Extending Concatenative Synthesis with Neural Network-based Sequence Modelling

Music Computing Final Project

Wojciech Kacper Werkowicz
ref. 33655149
Supervisor: Dave Griffiths

May 30, 2023
Goldsmiths University of London
UK

Contents

1 Overview

This project aims to extend the concatenative synthesis process by incorporating neural network-based sequence generation. It focuses on analyzing audio data on a micro scale to model its underlying structures and utilizes this information to interactively resynthesize the input in real-time with novel variations.

This project is motivated by a keen interest in neural audio synthesis and a recognition of the limitations present in existing approaches. Popular projects in this field often require significant computational resources and expertise, making them inaccessible to many individuals. This project aims to address these barriers by developing a user-friendly and low-power solution that allows users with varying levels of experience to engage with the possibilities of using neural networks in the audio domain. Simultaneously inspired by existing computer music tools such as CataRT,[1] the goal is to bridge the gap between the historically established techniques and more recent advances in Machine in a manner that empowers users to experiment with sonic exploration.

The designed workflow is designed to be accessible to users with varying levels of programming expertise, providing an easy entry point, while offering flexibility for creative exploration and customization. By leveraging music information retrieval techniques and analysis, the pipeline extracts relevant audio features. Then, ML unsupervised classification methods, such as K-means clustering, are employed to abstract the inherent structural characteristics of the audio data. In the subsequent stage, a Keras GRU model is utilized for time series modeling and, later. After fitting, the model can be used to generate new sequences, which are passed to the audio software of choice using the Open Sound Control (OSC) protocol. Enabling smooth communication and control of the synthesis process, this universally accommodates users' preferences and facilitates integration with their preferred synthesis methods.

To minimize power requirements, the training process can be executed efficiently on a cloud runtime, while the output is resynthesized directly from the original audio file. This ensures efficient and low-power processing, making the project accessible even on modest computer systems.

2 Background research

In the previous section, the project was introduced as an exploration of the intersection between granular synthesis and machine learning. In order to provide historical and technological context and enhance coherence, this section examines relevant resources that have influenced and shaped the conceptualization of the project.

[1]Schwarz, n.d.

2.1 Particle Synthesis

Elaborating on the research background, it is worth first to explain each of the digital audio techniques which will be referred to throughout this report.

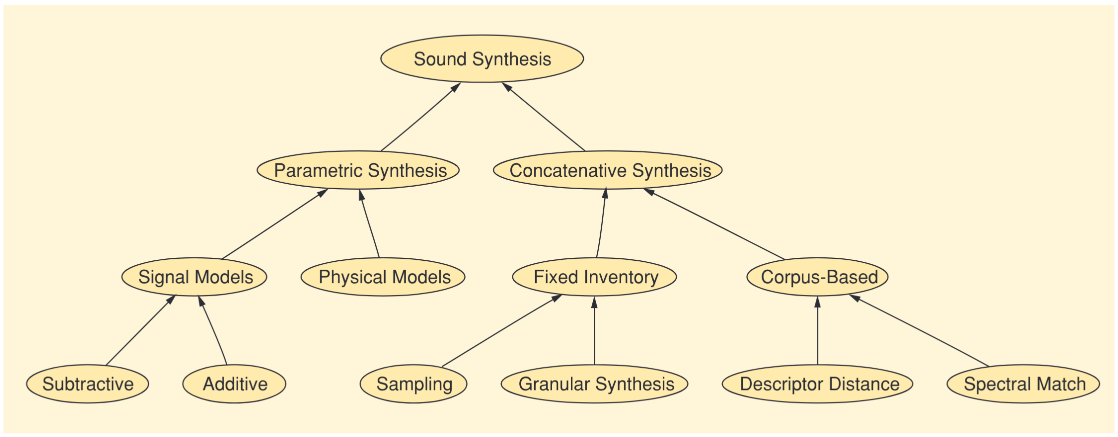


Figure 1: A diagram of synthesis methods by Diemo Schwarz[2]

2.1.1 Granular Synthesis and Granulation

Granular synthesis is a technique of constructing sound objects out of micro-acoustic events referred to as *grains*. Grains consist of a waveform, which can be synthesized, or sampled, shaped by an amplitude envelope, often called the *window function*. Splitting an existing audio sample into separate grains is a process called *granulation*. Doing so, allows for scanning through the temporal axis of the sound, which bears some resemblance to the often used *sliding window* method used in data analysis. *Microsound* by Curtis Roads is a staple resource on the topic of granular synthesis and other related methods generally described as *particle synthesis*.[3] It remains to be one of most comprehensive resources on the topic, even contemporarily, as various new tools and approaches to composing sound on micro scale are explored. Real-time granular synthesis has been first implemented by Barry Truax. On his website, he puts forward the basic ideas behind the latter alongside his personal impressions:

> The technique I have found the most striking in the way it facilitates moving inside a sound is real-time granulation of sampled sound. Briefly, the technique divides the sound into short enveloped grains of 50 ms duration or less, and reproduces them in high densities ranging from several hundred to several thousand grains per second.[4]

[3]Roads, 2004.

[4]Truax, n.d.

2.1.2 Timestretching

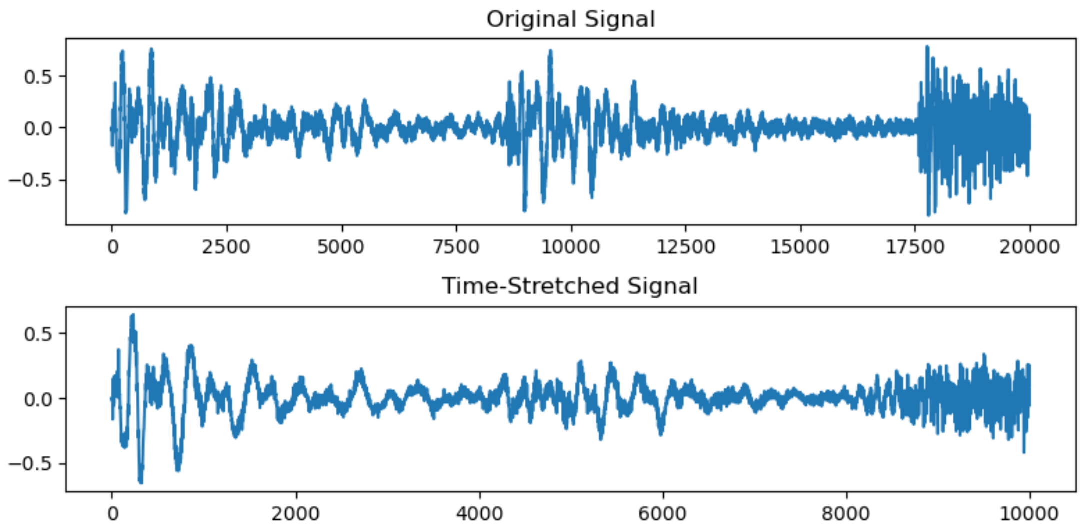


Figure 2: First second of Amen Break sped up by a factor of 2x using timestretching.

A technique closely related to granular synthesis is time stretching. It can be said to build upon the concept of *moving inside of a sound* mentioned by Truax[5]. Time stretching allows for the flexible exploration of the temporal dimension, offering the freedom to manipulate the rate at which grains move through the original material. By keeping the samples within the grain identical to those found in the original, time stretching preserves the pitch while enabling the combination of arbitrary sections of the material.[6] Notably, time stretching algorithms found their way into digital audio samplers such as the Akai S2000, significantly influencing the emergent sounds of 1990s like jungle[7].

[5]Parallels could be drawn to the approach of minimalist composer Tony Conrad:

> Our ”Dream Music” was an effort to freeze the sound in action, to listen around inside the innermost architecture of the sound itself. (...) We were announcing that the composer could sit within the sound, so to speak, and work with it as a plastic continuum extended in time(...). (Duguid, 1996)

[6]Truax, n.d.

[7]For a great text on jungle and time stretching, see the conversation between Robin Mackay and Christopher Haworth (Mackay and Haworth, n.d.)

2.1.3 Concatenative Synthesis

Another technique related to granular synthesis in its approach is concatenative synthesis. It works based on a similar principle of building objects out of micro-sounds. The main difference is that, in the case of concatenative synthesis, feature extraction is applied to each grain within the sound corpus. This way, a new representation of the audio is created which allows choosing grains informed by their sonic quality rather than their temporal arrangement in the source material.[8] The specific methods of analysis and resynthesis can vary across implementations, but the general principle remains the same.

Most notable ones are the Adaptive Concatenative Sound Synthesis (ACSS)[9] and the recognized CataRT interactive system for real-time synthesis. CataRT allows users to navigate and manipulate large sound databases using an extent of Music Information Retrieval and Machine Learning techniques. One of the strengths of CataRT is its abstract representation of sound and its intuitive interface, which provides high-level control. Within the above, the temporal context is completely left to the user and their skills in exploring the representational space, though there have been inquiries into introducing more elaborate algorithms for picking the next most suitable grain to play using distance functions.[10]

2.2 Data Science

2.2.1 Acoustic lexemes

Music Information Retrieval is a common step in organising soundbanks. Especially relevant to this project is the method proposed by Michael Casey.[11] To systematize a large amount of audio recordings of varied character, he would conduct segmentation and analysis, later followed by classification to identify sonic archetypes commonly occurring within the data. The causal-temporal relationships between them would be then modelled using a handful of ML algorithms such as Hidden Markov Models and the Viterbi algorithm. The capability to generate label sequences was implemented with database look-up in mind rather real-time creative use, but the workflow itself lies in close proximity to what was trying to achieve in my project.

[8] Magnusson, n.d.

[9] Sturm, 2006.

[10] Schwarz et al., 2008.

[11] Casey, 2005.

2.2.2 Fluid Corpus Manipulation Toolkit

The FluCoMa project[12] is a general-purpose toolkit for creative data mining of audio using Machine Learning. It provides a large number of refined functionionalities for data segmentation, analysis and processing. Due to the modular design, these can be combined to suit one's particular needs in either of environments: Max MSP, Pure Data, SuperCollider or even command line. An issue with FluCoMa is that it seems to presume a certain level of expertise from its user and contains very few ready-made examples, which can prevent less experienced artists or programmers from experimentation.

