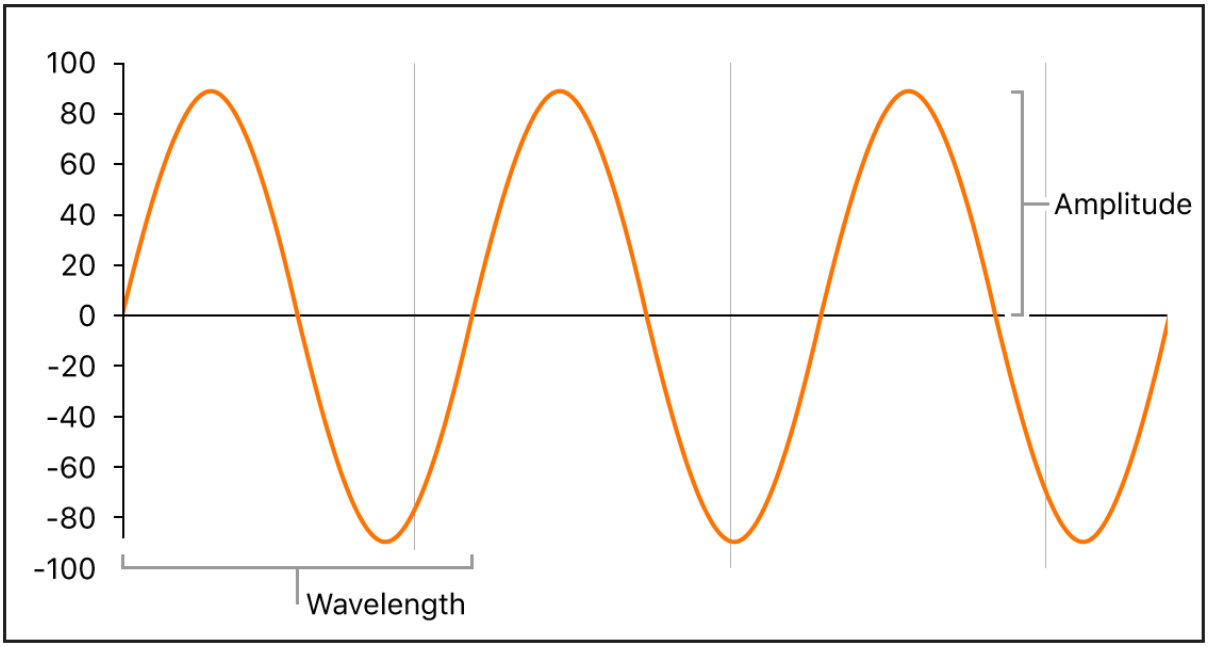
The project can evolve depending on your input as you gain more knowledge on this topic and depending on your own unique set of skills.

Useful notions for part 1

(1) Analog vs. digital

Sound and music are recorded on a medium (plural: media).
The medium may consist of tape, vinyl records, CDs, hard drives...
There are two main classes of media: analog and digital.

In analog recording, the information on the medium resembles the soundwave. Both the soundwave and the information stored on the medium look like this:



In digital recording, the information on the medium doesn't resemble the soundwave: it looks like 0101110100101010000001001010.

For more details, you may refer to the chapters "Analog Representations of Sound" and "Digital Representations of Sound" in the file "SynthesisChapt1.pdf", pp.21-22. There are many pages on the internet that explain analog and digital audio from a variety of perspectives without mathematical formalism[1].

(2) "Analogousness" and errors

A key aspect of analog versus digital is how errors in the medium translate into something that we can hear. Suppose you change only slightly the information stored on an analog medium. For instance, suppose you multiply the first half-period of the sine wave by 1.1. Then, upon playback, you will hear only a slight difference. **The restitution of the error is analogous to the error itself.**

