Wavize: A Deep Learning Application for Enhanced Categorization of Diverse Bass Sound Designs in Electronic Dance Music

Simon Andersen
SAND22@STUDENT.AAU.COM

Sound & Music Computing
Thesis

Aalborg University
Department of Architecture, Design and Media Technology
Sydhavn, Denmark

24th of May 2024

Summary

Samples are musical audio files used by musicians in their compositions, featuring sounds from instruments like guitars, basses, synthesizers, pianos, or drums. A synthesizer, an electronic instrument that generates audio signals, can replicate real instruments or create new sounds, widely utilized in Electronic Dance Music (EDM). In the digital era, music collaboration has become effortless as musicians can produce and distribute new samples for further composing. As "sampling" is a common approach to music creation, musicians using Digital Audio Workstations (DAWs) accumulate extensive sample libraries, making management a tedious task.

With the widespread use of DAWs, tools known as sample managers have been developed to help musicians organize these samples by instrument type or characteristic. They aim to alleviate the time it takes to find specific sounds, making the musician more productive. However, as musicians create new and distinct sound designs using synthesizers, traditional sample managers often fail to categorize these new interpretations accurately, especially for bass sounds, leading to overly broad groupings.

Audio recognition, and specifically audio classification, has emerged as a significant area of research within the field of artificial intelligence (AI). Audio classification involves the categorization of sounds into predefined classes based on their acoustic features. This thesis describes the development and implementation of a sample manager application that leverages artificial intelligence and audio classification to improve the categorization of new bass sound designs. The application focuses on seven specific bass categories: "808", "acid", "brass", "growl", "reese", "slap", and "sub". By utilizing AI-driven audio classification techniques, the application aims to enhance the ability to distinguish between these different bass sounds, offering a more precise and user-friendly sample management solution.

The research involved in this thesis encompasses both the technical development of the AI model and the practical implementation of the sample manager application. Overall, this thesis contributes to the field of music technology by providing an innovative solution for sample management. The AI-driven sample manager not only improves the organization and retrieval of audio samples but also supports musicians in their creative processes by making it easier to find the exact sounds they need. This advancement highlights the potential of artificial intelligence to enhance various aspects of music production and composition, paving the way for further innovations in the industry.

Contents

Abstract

The purpose of this thesis was to explore the realm of sample managers and deep learning techniques for automatic classification and sorting of bass samples. A balanced dataset was made containing 210 samples, upscaled to 34.020 using data augmentation. From this a deep learning model, fine-tuned from the DistilHuBERT model, was developed and integrated into a user-friendly graphical interface, specifically an Electron desktop application, capable of accurately categorizing bass sounds according to their sonic attributes. The model was evaluated using various performance metrics, achieving an overall accuracy of 91.3% and an AUC score of 0.81. It demonstrated high precision and recall across multiple categories such as '808', 'brass', 'growl', 'reese', and 'slap'. Usability testing confirmed the effectiveness and accessibility of the system. Additionally, latency percentiles were used as an evaluation technique to ensure the system's responsiveness, with the longest latency for 100 sample predictions yielding 6.17 seconds. While the model showed excellent performance, occasional misclassifications in categories like 'acid' suggest areas for improvement. Overall, this project successfully implemented a high-performing, user-friendly product that enhances sample management and classification, providing a strong foundation for future enhancements.

1 Introduction

Imagine stepping into a vast library filled with countless sounds, each whispering its own story. Sound categorization acts as the librarian, guiding you through this maze of audio, helping you find the perfect melody amidst the noise. Furthermore, the process of sound categorization involves grouping sounds based on their specific characteristics [34]. In music production, this practice is crucial for efficiently locating desired sounds [72]. Musicians working with digital audio workstations (DAWs) typically gather audio files, referred to as *samples*, for use in their compositions [58]. Samples are isolated instances of sound that can be manipulated by the musician to create patterns, rhythms, melodies or bass lines within a composition [58]. They are commonly stored in digital formats such as .mp3 or .wav and are often distributed in collections called *sample packs*, which can be obtained for free or purchased from companies or sound designers [58]. Sample packs may include various harmonic or percussive sounds, such as drums, instruments, or synthesizers, and are typically organized within a computer's file system as a *sample library* [58]. Musicians frequently download sample packs and organize them within their sample libraries by creating subfolders to categorize similar types of sounds [82]. However, maintaining an organized sample library can be challenging and time-consuming due to the continuous influx of new samples, resulting in difficulty locating specific sounds [82].

Companies such as XLN, Waves, ADSR, Algonaut, and Sononym have endeavored to address this issue by developing sample organizer applications. These applications are packed with a variety of features, including automatic organization and tagging of sample libraries [37]. Upon further investigation, these tools

demonstrate promising results for organizing drum sounds [17]. However, they appear to fall short when it comes to organizing harmonic samples like basses [17]. The bass samples are either organised in categories such as 'bass', 'other', 'genre' or 'texture' [17]. Nevertheless, manual keyword-based categorization options are also available to customize the categories. Ableton, the creator of the DAW 'Live,' has even introduced a sample organizer in a recent update of their product. However, this only provides options within the bass category, such as 'Analog,' 'Basic,' 'Dark,' and 'Distorted' [16]. Although these keywords somewhat describe the characteristics of the sound, they fail to incorporate the specific terms commonly used in EDM bass *sound design*.

As music and technology have evolved, new concepts have emerged, facilitated by innovative sound design, which refers to the synthesis of sound. Electronic Dance Music (EDM) has experienced significant growth in sub-genres since the turn of the millennium [19]. One such sub-genre is known as 'bass music,' which encompasses a wide range of styles [61]. According to a definition by 'Music Radar' magazine, it is characterized by the predominance of bass [61]. Additionally, it typically features heavy use of synthesizers in the production, which are manipulated, tuned, and mixed to emphasize bass design [61]. As DAWs now are accessible to most, new interpretations of music continues to evolve. This evolution has led to the term 'bass' encompassing a broad spectrum within contemporary music [56]. Consequently, a taxonomy has emerged within the EDM community to categorize how bass sounds are created using synthesizers [56]. This taxonomy has introduced specific terms for various bass sound designs, such as 'acid,' 'neuro,' '808,' 'reese,' and others [36]. In essence, distinct bass sound designs have become nearly as recognizable as real instruments within the EDM genre [63]. This underscores the need for improvement in organizer tools within the bass category.

Only Ableton and Algonaut advertise the use of artificial intelligence(AI) in their sample managers. However, state-of-the-art machine learning models have demonstrated high performance in audio recognition tasks [81]. Thus, it appears that these companies have omitted the bass taxonomy from their systems. For this reason, this thesis will investigate implementing a broader bass categorization within a sample manager system utilizing machine learning for automatic sound categorization.

2 Analysis

2.1 Background

Understanding musicians' tools is essential for gaining insight into their creative process. The primary tool used in music production is a DAW, an application capable of recording, editing, and mixing audio for compositions [5]. Most DAWs come equipped with default samples and virtual instruments, synthesizers, or

audio effect programs, known as *plugins* [5]. Plugins are additional programs that extend the functionality of the original software [30]. Moreover, users can add and utilize third-party samples and plugins within the DAW for various tasks [5].

Within a DAW, plugins can function as digital instruments, providing new sounds playable via the MIDI protocol [5]. Meaning, they can be played with external hardware like a keyboard. These sounds range from recorded piano, guitar, or drum samples to digitally synthesized or hybrid audio. Synthesizer plugins, for example, are instrumental in crafting bass sounds from the mentioned taxonomy, as they not only synthesize new sounds but also offer a variety of pre-made sounds known as *presets* [54]. Another widely used plugin concept is called *effect plugins* [76]. This type of plugin does usually not produce audio, but rather shapes it. They are utilized to the user's creative advantage and can i.e. be filters, distortion, pitch correction, etc. [76]

In addition to plugin compatibility, users can import their sample libraries from directories on their computers into the DAW [6]. These samples can be dragged into compositions or samplers and arranged accordingly. Samples are typically categorized as *loops* or *one-shots* [6]. Loops, comprising melodies, chord progressions, vocal or drum sections, are creatively manipulated and repeated, reflecting their name [6]. They are crafted by producers or sound designers in specific scales, tempos, and time signatures, ready for further composing. On the other hand, one-shots are shorter segments, often percussive sounds or single musical notes [6]. The sample packs mentioned earlier in section 1 may contain a variety of music composition loops, one-shots, or both, drawn from live recordings or digitally produced sources [6]. These packs frequently include bass one-shots as well.

2.1.1 Sound selection

One of the key tasks in digital music composition is sound selection [7]. This involves choosing or creating sounds that align with the direction of a musical piece and is crucial to the overall quality of the music being produced [7]. When a user loads their sample library into their DAW, a shortcut to the folder is added within the software [1]. This folder can then be accessed to view the individual samples contained in the library [1]. DAWs also enable users to preview their samples in real-time when they are highlighted, facilitating the sound selection process [18]. This preview feature allows users to assess whether the sampled sound fits the creative vision of the composition before incorporating it into the project.

2.1.2 Sample categorization

Categorizing samples within a sample pack is hence crucial for streamlining the sound selection process [18]. Proper sample categorization significantly impacts

the ability to convey the sonic attributes of a sample when musicians are searching for something specific [6]. Upon downloading a sample pack, initial sorting is typically done by the creator, but further organization falls to the customer [6]. This organization can be achieved manually or facilitated by sample management tools. For instance, some sample packs may include a 'bass' subfolder containing samples not yet sorted into specific bass subcategories. Organizing these samples manually requires reviewing each one and relocating them to appropriate subfolders within the sample library. As new files are frequently added to the sample library, this task becomes recurrent and time-consuming. However, several sample managers have been advertised as being a solution to this issue.

2.1.3 Sample managers

The core function of a sample manager is to aid musicians in organizing, cataloging, and manipulating audio samples, consolidating them into a centralized repository [37]. These tools often feature standalone and plugin capabilities, compatible with both internal and external DAWs, exemplified by products from ASDR, Algonaut, and Sononym [37]. They typically encompass a variety of sample types, including loops, melodies, vocals, and other sound recordings utilized in music composition [37]. Categorization is typically based on factors such as instrument type, genre, mood, or source [37]. Furthermore, they commonly offer metadata tagging options such as tempo, key, instrument type, and descriptive keywords, facilitating efficient organization and retrieval [37]. Additionally, many provide editing functionalities such as trimming, looping, and adjusting volume or pitch of samples [37]. Refer to Appendix A.1 and A.2 for snapshots of Sononym and ADSR application interfaces.

However, upon review of the produts, it becomes evident that considerable manual intervention is still necessary despite the presence of automated features [37]. While these tools excel at automatically categorizing drum samples, instances have been observed where harmonic bass samples are misclassified as drum samples. Furthermore, ADSR's Sample Manager necessitates manual tagging or grouping of samples into specific folders, with newly added samples requiring manual tagging as well. Additionally, Sononym and Ableton's products feature a "search for similar sounds" function, which initially performs well in categorizing popular bass sub-categories like '808'. However, when compared to other bass sub-categories, the similarity range of files varies significantly. Despite distinctions between instrument plugins and sample libraries, categorization within plugins tends to prioritize harmonic sounds and warrants further exploration.

2.1.4 Plugin preset categories

Silvin Willemsen, a post-doc and freelance audio plugin developer at Baby Audio, mentioned that the presets for his latest plugin-project 'Atoms' were man-

ually categorized. He also noted that some presets were challenging to label and were placed in broad categories. However, each company's approach to sound categorization varies, as there are no standardized ways of handling sub-categories. As Willemsen suggest, it could be assumed, manual labelling might introduce some subjective bias. Investigating categories in various instrument

Figure 1: Analog Lab Pro bass sub-categories

and synthesizer plugins such as Spectrasonics' 'Omnisphere,' XFer Records' 'Serum,' and Arturia's 'Analog Pro' and 'Pigments' revealed different labeling schemes. While they all share broad categories like 'leads,' 'pads,' 'keys,' and 'bass,' expanding the bass category unveils a variety of behavior-describing labels. These included 'sub', 'acid', 'analog', 'style', 'characteristic', 'texture' and, 'genre' as seen on figure 1. Presets labeled with 'sub,' 'acid,' and 'analog' actually refer to specific bass sound designs, indicating that some plugin companies have incorporated parts of the modern EDM bass taxonomy into their preset offerings [36]. Implementing a similar approach within sample managers could enhance bass categorization, thereby refining the sound selection process for musicians utilizing sample libraries. However, this deserves a detailed look into what a bass sound entail.

2.1.5 What is a bass?

The term 'bass' serves both as an instrument and an adjective, producing or describing low-pitched tones, respectively [85]. Synthesizers have enabled musicians to utilize low-pitched wave tables to create various bass sounds with different *timbres* (see Figure 2), [77] with timbre referring to the distinct quality of the sound. This evolution has led to the creation of sound design patterns recognized as their own category, such as the 'wobble,' 'UK organ,' 'jump up,' and 'slap' bass [36]. The term 'bass' has become synonymous with EDM and has altered perceptions of what a bass sound might entail [12]. Producers utilize synthesizers to design sounds rooted in low-pitched tones but incorporate techniques such as frequency modulation (FM) to introduce mid and high frequencies into the sound [77]. These sound design methods are prevalent in EDM genres such as Drum and Bass, Dubstep, Future Bass, Trap, UK Bass/Garage, or Bass House [63].

Figure 2: Wavetable in the Serum plugin

With this musical background established, the question arises: how can these bass samples automatically be categorized?

2.2 Audio data

As the thesis focuses on working with bass audio data, it necessitates an understanding of the file contents and how to manipulate them. A sound wave, can be defined as a continuous signal containing an infinite number of signal values within a given time frame (see Figure 3[1]) [28]. Converting such a continuous

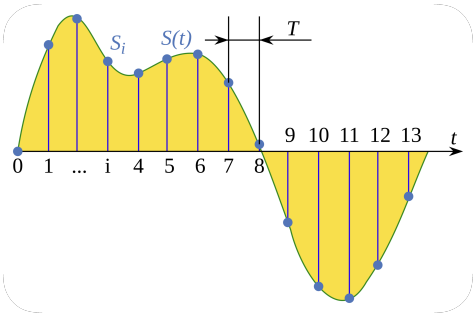


Figure 3: Sound wave defined by discrete values (courtesy of CC0 deed 1.0)

sound wave into a series of discrete values is termed a *digital representation* [28]. Typically, the digital representation of a bass one-shot is generated using a digital synthesizer [63]. These synthesizers employ algorithms to generate a stream of numbers, which can subsequently be converted into analog form [35].

The most commonly used file format for storing digital representations in music

[1] creativecommons.org/publicdomain/zero/1.0/deed.en

is the .wav format due to its uncompressed nature. This means that .wav files retain all of the original audio data without any loss of quality, unlike the .flac and .mp3 formats [83]. Audio files are saved at a specific *sampling rate*, which refers to the number of samples taken in one second, measured in hertz (Hz) [28]. To avoid confusion between the terms "sample" and "sample rate," it is important to note that in sample rate, "sample" refers to the exact amplitude values comprising a digital representation [28]. However, "sample" is also a term used in the music community to describe an audio file [6]. Typically, a sample rate of 44.1 kHz is chosen as it determines the highest frequency that can be captured from the signal [28]. This principle is known as the Nyquist theorem, which asserts that audio can be regenerated *"with no loss of information as long as it is sampled at a frequency greater than or equal to twice per cycle,"* as cited by [74]. Since the human ear can detect frequencies up to 20,000 Hz, a sample rate of 44,100 Hz covers this range of information [4].