2.2.3 Neural Networks

Significant developments in the field of Neural Networks and Deep Learning over the past decade have resulted in various new methods of sound synthesis. From the point of view of an individual user, research projects developed by major AI companies often have limitations due to the manner in which they curate and condition the training data, as well as design and optimize their model architectures.[13] However, more recently, open-source architectures of a more accessible scale have emerged.

Within these, a popular approach is auto-encoding, a method that involves training neural networks to learn compressed representations of audio data. In this process, the network learns to extract essential features from the data and subsequently decode the representation back into audio. This parallels the analysis-synthesis principle discussed throughout this chapter. Auto-encoding has been employed in different projects, such as neural granular synthesis[14] and currently notorious real-time audio generation,[15] both developed at IRCAM, also responsible for aforementioned CataRT tool.

One notable project is *Musika*. It demonstrates a generative architecture capable of training and running on consumer-grade hardware while maintaining good audio quality.[16] Out of the mentioned projects, it is also the only one which considers modeling longer structures. Notably, all three projects require a certain level of programming expertise to engage with and experiment effectively.

There are also simplified, user-friendly web-based interfaces for sound synthesis based on neural networks. For example, a toy webpage that wraps RAVE models[17] and Holly Herndon's *Holly+*[18] provide accessible means for timbre transfer and other sound

[12]Tremblay et al., 2021.
[13]Dhariwal et al., 2020.
[14]Bitton et al., 2021.
[15]Caillon and Esling, 2021.
[16]Pasini and Schlüter, 2022.
[17]Caillon, n.d.
[18]"Holly+", n.d.

synthesis techniques.

2.3 Influences

2.3.1 Carl Stone

While conducting research for this project, I happened to often come back to the work of Carl Stone, a pioneering computer musician whose contributions I deeply admire. In an enlightening interview featured in *Tone Glow*,[19] Stone delves into his experiments with a sampler and the modulation of micro-scale playback parameters, ultimately resulting in the creation of his groundbreaking 1986 piece, *Shing Kee*.[20] Stone's hands-on approach, reminiscent of the famous Phuture 303 anecdote[21], aligns closely with my own artistic framework, making him a significant source of inspiration for this project.

2.3.2 EVOL

EVOL is a duo of Roc Jiménez de Cisneros and Stephen Sharp, who algorithmically exploit the sound palettes of rave music. After the performance by Roc[22] in October 2022, I had the opportunity to engage in a conversation with him about the captivating rave acapella drone that initiated his set. Our discussion explored the techniques employed, including various implementations of timestretching algorithms and the use of hardware samplers like the Akai S2000.[23] He also recommended exploring the works of Theo Burt, which led me to an intriguing discovery: Burt is actually one half of The Automatics Group, whose contributions have significantly influenced my initial inquiries into Music Information Retrieval. A notable release by EVOL that caught my attention is *Speed Snake*,[24] showcasing their expertise in timestretching techniques. Additionally, their *Hardcore Vol. 4*[25] release, featuring several hours of material generated using a Recurrent Neural Network (RNN) by Jules Laplace, highlights the convergence of machine learning and experimental sound synthesis so crucial to my endeavour. The accompanying documented conversation between Laplace and Jiménez de Cisneros[26] covers a wide range of topics, including creative applications of ML and critical theory

[19]Minsoo Kim, 2020.

[20]"Shing Kee", 1986.

[21]See: Mark Fell on the birth of acid house and Martin Heidegger (Fell, n.d.)

[22]"Club Mutante with Evol, Covco, Kinlaw & Franco Franco, Gribs, Div Pro and Iele at Ormside Projects, London (2022) / RA", n.d.

[23]Jiménez de Cisneros, n.d.

[24]"Evol - Speed Snake", 2021.

[25]"Evol - Hardcore Vol. 4", 2020.

[26]Jiménez de Cisneros, 2020.

related to Artificial Intelligence, making it a highly relevant reference for this project and its direction.

2.3.3 HEXORCISMOS

Moisés Horta Valenzuela is a self-taught programmer and sound artist from Tijuana, México. His work encompasses various aspects of computer music, Artificial Intelligence, and the history and politics of emerging digital technologies. Among his output, two works were highly influential on my approach to creative use of new technology: generated versions of YouTube DJ podcast *HOR Berlin*[27] and the genial re-interpretation of Antonio Zepeda's *Templo Mayor*[28] made using Generative Adversarial Networks (GANs).

2.4 Conclusion

This chapter has provided an overview of key historical and cutting-edge techniques and practices in particle synthesis, ranging from granular synthesis and time stretching to concatenative synthesis and the use of neural networks. These approaches, combined with the influence of pioneering artists and advancements in machine learning, offer exciting possibilities for the creation and manipulation of sound in the digital realm, thus informing and motivating this project.

3 Context

The project is situated within the context of digital audio synthesis. It addresses the need for experimental approaches to sound design and musicking, particularly for individuals who may have limited access to high-performance hardware and proficiency in the fields of data science, sound design, or music composition. By focusing on low-power solutions, the project aims to offer an alternative to resource-intensive methods, making it accessible to a wide range of users.

3.1 User Base

The project is primarily aimed at individuals who are interested in exploring novel methods of synthesizing and composing audio. This includes amateur and professional musicians, sound designers, composers, or simply enthusiasts who are looking for accessible and efficient tools to experiment with sound synthesis. The target audience

[27] "EPOCH.000: HÖR Berlin Pt.1 - 22 March 2021 - YouTube", n.d.
[28] "Transfiguración, by HEXORCISMOS", n.d.

also encompasses those who may have limited technical expertise or access to powerful computational resources. By catering to users with diverse backgrounds and skill levels, the project aims to provide a platform for creative exploration and expression.

3.2 Types of Uses

The project offers an open workflow that can be easily adapted to different requirements. It allows for both real-time and non-real-time usage, providing flexibility in the creative process. Users can interact with the system in real time, manipulating and exploring the generated grain sequences as part of their live performances or improvisations. Additionally, users can generate and refine grain sequences in a non-real-time manner for later use in compositions or sound design projects. The project's focus, alongside low-power usage and efficiency, lies as well on modularity, making it easy to break, adapt and extend by more advanced users.

3.3 Testable Goals

To evaluate the effectiveness of the project in meeting its intended goals, several testable objectives can be considered. Firstly, it is important to assess the system's performance in generating coherent and musically interesting grain sequences. This evaluation should involve analyzing the quality of the generated sequences and whether they translate into sound that is engaging and musical (with a dose of reserve towards this last term).

Secondly, gathering feedback from users on the project's practical usability and adaptability is crucial. This should include user tests and interviews to understand the intuitiveness of the interface, clarity of the documentation, and overall workflow. User feedback will provide valuable insights for enhancing the user experience and addressing any potential challenges.

Furthermore, the project's efficiency in terms of low-power usage and resource optimization should be assessed. This includes measuring the project's computational requirements and comparing it to existing solutions to demonstrate its effectiveness in utilizing resources efficiently.

By defining and conducting tests based on these goals, it will be possible to assess the project's effectiveness in meeting the needs and expectations of its target audience.

These sections provide a comprehensive overview of the project's context, target audience, types of usage, and testable goals. They set the foundation for understanding the project's purpose and its potential impact on users and the field of sound synthesis and composition.

4 Implementation

In the overview, the workflow this project implements consists of several steps, making use of multiple ML-based MIR techniques, sequence modeling and concatenative synthesis. It is mostly implemented in Python using libraries such as *librosa*, *numpy*, *scikit-learn* and *Keras* while the synthesizer employed at the final stage has been created using Max MSP, though it can potentially be replaced by an equivalent in any other audio programming environment. Firstly, data is initially segmented into parts, after which feature extraction is applied to each segment. The averaged values of each of these features serve as the basis for classification using clustering. Secondly, each segment of the audio data is replaced accordingly with a class label, creating a simplified representation of the time series. Such a sequence can then be split into sub-sequences to construct the training dataset. Thirdly, the model is fitted to the data, after which it can be used to generate new labels based on provided input. Finally, given the generated labels, corresponding audio frames can be retrieved and concatenated to synthesize the audio output. Each part of this process will be explained in more detail in this section of the report.

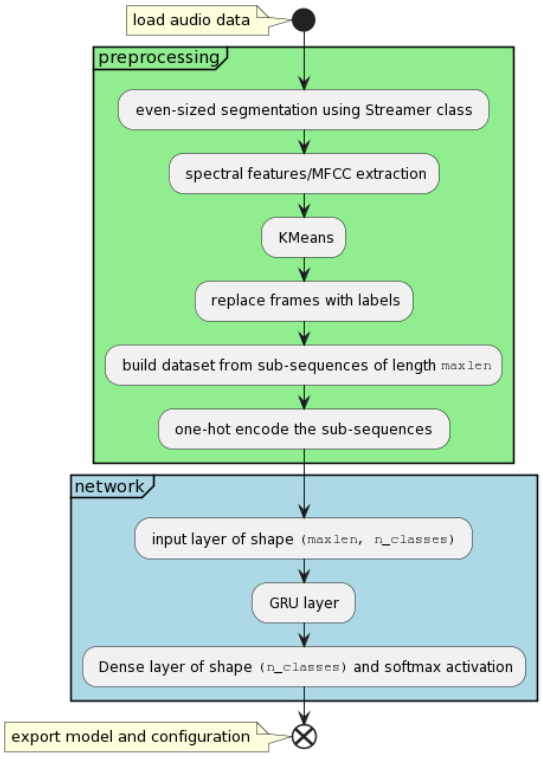

load audio data
preprocessing
even-sized segmentation using Streamer class
spectral features/MFCC extraction
KMeans
replace frames with labels
build dataset from sub-sequences of length maxlen
one-hot encode the sub-sequences
network
input layer of shape (maxlen, n_classes)
GRU layer
Dense layer of shape (n_classes) and softmax activation
export model and configuration

Figure 3: Preprocessing and diagram

4.1 Segmentation

Theory

For performance reasons, analysis of audio signal is commonly applied to uniform length, overlapping frames, with hop proportionately smaller than the frame size. The former tends to be between 10-100ms, which is the estimated optimum between resolution and preserving information when averaging. Smaller hop size increases the resolution, at the cost of increasing the complexity of computation.