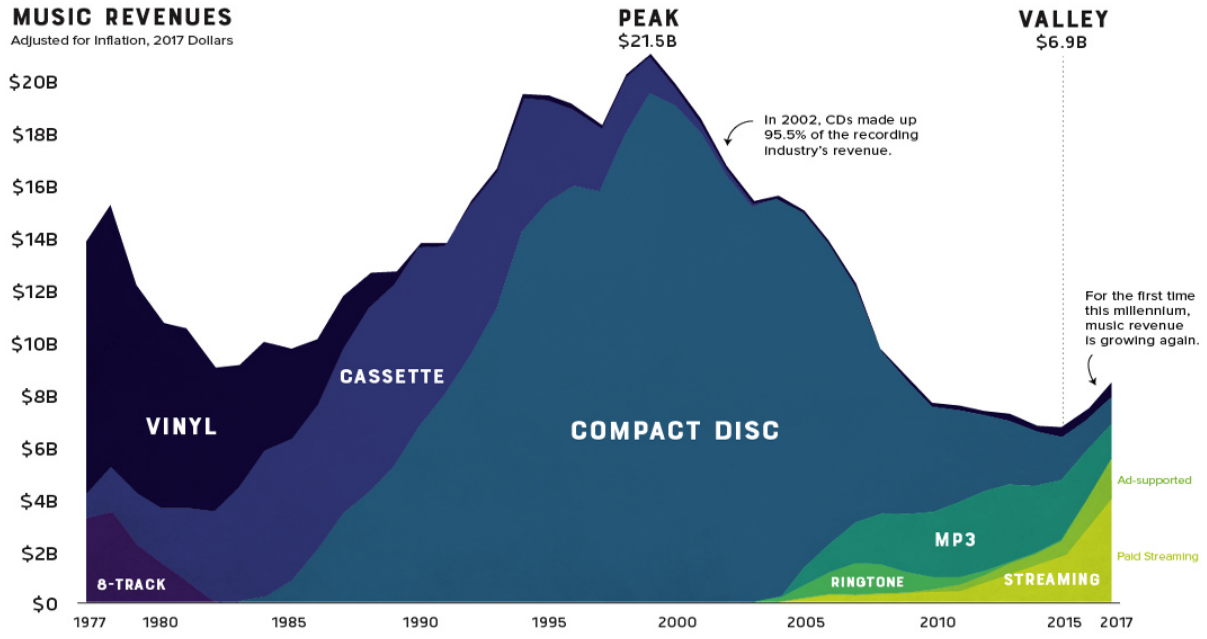
[1] https://craiganderton.org/basics-of-analog-audio/
https://recordingconnection.com/reference-library/analog-or-digital-what-is-the-difference
https://www.avaccess.com/blogs/guides/digital-audio-vs-analog-audio/

But in the digital domain, if you change 01011101001010100000100101O to 11011101001010100000100101O (only the first digit is changed), it is possible that you will a loud click. The error on playback is not proportionate to the error on the medium. **The restitution of the error is not analogous to the error itself.**

(3) Usage of digital and analog technology in music production

Historically, engineers have tried to devise media that reproduce audio as well as possible. Analog audio is prone to errors. It always makes slight errors. The values that are stored on the analog medium belong to a continuous space: you may want to record the instantaneous value "0.5," but it will record and playback "0.51". You may want to play back the audio at the right speed, but for some technical reason, the playback is slightly faster. The machine's temperature may be too high, prompting the electronic components to deviate from their usual behavior. Analog audio makes a lot of mistakes: it has noise, distortion, flutter…

On the other hand, digital audio, when properly engineered, doesn't make errors. You store 01011011000, and it plays back 01011011000. For this reason, digital audio was a revolution. When the CD was introduced, it was very successful, and it replaced almost all other media for a while.



But near the '90s, music producers started to realize that although digital audio was practical, it was not always interesting from a creative point of view. Music producers want to make interesting music, they want an interesting sound. From that prespective, the unpredictiveness of analog audio can be more appealing, and the predictiveness of digital audio may be boring.

Therefore, producers started to revert to using analog audio. A good example of music producers making heavy use of analog audio in the studio when digital audio was all the rage is Massive Attack's "Mezzanine" (1998). Tracks like "Man Next Door" are very heavy on analog audio production.

To this day, producers still buy expensive analog equipment.

This stereo analog compressor costs £4000.

This 1983 analog synth sells for £8000.