As previously mentioned, each sample within the digital representation denotes the amplitude of the audio wave at a specific point in time. Additionally, the *bit depth* of a sample determines the level of precision in describing its amplitude value. A higher bit depth leads to a digital representation that more faithfully reflects the original continuous sound wave [28]. Typically, the most common bit depths for exporting one-shot samples are 16-bit or 24-bit [28].

2.3 Audio analysis

Various tools can be utilized to analyze audio data. For instance, the *waveform* of a signal can visualize the amplitude relationship in the time domain. Let's consider two one-shots from the '808' and 'growl' bass sub-categories and compare them. Figure 4 illustrates a comparison of the amplitude waveform

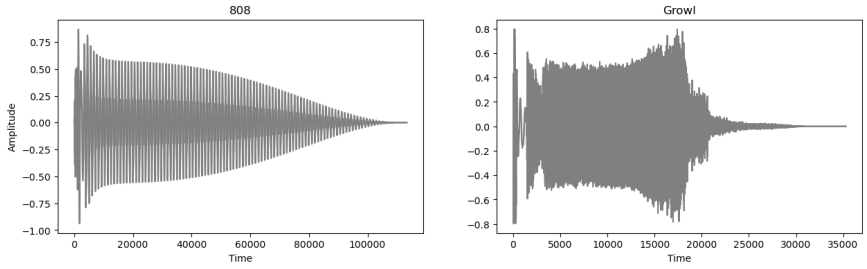


Figure 4: Wave form of an 808 (left) and growl (right) bass sample

between an *808* and a *growl* bass one-shot. It's evident that the waveform of the 808 (on the left) is more spread out, while the growl (on the right) appears densely packed together as one large block. This observation indicates that the growl contains a greater number of frequencies in the signal, although it obscures the actual presence of low frequencies. However, this characteristic can be elucidated with a frequency spectrum, which provides a representation in

the frequency domain. The frequency spectrum in Figure 5 reveals that despite containing a significant amount of mid-frequency content, the growl is primarily rooted in low frequencies. This characteristic arises from the production method of the sound. However, it's important to note that the frequency spectrum only presents a snapshot of the signal. To comprehensively analyze how the sound changes over time, a spectrogram can be employed. Additionally, the

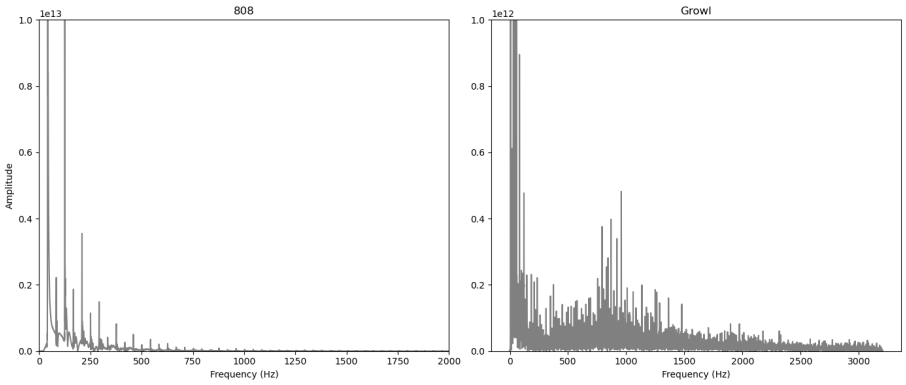


Figure 5: Frequency spectrum of an 808 (left) and growl (right) bass one-shot

spectrogram of the same one-shot can be observed in Figure 6, revealing that the growl one-shot exhibits frequencies that vary more compared to the 808. These characteristics highlight the unique behavior of each bass sub-category, which a machine learning model should be capable of capturing to automatically categorize the one-shots.

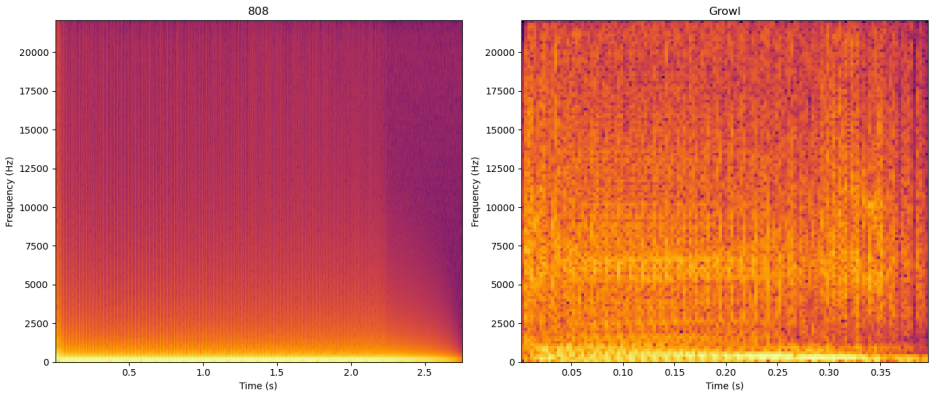


Figure 6: Spectrogram of an 808 (left) and growl (right) bass one-shot

2.4 Audio in ML

Now, how can audio data be implemented in a machine learning (ML) context? As defined by IBM, *"ML is a branch of AI and computer science that focuses on using data and algorithms to enable AI to imitate the way that humans learn, gradually improving its accuracy"* [51]. This means that audio data, which essentially tries to replicate reality, can be used within machines to mirror human-like capabilities, such as recognizing and distinguishing the difference between bass sound designs. The current landscape of audio-related ML has shown many solutions that are sufficiently able to recognize and distinguish between different types of audio, such as music, speech, and environmental sounds [55]. This is widely known in the ML space as *audio classification* and is one of the most common applications in audio and speech processing [45]. It refers to the act of assigning one or more labels to a digital representation based on its content [45]. For instance, detecting words to interact with a smartphone, inferring keywords for communication, and quite relevantly, automatically distinguishing between sounds of different bass categories.

ML models cannot directly interpret audio waves, necessitating the use of digital representations. This is addressed through *feature extraction* techniques, which organize audio data into meaningful representations [55]. These features encapsulate characteristics of the audio signal, aiding models in distinguishing between different sounds [55]. Features can span time and frequency domains, as well as encompass statistical, pitch, tempo, or timbre aspects. Among frequency domain features, *Mel-frequency Cepstral Coefficients* (MFCCs) are commonly employed [55]. MFCCs transform raw audio into a more manageable and informative format, generating a *mel-spectrogram* based on human auditory perception. This process discards less relevant information based on human hearing capabilities [55], effectively converting audio data into image data for use alongside image detection algorithms.

However, advancements in ML algorithms now enable the utilization of raw audio for audio classification [59]. Unlike images, raw audio data contains abundant information [59]. Modern algorithms can extract meaningful features directly from the raw waveform for classification tasks [59]. Regardless of feature extraction method, data must undergo pre-processing to be suitable for a ML *architecture*.

ML algorithms, often referred to as the architecture, define how the data is processed. The evolution of AI has introduced a hierarchy of techniques, such as deep learning (DL), a sub-field of neural networks (NNs), which itself is a subfield of ML [51]. Classical ML relies more on human intervention for learning, whereas newer DL methods can automatically determine distinguishing features between different data categories [51]. The term "deep" in DL simply denotes the number of layers in a NN [51]. Additionally, DL and NNs have greatly accelerated progress in computer vision, natural language processing, and speech

recognition [51].

To summarize, an automatic audio classification model can be achieved by collecting a dataset of audio samples and feeding either raw audio or extracted features into the DL architecture to make predictions regarding the categories.[51].

3 Related work

Numerous experiments have been conducted in audio classification, spanning speech, music genre, instrument, foley, and environmental sound classification. Pertinent to this thesis is the work by Anubhav Chhabra et. al. [15], who presented a drum sample classification model employing machine learning techniques with samples from a drum simulator. Their study focused on drum instruments such as bass (kick), snare drum, toms, and cymbal samples. Features like Zero Crossing Rates, Spectral Centroid, and RMS Energies were extracted and fed into three different machine learning architectures: K Nearest Neighbors, Random Forests, and Naive Bayes. The Naive Bayes architecture yielded the best performance with a 96% accuracy, although the others scored above 90% [15]. Similar results were achieved in the work of Nicolai Gajhede et. al. [29], who tackled the same drum classification problem. They employed mel-spectrograms and a convolutional NN (CNN) architecture, which achieved a 97% accuracy [29]. These findings demonstrate the feasibility of accurately classifying sounds using ML or NN techniques.

In another relevant aspect, the work of Wei-Han Hsu et. al. also becomes pertinent with their EDM sub-genre classification model [38]. Their project entailed the classification of 30 EDM sub-genres, utilizing a collection of 2,500 songs per sub-genre. Some of the genres included were 'dubstep', 'drum and bass', 'progressive house', among others. They extracted features such as Mel-spectrograms (MFCCs) and tempograms as the input data for a 'short-chunk CNN+Resnet' DL architecture. Their results showcased a 70% accuracy in EDM sub-genre classification [38]. It can even be assumed that some compositions included various harmonic bass sound designs, as previously mentioned. Despite their notable work, it focused on entire compositions rather than the isolated instances of samples used within the music. This might suggest the possibility of achieving a much higher accuracy rate if only the bass sound designs are considered, akin to the work of A. Chhabra and N. Gajhede's teams.

Strongly connected to the drum sample classification is the work of Kleanthis Avramidis et. al. [8]. However, their project focused on classifying polyphonic instrument audio samples, which are audio files containing multiple instruments. Instead of extracting features from the audio data, they directly inputted the raw audio data into a machine learning model. The architecture of their model employed a hybrid combination of convolutional and recurrent neural networks, utilizing a parallel 'CNN-BiGRU' structure. According to the article, the results

yielded *"competitive classification scores"* compared to state-of-the-art alternatives [8]. This work demonstrates the capability of utilizing raw audio data for audio classification.

4 Problem Description

The research unveiled a lack of projects focusing specifically on bass samples, although some small advancements have been noted in commercial products. Nevertheless, substantial evidence from other researchers supports the enhancement of a DL model capable of broadening the traditional bass category. However, existing articles predominantly emphasize the performance of these models rather than delving into their practical implementation. Hence, the primary problem addressed in this thesis is the implementation of a viable model that extends the functionality of an audio classification model with the modern EDM bass taxonomy using bass one-shots alongside an interactive application. This led to the following problem statement.

4.1 Problem statement

In the realm of music production, efficient sorting and classification of audio samples have been explored for various solutions. Traditional methods often rely on manual categorization or rudimentary keyword-based systems, which are time-consuming and prone to subjective biases. Addressing this gap, this thesis investigates the implementation of deep learning techniques for the automatic classification and sorting of bass samples. The primary objective is to design and develop a user-friendly graphical interface that integrates a trained deep learning model capable of accurately categorizing the modern bass taxonomy according to their sonic attributes.

5 System Requirements

In constructing the system outlined in Section 4, several requirements were established for the project. Primarily, an audio classification model is necessary to predict the category of bass samples and provide feedback. The model should possess the capability to process multiple samples either rapidly one after the other or simultaneously. Additionally, the model is expected to achieve a minimum accuracy rate of 90% for predictions, as determined by the related work of other researchers (Section 3).

In order to interact with the model, an application with a graphical user interface(GUI) should be developed to initiate the model functionality and exchange data in between the processes. More over, the application should be able to select, create and save references for two directories. One of them should represent the samples that should be sorted and the other one should represent the output of the sorting. The expected output of the sorted files should be a

directory containing sub directories for each category represented in the model. Which means there should be some functionality implemented to handle the creation of new sub directories within the output directory dependant on the output of the model prediction. The user interface should in return show the predicted output and be able to preview the audio as well as support correction of category placement. In addition, also functionality to open the destination folder upon completed predictions.

6 Design

6.1 Available data

In order to design the system described in Section 5, it is crucial to understand the available data for the project. Creating a model capable of classifying bass samples requires individual bass one-shots to feed the system, as opposed to genre classification where entire compositions are used [45]. The expected input for the machine learning model is also bass one-shots used within compositions, with the output being the label of the category.

However, no specific dataset of individual bass one-shots was found, necessitating the manual creation of the dataset. The chosen categories for the project were narrowed down to "808", "acid", "brass", "growl(FM)", "reese", "sub", and "slap". These categories were selected from personal resources such as sample packs and libraries, as well as presets from the Serum plugin. Additionally, some categories overlapped with an article describing 9 crucial synth basses in music production [36].

Each category represents a distinct sonic characteristic commonly found across various EDM genres [36]. The label selection aims to ensure that the model becomes adept at recognizing and classifying different types of bass sounds, ranging from the deep and resonant tones of the 808 to the aggressive and gritty textures of the growl(FM) bass. This variety should, in turn, enable the model to learn nuanced features inherent to each category, enhancing its ability to accurately classify bass sounds.

6.2 Dataset curation

When manually curating a dataset, it's crucial to consider its impact on the quality and effectiveness of the final model. Factors such as audio data quality, relevance, diversity, and balance must be carefully evaluated during selection [60]. Low data quality can affect recognition accuracy and cause the model to respond differently to similar data of higher quality [60]. Data should be relevant to its represented class, with well-defined boundaries enforced in the selection process [60]. However, it's essential to avoid excessive homogeneity within the dataset to capture the diverse characteristics of subcategories[60].

Put in to perspective, imagine recognizing different breeds of cats in images — specific breed-defining traits should be represented, but variations due to different camera angles or environments should also be considered. Audio mixing techniques like distortion and reverb, commonly used in bass production, may obscure defining characteristics and should be accounted for in the dataset. Diversity within class boundaries enhances the variability of scenarios encountered by the model, preventing bias and improving generalization to unseen data[60]. Furthermore, maintaining balance throughout the dataset is crucial to avoid bias and ensure optimal performance [60].

6.3 Data augmentation

Despite the availability of resources, the data remains relatively scarce compared to state-of-the-art models. However, this challenge can be addressed through data augmentation. Data augmentation involves generating additional data similar to the original dataset, effectively upscaling a small sample size to a larger one. Augmented data contributes to improving model performance by reducing overfitting, enhancing generalization, and increasing model robustness.

Popular augmentation techniques within the audio domain include polarity inversion, which generates a copy of the original signal with inverted phase. Pitch scaling is also commonly used to produce signals in different pitch regions, ensuring that the model learns how the signal sounds at various pitches. Another useful audio augmentation technique is time-stretching, which generates signal versions that are either shorter or longer. This technique strengthens recognition capabilities, particularly in cases where a sample has a fast or slow decay rate.

6.4 Learning methods

When working with a DL system categorizing data, two overall learning methods are available: *supervised* and *unsupervised learning*. These methods determine how the models are trained and the type of training data the algorithms use [33]. The biggest difference between the methods lies in the type of data used. Supervised learning utilizes labeled training data, whereas unsupervised learning does not [33]. In comparison, unsupervised learning figures out the nuances of the data on its own [52].

However, when categorizing data using supervised learning, it is known as classification, whereas the unsupervised equivalent is called clustering. Since the data are well-defined within their categories, they can be utilized as labels, making supervised learning the preferred approach [33].