Another manner in which segmentation can be executed is *onset detection*, which attempts to track the beginning of rapid increases in energy - *transients*. Among many existing approaches, some are simple such as thresholding the signal, while the more involved commonly use features such as spectral flux,[29] which is calculated as the Euclidean distance between the magnitudes across all bins in two different spectra.[30]

Implementation Loading and segmentation of provided audio file are handled as a single process in the `Streamer` class, which itself is a wrapper of the *librosa* block processing function `stream`.[31] Hyperparameters such as frame and hop size can be specified in samples, or more rudimentarily, in seconds, or metered measures at a tempo estimated by *librosa* `beat_track`,[32] itself based on a beat tracking routine by Daniel P.W. Ellis.[33] Notably, further audio analysis is applied entirely to single frames returned by the `Stream` object, apart from the optional beat detection.

4.2 Analysis

Theory Features that are extracted from each frame can be divided into two main groups. The first is specific spectral descriptors such as the spectral centroid, bandwidth, flatness and roll-off. Intuitively they respectively correspond to the ”balance point” of the spectrum given the magnitudes of its bins, the deviation from it, the tonality measure and the frequency below which a specified portion of the total spectral energy lies.[34] The second is *Mel Frequency Cepstral Coefficients* (MFCCs), which concisely describe the shape of the spectral envelope of the frame. Its calculation amounts to mapping the spectrum magnitudes to the Mel scale in an attempt to approximate human perception, finally using the *Discrete Cosine Transform*(DST) to derive a

[29]“MIREX 2012: Audio Onset Detection - MIREX05 Dataset - Introduction”, n.d., MIREX 2012.
[30]Dixon, 2006.
[31]“Librosa.Stream — Librosa 0.10.0 Documentation”, n.d.
[32]“Librosa.Beat.Beat_track — Librosa 0.10.0 Documentation”, n.d.
[33]Ellis, 2007, Ellis.
[34]Tjoa, n.d.-a.

spectrum-of-the-spectrum, commonly referred to as a *cepstrum*.[35] Normally, between 10-20 coefficients provide sufficient information. Importantly, if the features are extracted from a single frame using their default parameters, the results take the shape of a $N \times T$ matrix where N is the number of features extracted and T is the number of sub-frames the feature function internally split the analyzed signal into.

Implementation For each audio frame returned by the `Streamer` object, all of the features are calculated using their respective *librosa* implementations. MFCC extraction, as well as `n_fft` and `hop_length` parameters passed onto the feature extraction functions are optional hyperparameters.

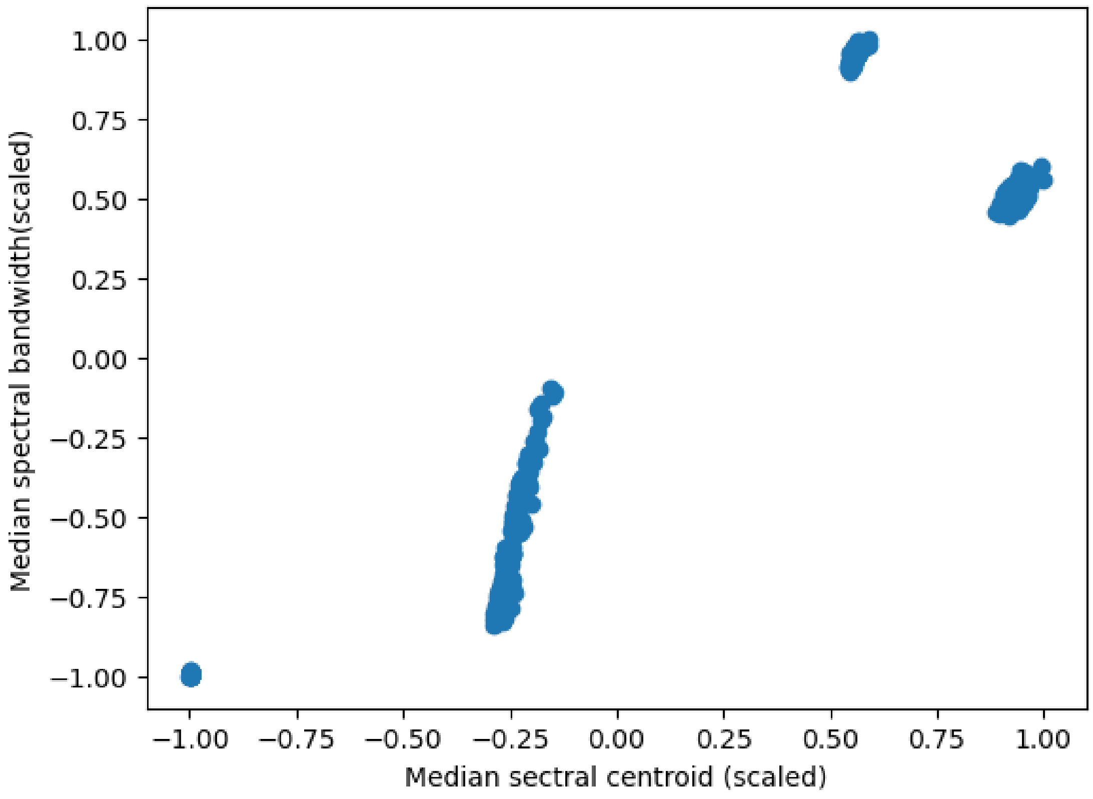


Figure 4: 2D representation of a sound corpus

4.3 Classification

Theory One of the ways to approach unsupervised classification is to do so by using a clustering algorithm such as KMeans, which assigns data points to K classes based on

[35]Tjoa, n.d.-b.

their spatial interpretation..[36] To reduce the complexity of such computation that comes with high dimensionality, the median of each extracted feature is calculated, effectively reducing it to an N-dimensional vector where N is the number of features extracted. Dependent on whether the MFCCs were included $N = 4$ or $N = 17$. Subsequently, each frame can be assigned a corresponding label, which effectively provides a new representation of the data in the form of an integer label sequence.

Implementation After calculating the median across each feature, the values are then normalized to the range $(-1, 1)$ using the `MinMaxScaler` from *scikit-learn* library. Concurringly, data points are clustered with `KMeans`, also using the *scikit-learn*. Critically, the value of K is a hyperparameter to be set by the user. As the last step, a dictionary of labeled frames is created and exported into a file for later use, where each label acts as a key for an interleaved array of start-end indices for each frame.

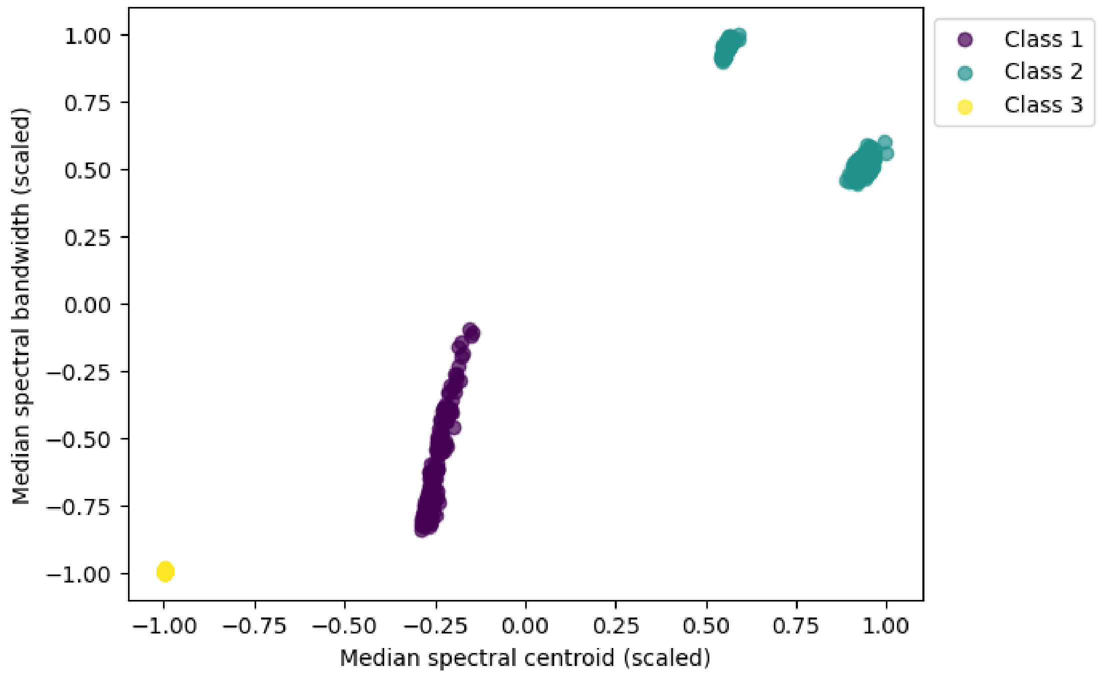


Figure 5: 2D representation of a classified sound corpus

4.4 Dataset construction

Theory Given a sequence of length L, it can be split into sub-sequences of length N using a sliding window similar to the method used in audio processing. Considering

[36]Tjoa, n.d.-c.

such a sub-sequence (also called an *N-gram*).[37] as an array of features and the element following it as the target allows for a model that given a part of a sequence can predict its continuation. As all of the targets and the features are integer values, they can be encoded into one-hot vectors where for label value l the corresponding representation is a $N - dimensional$ vector which contains zeros on all positions apart from l, and where N is the number of classes considered. This allows the network to predict soft probabilities instead of single labels, the significance of which will be discussed in the next subsection.

Implementation The dataset of sub-sequences and their target labels is extracted from the original data sequence with audio frames replaced by their labels by using a simple for loop and list comprehensions. Integer labels are encoded into one-hot vectors using the Keras `to_categorical` function.