The unpredictiveness and imperfection of analog audio has also been transferred to the digital domain:

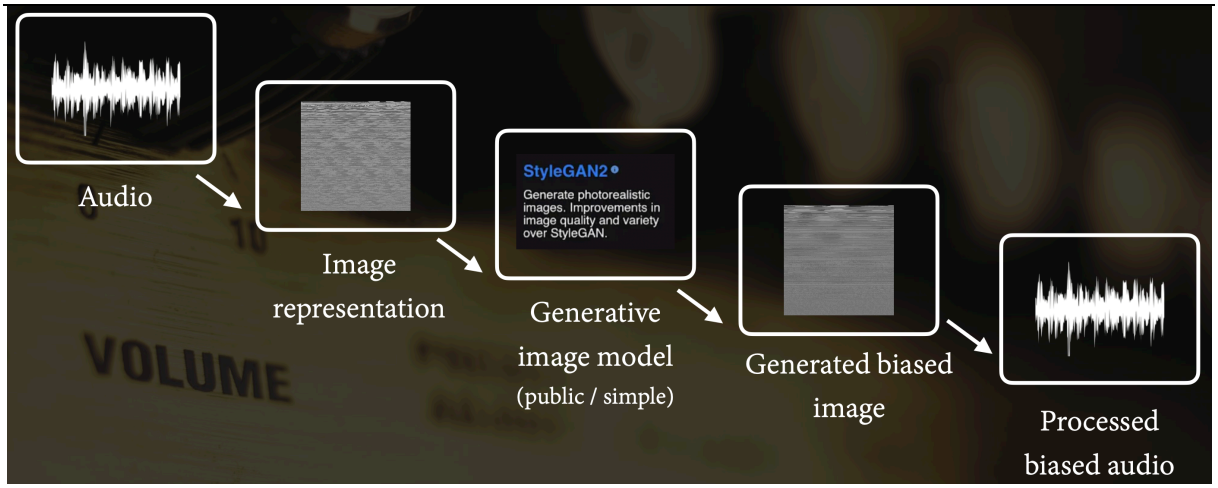

Tape emulation plug-in.


Vinyl emulation plug-in.

Producers use such analog emulation plug-ins because they like the "grain" of the analog process. The word "grain" in this case comes from photography. In the picture below, the image on the right is grainy.

A ‘grainy’ sound is less clean and more organic. It is interesting in many music genres.

(4) **The final goal of the course** is to create “grain” in relation to one particular technique of A.I. audio generation. In the image below, the “processed biased audio” is equivalent to the “grainy” image above.



StyleGAN is an AI model. One classic application of StyleGAN is human face generation. You give the model 10000 real human faces, and it will learn how to generate faces that don’t exist, like https://generated.photos/faces.

It follows that, in theory, if you give the model 10000 images that represent a sound, it will learn to generate new images that represent a sound. Therefore, it will learn to generate new sounds.

(5) This is where the notion of "analogue-ness" is again relevant.

Suppose that you give the model 10000 similar images that all represent variations of a single sound: the model will learn how to generate this sort of image – and, as a result, this sort of sound. But an AI model is never perfect: it will always make small errors.

Read paragraph (2) again. From (2), it follows that if the image representation is analogous to the sound, then the small errors made by the model will translate into small audible errors. But it the image representation if not analogous to the sound, then the small errors made by the model will translate into important audible errors.

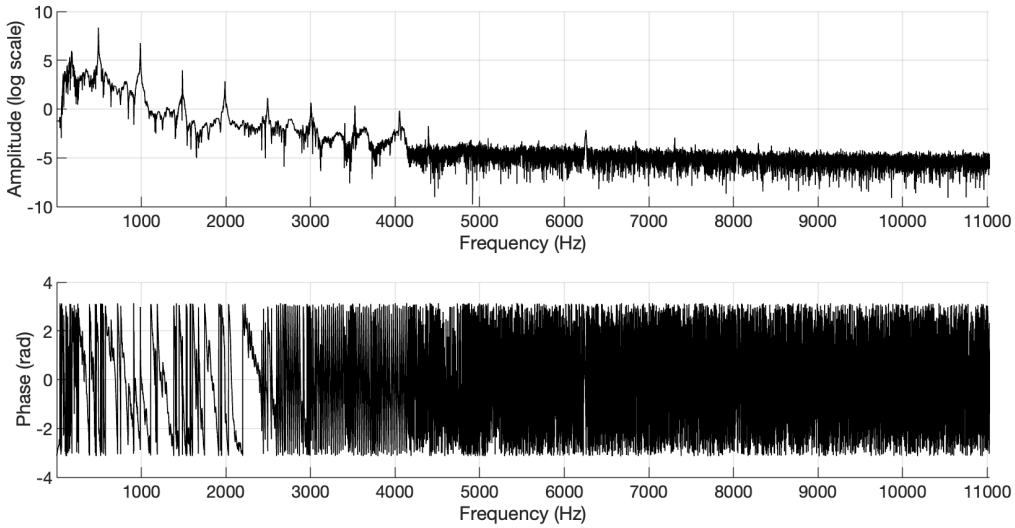
The key question to the process shown in (4) is therefore: to what extent is the image representation of the sound analogue to the original sound?