6.5 Machine learning system

Given the expected behavior of the ML model to predict the subcategory of individual bass samples, the *predictive ML* approach should be considered. Predictive ML encompasses tasks where predictions are made, such as determining email credibility, making weather predictions, or voice recognition [32]. In contrast, *generative ML* handles tasks like generating output based on the user's intent, such as generating audio from a user prompt [32]. The advantage of the predictive approach is that the output can often be easily evaluated against reality [32].

However, the specific predictive system should be determined to ensure the correct output. The most popular predictive ML systems are *numerical* and *classification* [32]. The numerical system is often associated with tasks such as stock market predictions, disease progression, and risk of complications [32]. On the other hand, the classification system is linked to tasks such as image, object, or audio classification [32], making it the most optimal approach to categorize audio samples into specific categories.

Nevertheless, the output techniques should also be examined. The outputs available in the classification system are *binary*, *multi-class single-label*, and *multi-class multi-label* [32]. Binary output, for example, is used for classifying emails as spam or not. Multi-class single-label output is used for classifying an animal in an image, while the multi-label version is for classifying every animal in an image [32]. Dealing with samples of only one category suggests that the multi-class single-label output is the best fit for the project.

6.6 ML architecture

In machine learning, models are categorized into two types based on their depth: deep or shallow [53]. The depth signifies the number of layers in a neural network, defining the complexity of the model's architecture [53]. Shallow models are often characterized by their simplicity and small number of processing steps [53]. They typically have limited capacity to learn complex patterns from data. Some algorithms of shallow learning include linear regression, logistic regression, and k-nearest neighbors [53]. They are commonly employed when the patterns present in the data or relationships between features and outcomes are relatively simple [53].

On the other hand, deep learning is characterized by having multiple layers in its neural network to automatically learn intricate patterns and representations from data [53]. Algorithms in this category include convolutional neural networks (CNNs) and recurrent neural networks (RNNs). These models are capable of capturing complex relationships in data by progressively extracting hierarchical features [53]. Over the last couple of years, deep learning has demonstrated impressive performance in various fields such as image and speech

recognition, and natural language processing [53]. However, training an optimized deep learning model requires a much larger dataset compared to shallow learning, in order to solidify the algorithmic capabilities of the neural network [53]. Despite this challenge, techniques such as data augmentation can be utilized to address the issue [10].

Due to multiple instances of high-performing deep learning models using smaller datasets alongside data augmentation approaches, the deep learning architecture was further explored for the implementation [73; 10; 66].

6.7 DL algorithms

In running supervised deep learning, various deep learning algorithms are available, as mentioned in Section 6.6. Apart from convolutional and recurrent neural network (RNN) algorithms, there are Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU), both types of RNN. Additionally, there are the Transformer, Diffusion, and Variational Autoencoder algorithms. Each of these algorithms has demonstrated capabilities suitable for audio classification [20].

However, the Transformer algorithm has shown state-of-the-art performance in various natural language processing tasks, which often involve sequential data like audio. Noteworthy examples include Meta AI's Wav2vec 2.0 [9] and HuBERT [39] transformer models. This architecture efficiently captures long-range dependencies in audio data, essential for understanding contextual information in audio samples [31]. Moreover, the self-attention mechanism allows the model to focus on relevant parts of the audio sequence, facilitating better feature extraction and classification [14; 84]. The literature review of Siddique Latif et. al. shows the popularity and strong performances of the transformer architecture in speech recognition [57]. These findings highlight the effectiveness of the Transformer model and its potential for achieving high accuracy in audio classification tasks. For these reasons, the transformer was selected as the architecture for the system.

6.8 Fine-tuning

Audio classification is not a new concept in the machine learning space, as discussed in Section 3, and has been utilized for various tasks. This has led to the release of numerous models and datasets to the public via different platforms, which perform exceptionally well for audio classification [80; 81]. These high-performing models can be customized for other applications than their original version through a process known as *fine-tuning* [70]. Fine-tuning involves taking a pre-trained model and further training it on a new dataset to improve its performance or adapt it to a specific domain [70]. While model training can be time-consuming when done from scratch, fine-tuning leverages the existing knowledge encoded in the model, ultimately saving time and computational resources [70]. Technically, fine-tuning involves modifying the model's parameters

through new training iterations using the new data, enabling the model to specialize in the new problem while retaining the general knowledge it gained from the original training [70].

HuggingFace[2] is a platform where researchers can release their trained models or datasets for various machine learning concepts. These models can be freely fine-tuned and the platform has a dedicated category for audio [41] and audio classification [40]. One of their web courses delves into fine-tuning pre-trained models for music genre classification [42], which is relevant to this project's goals. Instead of classifying genres from compositions, it should approach to classify the category of individual bass samples.

The course suggests the use of different pre-trained audio classification models such as Wav2Vec 2.0 [9] and HuBERT [39]. However, it ultimately recommends DistilHuBERT, a distilled version of the HuBERT model, which has undergone *knowledge distillation*. Knowledge distillation involves transferring knowledge from a complex, larger, or collection of models to a simpler, smaller model, making it suitable for deployment in real-world scenarios [79]. The distilled version has resulted in a more lightweight size reduced to 75% and 73% faster training while retaining most of its performance [13]. For optimal use, this pre-trained model will be utilized for the project.

6.9 Validation

After fine-tuning the model, it is essential to validate its accuracy. This involves quantifying the relationship between the model's predictions and data it has not seen before. This measure determines if the model is *overfitting*, *underfitting*, or generalizing well. Overfitting means the model predicts well on training data but poorly on new data, while underfitting means it fails to capture patterns in the data. The goal is to achieve a balance for good generalization [62].

To validate the model's performance, cross-validation techniques can be used during the fine-tuning process [62]. The main techniques are the *train-test* split and *K-fold* [62]. Train-test split involves randomly dividing the dataset into training and validation parts, which works well with larger datasets [62]. K-fold is useful when data is scarce [62]. It involves splitting the dataset into multiple folds (usually 5-10). The model is trained on one fold and validated on the remaining folds, repeating this process for each fold [62]. The average prediction score across all folds is the model's accuracy.

With a medium-sized dataset using data augmentation, both techniques can be employed to compare their performances.

[2] huggingface.co

6.10 Inference API

Models hosted on the HuggingFace servers can be accessed through its Inference API [48], provided the model is an inference model. Upon reviewing the DistilHuBERT model, it is found to be compatible with the API [43]. This enables the exchange of HTTP requests, allowing users to receive prediction results from HuggingFace's servers. Essentially, the model's functionality can be achieved by running it on their servers. However, it's worth noting that the API is rate-limited, although a paid alternative named Inference Endpoints is available for production use [44]. Despite this, the API documentation indicates that if the free version is used and a client suddenly sends 10,000 requests, it may result in 505 errors [47]. While the rate limit suffices for this proof-of-concept, the production version would be necessary for a final product, depending on its usage. Additionally, it should be mentioned that when the free version of the Inference API is utilized, the models may become inactive after a certain period [46]. However, given the model's ability to interact with HTTP requests, a web-based application can be developed as the graphical user interface of the system.

6.11 User-interface

A variety of frameworks are available for creating graphical user interfaces. One of the application requirements is to utilize operating system-level functionalities to generate directories containing sorted sample categories, as depicted in Figure 7. Given that the model can run on the HuggingFace servers and be ac-

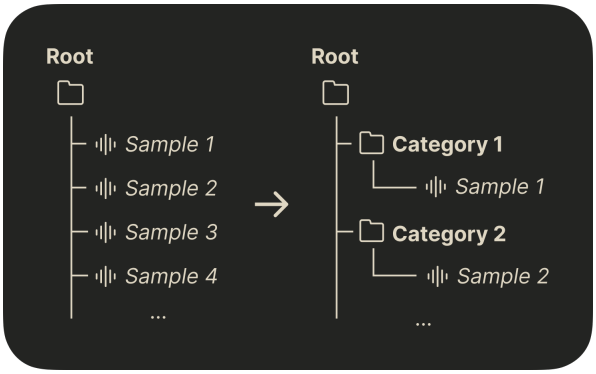


Figure 7: Expected folder sorting

cessed through the Inference API, a desktop web application appears to be the most suitable choice for the project. To utilize the `@huggingface/inference` `Node.js` framework, the web application framework should also be compatible with Node.js. Some of the available cross-platform frameworks and tools with native and Node.js capabilities include Electron, NW.js, Proton Native, and Tauri. Electron was selected as the web application framework due to its cross-

platform compatibility, easy learning curve and simple syntax. One drawback of the framework is that it runs on the Chromium web browser and is known to consume a lot of memory. However, it is still widely used for its compatibility with web tools, large community, and comprehensive documentation. Additionally, a mock-up for the user interface was designed in Figma[3], as shown in Figure 8 and additional snapshots in Appendix B.1.

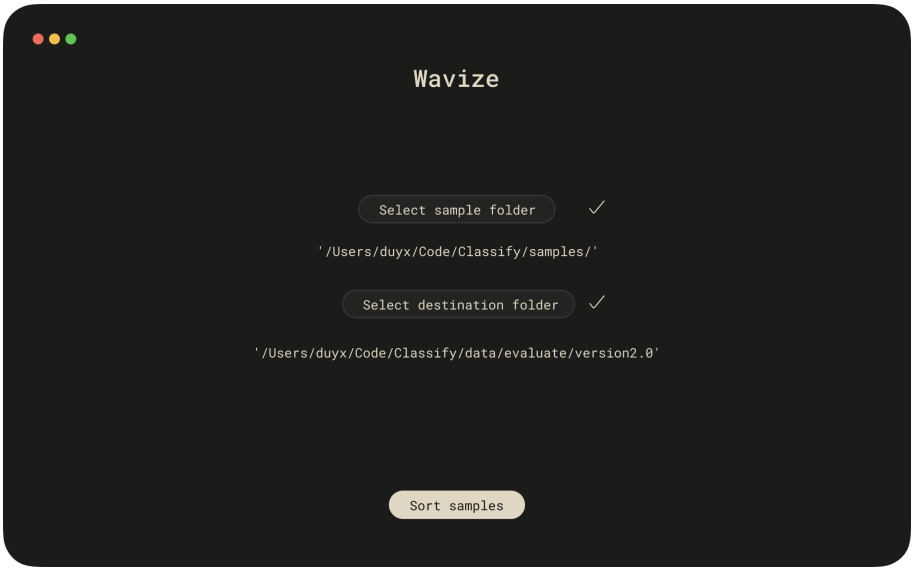


Figure 8: User Interface mock-up

6.12 Resources

To perform the fine-tuning of the DistilHuBERT model, several platforms have been made available for use. These include SDU's UCloud[4], AAU's ML workstation[5], and Google's Colab environment[6]. Each platform offers computing resources as well as instances of Jupyter notebooks or Visual Studio Code to execute the fine-tuning process.

6.13 Programming languages and frameworks

Python will be used for fine-tuning the HuggingFace model alongside the HuggingFace libraries as `transformers`, `evaluate`, `datasets` and `PyTorch's` machine learning library. The desktop web-application will be implemented using

[3] figma.com
[4] docs.cloud.sdu.dk
[5] aalborg-university.gitbook.io/machine-learning-workstation
[6] colab.google.com

web-based programming languages such as `HTML`, `CSS` and `JavaScript`. For data augmentation, the Python library `Librosa` will be utilised.

7 Implementation

7.1 Collecting data

To compile data for each category, a personal sample library and 'xfer records' synthesizer plugin, Serum, were utilized. The sample library consists of approximately 200,000 audio files, each carefully selected to match the characteristics of individual categories. Similarly, within the Serum plugin, specific presets were chosen and exported for each category. This process resulted in a dataset containing 30 samples per category totalling 210 samples, prepared for further processing as required. These samples included various versions of the represented classes. This ranged from clear signals to being subjected to various effect plugins as distortion, reverb, chorus or phase. Due to these sound effects the sounds have similar characteristics but the underlying sound design method remained consistent for each subcategory.

Additionally, an evaluation dataset was created using samples from the sample library and Splice, amounting to 138 samples. Importantly, none of the evaluation data overlapped with the training dataset. However, subjective bias is introduced with each sample being manually hand-picked as it was influenced by a personal judgement and preference. Possibly leading to inclusion or exclusion based on opinions rather than objective criteria, potentially resulting in a lack of diversity and undermining generalization.

7.2 Data augmentation

The data augmentation process was facilitated through a Python script designed to handle a root directory containing various subdirectories housing audio files in the .wav format. Organizing the data into respective folders simplified the task of maintaining category separation when utilizing the augmented data for training. When the augmentation script was initiated, it generated four output folders: one for overall output and three for temporary use, labeled "inverted," "pitch_shifted," and "time_stretched."

7.2.1 Polarity inversion

The first step of the data augmentation script involves polarity inverting the samples. It begins by reading the first category of the root directory and proceeds to the next once processing is complete. This is achieved using the `librosa.load()` function to load the sample, followed by multiplying the entire signal with -1 to invert its polarity. Subsequently, the polarity inverted sample is written to the inversion folder, while a copy of the original signal is

saved to the same folder. Before writing the inverted signal, an envelope is applied to reduce the silence in the audio. The mentioned copy serves to provide two versions of the same signal, thus increasing the size of the dataset.

7.2.2 Envelope/threshold

The previous envelope function was implemented by taking the signal, calculating the rolling maximum of its absolute values, comparing each rolling maximum value to a threshold, and creating a binary mask indicating whether each rolling maximum is above the threshold or not. Finally, it returns this mask along with the calculated rolling maximum values. The mask can then be applied to the signal, effectively removing excess silence. This method can be reviewed on Appendix C.1

7.2.3 Pitch shifting

The next step in the processing involved pitch shifting. Different pitches of the audio were needed to account for classifying data with varying pitch. An array of integers was created to select the amount of semitones to pitch shift the original and inverted signal. The lowest semitone selection was -6, and the highest was 3, meaning the signal could be shifted down by 6 semitones and up by 3 semitones. As the samples are already low pitched, extreme pitch shifts could distort the signal. Pitching them too high would remove the low-frequency information.

The script starts by reading the samples one by one from the previously mentioned folder, which contains copies of the original signal and the polarity inverted one. Then, it iterates over the amount of semitones being shifted. The pitch shift is applied using the librosa effects library in Python with the `pitch_shift()` function, using the signal, sample rate, and semitones as parameters, and the output is the pitch-shifted audio. The envelope is applied once more to ensure there is not any additional silence present in the signal. This process continues until all samples have been processed with each semitone shift. Finally, the pitch_shifted folder is filled with the shifted versions of the original and inverted signal.

7.2.4 Time stretching

Following the pitch shifting, the script proceeded with time stretching of the signal. An array was employed to store float values representing the percentage of time stretching. Extreme time stretching, either too much or too low, could corrupt the signal by losing valuable information. Hence, the lowest value chosen was 80%, indicating a shorter signal, while the highest was 120%, indicating a longer signal. In each iteration, 5% was added (80%, 85%, ..., 120%).

The pitch_shifted folder served as input, reading the samples one by one and iterating over the array of time stretching values. Time stretching was applied

using the same effects library in librosa, specifically the `time_stretch()` function, with the signal and rate as parameters. Additionally, the envelope reducing silence was applied again for safety. All of the samples were then written to the time_stretched folder.

7.2.5 Data structure

To streamline the data organization and reduce clutter, the contents of the time_stretched folder, which contained all the processed data, were copied to a new directory. Subsequently, all the previous samples were deleted from the output folders. This step ensured that only the processed and relevant data remained in the output directory, minimizing confusion and improving data management.