4.5 Model

Theory Gated Recurrent Unit (GRU), is a simplified version of the more complex Long Short-Term Memory (LSTM) cell commonly used for sequence modeling in recurrent neural networks. It is designed to capture and utilize long-term dependencies in sequences by selectively updating and resetting its hidden state.[38] By doing so, the GRU can effectively capture relevant information from the past and combine it with new input to make predictions about the next element in the sequence. This makes it particularly suitable for tasks involving sequential data, such as generating new labels based on provided input. Followed by a fully connected layer with softmax activation function, which returns probabilities for each fo the classes. Categorical crossentropy loss is then used to compare the predicted distribution with the true labels, measuring the dissimilarity between them and providing a gradient signal for training the model.

Implementation For modeling the label sequence, a simple but efficient at this level of complexity model architecture is employed. Adapted from char generation tutorial by Lukas Biewald.[39] It can be described as *shallow*, as it consists of three layers only: an input layer of shape (N, C), where N is the size of the previously constructed N-grams and C is the number of classes; a *Gated Recurrent Unit* (GRU) layer, in which the number of units is a tweakable hyperparameter; and the fully connected output layer with C number of neurons with *softmax* activation. *Adam* is the optimizer of choice, while the loss function is *categorical crossentropy.* T

[37]Chollet, 2021.
[38]Cho et al., 2014.
[39]Biewald, 2015, September 1/2019.

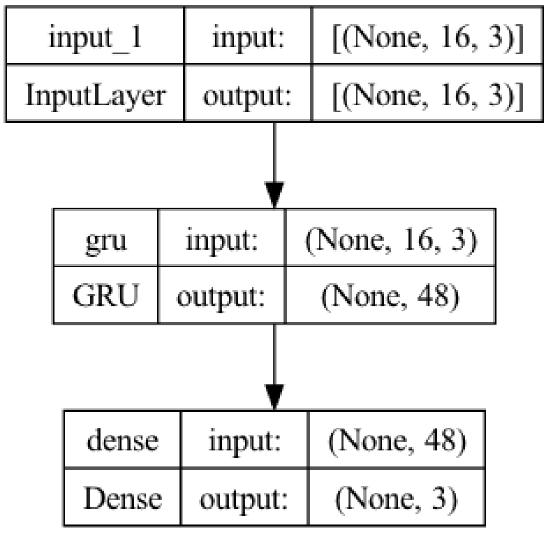


Figure 6: Model architecture diagram

4.6 Generation

Theory After fitting the model, provided with a sequence of length N it is possible to generate a distribution of the next possible labels to follow. Parametrised sampling can be applied to such a distribution, in this case, *temperature sampling*, where the temperature parameter controls the diversity of the output. It is especially applicable in the generative context of this workflow. In temperature sampling, a parameter called temperature is applied to the softmax output probabilities. A higher temperature value $T > 1$ makes the distribution more uniform, leading to increased randomness in the generated output. Conversely, a lower temperature value $T < 1$ sharpens the distribution, making the generated output more focused and deterministic. The sampled label can be added to the sequence used as the prompt to generate the next one, allowing the process to be repeated ad infinitum to generate a sequence of any length, even though its undestanding of it is limited to a local context of the N-gram length.

Implementation Generation takes place in a separate Python script `online.py`, which runs the generation on loop in its own thread, by default prompted with the first N-gram from the original sequence. After each generation, it removes the first element of the N-gram, replacing it with the generated token. When a sequence of specificed length is generated, it is sent as an *Open Sound Control* (OSC) message to the target environment.[40] Prompt, temperature and length of sequence to generate are parameters, all controllable through OSC. A separate thread runs an OSC server which

[40]"OSC Index", 2021.

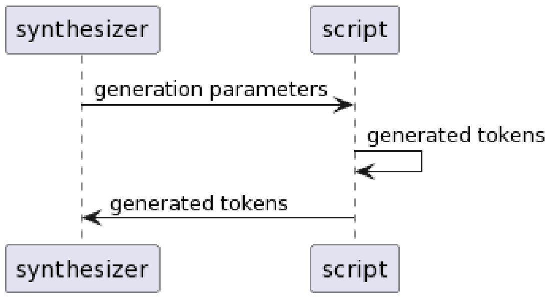


Figure 7: OSC communication between generation script and synthesizer

listens for messages and passes any changed parameters to the generation thread. To make the temperature changes effective instantenously, all tokens generated so far are cleared and generation re-starts from scratch (with default or specified prompt).

4.7 Synthesis

Theory While the more standard overlap-add method is still widely used in various software, programs such as Max MSP and SuperCollider contemporarily provide the tools to work with multiple channels. This opens up possibilities for elaborate spatialization techniques and simplifies the implementation of techniques like granular synthesis. For each start/end index pair, a grain instance should be created on one of the channel and traverse through the appropriate part of the original audio file. A window function or a linear ramp should be applied at the start and end of the grain to avoid artifacts such as clicks.[41] Notably, the number of channels is flexibly chosen and can be much higher than the number of target channels to mix down to. The selection of grains informed by the generation script makes this synthesizer an expansion on the method of concatenative synthesis, as described in the background research section.

Implementation An example synthesizer has been implemented using Max MSP, though this is rather straightforward to achieve in any sound programming environment which is compatible with the OSC standard and has the capability of reading dictionaries in JSON format. As the label sequences received from the generation script via OSC and are pushed to the end of a FIFO queue, particular elements are retrieved from the queue at a synchronous rate controlled by the `metro` object. To further enhance the creative control over the audio output, it is possible to lock the last generation sequence, playing it on a loop and ignoring the queue. This allows additional periodicity which is highly useful in some contexts. The synthesizer provides multiple

[41]Roads, 2004.

methods to decode the label into an audio frame, including fixed indexing, sequential traversal, probabilistic traversal, and random selection. These methods allow for flexibility in generating different sonic outcomes.

5 Process

5.1 Introduction

The project followed an iterative design approach, consisting of cycles of ideation, implementation, testing, and refactoring. These cycles allowed for improvements in my understanding of crucial machine learning (ML) concepts. The project was tracked using a GitHub repository, which facilitated version control, code documentation, and organization. I also used a Wiki as a personal blog to record dilemmas and insights. Meetings and discussions with colleagues and mentors provided valuable feedback and guidance, stimulating critical thinking and shaping the project's direction.

5.2 Point of Departure

The project began with some superficial ML intuition gained from previous courses and experience in Music Information Retrieval. Building upon my previous projects, ”Sound and Signal 2” and ”Artificial Intelligence,” I used the latter as a starting point for my Final Project. Although aware of my limited understanding of the problems I aimed to solve, my previous use of neural networks for synthesis in Max MSP gave me hope that my intuitions would translate to the lower-level workflow of the project. Encouraged by my AI lecturer, Jérémie Wenger, I embraced the challenge and embarked on confronting my concept with reality.

5.3 Gaining Orientation

Early meetings related to the project highlighted the challenges of explaining the scope to others unfamiliar with my previous work. Following advice from my supervisor, Dave, I took a step back to evaluate what I already had in place. This process helped me clarify the project's direction. To create something more than simple edits, I aimed to make the project work in real-time.

5.4 Initial Refactor

Before considering additional features, I addressed the instability of the existing code in the segmentation/analysis/modeling/generation/synthesis pipeline. Applying data processing best practices, I balanced optimization and functionality. Optimization was prioritized enough to prevent code crashes while allowing for efficient handling of Git issues. The overall pipeline design was refined to ensure coherence and correct functioning.

5.5 Novelty Factor

After addressing previous mistakes, I replaced the unwieldy Python synthesis script with a quick and dirty solution in Max. This design choice necessitated the use of two separate applications due to the complexity of running a Keras model from within Max. However, this brought me closer to the goal of integrating the project to universal compatibility with various audio engines for synthesis.

5.6 First Tests

Despite the processing and synthesizer not being user-friendly, I conducted user testing to gauge progress. My classmate, Shang Ren, volunteered to try the prototype. During testing, we encountered challenges with setting path variables for training a new model. However, the generated audio showcased interesting collaging of sounds, and real-time attempts revealed the importance of parameter range restrictions for stability.

5.7 Clean-ups

During this phase, I focused on code and patch clean-up. I also attempted to write an alternative synthesis algorithm using the gen language in Max, although this proved to be counter-intuitive due to its unconventional syntax and limited documentation. Discussions with members of the RAVE Discord server provided ideas for future work, particularly in the area of artificial augmentation of datasets. Additionally, I used the patch and a trained breakbeat model during a live improvisation gig, which highlighted timing issues.

5.8 Catching Up

To benchmark the project's progress, I conducted thorough testing. However, even with a simple toy dataset, the generated results remained chaotic. Extensive debugging revealed that the neural net architecture had been generating random values due to errors made six months prior. This discovery led to valuable insights into the challenges of generative approaches and the need for rigorous testing and debugging[42].

[42]For a concise description made on the spot after fixing these issues, see blog post from 2023/05/08: https://github.com/wwerkk/MC-FP/wiki/2023-05-08

Figure 8: dupnac with Michelle Olczak (left) and Lorelai b. Woch (right), 25.04.2023 at Mlodsza Siostra, Warsaw, Poland

5.9 Explaining Over Again

Presenting my project at the graduation show provided valuable feedback and opportunities to refine the project's narrative. Conversations with experts, such as my C++ lecturer Lance Putnam, informed my understanding of handling random distributions in a creative context and searching for the middle ground between what human perception is capable of appreciating and procedurally generated chaos. Inspired by that, I eventually ended up adding a looping functionality to my patch, to account for the high variance of generated data, but likely this was just a small manifestation of the thought process which started in my head at that point. Inspired by that is the addition of a locking receieved prompts in a repeating loop, to account for the high variance of generated data. Performing on a multi-channel sound system highlighted the engagement potential of the project, as I have heard a handful of positive impressions about the textures emerging from the synthesis process which I did not consider capable of standing for itself in the context of a solo piece.