The more analogue the image representation, the smaller the output error. By selecting different image representations, we may adjust the "grain" 's intensity.

(6) What representations for the sound?

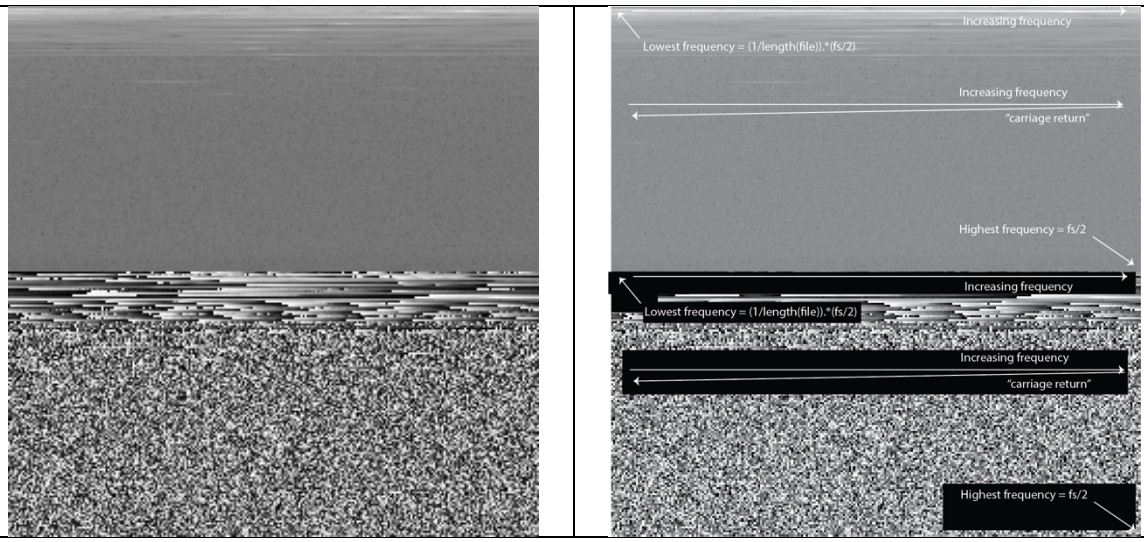
The images we use may be based on the Fourier Transform, the polar coordinates of the Fourier transform, and the Discrete Cosine Transform.

Let's consider the polar coordinates of the Fourier transform. The polar coordinates are: amplitude and angle (phase). Here are the polar coordinates for a piano note.



Amplitude is analogous. If you change one Y coordinate, the reconstructed sound will be similar. Phase is not analogous. If you change one Y coordinate, the reconstructed sound may be entirely different.

The representations above can't be learned by StyleGAN or similar networks. StyleGAN accepts square images. Therefore, let's turn these representations into square images (256*256 pixels, grayscale).



Top: amplitude. Bottom: angle/phase.

(6) To what extent is the 'square image' representation analogous?

Both the top and bottom parts of the square image inherit the properties of the polar coordinates. The top part is more analogous than the bottom part.

Folding the frequency values to make the representation square makes both parts less analogous:

- Consider one point in one line. It corresponds to a frequency. Consider the point just at the right of it. It corresponds to a very close frequency. This is analogous: close points in the representation correspond to close points in the physical reality.
- Now consider the same point in one line. It corresponds to a frequency. Consider the point just at the bottom of it. It corresponds to a far frequency. This is not analogous: close points in the representation don't correspond to close points in the physical reality.

As a result, if we use this representation to train a neural network, the small errors made by the neural network will result in much bigger errors in the audible result. One way to control the errors made by the neural network is to control to what extent the representation is analogous.

Course tasks for part 1

The first set of tasks in the course deals with the representations without the neural network.

As stated at the beginning of the document, this is a practical course. You should focus on being able to understand, explain, and implement concepts rather than executing instructions. Therefore, the following are suggestions. You may follow a different path if it helps you understand the concepts at hand.