7.2.6 Signal length reduction

As a final processing step, signals exceeding 1 second in duration were trimmed since the initial seconds of bass samples are highly defining of their type. This was accomplished by utilizing the final output directory from the data management step, listing the subdirectories, and iterating over the samples in each subfolder. The sample rate was then used to determine if the sample exceeded 1 second, and if so, it was trimmed accordingly.

This concludes the data augmentation process for this project. Additionally, small quality-of-life components were added to provide a visual track of the performance during processing, and a smaller version where only one category would be augmented was also created.

7.3 Dataset preparation

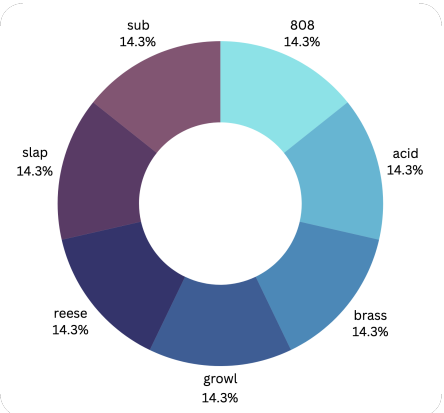


Figure 9: Balance percentages of the dataset

Through data augmentation, a dataset consisting of 34,020 audio samples was generated. Each class contained 4.860 samples as seen on Figure 9. Leveraging the HuggingFace platform in the project, the dataset could be uploaded to their platform, facilitating easier tracking of different versions used in training. HuggingFace offers a library called `datasets`, which was utilized for creating datasets in the HuggingFace standard.

The `load_dataset()` method is capable of loading an online or local dataset. In this case, the local augmented dataset was loaded by specifying the path of the augmented data directory. This function utilizes the names of the folders and prepares them as labels for each sample, generating a dictionary structure as depicted in Listing 1.

```
DatasetDict({
    train: Dataset({
        features: ['audio', 'label'],
        num_rows: 38232
    })
})
```

Listing 1: Dataset structure

To evaluate the model performance, the train-test split cross-validation technique was implemented. The library includes a method named `train_test_split()`, which yields a dictionary comprising two random train and test subsets with a split of 90% and 10%, respectively. The resulting dictionary is illustrated in Listing 2. The 'test' dataset provide evaluation during training. However, as data augmentation was used to upscale the data, a lot of similar audio files is present in the test split which can skew the accuracy rate. Regardless, it still serves as valuable information while undergoing training. It is because of this reason, the evaluation dataset mentioned in 7.1 were created can be utilised as completely unseen data.

```
DatasetDict({
    train: Dataset({
        features: ['audio', 'label'],
        num_rows: 34408
    })
    test: Dataset({
        features: ['audio', 'label'],
        num_rows: 3824
    })
})
```

Listing 2: Dataset with training split

The model expects input data to be encoded in a format that it can process, which is facilitated by the `AutoFeatureExtractor` class in the `transformers` library. This class automatically selects the appropriate feature extractor for the DistilHuBERT model using the `.from_pretrained()` method, aiding subsequent learning and generalization across various process domains. Additionally, the model requires data to have a sampling rate of 16,000 Hz, a condition handled in the data augmentation script, ensuring all samples are in the correct

sampling rate.

HuBERT-like models anticipate a float array corresponding to the digital representation in a one-dimensional aspect, a transformation also managed in the augmentation script. While the model expects audio in mono instead of stereo format, meaning one channel instead of two, it's crucial for all data to fall within the same dynamic range to ensure optimal performance. This uniformity enhances stability and convergence during training by ensuring a similar range of activation's and gradients for all samples. This is achieved through feature scaling or normalization of audio data, scaling each sample to zero mean and unit variance, a process directly performed by the feature extractor.

Finally, the feature extractor returns a dictionary containing an input value and attention mask array, utilized for input and batch processing in the HuBERT model. Additionally, the DistilHuBERT model has a length threshold of 30 seconds. Since the data augmentation reduces all samples to 1 second, this threshold constraint is not an issue.

Since the labels are mapped to integers, they must correspond to the actual bass categories present in the dataset, allowing use in any downstream application with the `int2str()` method. This was accomplished by fetching the labels of the original classifications and applying them to the respective integer, resulting in the encoded dataset.

The final step in dataset creation involved uploading the dataset to the HuggingFace hub using the code depicted in Listing 3.

```
1    dataset_encoded.push_to_hub("profile-name/dataset-name")
```

Listing 3: Uploading the data to HuggingFace

7.4 Fine-tuning

The uploaded dataset was loaded using the `load_dataset()` function, which included the predefined train-test set. In working with the HuggingFace framework, the `Trainer` class from `Transformers` is essential as it handles the most common training scenarios. This is achieved by loading the DistilHuBERT model with the `AutoModelForAudioClassification` class, automatically adding the appropriate classification header for the framework. Additionally, the labels were retrieved from the dataset and used as an argument.

Next, the training arguments was defined. These include the batch size, gradient accumulation steps, amount of training epochs, and the learning rate. Finally, the model was initiated and trained using the `.train()` method. The setup of training arguments can be seen reviewed in C.2.

Additionally, to compare the performance of different cross-validations schemes,

the K-fold technique was implemented for the fine-tuning process. The KFold class from `sklearn.model_selection` was used to create 7 splits and each fold was used for performance validation.

7.5 Evaluation script

To evaluate the fine-tuned models on unseen data outside the data augmented training set, two Python scripts were created to compare the accuracy of each model and generate confusion matrices and ROC-curves. These scripts utilized the evaluation dataset mentioned in 7.1.

7.5.1 Single evaluation script

The first script evaluated a specific model loaded from the `pipeline` library and created a log file containing the prediction for each sample in the evaluation dataset. It also generated a confusion matrix based on the true and predicted labels. This was implemented by iterating over each subdirectory in the evaluation dataset. The files were then read one by one using the `torchaudio.load()` method, and the selected model was employed to classify the audio file. The label with a score closest to 1 was extracted, and if it matched the subdirectory name, it indicated a correct prediction by the model. The number of correct predictions was stored in a variable representing the total number of correct guesses. To calculate the accuracy of each label individually, another variable was maintained to track the number of correct guesses for each subdirectory. The true and predicted labels were saved to arrays for later use in generating the confusion matrix and ROC-curve.

Each prediction was written to the log file, and in the end, a comprehensive accuracy report was generated based on the recorded counts. This report included the accuracy percentage for each label and the overall accuracy. Finally, the confusion matrix was generated using the `confusion_matrix()` function from the `sklearn.metrics` library, using the true and predicted labels.

7.5.2 Comparison script

During the fine-tuning process, additional models were created. To facilitate quick comparison among these models, a simple script was developed to generate only the overall accuracy score. Each model reference was loaded and iterated over, replicating the process used in section 7.5.1.

7.6 Hyper parameter adjustments

In the process of fine-tuning, additional changes to the hyper parameters were manually determined by using the evaluation scripts mentioned in section 7.5. Other automated tuning methods could have been utilized, such as Bayesian optimization, Random search, or Grid search [75]. These methods, if set up

properly, can alleviate the tediousness of manual tuning [75]. Despite this, the manual experiments quickly exhibited fine results. The first couple of training's were set at around 10 epochs with various batch sizes, gradient accumulation steps, and learning rates. It revealed an unstable training and validation loss throughout the epochs but maintained a high training accuracy at around 100%. However, when tested on the unseen evaluation dataset, the performance was deemed mediocre, scoring around a 60-80% accuracy.

After additional parameter adjustments and increasing the epochs the model performance became worse. Then, epochs was adjusted to a lower value and enhancements in the model evaluation started to generalize better. It was lowered at the epoch steps right before an increase in training and validation loss were occurring. Moreover, as epochs were lowered, new experiments were made by adjusting, gradient accumulation, batch size and learning rate. Increasing the batch size revealed better generalization to the unseen data at around 80-87% accuracy.

At last, by increasing the gradient accumulation, the model finally crossed the 90% mark as being the minimum requirement for the model accuracy. This was true to both the train-test and K-Fold cross-validation. This performance was achieved with around 3-4 epoch steps, 8-16 batch size, and 3-5 gradient accumulation and a learning rate of 5e-05. The validation loss started at 0.0492 and ended on 0.0024 and training loss from 0.043 to 0.001. Additionally the model training accuracy was 99% as seen on the training results on Table 1.

Training loss	Epoch	Step	Validation Loss	Accuracy
0.0433	1.0	382	0.0492	0.9877
0.0022	2.0	765	0.0061	0.9982
0.0013	3.0	1146	0.0024	0.9994

Table 1: Training result for the highest performing train-test model

7.7 Inference API request

The HuggingFace platform provides a simple JavaScript HTTP POST request proposition [49] to be used for communicating with uploaded models shown on listing 4.

```
1     // request function
2     async function query(filename) {
3         const data = fs.readFileSync(filename);
4         const response = await fetch(
5             "api-inference-url",
6             {
```

```
                headers: { Authorization: "Bearer Token" },
                method: "POST",
                body: data,
            }
        );
        const result = await response.json();
        return result;
    }

    // usage
    query("sample.wav").then((response) => {
        console.log(JSON.stringify(response));
    });
```

Listing 4: Inference API example

Since the model is tagged with `audio-classification` the API accepts audio data by default in various formats. An example of the POST request made to one of the models is shown on the following JSON data on listing 5.

```
[
  { score: 1, label: 'slap' },
  { score: 1.4187427274658937e-12, label: 'growl' },
  { score: 6.623589480711511e-13, label: 'acid' },
  { score: 6.345010237704396e-13, label: 'brass' },
  { score: 2.0200807443904872e-13, label: '808' }
]
```

Listing 5: JSON result

With this end-point available the only task left is to connect the InferenceAPI with the Electron framework and handle the networking communication as well as updating the GUI to show when the application is waiting for a response.

7.8 Electron app

The Electron framework comprises a `main` and `renderer` process [21]. The `main` process serves as a Node.js backend process responsible for running the Electron instance and managing its functionalities. This process initializes the application window and manages system-level events and interactions [21]. Subsequently, it can be employed to load existing directories and create new ones for the output of classifications. Conversely, the `renderer` process is tasked with rendering the user interface of the application using various web technologies. It facilitates displaying the results of classifications and interacting with the model. The actual GUI can be implemented using standard `HTML` and `CSS` code, utilizing the `renderer` script to alter elements during runtime.
The data flow of an Electron application is maintained by Inter-Process Communication (IPC) [24], which is implemented by subscriber functions that listen whenever a specific keyword is sent to the IPC process. These channels, known respectively as `ipcMain` and `ipcRenderer`, can exchange data within the arguments of the communication methods `.on()` and `.send()` [21].

7.8.1 main.js process

Electron is shipped with a class called `dialog`, which opens the file explorer of the operating system [22]. The parameters of the `dialog` class can be modified to only include directories with file formats of `.wav` and `.mp3`. Additionally, by passing the `'openDirectory'` keyword, the dialog window is informed to return the chosen directory name. The directories are then saved as variables that can be used to load the samples within the directory and create new directories for the categories in the target directory.

To create new directories, move data from one directory to another, and load filenames, the `node:fs` (filesystem) class in Node.js can be utilized [68]. It is important to mention that Electron embeds Node.js into its binary, making the default node modules available [25]. The selected directories could be used in conjunction with `fs.readdirSync()` to retrieve the file names of the samples [68]. However, the `fs.readdirSync()` only returns the actual file name and in order to use the sample for the InferenceAPI request, the entire file path is required. To manage the sample paths the `node:path` class can be put to use [69]. The `path.join()` method can be used to join the selected sample folder with the file names which finally can be passed into the InferenceAPI request. All the paths were then connected by passing the root path with the file name and collected in an array.

Now, with the sample paths concatenated, the request mentioned in section 7.7 could be implemented. The API is only able to handle one sample at a time which means a request for each individual sample is required. This problem can be handled by using the JavaScript `Promise` object which represents the completion of an asynchronous operation [64]. First of all, the request was implemented with the proper headers in an `async` function called `apiCall(filePath)` which accepts a single file path. The `fs.readFileSync()` method was then used inside the function to pass the audio data to the `data` argument of the request. Another `async` function was then implemented called `runMultipleApiCalls()` to map each file path into a `Promise` object of the `apiCall()` function. This essentially creates a queue of requests that are able to be executed by running the `Promise.all(promises)` method which is shown on listing 6.

```
1     async function runMultipleApiCalls(filePaths, callback) {
2         const promises = filePaths.map(filePath => apiCall(filePath
              ));
3         try {
4             const results = await Promise.all(promises);
5             // Call the callback function with results
6             callback(null, results);
7         } catch (error) {
8             // Call the callback function with error if any
9             callback(error);
10        }
11    }
```

Listing 6: Request queue function

Dependent on whether any errors occurred during the request process, the method returns the JSON data shown on listing 5 in section 7.7.

The results of the response were collected in an array, which was used to unpack the JSON data. Since the received data was a dictionary of the predicted score and the respective label, the label and score closest to 1 had to be extracted. An extraction function was implemented and can be reviewed in appendix C.3. The function returns the predicted label, which could be used to create a new sub-directory in the target folder with the same name (see appendices C.4 and C.5). These aggregated functions can then be run for each response, copying the samples to each category sub-folder that matches the label.

The `shell` class in Electron was used to implement opening the containing folder after the samples had been sorted in a file explorer instance [27]. The `.openPath()` function was used and passed the `'destinationFolder'` as the argument.

7.8.2 renderer.js process

A simple HTML and CSS page was created to interact with the system, containing elements such as labels and buttons. Additionally, it provided feedback whenever something was executed in the program. Each button was assigned an `eventListener()` in the `renderer` script, which waits for a click event to occur [65]. All interactions are connected to either a `ipcRenderer .send()` or `.invoke()` method, which notifies the `main` process to execute its back-end functionality. `.send()` is specifically used for one-way communication [26], whereas `.invoke()` is used for two-way communication [23]. Thus, invoking a process allows data to be exchanged between the `renderer` and `main` processes within the same function. Additionally, a loading animation screen was added to indicate that the system was performing a time-consuming task

8 Evaluation

To conduct the evaluation various elements of the system can be assessed.

8.1 Baseline testing

Given that Sononym offers a trial period, ADSR Sample Manager is free, and Ableton Live 12 is owned, a baseline test can be conducted. This test will involve deploying the evaluation dataset into each system to assess their performance and compare them with the implemented system.

8.2 Model metrics evaluation

Analyzing a DL model's metrics reveals its prediction performance in certain areas [11]. In classification models, common metrics include accuracy, precision,

recall, F1 score, AUC-ROC and the confusion matrix [11]. The confusion matrix shows actual versus predicted values. Accuracy measures overall prediction correctness. Precision indicates how often the model is correct when predicting a class [3]. Recall indicates whether the model can find all objects of the class [3]. The F1 score, or F-measure, is the harmonic mean of precision and recall [11]. The AUC score indicate how well a model can produce relative scores to discriminate between positive or negative instances across all classification thresholds [78].