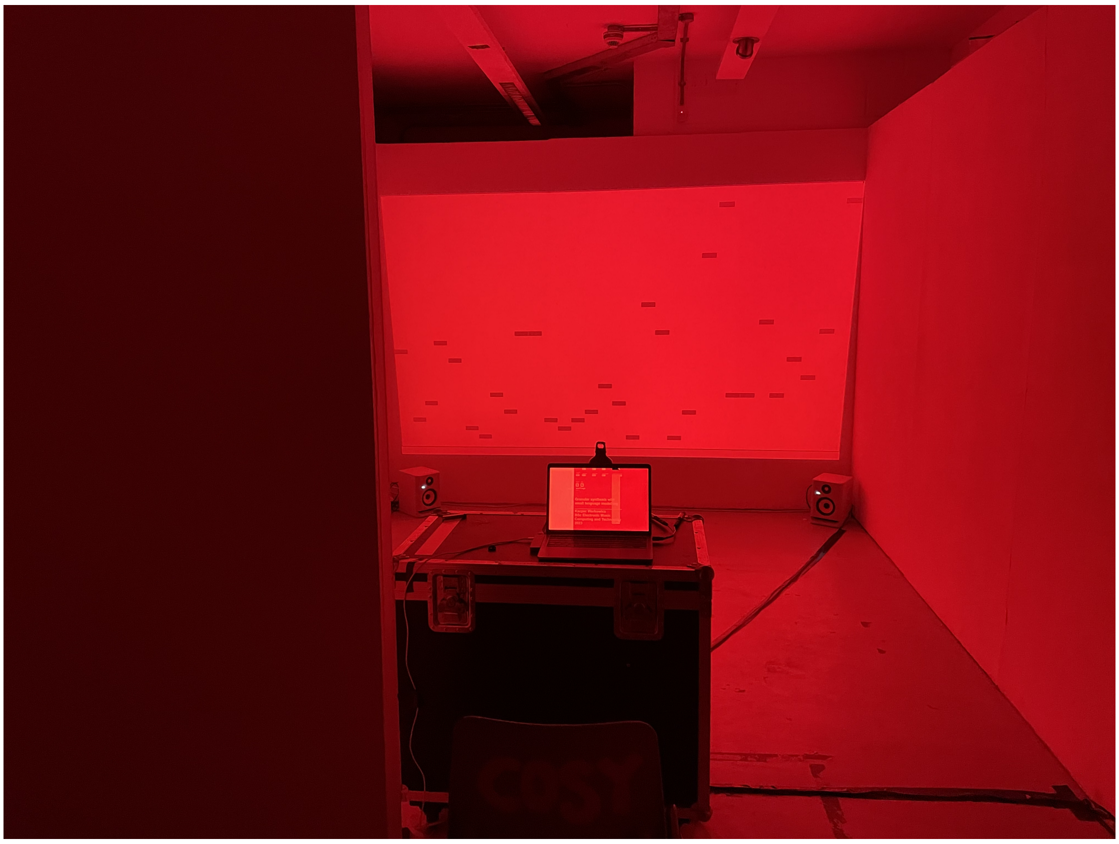


Figure 9: Project presentation at Music Computing graduation show on 11.05.2023

5.10 Failing Again

During further benchmarking, issues with clustering accuracy emerged, revealing another bug in the system that had grown in complexity.

5.11 Final?

As the project reached its final stages, I had to remove several planned features to prioritize functionality. Time constraints led to the conscious decision to focus on the well-functioning aspects of the workflow, even if they were once the most problematic components.

6 Debugging

Given the large scope of this project, multiple challenges have arisen during its development. Git issues[43] were the chosen method of organising and keeping track of bugs and other encountered problems related to optimization or new functionalities. Considered below are the obstacles picked on the grounds of meaningful contribution to advancing this project and my personal expertise. In perspective, these endeavours seem to place themselves within one of two areas: 1. handling of data Input/Output (IO) and computational efficiency specific to this project, 2. audio analysis and Machine Learning problems.

6.1 Data Processing

6.1.1 Analysis Pipeline Input

Description Processing longer audio files would take a lot of time or result in *Out Of Memory* (OOM) errors causing the kernel to crash.

Breakdown The audio file would be loaded into the memory all at once, using either of two functions: *librosa* library `load` or `open_audio` from *audio2numpy* Python package. This heutiristically proved to be a severe performance bottleneck. After segmentation, sample values of each frame would be stored in a multi-dimensional numpy array, further increasing the computational complexity and memory usage.

Solution After researching the issue, it has been discovered that the preferred approach is to load the audio block by block, and that *librosa* `stream` function[44] serves that exact purpose. It has been wrapped in a custom `Streamer` class which can be used to convniently simultaneously segment and analyse the data.

6.1.2 The Inffeciency of Implenting Audio Synthesis Python

Description Synthesizing audio output in Python using the overlap-add[45] method would often take a lot of time to compute or cause OOM errors, also being difficult to experiment with and debug.

Breakdown As suggested by the previous problem: given the high sample rate used for calculating audio, *numpy* is not efficient in doing so on sample-per-sample basis.

[43]"Issues · Wwerkk/MC-FP", n.d.

[44]"Librosa.Stream — Librosa 0.10.0 Documentation", n.d.

[45]Roads, 2004, Microsound p.257.

Solution The pipeline originally implemented in a single Jupyter notebook has been broken into three separate parts: analysis/training, label sequence generation and an accompanying audio synthesizer implemented in Max MSP. This improved the performance and modularity of the project. The training notebook itself got reduced down to picking hyperparameters and calling the training script.

6.1.3 Analysis Pipeline Output

Breakdown Given the generation script and synthesizer both require access to some of the processing results, as well as the original audio file in case of the latter; it had to be considered what data needs to be passed on from the training/analysis script and how to execute such communication.

Solution The design choices at this stage had to be deliberate, as they were somewhat critical to reaching the accessibility goals of the project. For every configuration with a unique name, two .json files would be exported alongside the Keras model. The first would store hyperparameters such as path to the source audio file, sample rate, frame and hop size, etc.; while the second would be a dictionary of labelled frames, where the value for each key (label) would be an array of frame start/end indices. This information turned out to be sufficient to successfully execute synthesis on toy examples in a separate Python script (`offline.py`, later removed from the repo). Achieving the same result with Max was mainly constrained by problems with reading out the values from the dictionary. After searching the Cycling'74 Forum (several threads on using dict in dict) and some experiments, the problem turned out to be simple but confusing: Max `dict` object is not capable of reading 2D Python arrays, therefore the start/end indices had to be saved as an interleaved 1D array.

6.1.4 Real-time Generation and Open Sound Control

Breakdown The script was a single-thread OSC server listening for a *generate* containing the generation parameters. After receiving it would generate a label sequence and send it, again using OSC. When the p-th last label in the current sequence was triggered to play in the patch, the patch would prompt the next generation. Combinations of low sequence length and fast trigger rate would cause large amounts of messages to be sent from the synth to the script, effectively clogging the OSC queue. Running out of labels in the FIFO queue, synth would go silent.

Solution During consultation with Dave which followed him testing the patch, he pointed out the weak points of my buffer-ahead solution: low performance and delay in generation parameter adjustments affecting the playback output. His suggestion was to split the script into two threads: one generating ahead in an infinite loop and. At the same time, a separate thread would run the OSC server listening for any parameter changes and pass them to the generation thread. Initially hesitating due to my lack of proficiency with thread management in Python, I finally decided to attempt this approach which turned out to be an interesting challenge to solve. Main obstacles were related to scope issues such as updating variable state across different threads.

6.2 Audio Analysis and Machine Learning

6.2.1 Sequence Generation

Breakdown During generation benchmarks on a toy dataset[46] it turned out that during training, the accuracy metric does not provide any coherent information, while the validation loss starts oscillating between two values almost immediately and does not go down as the training progresses. Given the simplicity of the data, the results were straightforwardly random regardless of the sampling temperature.

Solution Backtracking the issues I decided to compare my adapted code with the original code in the tutorial I was initially using to build the model. There was one crucial difference which was the choice of the loss function - the tutorial model was using *categorical cross-entropy*, while my model used *sparse categorical cross-entropy*. After consulting Stack Overflow[47] and, followingly, the Keras documentation,[48] it had turned out my loss function has been incompatible with the way my training data was formatted. This was already an issue in the former AI project I was basing on, which explains the excessive effort I had to put into adapting Biewald's code to process my data. The reason for it was my initial lack of understanding of the pre-processing involved in his, but most crucially, of specific loss functions and how they affect the model architecture and effectively its behaviour. After debugging the model architecture validation loss and accuracy metrics finally started to make sense. Yet, even with accuracy reaching the value of 1.0 in the case of the toy dataset, the results generated with low or no temperature were often not correspondent with the original data. This leads onto the next issue.

6.2.2 Data Clustering

While benchmarking with the toy dataset, it appeared the label representation of audio frames introduces a significant amount of error to the output.

[46](*drumloop*) - a repeated one bar 120BPM loop consisting of three samples (kick, hihat, clap) arranged as KHCH.

[47]Bitswazsky, 2021.

[48]Team, n.d.

Breakdown To explain this issue clearly, let us consider the sequence consisting of multiple exact repetitions of (K, H, C, H), where K, H and C correspond to kick, hihat and clap samples spaced out evenly across four beats at 120BPM. After segmenting the data at that exact rate and conducting feature extraction, followed by a clustering algorithm, the first three sounds appearing in the data have been assigned the labels $(0, 1, 2)$. Intuitively, $(0, 1, 2, 1)$ would be the correct representation of that sequence, meaning that using these labels we can go back to the original audio sequence, perhaps with slight variation. This in practice proved to be impossible with the exception of retrieval of the first audio frame available for each label (as they have been added to the dictionary in order of appearance in the data). This effectively prevents any accurate modeling from taking place, which extends to any pursuits of coherent generation and synthesis.

Solution Diagrams for KMeans, DBSCAN, UMAP and combinations. As it turned out, even in the ideal case of a simple toy dataset, where the exact timing and number of sounds is known, the labels get mixed up. Alternative approaches have been attempted experimentally, such as using the DBSCAN algorithm instead of KMeans, or preceding the clustering by dimensionality reduction with UMAP. The dimensionality reduction did mildly improve the classification, but the described problem persisted and required further investigation.