- Start with the audio file I previously provided: "GI_GMF_B3_353_20140520_n.wav".
- In Matlab or Python, evaluate and represent its Fourier Transform. Do the reverse transform, save the audio, and check that it's the same as the input audio.
- Evaluate and represent the polar coordinates. Do the reverse transform, save the audio, and check that it's the same as the input audio.
- Change one value in the amplitude of the polar coordinates slightly. Perform the reverse transform. Compare the output with the original files.
- Change one value in the angle/phase of the polar coordinates slightly. Perform the reverse transform. Compare the output with the original files.
- Convert the polar coordinates of the Fourier transform to square images. Save the image, reload it, do the inverse transform, and check that it's the same as the input audio.
- Implement changes to the square images, do the inverse transform, and listen to the difference in the output audio.

Generally speaking, if you can showcase different artifacts in the output audio, link them with the transformations you performed on the images, and explain what transformation prompted these artifacts, then you're off to a good start.

Part 2

Resources

- Dataset:
  - C3 dataset (around 1,000 sounds of piano of the C3 note - start with this)
  - full dataset of piano sounds (88 classes, one per key of an acoustic piano - use it if you need to scale up)

- Image generation AI models:
  - original github repository used for the Ki project: stylegan2-ada-pytorch
  - custom fork of a previous version of styleGAN to deal with non-square images
  - newer model: stylegan3

- Google Colab - use it if you need to train a neural network but don't have access to GPUs

- Digital Audio fundamentals

- Digital Image characteristics :
  - video 1
  - video 2

Possible directions for part 2

- Try as many image representations of sounds as you can. Find some that are invertible, find some that are destructive. Don't stick to black and white, adding RGB to the input data may be a good idea.

- Retrain a model using the same representations but using a newer model, like stylegan3.

- Try to adapt the StyleGAN architecture to support non-square input training data. For example, is it possible to train a StyleGAN model with 2x131072 pixels images instead of 512x512? This way, we would be able to check if a more "analog" representation of the sound can lead to better results. I'm not sure this is possible.

- The output of this project, developing a music plugin, could have various levels of complexity depending on how you move forward in this project.

Part 2 - Ki: an example of what we try to achieve

Ki is a neural piano sound synthesizer. It can generate an infinite number of strange, unique sounds that cover various keyboard types of tones. It resulted from an experiment by Amaury Delort, Emmanuel Deruty, and Cyran Aouameur at Sony CSL Paris.

What is Ki

Video demo

Try it (Please don't share it outside of the scope of this project)

You will need to install it at the system level for it to be used in a DAW. I suggest using REAPER as it's free (you just have to close a popup every time you launch the program), highly customizable, and available on all platforms. Ki is a virtual instrument plugin, so you will need an external keyboard or the keyboard of your computer to generate sound with it. Google, YouTube and ChatGPT are your friends if you struggle at some moments in the process.

How we did it

Use state-of-the-art image generation model (stylegan2-ada-pytorch)

Find a good image representation

After testing different representations (FFT, DCT, ArgAngle), we decided to go with DCT (Discrete cosine transform)

Create a good dataset & train a custom model

For this project, we created a dataset made of recordings of piano notes. They are categorized by pitch, and there are about 1,000 samples for each category. You can find this dataset in "Ressources" above.

Initially, we ran our experiments on mono sounds converted to 256x256 pixels black and white images. It led to the generation of mono samples of duration 1,4860771 seconds at 44,1 kHz. Then we updated our input data to 512x512 images, enabling us to generate stereo samples of duration 2,9721542 seconds. This is the workflow we kept to train the final version of Ki.

Since we used black and white images, we had to modify the original "generate.py" file of the repo :

custom generate.py stylegan2-ada-pytorch

You can find below the functions used in our code to:

- preprocess the input data (from audio to image) to feed the AI model
- once the model is trained, convert back generated images to audio

Ki - python functions from audio to image, from image to audio

Course task for part 2: do the same with the representations you found in part 1.

Final task for part 2, if there is time: create a plugin with the trained model

Once the model is trained, you can "encapsulate" the trained network and emulate its behavior. It's called "model tracing". Then, you can export it as some kind of function you can use in other projects. Then, with the help of frameworks like JUCE you can build an audio plugin from it.