8.3 Model performance

It is important that prediction requests sent to the HuggingFace API do not take too long. The prediction latency of the DL model can be evaluated by describing the distribution of prediction times [50]. This latency can be used to calculate desired percentiles (e.g., p50, p90, p95, p99) [50]. These percentiles define the latency experienced by users [50]. The p50 percentile is the median measure of typical user latency, p90 represents the majority, p95 covers almost all users, and p99 helps to understand the tail latency, ensuring that the slowest 1% of requests are within acceptable limits [50]. Additional experiments can simulate multiple requests as the application would make, such as 50, 75, and 100 prediction requests sent to the model. Since the number of unsorted bass samples can vary, extreme cases should be considered. The same measures are used but account for additional requests. The latency can be timed by starting a timer before a prediction is made and stopping it when the model is finished.

8.4 Interview & usability testing

To understand how real users interact with the system, a usability test will be conducted. If the testers have in-depth experience with a DAW, additional interview questions will be asked regarding music composition. The test will begin with background questions, followed by usability tasks during which participants will answer questions after certain tasks. Post-test questions will be asked afterward. If the user is knowledgeable about DAWs and music composition, the interview will conclude with creative-related questions.

8.5 Listening test

Various online forums are highly passionate about music composition, with some focusing on specific areas or sub-genres. These forums can be leveraged to evaluate the EDM community's ability to recognize modern bass sounds. This will be done through an online listening test questionnaire. Participants will listen to 7 audio files representing different bass sounds and select the correct category from a set of multiple-choice options. The questionnaire was posted to two Reddit communities, */r/musicproduction* and */r/edmproduction*, which have 407,000 and 751,000 members, respectively.

9 Results

9.1 Framework comparison

9.1.1 Sononym

Sononym is both a plugin and a standalone application, which means it can be utilized within and outside of a DAW. The evaluation dataset was integrated into the Sononym sample manager. During the application's usage, it quickly became apparent that it categorizes nearly all low-frequency samples into a broad category labeled "Bass & Low Keys," as illustrated in Figure 10a.

(a) Bass categories in the Sononym App

(b) Sononym similarity accuracy with growl bass

Figure 10: Sononym

A few of the brass samples were categorized as "Stabs & Orchs," referring to orchestral sounds. However, all samples were clearly synthesized. Additionally, the brass sounds were categorized as "Explosion & Shots." The acid category was very vague, with some samples being categorized as drum sounds like "Toms" or "Snare," or simply not recognized in any category. The Sononym

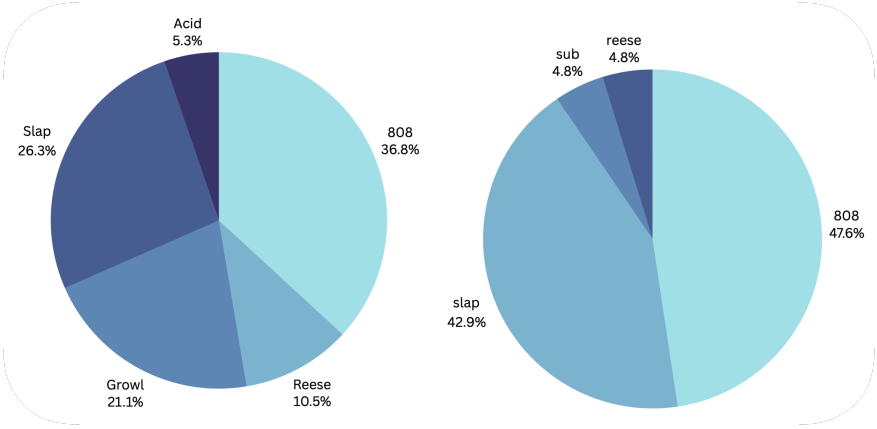


Figure 11: Sononym similarity accuracy of growl (left) and 808 bass (right)

app features a "search-by-similarity" function. A growl and an 808 sample were

selected, and a pie chart of the first 20 recommended files was created, as shown in Figure 11. The left side shows the growl bass predictions, and the right side shows the 808 predictions. The app did not categorize them but identified them as similar sounding and displayed them in random order. The similarity accuracy for the growl ranged from 49-57%, as seen in Figure 10b, while the 808 sample showed a higher similarity range of 61-69%.

9.1.2 ADSR Sample Manager

The ADSR Sample Manager, like Sononym, functions as both a plugin and a standalone application. Unlike Sononym, it employs a different approach to keyword tagging, eschewing AI in favor of utilizing file and folder names. For instance, a file named "loop_reese_bass.wav" would be recognized as containing the tags 'loop', 'reese', and 'bass', categorizing it accordingly. However, generic names like "sample_1.wav" do not yield any category registration. Consequently, all evaluation dataset names were altered to remove any categorical associations, preventing automatic sorting. Nevertheless, the ADSR Sample Manager successfully identified samples as loops or one-shots.

When using folder names, it recognized categories such as '808', 'Acid', 'Brass', and 'Reese', sometimes placing them under an overarching 'bass' category. However, if all samples are placed in a 'bass' directory, they will all receive the 'bass' tag indiscriminately.

9.1.3 Ableton Live 12

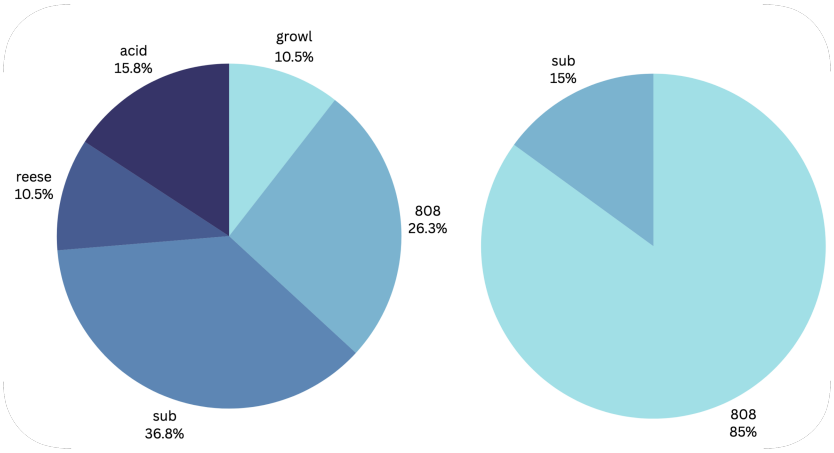


Figure 12: Ableton similarity accuracy with growl (left) and 808 bass (right)

The new similarity search feature introduced in Ableton Live 12, as cited by [2], *"calculates a similarity score between sounds based on spectral and temporal*

characteristics, timbre, pitch, and other attributes". This functionality is accessible only when the DAW is launched. However, it lacks the option to exclude specific content, meaning the entire DAW library is utilized. Nonetheless, users can narrow down the search to 'user-added' and 'samples' categories. A similar approach to Sononym was adopted, displaying the first 20 similar files of a growl (left) and 808 (right) bass in Figure 12.

9.1.4 Model comparison

The functionality of the implemented model differs slightly from the similarity search in other products. Predictions can be retrieved by deploying the evaluation dataset to the model. Once again, the 808 and growl bass were selected for comparison against the products. The predictions are shown in Figure 13.

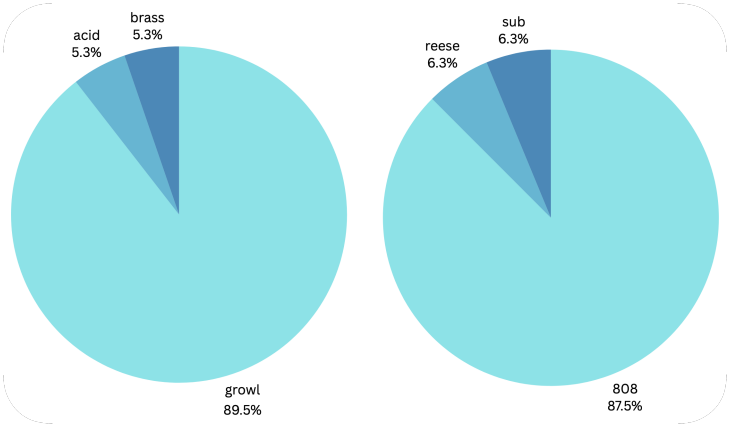


Figure 13: Model predictions of growl (left) and 808 bass (right)

9.2 Model metrics

9.2.1 Accuracy

808	Acid	Brass	Growl	Reese	Slap	Sub	Overall
87.5%	77.78%	100%	89.47%	100%	95.24%	93.33%	91.3%
28/32	14/18	16/16	17/19	17/17	20/21	14/15	128/138

Table 2: Accuracy table

The accuracy was measured based on predictions made on the unseen dataset, as shown in Table 2. It indicates that the best-performing labels are Brass and Reese, achieving a 100% accuracy, while the lowest accuracy is observed for the Acid label, with 77.78%. The overall accuracy across all labels was calculated to be 91.3%.

9.2.2 Confusion matrix

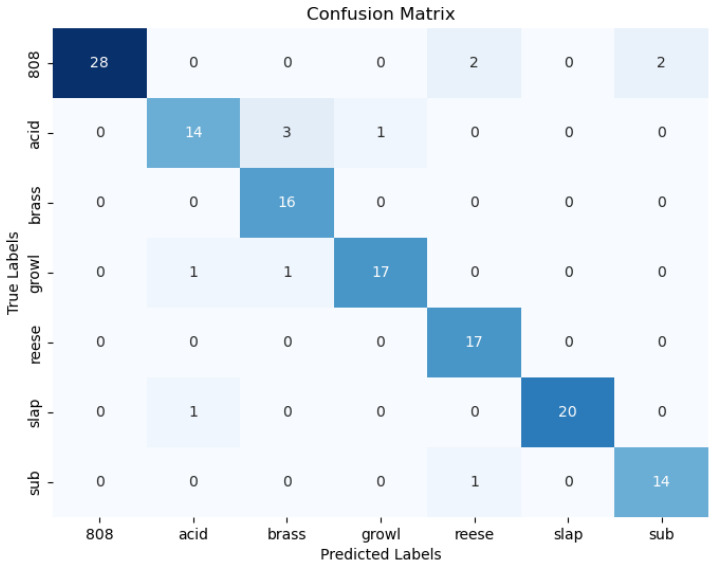


Figure 14: Confusion Matrix

Since accuracy solely reflects correct predictions, the confusion matrix becomes instrumental in analyzing the incorrect ones. Illustrated in Figure 14, the majority of high values on the diagonal line signify correct predictions, while numbers off the curve denote incorrect guesses. Evaluating the model's recognition performance involves deriving insights from the classes the model deems similar, shedding light on labels that require improvement.

9.2.3 Precision, recall and F1-scores

A precision, recall, and F1-score measurement table is presented in Figure 3. A high precision value indicates that the class labels predicted as positive are mostly correct [11]. Similarly, a high recall value indicates that most of the actual positives are correctly identified by the model [11]. Lastly, a high F1-score signifies balanced performance, exhibiting concurrent high precision and high recall values.

	precision	recall	f1-score	support
808	1.00	0.88	0.93	32
acid	0.88	0.78	0.82	18
brass	0.80	1.00	0.89	16
growl	0.94	0.89	0.92	19
reese	0.85	1.00	0.92	17
slap	1.00	0.95	0.98	21
sub	0.88	0.93	0.90	15
accuracy	0.91	0.91	0.91	138
macro avg	0.91	0.92	0.91	138
weighted avg	0.92	0.91	0.91	138

Table 3: Precision, recall, and f1-scores

9.2.4 AUC-ROC

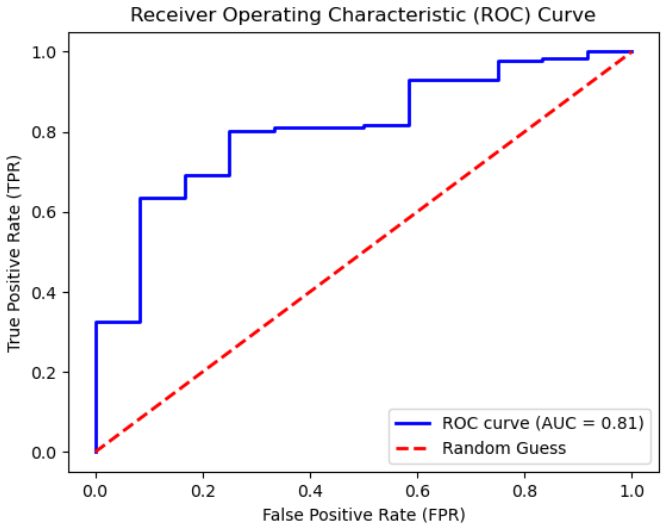


Figure 15: ROC curve and AUC score

The AUC-ROC (Area Under the Receiver Operating Characteristic Curve) is a crucial metric for evaluating the performance of classification models, particularly in terms of their ability to distinguish between classes [78]. A perfect model would achieve an AUC of 1, indicating perfect classification of all positive and negative instances. Conversely, a model performing no better than random guessing would yield an AUC of 0.5 [78]. In the ROC curve illustrated in Figure 15, the AUC is 0.81.

9.3 Prediction Latency

Table 4 presents the prediction percentiles measured in milliseconds (ms) by the implemented model. It indicates that the typical user will experience latencies lower than 38.27 ms, suggesting that the model performs well for the vast majority of use cases. The 90th and 95th percentiles of predictions are under 77.67 and 80.42 ms, respectively, indicating that the model maintains good performance even under higher loads. Finally, the 99th percentile of predictions is under 124.20 ms, with the remaining 1% possibly experiencing slower response times.

requests	p50(ms)	p90(ms)	p95(ms)	p99(ms)
1	38.27	77.67	80.42	124.20
50	4026.14	4307.55	4360.1	4525.91
75	5998.46	6328.57	6438.35	6578.97
100	8005.75	8397.23	8523.02	8743.86

Table 4: Latency simulation of 50, 75 and 100 requests

In contrast to individual requests, additional latency simulations of multiple requests are shown in Table 4. For instance, 50 sample predictions may take up to 4.5 seconds or more but should perform at around 4 seconds for most use cases. The extreme case at 100 reveals a time of 8.7 seconds. When tested on 50 samples using the inference API, it took 2.33 seconds to complete, encompassing, file load, prediction latency, and networking latency. This suggests that the HuggingFace servers may accelerate predictions, as it performs even better than the local equivalent. Additionally, processing 100 samples using the InferenceAPI took 6.17 seconds, as shown in Table 5. Scripts for percentile calculations can be reviewed in Appendices C.6 and C.7.

requests	latency
50	2330 ms
100	6170 ms

Table 5: Inference API latency from the web-app

9.4 Usability test

The usability test involved three participants, whose details are available in Appendix D. Participants had musical experience ranging from 7 to 16 years, with

two having 4 to 14 years of DAW experience. Their musical tastes encompassed genres like hip hop, metal, R&B, pop, rock, and orchestral. Only one participant was familiar with sample managers and had experience with the XLN's XO sample manager plugin mentioned in Section 1, finding it too complicated.

When asked to define a bass sound, all participants described it as characterized by low tones on the frequency spectrum, typically ranging from 20 to 500 Hz, and noted that various instruments could produce these low notes. They were also asked about bass categories they knew, with participants mentioning categories like 808, growl, wide synth EDM, and analog bass, among others.

After the personal background interview, participants were introduced to the application interface and asked to describe what they saw. All participants noted the application title and three buttons and correctly identified their functions. Usability tasks were conducted next, with all participants completing them without issues. However, two participants initially did not realize that the dialog box was scrollable, causing confusion until they discovered its scrollability. The third participant recognized it immediately.