6.2.3 Audio Representation

Given the classification issues and intuitive interpretation of visualized feature space, the extracted features and their properties were unsatisfying.

Breakdown Something, a diagram. Deducted from the lack of improvement over different classification methods was the possibility of wrong set of features which have been chosen to represent the data.

Solution Initial approach to feature extraction included a combination of so-called *crude features*: zero-crossing rate and mean energy, as well as spectral features and MFCCs. Multiple experiments have been executed to determine the most robust combination of these to describe the two toy datasets[49]. The results, as well as reviews of previous iteration of the pipeline (SNS2 project), proved that the spectral features are usually the most performant out of all three subsets. At the same time, MFCCs are overall rather robust for representing even more percussive data that includes non-pitched

[49]aforementioned *drumloop2* as well as the *sweep* dataset, both synthesized to simulate different traits found in real-world audio: first - looped structures of percussive elements, second - structures focused on modulations of pitch

sounds if the first coefficient is not discarded, which is against common practice. It has been decided to stick to the spectral features as default, while the MFCC extraction is an optional hyperparameter, as they do contribute meaningfully to data representation in cases of analysing audio with a lot of periodic components. However, the experiments at this stage have again achieved only mild improvement. What should be considered is the general problem of meaningfully representing data in a compact manner. Towards the end of project development, I have attended virtually the *Creative Machine Learning* summer classes taught by Philippe Esling at University of Tokyo. Through discussion on Discord server related to the class, I have stumbled upon the following exchange related to building a multi-label classifier for the MuscleFish dataset:

> `cakste`: *[...] the dataset is very unbalanced, and plotting the mean/std of the features doesn't seem to visually be discriminative enough.*
>
> `Philippe Esling`: *You are plain right, mean features have a glass ceiling of discrimination*
>
> `Philippe Esling`: *Tips / tricks are all the regularisations we saw (ReLU, dropout, BatchNorm)*
>
> `Philippe Esling`: *But a full CNN based on the STFT would improve over mean features and MLP !*

Though this knowledge was highly relevant for my project, applying it in practice would unfortunately require more time than I had on my hands and had to be left to future experimentation.

6.2.4 Audio Segmentation

Even segmenting data in sync with exact metered beat, frame start points would shift slowly over time, introducing error to further analysis steps.

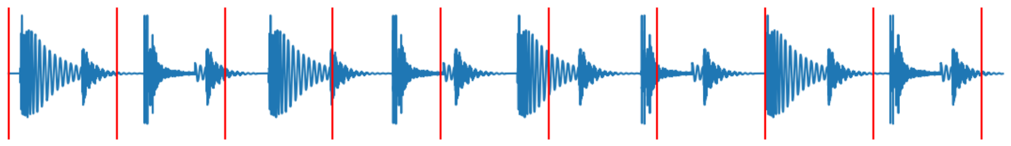


Figure 10: Unmatching segmentation of a 4-element drum loop at 120BPM

Breakdown Following the accuracy problems at each step of the pipeline has led to discovering the issues inherent in the very first step of analysis, as the segmentation even at a known tempo was very obviously wrong. The default way of passing the frame length to the training script was to give it a metered measure specified in beats.

The conversion from beats at a specific tempo involved a bit of division/multiplication and was not implemented transparently and was not ever properly checked after its implementation.

Solution The efforts have concentrated on potential work-arounds, mainly coming back to onset detection, as it has been used successfully in the past projects. Doing so, they involved either *librosa* or the *FluCoMa CLI* Python bindings by James Bradbury.[50] Neither of these has solved the issue, complicating it more instead. During the writing of this report, not debugging the numerical measure-sample conversion appeared as an obvious oversight and after looking into has been fixed without much issue.

6.3 Overview

The groundwork for this project was laid by previously designed pipelines, often sub-optimal or working only within a highly restricted set of parameters and required plenty of adaptation work. Most of the bugs specific to this project and its implementation, especially when it comes to compatibility between particular parts of the workflow have been resolved successfully. Certain issues, upon being researched, turned out to be related to more general ML problems which could surely benefit from being assessed in their own respect. This is even more so as the problems at given point tend to cascade into next stages in a manner that makes the actual problem difficult to position. The optimistic interpretation would be that that performance of further stages of analysis could be improved by further fixes of relatively simple, but critical bugs such as segmentation problem desribed at the end.

7 Evaluation

Throughout the project, I aimed to create a real-time audio generation system using machine learning techniques. In simple terms of constructing a workflow or a pipeline, I consider the project to be successful. Despite encountering challenges and setbacks, I managed to develop a piece of software that is functional and that generates interesting and varied audio outputs. It can be trained, run and, most importantly, experimented with efficiently assumed only some basic programming knowledge and using a personal computer.

It was challenging to narrow down my aims more and more, as I gained a deeper understanding of the project's complexity. I realized the drastic need to focus on ensuring each of the existing parts works well, rather than expanding it with additional features. In retrospect, the project scope was likely too wide and I would have had an easier time focusing my research on a single method out of many employed here.

[50]Bradbury, 2020, June 8/2023.

To test the project, I conducted user testing and benchmarking. These tests were appropriate as they provided valuable insights into the state of the system, its shortcomings and potential improvements in regards to interfacing with it. Both testing and benchmarking proved to be often ambiguous in context of generative tools like this, which opens a lot of room for critical discussion, also from the point of view of aesthetics.

The use of a GitHub repository and a personal Wiki for documentation and organization proved effective. However, there is plenty of room for improvement in terms of my project management and programming habits.

Problem-solving was an integral part of the project, and my approach was generally successful, even in cases which required deep investment and plenty of time. However, in hindsight, I could have incorporated more structured testing and debugging from the start, perhaps using synthesized data to throughouly test the processing at each stage.

Looking to the future, the project has potential for further development. The system could benefit from being split into parts, as it is already quite modular. This would enable for a wider range of applications, such as experimental data augmentation methods, audio scraping and sample organization. Integration with other environments is ensured to be smooth, as the analysis results are saved in a standardized format and take up a low amount of space, allowing them to be easily shared.

Notably, I have realized the depth of the research, creative and sometimes ethical problems I encountered. Many projects and resources relevant to this highly specific field of neural audio operate on a level of complexity that makes them inaccessible for the average person. From perspective, my endeavour operated at mixed and often contrasting levels of expertise, as I was simultaneously catching up with basic data science competence and building intuition with advanced and cutting-edge AI tools.

Trying to take a step back, it had been very meaningful for me to talk about this project with my lecturer in Improvisation, Iris Garrelf. The conversation we had helped me take a step back and approach this project critically, including considerations in a context that is not only technological, but also related to my personal practice. Notably, until that point I have only been judging this work on the basis of its technological accomplishment or potential foor achieving such, denying it any notion of value otherwise. What is worth considering, is whether making a musical instrument that is a polished state of the art technological object a goal worth pursuing? Would it effectively make for a better musician? Alike to the rather fundamentall Machine Learning problems I have encountered during implementing the presented idea, these are questions which likely require interrogating in a research project of their own.

8 References

References

Schwarz, D. (n.d.). *CataRT*. Ircam. Retrieved January 30, 2023, from http://catart.concatenative.net/

Schwarz, D. (2007). Corpus-Based Concatenative Synthesis. *IEEE Signal Processing Magazine*, *24*(2), 92–104. https://doi.org/10.1109/MSP.2007.323274

Roads, C. (2004). *Microsound* (1. paperback ed). MIT Press.

Truax, B. (n.d.). *Granulation of Sampled Sound*. Retrieved May 29, 2023, from https://www.sfu.ca/∼truax/gsample.html

Duguid, B. (1996, June). *Interview with Tony Conrad*. Retrieved May 29, 2023, from http://media.hyperreal.org/zines/est/intervs/conrad.html

Mackay, R., & Haworth, C. (n.d.). *Towards a Transcendental Deduction of Jungle (Interview) (Part 1)*. Robin Mackay. Retrieved May 29, 2023, from https://readthis.wtf/writing/towards-a-transcendental-deduction-of-jungle-interview-part-1/

Magnusson, T. (n.d.). *Granular Synthesis — scoring*. Retrieved May 29, 2023, from https://thormagnusson.gitbooks.io/scoring/content/PartII/chapter10.html

Sturm, B. L. (2006). Adaptive Concatenative Sound Synthesis and Its Application to Micromontage Composition. *Computer Music Journal*, *30*(4), 46–66. https://doi.org/10.1162/comj.2006.30.4.46

Schwarz, D., Cadars, S., & Schnell, N. (2008). What Next? Continuation in Real-Time Corpus-Based Concatenative Synthesis. *International Computer Music Conference (ICMC)*, 1–1. https://hal.archives-ouvertes.fr/hal-01161402

Casey, M. A. (2005). Acoustic lexemes for organizing internet audio. *Contemporary Music Review*, *24*(6), 489–508. https://doi.org/10.1080/07494460500296169

Tremblay, P. A., Roma, G., & Green, O. (2021). Enabling Programmatic Data Mining as Musicking: The Fluid Corpus Manipulation Toolkit. *Computer Music Journal*, *45*(2), 9–23. https://doi.org/10.1162/comj_a_00600

Dhariwal, P., Jun, H., Payne, C., Kim, J. W., Radford, A., & Sutskever, I. (2020, April 30). *Jukebox: A Generative Model for Music*. arXiv: 2005.00341 [cs, eess, stat]. Retrieved January 16, 2023, from http://arxiv.org/abs/2005.00341

Bitton, A., Esling, P., & Harada, T. (2021, July 3). *Neural Granular Sound Synthesis*. arXiv: 2008.01393 [cs, eess]. Retrieved January 16, 2023, from http://arxiv.org/abs/2008.01393

Caillon, A., & Esling, P. (2021, December 15). *RAVE: A variational autoencoder for fast and high-quality neural audio synthesis*. arXiv: 2111.05011 [cs, eess]. Retrieved January 16, 2023, from http://arxiv.org/abs/2111.05011

Pasini, M., & Schlüter, J. (2022, August 18). *Musika! Fast Infinite Waveform Music Generation*. arXiv: 2208.08706 [cs, eess]. Retrieved January 16, 2023, from http://arxiv.org/abs/2208.08706

Caillon, A. (n.d.). *RAVE.js*. Retrieved January 16, 2023, from https://caillonantoine.github.io/ravejs/

Holly+. (n.d.). Retrieved May 29, 2023, from https://holly.plus

Minsoo Kim, J. (2020, September 25). *Tone Glow 032.6: Carl Stone*. Tone Glow. Retrieved May 28, 2023, from https://toneglow.substack.com/p/0326-carl-stone

Shing Kee. (1986). Retrieved May 28, 2023, from https://carlstone.bandcamp.com/track/shing-kee-1986

Fell, M. (n.d.). *Collateral Damage: Mark Fell - The Wire*. The Wire Magazine - Adventures In Modern Music. Retrieved May 30, 2023, from https://www.thewire.co.uk/in-writing/essays/collateral-damage-mark-fell

Club Mutante with Evol, Covco, Kinlaw & Franco Franco, Gribs, Div Pro and Iele at Ormside Projects, London (2022) / RA. (n.d.). Resident Advisor. Retrieved May 29, 2023, from https://ra.co/events/1596527

Jiménez de Cisneros, R. (n.d.). Personal communication.