The participant with no DAW experience found the categories confusing but understood that some sorting had occurred. Opening the output folder revealed organized samples, which surprised some participants, eliciting positive responses.

Post-test questions revealed that while some struggled with the scroll bar, they found the application simple and fulfilling its promises. Suggestions for improvement included enabling the sort button only after selecting both directories and providing more on-screen text like a brief tutorial.

Participants with DAW experience found the application easy to use, noting its clarity in categorizing bass samples. However, the participant with no DAW experience found it less intuitive but still easy to use.

Suggestions for improvement included disabling the sort button until both directories are chosen, providing more on-screen text, enabling the selection and sorting of entire sample libraries, and creating a more breathable interface.

In response to a creative-oriented question, participants with DAW experience mentioned using Splice for gathering samples. They described methods for selecting bass samples for use in music production, such as downloading samples and using Ableton sample explorer or directly dragging and dropping samples from Splice into their DAWs.

9.5 Questionnaire

The questionnaire received 21 answers. The first question revealed the wide span of musical backgrounds of the participants seen on Appendix E.1. The results of the 7 listening tests can be seen on Appendix E.2 and the overall accuracy of the participants can be seen on Table 6.

808	Acid	Brass	Growl	Reese	Slap	Sub
86.4%	61.9%	100%	85.7%	76.2%	90.5%	85.7%

Table 6: Participants listening test

10 Discussion

One of the primary goals of this research was to develop and evaluate a machine learning model for classifying different types of musical sounds, including '808', 'acid', 'brass', 'growl', 'reese', 'slap', and 'sub'. Specifically, the study sought to determine the effectiveness of the model in accurately identifying these sounds and to compare its performance against existing sample manager products.

The results demonstrate that the model trained using the train-test split cross-validation achieved high overall performance, with an accuracy of 91.3%. Meanwhile, the model evaluated with k-fold cross-validation also performed well, reaching an accuracy of 90.58%. Notably, the precision and recall metrics were particularly strong for the 'slap' and 'brass' categories, with precision scores of 1.00 and 0.80, respectively, and recall scores of 0.95 and 1.00. The F1-scores for these categories were also high, indicating a balanced performance in terms of precision and recall. The high precision and recall for the 'slap' class suggest that the model is highly effective at identifying this type of sound, likely due to its distinct auditory features. In contrast, while the 'acid' category showed a lower recall of 0.78, the precision remained relatively high at 0.88, indicating that the model was more conservative in its predictions, potentially avoiding false positives but missing some true positives.

However, compared to the work of A. Chhabra and N. Gajhede et. al. mentioned in Section 3, the overall accuracy does not perform as well. This might attributed to the training data was selected with a personal bias. It can be assumed that with access to more resources, the generalization can improve even further. Using data augmentation may have enhanced the overall generalization, as the original dataset only consisted of 210 audio samples, but it could also have affected the quality of some samples. Conclusively, affecting its generalization. Since the hyperparameters were manually configured for this project, it would be valuable to explore the results of using automated techniques such as Bayesian optimization, Random search, or Grid search. Additionally, only some models in the HuggingFace environment support direct regularization techniques as

dropout-layers. However, it should be possible to add custom configurations with non-supported models like the DistilHuBERT model but would require further exploration. This could effectively also enhance the generalization of the model.

One significant aspect of the model's performance is the AUC score of 0.81. An AUC score of 0.81 indicates that the model has a good level of discrimination, effectively distinguishing between positive and negative classes approximately 81% of the time [78]. However, this score should be improved for a commercial application.

The confusion matrix presented provides a detailed breakdown of the model's classification performance across different sound categories. This matrix helps to visualize the true positives, false positives, false negatives, and the overall accuracy for each class. Overall, the confusion matrix indicates a robust performance of the model across most categories, with particularly strong results for '808', 'brass', 'growl', 'reese', and 'slap'. The minor misclassifications suggest areas for potential improvement, particularly in distinguishing 'acid' sounds from 'brass' and 'growl'. These insights align with the high AUC score of 0.81, confirming the model's effectiveness in distinguishing between the various classes while also highlighting specific areas for enhancement.

The latency results indicate a well-performing model with generally fast response times, which is promising for real-time applications. Response time is highly dependent on the number of samples the user wants to sort. For example, it took 6.17 seconds using the Inference API to sort 100 samples, but users could likely surpass this amount. Sorting many samples will eventually exceed 10 seconds. As researchers have found, users lose interest after 10 seconds [67], so it might be beneficial to include a progress bar for additional user-friendliness when processing takes longer. Despite this, a study in the realm of AI-based task management tools suggests that users appreciate the enhanced productivity and efficiency these tools offer, which can justify longer wait times [71]. Users would likely start the sorting algorithm and gladly wait while doing other things in the meantime, especially when the AI adds significant value or enhances the user experience.

Comparing the quality of commercial products revealed that their automatic organization features failed to subcategorise the provided bass samples independently. Manual tagging, sorting, and labeling are required to sort the bass samples correctly. With this in mind, the implemented system far outshines the competitors in terms of autonomous subclass organization. This superiority was demonstrated by the web app's ability to categorize every sample with few misclassifications. However, the "search-by-similarity" function available in Ableton and Sononym provided some comparable results. It should be noted that this function is different from automated organization since it only temporarily recommends similar-sounding samples. Nevertheless, it still

provides insights into whether the function can recognize bass samples in the same subcategory. Ableton showed similar performance in recommending samples in the 808 category, with the implemented system achieving around 85% accuracy. Sononym, however, only achieved an accuracy of close to 50%. The differences between the systems were particularly evident with the growl bass sample. Ableton and Sononym recommended similar samples with accuracies of 10% and 21.1%, respectively, while the implemented system scored 89.5% for the growl bass. This demonstrates that even the "search-by-similarity" function is outperformed by the implemented system.

As additional research for the project, empirical evidence was gathered to assess how well the modern bass taxonomy is recognized by the EDM community. The data collected through the questionnaire indicates a significant proficiency among EDM musicians in distinguishing between different sound categories. Participants consistently identified and categorized sounds with a high degree of accuracy. Specifically, brass received a 100% score, indicating that this sound is more recognizable due to its association with an acoustic instrument, despite being synthesized. The 808 and sub categories also scored well, as these are widely known synthesized basses. The growl bass showed a similar result, as its distinct sound sets it apart from others. The sound recognized the least was the acid bass, suggesting that this sound design technique is not as widely known in the community. Overall, this research demonstrates that musicians in the EDM community possess the ability to accurately distinguish sounds based on their categorical characteristics and respective modern terms.

The other goal of the study was to implement a user-friendly interface evaluated with thorough usability testing. Initial reactions found the interface intuitive with some room for improvement as adding more descriptive text for buttons and labels and highlighting the ability to scroll through sorted samples dialog box. The usability tests proved that participants with DAW experience quickly understood the purpose. The tests included remarks like despite making some misclassifications the app would most likely assist musicians in sorting their bass samples. As it was suggested having the bass sample sorted to some accurate degree is still better than not sorting them at all. Ultimately showing acceptance of mishaps in the algorithm. Though a difference in usage of bass samples were revealed as one stated they mostly use plugin presets for bass sounds. This could be attributed to plugins presets provide more customization than a sample. Though, it ultimately comes down to the individual musicians workflow. The implemented web-app proves leverage in the sound selection process for musicians that tends to use bass samples rather than the use of plugins.

11 Conclusion

The purpose of this thesis was to explore the realm of sample managers and deep learning techniques for automatic classification and sorting of bass samples, as

well as implementing a user-friendly graphical interface that integrates a deep learning model capable of accurately categorizing the modern bass taxonomy according to their sonic attributes.

Through comprehensive evaluation, the deep learning model demonstrated excellent performance metrics. The model achieved an accuracy of 91.3% and an AUC score of 0.81, indicating a strong ability to distinguish between different classes of bass sounds. Additionally, the precision and recall scores across various categories such as '808', 'brass', 'growl', 'reese', and 'slap' were notably high, underscoring the model's robustness and reliability in real-world applications.

Usability testing of the graphical interface also yielded positive results, confirming that the system is not only effective but also accessible and intuitive for users. The integration of the deep learning model within this interface allows users to efficiently manage and classify their bass samples with minimal effort, enhancing both productivity and user satisfaction.

Despite these successes, there is room for improvement. Some categories, such as 'acid' sounds, exhibited occasional misclassifications, suggesting that further refinement of the model and additional training data could enhance overall accuracy. Additionally, ongoing user feedback will be essential in iterating on the interface design to ensure it meets the evolving needs of users.

In conclusion, this project has successfully implemented a high-performing, user-friendly product that significantly advances the field of sample management and classification. While there are areas for further development, the foundation laid by this thesis provides a strong basis for future enhancements, ensuring that the system remains both cutting-edge and highly functional.

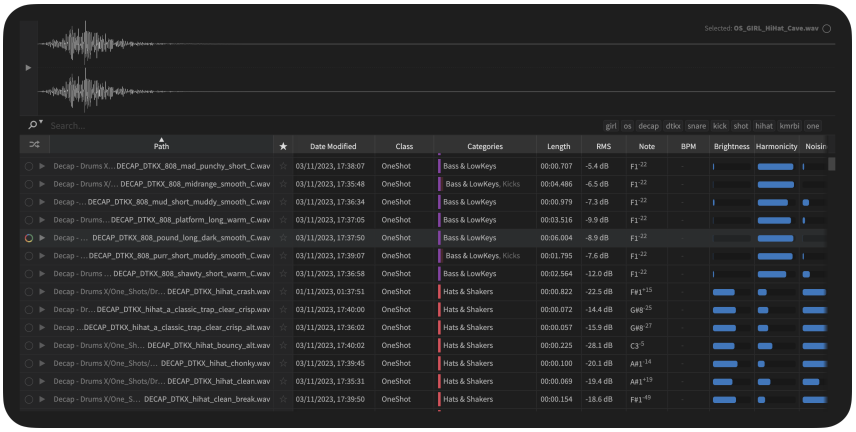
12 Further work

While this thesis successfully developed and evaluated a deep learning model for the classification and sorting of bass samples, several avenues for future research and development can enhance the system's performance and usability. Enhancing the training dataset with more diverse samples could improve the model's generalization capabilities. Exploring different architectures or fine-tuning existing ones, such as experimenting with larger or more complex versions of the DistilHuBert model, could yield better performance. Focused efforts to reduce misclassifications, particularly in categories like 'acid', can involve refining feature extraction methods or using ensemble methods to combine predictions from multiple models. Implementing techniques like model pruning, quantization, or using more efficient inference engines can help reduce prediction latency. Exploring a local model runtime instead of being dependant on the InferenceAPI as well as hardware acceleration options such as GPUs can significantly decrease prediction times.

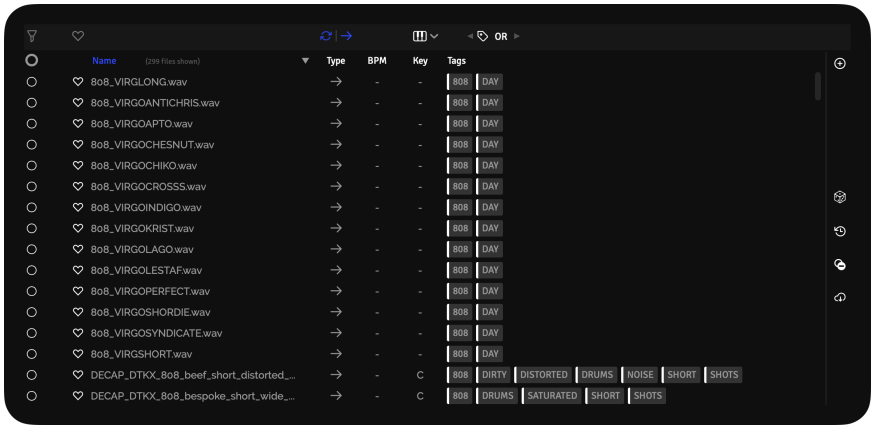
Integrating real-time feedback mechanisms to provide users with immediate insights on their samples can improve usability. Allowing users to customize classification parameters or add new categories dynamically can make the system more flexible and user-friendly. Conducting more extensive usability testing with a broader range of users, including professional sound engineers and casual users, can provide deeper insights into the interface's strengths and areas for improvement. Integrating plugin compatibility with DAWs can streamline workflows for music producers even further. Extending the classification capabilities to other types of sound samples, such as instruments, vocals, or effects, can broaden the system's applicability. Implementing similar techniques for the classification of genre or mood can open up new research and application areas. By addressing these areas, the project can evolve into a more robust, versatile, and user-friendly tool that meets the diverse needs of its users. Future work in these directions will not only enhance the current system but also contribute to the broader field of audio classification and music technology.

A Sample Manager Interface

A.1 Sononym interface



A.2 ADSR Sample Manager interface



B UI mockup

B.1 Web App Design

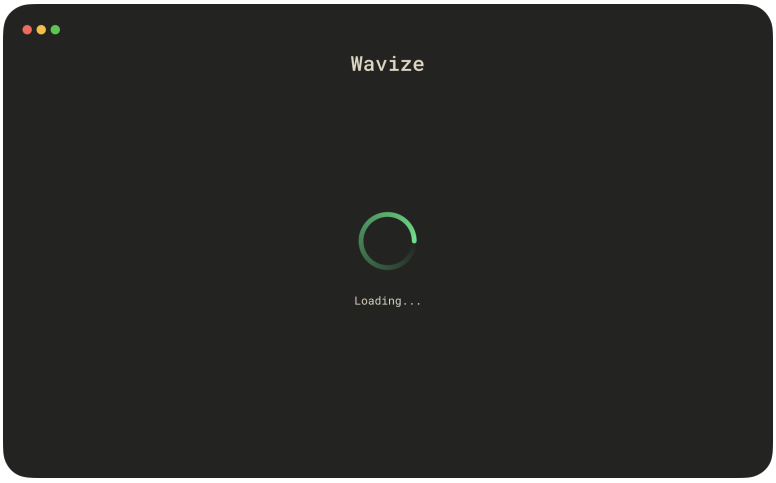


Figure 16: Web app waiting on request

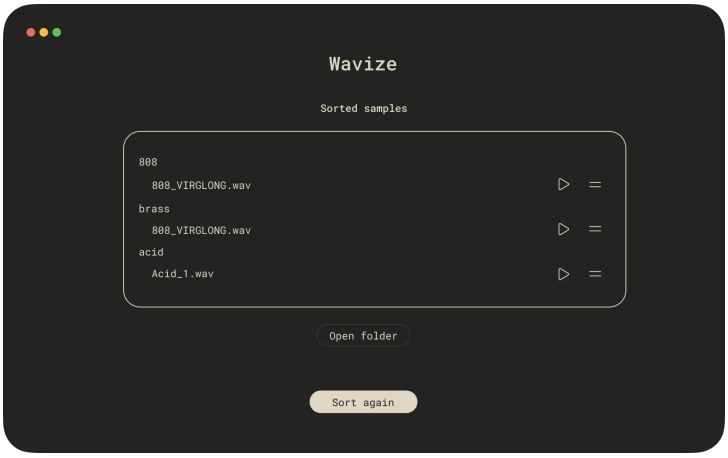


Figure 17: Web app finished fetching predictions and sorting samples

C Code examples

C.1 Envelope method

```
    def envelope(y, rate, threshold):
        mask = []
        y = pd.Series(y).apply(np.abs)
        y_mean = y.rolling(window=int(rate/20),
                           min_periods=1,
                           center=True).max()
        for mean in y_mean:
            if mean > threshold:
                mask.append(True)
            else:
                mask.append(False)
        return mask, y_mean
```

C.2 Training arguments

```
    training_args = TrainingArguments(
        f"{model_name}-{pre_name}",
        evaluation_strategy="epoch",
        save_strategy="epoch",
        learning_rate=5e-5,
        per_device_train_batch_size=batch_size,
        gradient_accumulation_steps=gradient_accumulation_steps,
        per_device_eval_batch_size=batch_size,
        num_train_epochs=num_train_epochs,
        warmup_ratio=0.1,
        logging_steps=5,
        load_best_model_at_end=True,
        metric_for_best_model="accuracy",
        fp16=False,
        push_to_hub=True,
    )
```

Listing 7: Dataset with training split

C.3 Label extraction

```
    function findClosestTo1(arr)
    {
        // Initialize variables to store the closest object and its
             distance from 1
        let closestObj = arr[0];
        let minDistance = Math.abs(arr[0].score - 1);

        // Loop through the array to find the object with a score
            closest to 1
        for (let i = 1; i < arr.length; i++) {
            // Calculate the absolute difference between the
                current object's score and 1
            let distance = Math.abs(arr[i].score - 1);
```

```
11
12              // If the current object's score is closer to 1 than
                    the previous closest object, update the closest
                    object and the minimum distance
13              if (distance < minDistance) {
14                  minDistance = distance;
15                  closestObj = arr[i];
16              }
17          }
18          console.log(closestObj.label);
19          return closestObj;
20      }
```

C.4 Prediction handling

```
1       function handlePrediction(filePath, result) {
2           console.log(result);
3           prediction = findClosestTo1(result);
4
5           const categoryPath = path.join(_destinationFolder,
                prediction.label);
6           console.log(categoryPath);
7
8           if (folderExistsOrCreate(categoryPath)) {
9               copyFileToDirectory(filePath, categoryPath);
10          } else {
11              return;
12          }
13      }
```

C.5 File copying

```
1       function copyFileToDirectory(filePath, destPath)
2       {
3           // Get the filename from the filePath
4           const fileName = path.basename(filePath);
5
6           // Construct the destination path
7           const finalDestination = path.join(destPath, fileName);
8           console.log(finalDestination);
9
10          // Copy the file to the target directory
11          fs.copyFile(filePath, finalDestination, (err) => {
12              if (err) {
13                  console.error('Error copying file:', err);
14                  return;
15              }
16              console.log(`File ${fileName} copied to ${destPath}`);
17          });
18      }
```

C.6 Single latency script

```
    for sample in os.listdir(eval_dir):
        waveform, sample_rate = torchaudio.load(f"{eval_dir}/{
            sample}")

        if sample_rate != 16000:
            resampler = torchaudio.transforms.Resample(sample_rate,
                 16000)
            waveform = resampler(waveform)

        waveform_np = np.array(waveform[0]) # convert to numpy
            array and cut channel

        start_time = time.time()
        _ = model(waveform_np) # classify sample
        end_time = time.time()
        latency = (end_time - start_time) * 1000  # Convert to
            milliseconds
        latencies.append(latency)
```

C.7 Multiple latency script

```
    def simulate_multiple_requests(latency_distribution,
        num_requests=1000, num_predictions=100):
        total_latencies = []
        for _ in range(num_requests):
            simulated_latencies = np.random.choice(
                latency_distribution, num_predictions)
            total_latency = np.sum(simulated_latencies)
            total_latencies.append(total_latency)

        total_latencies = np.array(total_latencies)

        p50 = np.percentile(total_latencies, 50)
        p90 = np.percentile(total_latencies, 90)
        p95 = np.percentile(total_latencies, 95)
        p99 = np.percentile(total_latencies, 99)

        return p50, p90, p95, p99
```

D Evaluation interviews

D.1 Person A

The goal of this project is to enhance sample managers' ability to distinguish between harmonic bass samples categories by using artificial intelligence to automatically sort them. Your tasks will involve performing specific actions and providing feedback on your observations or any unexpected results. But first of all we will start with some questions.

D.1.1 Pre-questions

1. **Q: How many years of musical experience do you have?**
2. *A: 15 years experience*
3. **Q: How many years of experience do you have using a digital audio workstation?**
4. *A: 4 years experience*
5. **Q: Do you know what a sample manager is?**
6. *A: No, not really*
7. **Q: If yes, could you try to explain it?**
8. *A:*
9. **Q: What genres of music do you listen to?**
10. *A: Hip hop, Metal, EDM, Pop, Rock, Orchestral, Folk*
11. **Q: How would you define a bass sound?**
12. *A: Sound that are based on frequencies in the low end of the frequency spectrum around 0-500 Hz. Low octaves. Placed to the left of middle C.*
13. **Q: To the best of your recollection, can you name any bass sound categories?**
14. *A: growl, 808, el-bass, contrabass, MIDI-bass, sang-stemme bass, stortromme*

Now I'm gonna show you the interface of the sample manager application.

D.1.2 Interface exposure

1. **Q: With the exposure of the graphical interface what do you see on the screen?**

2. *A: I see 3 buttons, 2 dark and one white. I assume that the two first button is for finding folder which contains audio samples. The destination one is probably the folder where the sorted versions appear.*

Now the tasks begin, your goal is to select a folder on the computer filled with bass samples and sort in relevance to their specific characteristic defining behavior. However, we will do this in small steps.

D.1.3 Usability tasks

- **TASK 1: Select the first directory from the desktop called "unsorted_samples" containing various sound samples.**

- **TASK 2: Select the second directory from the desktop called "sample_folder" a output target directory for the sorted samples.**

- **TASK: 3 Submit your selection.**

- **Q: What do you see on the application now?**

- *A: It has taken the samples I gave to the AI and it has sorted it for me, in the various categories. The closer it is to 808, the more it is recognised as a 808 bass. The button underneath will probably open the folder with the sorted samples.*

- **Q: How would you interact with the application to see if the rest of the files?**

- *A: Ahh, this make more sense. I would like a indicator that I could actually scroll through the samples. Now it show the application has sorted all the samples in the various categories (So he didn't find the scroll function at first, and one acid bass was placed in the brass category)*

- **TASK 4: Open the output folder from the application.**

- *A: Ahh nice, the application actually made folders for all the categories. That's smart, I didn't expect that. I didn't realise at first that, that was what it did. But it makes great sense in a technical way in how it is done. I would like the folders to be capitalized*

- **TASK 5: Close the file explorer.**

- **TASK 6:Return to the previous directory selection page.**

D.1.4 Post-questioning

- Q: Did anything unexpected happen or areas where you felt restricted?
- *A: The scroll thing. I felt like the concept was very intuitive. I already feel like I know how to use it. It pretty much did what it promised. I like this because programs are always filled with so many functions, social media advertisements, and analytics that they are tedious to use. But it is really cool the applications is straight on the problem, like "I only do this". It almost feels foreign because I'm used to so many functions at a time, but i really enjoy the simplicity of this app.*
- **Q: Could you try to explain why you had to select two different folders?**
- *A: I needed to choose the destination folder because i don't want to clutter the original folder. Since the first folder isn't categorised with folder. However, you could just pick the same folder and then the application would sort the files in that instead. But if I want to select a specific folder to sort it to it is nice to know where it goes. It feels a bit like winraw for audio files, like having different possibilities of unfolding audio data in different ways. It is really nice as if you have a sample library you can just select the destination folder to the sample library and then it would sort and make it easier for me to organise my files. This is actually amazing. This is a much better instead of manually creating folders.*
- **Q: Should the application only have one folder selection?**
- *A: No, i actually prefer the possibility to set the specific destination as i know exactly where it goes at the same time. Then I don't have to manually drag it to my samples library. I think the button should say something like "Select your library" - then i would assume it would nudge the users to choose their existing library.*
- **Q: Do you think this would help musicians organizing their audio files?**
- *A: It would 100 percent make it much easier. Especially since other examples almost have too specific categories, or are really bad categorised.*
- **Q: How satisfied are you with the overall usability of the application?**
- *A: I'm satisfied. It is easy, and a huge contrast to other programs. Really easy and remind me of the "Windows Deeploader" application. It's nice it only embraces the sorting of samples problem. It could be nice if I could just select my sample library and it would sort all of my samples at once. But it is also because mine is incredible unsorted and confusing.*

- **Q: Could you think of any elements that could be improved in the application to your opinion?**
- *A: I know there is not a lot of functionality to it could be nice to spread the items a bit more making it a more breathable interface.*

D.1.5 Creative questioning

- **Q: Imagine you're starting a new track. How would you typically go about selecting bass samples to use?**
- *A: I typically use a sample, drum-loop, or a song that i feel like could be cool to reproduce. I use Splice for the samples, or like-minded distributors. I use Ableton's file hiearchy but also the Splice library. Actually i find the place where the downloaded samples are placed in Splice and drag them to my own sample library. It would make it easier if i could use this (Wavize) instead. I also search for specific samples when i make music in specific bpm as such. And also i use Splice to find other samples which works really great. But I drag the files from Ableton's file system. If i i.e. want a saturated piano Splice tags are just better. It could be really nice if you could take the application and just go through my entire sample library. I want a program that quickly could unfold the folders to other specific folders*

D.2 Person B

D.2.1 Pre-questions

1. **Q: How many years of musical experience do you have?**
2. *A: 7 years*
3. **Q: How many years of experience do you have using a digital audio workstation?**
4. *A: None really, but I know what the program does and have used it a few times.*
5. **Q: Do you know what a sample manager is?**
6. *A: No*
7. **Q: What genres of music do you listen to?**
8. *A: Everything, besides pop, rap, and EDM*
9. **Q: How would you define a bass sound?**
10. *A: 100 Hz low frequent. Can play a higher keys on a bass with higher overtones. Something without too much overtones.*

11. **Q: To the best of your recollection, can you name any bass sound categories?**

12. *A: Brass, String, electro acoustic instruments. Kick drum. Percussion instruments using the low keys, like bells and bars are able to make bass tones.*

Now I'm gonna show you the interface of the sample manager application.

D.2.2 Interface exposure

1. **Q: With the exposure of the graphical interface what do you see on the screen?**

2. *A: There are three buttons, a text and title. It shows that you can select a folder where the files are located and the other is where the files ends. The last one is the button that starts the algorithm.*

Now the tasks begin, your goal is to select a folder on the computer filled with bass samples and sort in relevance to their specific characteristic defining behavior. However, we will do this in small steps.

D.2.3 Usability tasks

- **TASK 1: Select the first directory from the desktop called "unsorted_samples" containing various sound samples.**
- **TASK 2: Select the second directory from the desktop called "sample_folder" a output target directory for the sorted samples.**
- **TASK: 3 Submit your selection.**
- **Q: What do you see on the application now?**
- *A: Dialog box, an open folder button. Then it shows some file names. As I don't really know the labels and categories it is quite confusing.*
- **Q: How would you interact with the application to see if the rest of the files?**
- *A: (He wasn't able to see that the dialog box could scroll).*
- **TASK 4: Open the output folder from the application.**
- *A: I can see folders of the categories and the folders contain the files.*
- **TASK 5: Close the file explorer.**
- **TASK 6: Return to the previous directory selection page.**

D.2.4 Post-questioning

- Q: Did anything unexpected happen or areas where you felt restricted?
- *A: It sorted and acid bass as brass.*
- **Q: Could you try to explain why you had to select two different folders?**
- *A: Input and output made great sense. But it could be good with some indication of what is supposed to happen.*
- **Q: Should the application only have one folder selection?**
- *A: No I think it is fine like this, I can't relate to what it should be used for but seems like it is fine. But if you just wanted to select the same folder it could be fine with one folder.*
- **Q: Do you think this would help musicians organizing their audio files?**
- *A: I think so.*
- **Q: How satisfied are you with the overall usability of the application?**
- *A: With my understanding, it don't feel as intuitive, but it is probably because I don't really know what it should be used for. I think the "sort samples" button should be disabled before you choose the folders. Then a subtitle could be nice that tells what the application's purpose it.*
- **Q: Could you think of any elements that could be improved in the application to your opinion?**
- *A: (See last question)*

D.3 Person C

D.3.1 Pre-questions

1. **Q: How many years of musical experience do you have?**
2. *A: 16 years*
3. **Q: How many years of experience do you have using a digital audio workstation?**
4. *A: 14 years*
5. **Q: Do you know what a sample manager is?**
6. *A: Yes*

7. **Q: If yes, could you try to explain it?**
8. *A: Use it to categorize samples and it is often used for drum samples. They categorize the samples in the type of drum it is like snare or kick drum. Also sometimes they categorize them as soft or harsh, depending on the samples transients. I would say the XO plugin is a sample manager which I used, but it felt like it was too complicated and took too much time to get into.*
9. **Q: What genres of music do you listen to?**
10. *A: Hip hop and R&B*
11. **Q: How would you define a bass sound?**
12. *A: It is a sound that dominates the lower frequencies and they can be entirely different. Bass sound often maintain a constant rhythm in contrast to the higher harmonic sounds like keys which is more used for melodies.*
13. **Q: To the best of your recollection, can you name any bass sound categories?**
14. *A: 808 in Hip Hop, wide synth EDM style basses, an also like analog bass*

Now I'm gonna show you the interface of the sample manager application.

D.3.2 Interface exposure

1. **Q: With the exposure of the graphical interface what do you see on the screen?**
2. *A: I see three buttons, the first is where I can select my samples and then the other one should be were the sorted samples are placed.*

Now the tasks begin, your goal is to select a folder on the computer filled with bass samples and sort in relevance to their specific characteristic defining behavior. However, we will do this in small steps.