Evol - Speed Snake. (2021). Retrieved May 28, 2023, from https://www.discogs.com/master/2358472-Evol-Speed-Snake

Evol - Hardcore Vol. 4. (2020). Retrieved May 28, 2023, from https://www.discogs.com/master/1727054-Evol-Hardcore-Vol-4

Jiménez de Cisneros, R. (2020). SampleRNN Trax conversation.

EPOCH.000: HÖR Berlin pt.1 - 22 March 2021 - YouTube. (n.d.). Retrieved May 30, 2023, from https://www.youtube.com/watch?app=desktop&v=UPs5L-1Amo8&t=939s

Transfiguración, by HEXORCISMOS. (n.d.). HEXORCISMOS. Retrieved May 30, 2023, from https://hexorcismos.bandcamp.com/album/-

MIREX 2012: Audio Onset Detection - MIREX05 Dataset - Introduction. (n.d.). Retrieved May 25, 2023, from https://nema.lis.illinois.edu/nema_out/mirex2012/results/aod/

Dixon, S. (2006). ONSET DETECTION REVISITED.

Librosa.stream — librosa 0.10.0 documentation. (n.d.). Retrieved May 25, 2023, from https://librosa.org/doc/latest/generated/librosa.stream.html

Librosa.beat.beat_track — librosa 0.10.0 documentation. (n.d.). Retrieved May 25, 2023, from https://librosa.org/doc/latest/generated/librosa.beat.beat_track.html

Ellis, D. P. W. (2007). Beat Tracking by Dynamic Programming. *Journal of New Music Research*, *36*(1), 51–60. https://doi.org/10.1080/09298210701653344

Tjoa, S. (n.d.-a). *Spectral_features*. Retrieved May 25, 2023, from https://musicinformationretrieval.com/spectral_features.html

Tjoa, S. (n.d.-b). *Mfcc*. Retrieved May 25, 2023, from https://musicinformationretrieval.com/mfcc.html

Tjoa, S. (n.d.-c). *Kmeans*. Retrieved May 25, 2023, from https://musicinformationretrieval.com/kmeans.html

Chollet, F. (2021). *Deep learning with Python* (Second edition). Manning Publications.
OCLC: on1289290141.

Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. https://doi.org/10.48550/ARXIV.1406.1078

Biewald, L. (2019, May 24). *Text generation*. Retrieved May 26, 2023, from https://github.com/lukas/ml-class

OSC index. (2021, August 13). Retrieved May 26, 2023, from https://opensoundcontrol.stanford.edu/index.html

Issues · wwerkk/MC-FP. (n.d.). Retrieved May 30, 2023, from https://github.com/wwerkk/MC-FP/issues

Bitswazsky. (2021, August 2). *Answer to "What is the difference between sparse_categorical_crossentropy and categorical_crossentropy?"*. Stack Overflow. Retrieved May 27, 2023, from https://stackoverflow.com/a/68617676

Team, K. (n.d.). *Keras documentation: Probabilistic metrics*. Retrieved May 27, 2023, from https://keras.io/api/metrics/probabilistic_metrics/

Bradbury, J. (2023, May 21). *Python-flucoma*. Retrieved May 27, 2023, from https://github.com/jamesb93/python-flucoma

9 Appendix

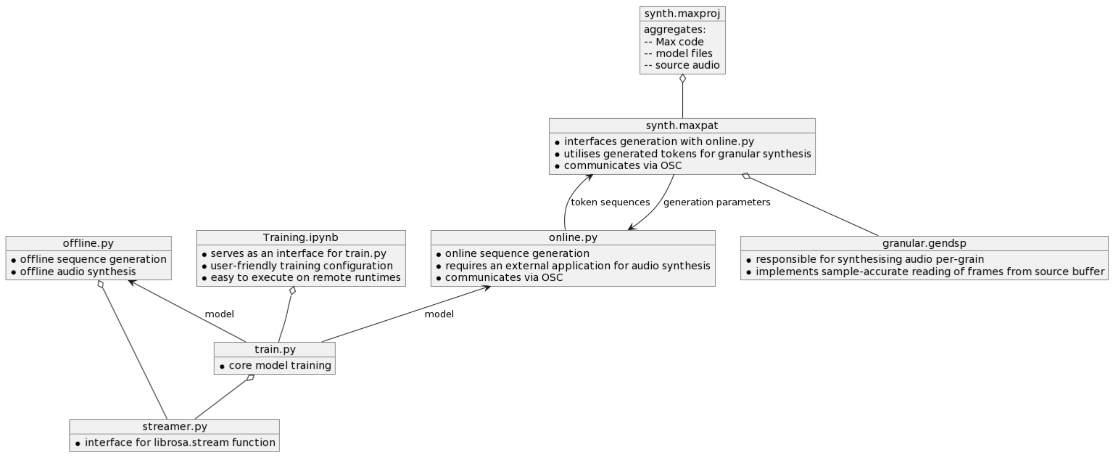
9.1 Appendix A: Code Repository

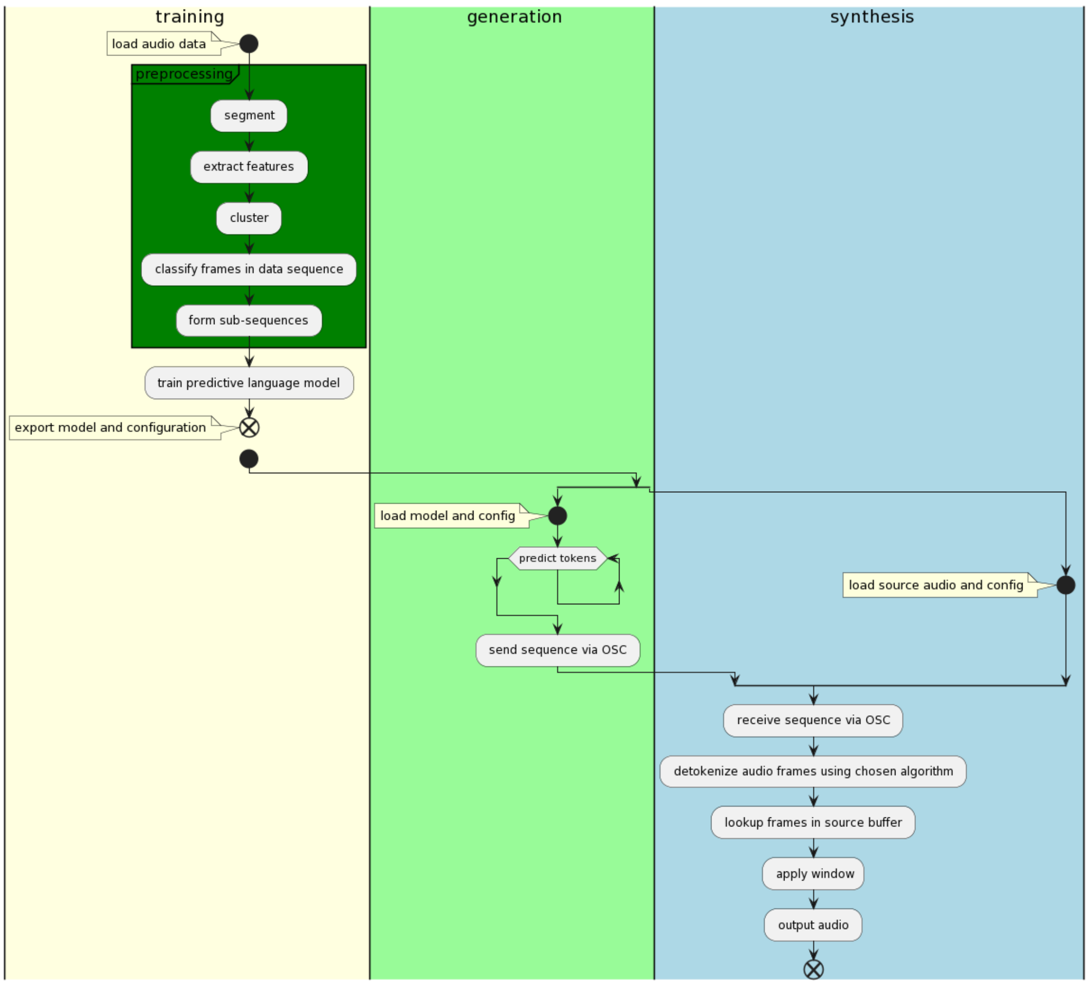
https://github.com/wwerkk/MC-FP

9.2 Appendix B: Project Blog

https://github.com/wwerkk/MC-FP/wiki

9.3 Appendix C: File Structure and Workflow Diagrams


synth.maxproj
aggregates:
-- Max code
-- model files
-- source audio
synth.maxpat
• interfaces generation with online.py
• utilises generated tokens for granular synthesis
• communicates via OSC
token sequences
generation parameters
offline.py
• offline sequence generation
• offline audio synthesis
Training.ipynb
• serves as an interface for train.py
• user-friendly training configuration
• easy to execute on remote runtimes
online.py
• online sequence generation
• requires an external application for audio synthesis
• communicates via OSC
granular.gendsp
• responsible for synthesising audio per-grain
• implements sample-accurate reading of frames from source buffer
model
model
train.py
• core model training
streamer.py
• interface for librosa.stream function



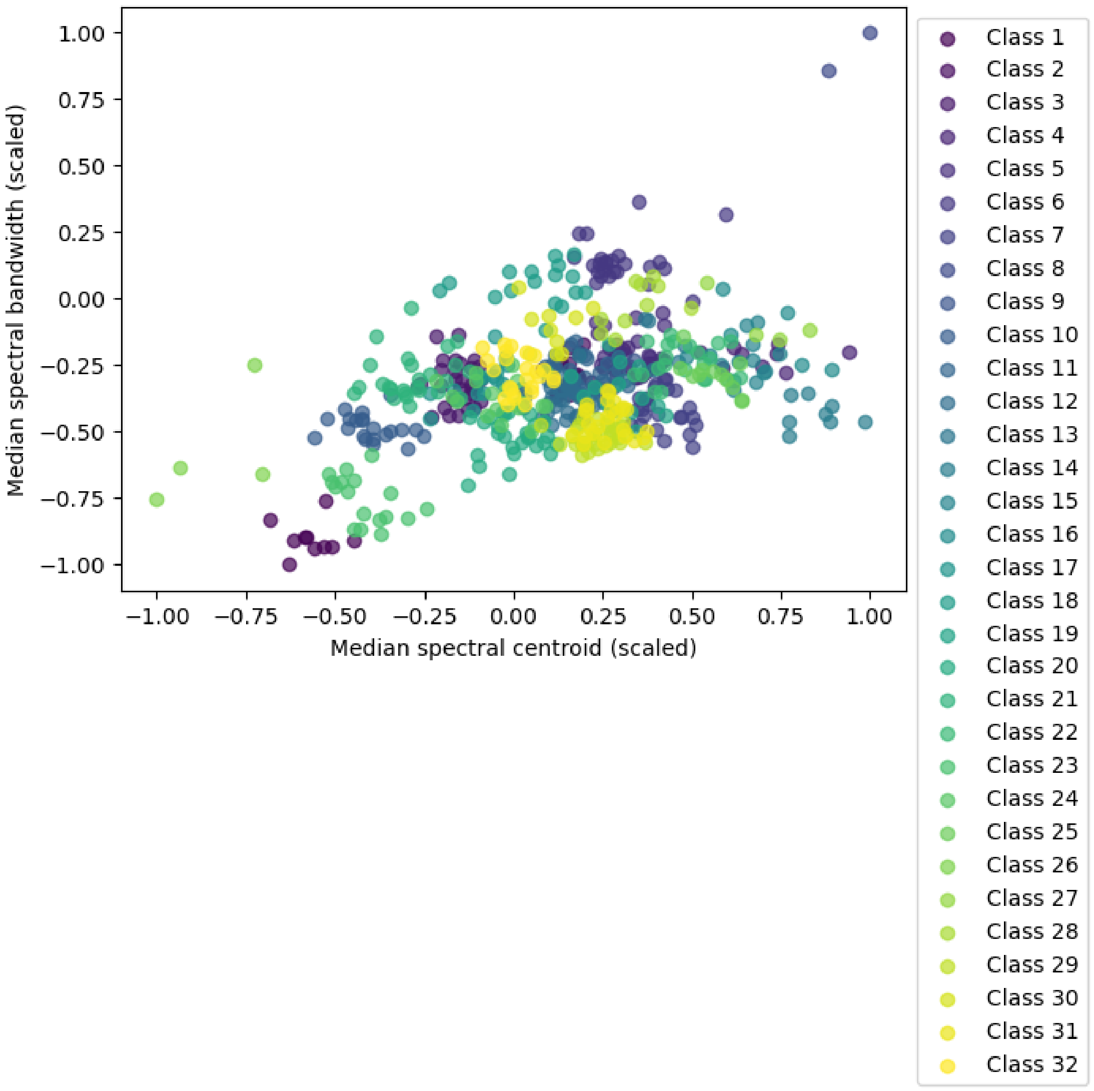
9.4 Appendix D: Clustering and Training Results for `collins` dataset

9.5 Appendix E: textitSound and Signal 2 MiniProject

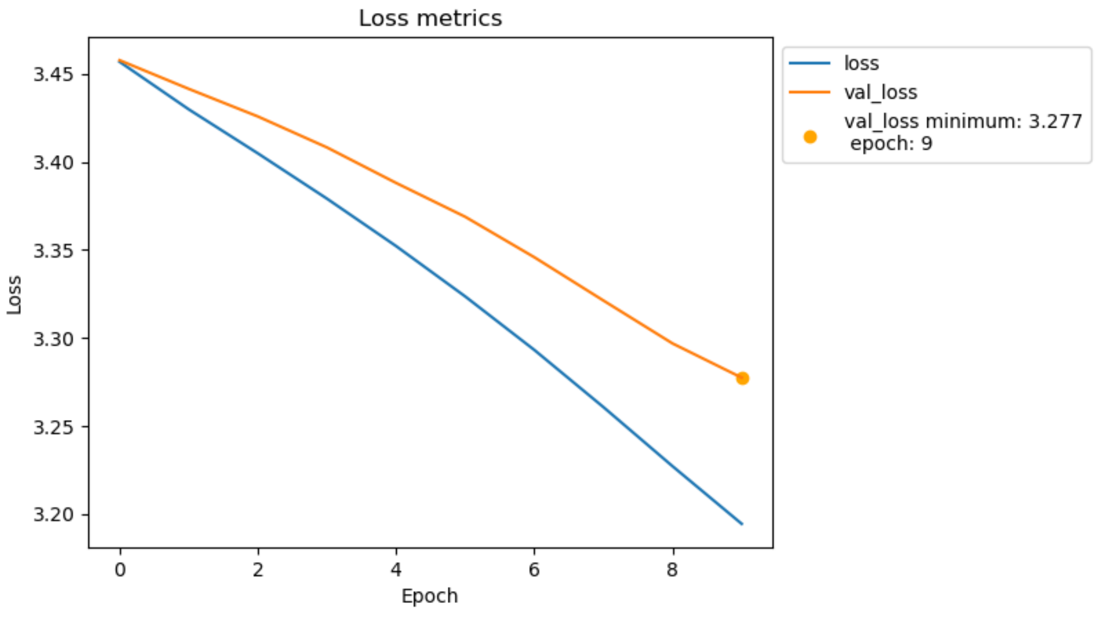
https://github.com/wwerkk/audio-segment/blob/main/audioSlicer.ipynb

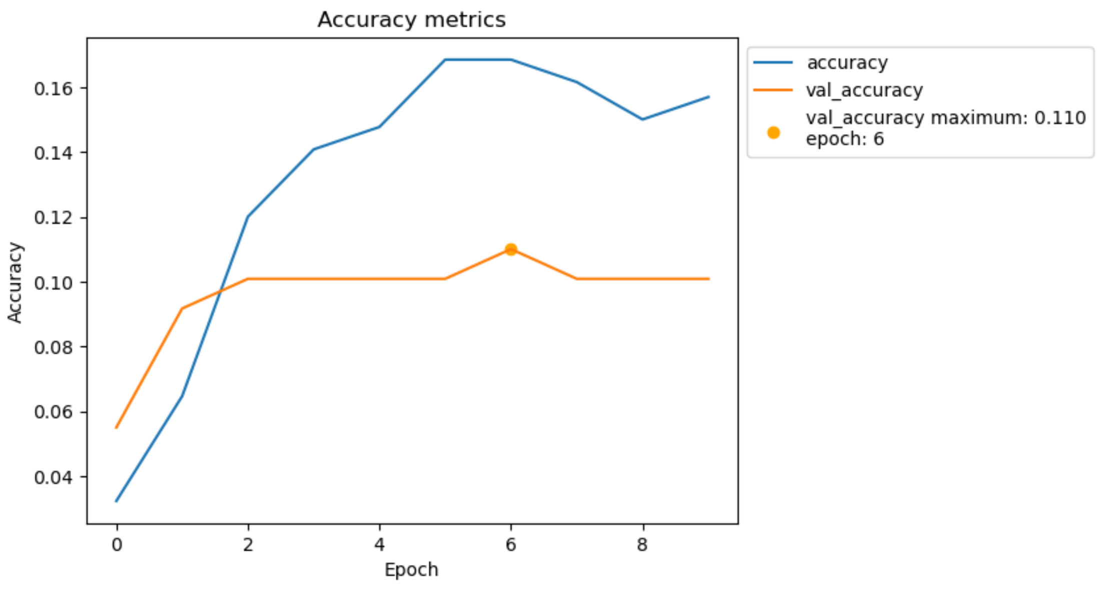
9.6 Appendix F: textitArtificial Intelligence Project 2

https://github.com/wwerkk/audio-segment/blob/main/GrainModelling.ipynb



Median spectral bandwidth (scaled)
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
Median spectral centroid (scaled)
Class 1
Class 2
Class 3
Class 4
Class 5
Class 6
Class 7
Class 8
Class 9
Class 10
Class 11
Class 12
Class 13
Class 14
Class 15
Class 16
Class 17
Class 18
Class 19
Class 20
Class 21
Class 22
Class 23
Class 24
Class 25
Class 26
Class 27
Class 28
Class 29
Class 30
Class 31
Class 32


Loss metrics
3.45
3.40
3.35
3.30
3.25
3.20
Loss
0
2
4
6
8
Epoch
loss
val_loss
val_loss minimum: 3.277
epoch: 9


Accuracy metrics
0.16
0.14
0.12
0.10
0.08
0.06
0.04
Accuracy
0
2
4
6
8
Epoch
accuracy
val_accuracy
val_accuracy maximum: 0.110
epoch: 6