D.3.3 Usability tasks

- **TASK 1: Select the first directory from the desktop called "unsorted_samples" containing various sound samples.**
- **TASK 2: Select the second directory from the desktop called "sample_folder" a output target directory for the sorted samples.**
- **TASK: 3 Submit your selection.**
- **Q: What do you see on the application now?**
- *A: The application have sorted the samples in different categories.*

- **Q: How would you interact with the application to see if the rest of the files?**
- *A: (The person knew that the dialog box was able to be scrolled through)*
- **TASK 4: Open the output folder from the application.**
- *A: Now I see the categories of the folders with the samples.*
- **TASK 5: Close the file explorer.**
- **TASK 6:Return to the previous directory selection page.**

D.3.4 Post-questioning

- Q: Did anything unexpected happen or areas where you felt restricted?
- *A: It does exactly what you ask it to and it seems to be usable in this case where it did not make any mistakes.*
- **Q: Could you try to explain why you had to select two different folders?**
- *A: Because you would want the original sample set. I think it is great just in case. But I would want it to do it like this instead of just removing the original folder.*
- **Q: Should the application only have one folder selection?**
- *A: I don't think so, as you could just choose the same folder.*
- **Q: Do you think this would help musicians organizing their audio files?**
- *A: Definitely, I would say that. If you have a large sample library it would make it much easier just to use this as it is so simple. Even if it made some mistakes, it is still better than being totally unsorted. I personally use Splice for most of my samples. But if I took my samples in a folder, this would help a lot.*
- **Q: How satisfied are you with the overall usability of the application?**
- *A: Very simple, it does exactly what you ask it to do. I feel like it is very intuitive. There could be some more text on the screen, like telling what it does. Like a little tutorial saying what it does and how it does it.*
- **Q: Could you think of any elements that could be improved in the application to your opinion?**

- *A: Nothing really specific, but it would be really nice if it was able to sort more than just bass samples if you have a cluttered sample library. Another feature could be if you could exclude something that wasn't a bass. But hopefully it is able to sort all samples in the future. Could be very nice as other sample managers are filled with so many functions it makes it very confusing and overwhelming.*

D.3.5 Creative questioning

- **Q: Imagine you're starting a new track. How would you typically go about selecting bass samples to use?**

- *A: I use logic as my DAW - but usually I do not use samples for basses. I often use synthesizers or plugins. But if I do i drag and drop them from Splice into a sampler. I do not use the logic sample previewer. Also I can search for kick on the system and preview the samples like that, and scroll through the different samples on the hard disk.*

E Questionnaire results

E.1 Musical experience

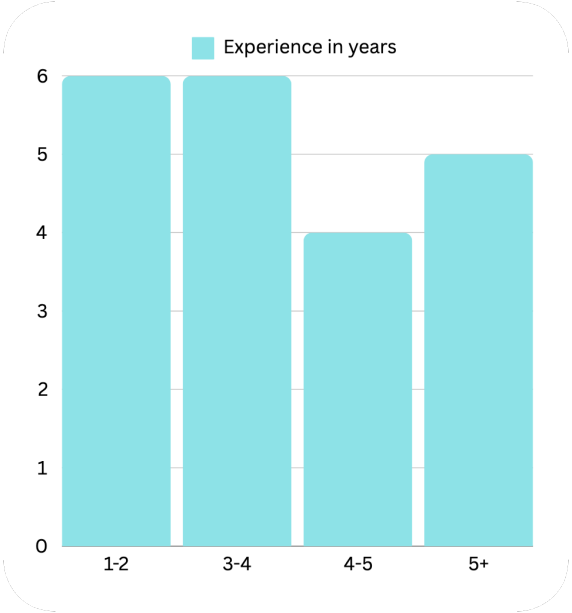


Figure 18: Musical experience of attendants

E.2 Listening test

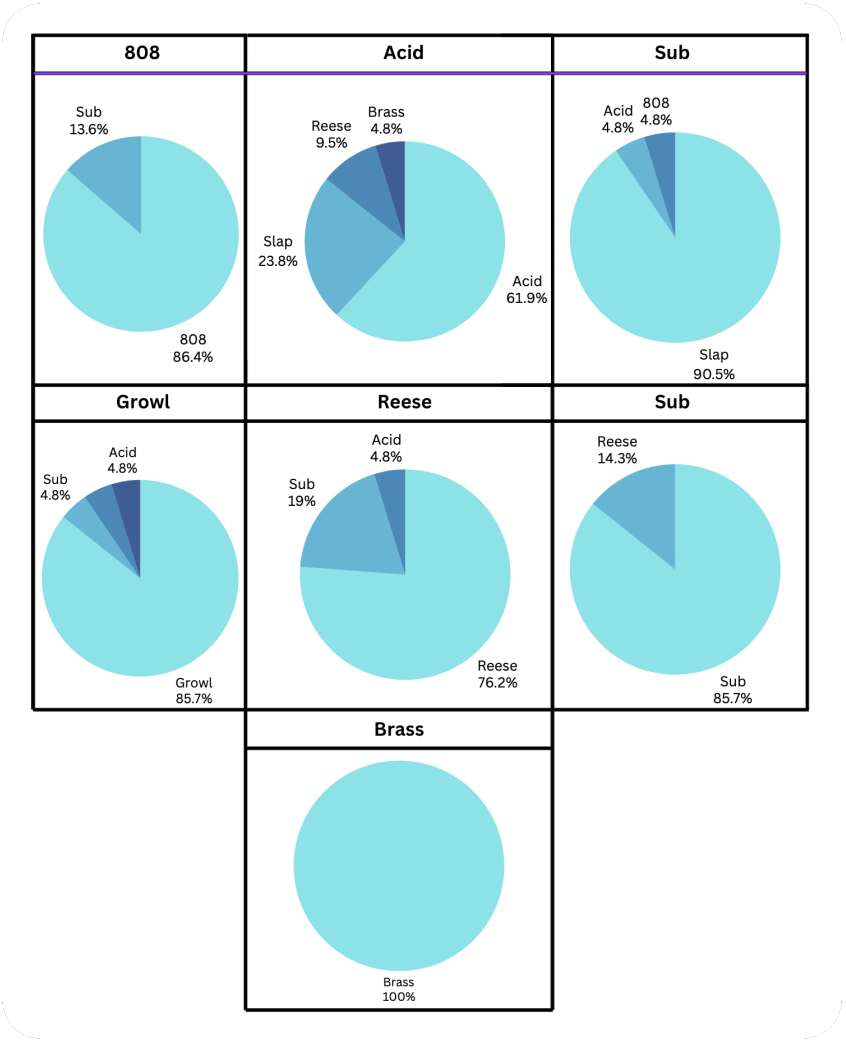


Figure 19: Results from the listening test

References

[1] Ableton. First Steps: ableton reference manual version 11, May 2024. URL http://ableton.com/en/live-manual/11/first-steps/#library.

[2] Ableton. Sound similarity search faq, Feb 2024. URL http://help.ableton.com/hc/en-us/articles/11386675465628-Sound-Similarity-Search-FAQ.

[3] Evidently AI. Accuracy vs. precision vs. recall in machine learning: what's the difference?, May 2023. URL http://evidentlyai.com/classification-metrics/accuracy-precision-recall.

[4] Amplifon. The human hearing range, Jul 2021. URL http://amplifon.com/uk/audiology-magazine/human-hearing-range.

[5] Avid. What is a daw? your guide to digital audio workstations, Nov 2023. URL http://avid.com/resource-center/what-is-a-daw.

[6] Avid. Guide to using samples in music and digital audio production, Oct 2023. URL http://avid.com/resource-center/music-sampling-guide.

[7] Avid. Sound Selection: Creating a sonic palette, Sep 2023. URL http://avid.com/resource-center/sound-selection.

[8] Kleanthis Avramidis, Agelos Kratimenos, Christos Garoufis, Athanasia Zlatintsi, and Petros Maragos. Deep convolutional and recurrent networks for polyphonic instrument classification from monophonic raw audio waveforms. In *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3010–3014, 2021. doi: 10.1109/ICASSP39728.2021.9413479. URL http://dx.doi.org/10.1109/ICASSP39728.2021.9413479.

[9] Alexei Baevski, Henry Zhou, Abdelrahman Mohamed, and Michael Auli. wav2vec 2.0: A framework for self-supervised learning of speech representations. *CoRR*, abs/2006.11477, 2020. doi: 10.48550/arXiv.2006.11477. URL http://doi.org/10.48550/arXiv.2006.11477.

[10] Wenhao Bian, Jie Wang, Bojin Zhuang, Jiankui Yang, Shaojun Wang, and Jing Xiao. Audio-based music classification with DenseNet and data augmentation. In *PRICAI 2019: Trends in Artificial Intelligence*, PRICAI 2019: Trends in Artificial Intelligence, pages 56–65. Springer International Publishing, 2019. doi: 10.1007/978-3-030-29894-4_5. URL http://dx.doi.org/10.1007/978-3-030-29894-4_5.

[11] Nikolaj Buhl. F1 score in machine learning, Jul 2023. URL http://encord.com/blog/f1-score-in-machine-learning.

[12] Stage Center. Edm and the dilution of today's music - center stage music center, Aug 2013. URL http://centerstagemusiccenter.com/edm-and-the-dilution-of-todays-music.

[13] Heng-Jui Chang, Shu wen Yang, and Hung yi Lee. DistilHuBert: Speech representation learning by layer-wise distillation of hidden-unit bert. In *Proc. IEEE Intl. Conf. Acoustics, Speech and Signal Proc. (ICASSP)*, pages 7087–7091, 5 2022. doi: 10.1109/icassp43922.2022.9747490. URL http://dx.doi.org/10.1109/icassp43922.2022.9747490.

[14] Nanxin Chen, Yu Zhang, Heiga Zen, Ron J. Weiss, Mohammad Norouzi, and William Chan. WaveGrad: Estimating Gradients for Waveform Generation. *arXiv e-prints*, art. arXiv:2009.00713, sep 2020. doi: 10.48550/arXiv.2009.00713. URL http://dx.doi.org/10.48550/arXiv.2009.00713.

[15] Anubhav Chhabra, Aryan Veer Singh, Ritesh Srivastava, and Veena Mittal. Drum instrument classification using machine learning. In *2020 2nd International Conference on Advances in Computing, Communication Control and Networking (ICACCCN)*, pages 217–221, 2020. doi: 10.1109/ICACCCN51052.2020.9362963. URL http://dx.doi.org/10.1109/ICACCCN51052.2020.9362963.

[16] Ableton Creator. Learn live 12: Live's browser, Mar 2024. URL https://www.youtube.com/watch?v=KcmPBZTwq78.

[17] EDMProd Creator. I tested out every sample manager (so you don't have to), Jan 2024. URL https://www.youtube.com/watch?v=8bNzgJSPyFc.

[18] James Cullen. Tips for organising your sample library, Feb 2023. URL http://topmusicarts.com/blogs/news/tips-for-organising-your-sample-library#:~:text=A%20Folder%20System%20is%20your.

[19] Discogs. Bass music music description, May 2024. URL http://discogs.com/style/bass+music.

[20] ProjectPro Editorial. 8 deep learning architectures data scientists must master, May 2024. URL http://projectpro.io/article/deep-learning-architectures/996.

[21] Electron. Electron docs, Jan 2018. URL http://electronjs.org/docs/latest.

[22] Electron. dialog: electron, May 2024. URL http://electronjs.org/docs/latest/api/dialog.

[23] Electron. ipcRenderer invoke: electron, May 2024. URL http://electronjs.org/docs/latest/api/ipc-renderer#ipcrendererinvokechannel-args.

[24] Electron. Inter-Process Communication: electron, May 2024. URL http://electronjs.org/docs/latest/tutorial/ipc.

[25] Electron. Quick Start: electron, May 2024. URL http://electronjs.org/docs/latest/tutorial/quick-start.

[26] Electron. ipcRenderer send: electron, May 2024. URL http://electronjs.org/docs/latest/api/ipc-renderer#ipcrenderersendchannel-args.

[27] Electron. shell: electron, May 2024. URL http://electronjs.org/docs/latest/api/shell.

[28] Hugging Face. The hugging face course, 2022, 2022. URL http://huggingface.co/course.

[29] Nicolai Gajhede, Oliver Beck, and Hendrik Purwins. Convolutional neural networks with batch normalization for classifying hi-hat, snare, and bass percussion sound samples. In *Proceedings of the Audio Mostly 2016*, AM '16, page 111–115, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450348225. doi: 10.1145/2986416.2986453. URL http://doi.org/10.1145/2986416.2986453.

[30] Anite George. Learn all about plugins and how they work, Oct 2021. URL http://lifewire.com/what-are-plugins-4582189.

[31] Yuan Gong, Yu-An Chung, and James R. Glass. AST: audio spectrogram transformer. *CoRR*, abs/2104.01778, 2021. doi: 10.48550/arXiv.2104.01778. URL http://doi.org/10.48550/arXiv.2104.01778.

[32] Google. Framing a ml problem - machine learning, May 2024. URL http://developers.google.com/machine-learning/problem-framing/ml-framing.

[33] Google. Supervised vs. unsupervised learning, May 2024. URL http://cloud.google.com/discover/supervised-vs-unsupervised-learning.

[34] Catherine Guastavino. Everyday sound categorization. In Tuomas Virtanen, Mark D. Plumbley, and Dan Ellis, editors, *Computational analysis of sound scenes and events*, pages 183–213. Springer International Publishing, Cham, 2018. ISBN 978-3-319-63450-0. doi: 10.1007/978-3-319-63450-0_7. URL https://doi.org/10.1007/978-3-319-63450-0_7.

[35] Jeff Harder. How synthesizers work, Jun 2012. URL http://electronics.howstuffworks.com/gadgets/audio-music/synthesizer4.htm#:~:text=Digital%20synthesizers%20use%20processors%20and.

[36] Simon Haven. Synth Bass: 9 crucial bass sounds you need to know, May 2022. URL http://edmprod.com/synth-bass.

[37] Simon Haven. Need a sample manager? 3 killer options to browse samples in seconds, Dec 2023. URL http://edmprod.com/sample-manager.

[38] Wei-Han Hsu, Bo-Yu Chen, and Yi-Hsuan Yang. Deep learning based EDM subgenre classification using mel-spectrogram and tempogram features, 2021. URL http://dx.doi.org/10.48550/arXiv.2110.08862.

[39] Wei-Ning Hsu, Benjamin Bolte, Yao-Hung Tsai, Kushal Lakhotia, Ruslan Salakhutdinov, and Abdelrahman Mohamed. HuBERT: self-supervised speech representation learning by masked prediction of hidden units. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, PP: 1–1, 10 2021. doi: 10.1109/TASLP.2021.3122291. URL http://dx.doi.org/10.1109/TASLP.2021.3122291.

[40] HuggingFace. Models - hugging face, Sep 2023. URL http://huggingface.co/models?pipeline_tag=audio-classification&sort=trending.

[41] HuggingFace. Audio classification, May 2024. URL http://huggingface.co/docs/transformers/en/tasks/audio_classification.

[42] HuggingFace. Fine-tuning a model for music classification - hugging face audio course, May 2024. URL http://huggingface.co/learn/audio-course/en/chapter4/fine-tuning.

[43] HuggingFace. ntu-spml/distilhubert hugging face, Jan 2024. URL http://huggingface.co/ntu-spml/distilhubert.

[44] HuggingFace. Inference endpoints - hugging face, May 2024. URL http://huggingface.co/inference-endpoints/dedicated.

[45] HuggingFace. Unit 4. build a music genre classifier - hugging face audio course, May 2024. URL http://huggingface.co/learn/audio-course/en/chapter4/introduction.

[46] HuggingFace. Detailed parameters, May 2024. URL http://huggingface.co/docs/api-inference/detailed_parameters?code=js.

[47] HuggingFace. More information about the api, May 2024. URL http://huggingface.co/docs/api-inference/en/faq#rate-limits.

[48] HuggingFace. Inference api (serverless) - hugging face, May 2024. URL http://huggingface.co/inference-api/serverless.

[49] HuggingFace. Inferenceapi connection, May 2024. URL http://huggingface.co/TheDuyx/distilhubert-bass-classifier9?inference_api=true.

[50] C. Huyen. *Designing machine learning systems: An iterative process for production-ready applications*. O'Reilly Media, Incorporated, 2022. ISBN 978-1-09-810796-3. URL https://books.google.dk/books?id=YISIzwEACAAJ. tex.lccn: 2023275143.

[51] IBM. What is machine learning?, Jan 2023. URL http://ibm.com/topics/machine-learning.

[52] Javatpoint. Machine learning techniques - javatpoint, May 2024. URL http://javatpoint.com/machine-learning-techniques.

[53] Aamir Kalimi. Deep learning vs shallow learning, Aug 2023. URL http://medium.com/@codekalimi/deep-learning-vs-shallow-learning-6af132bb6096.

[54] kits.ai. Synth preset, May 2024. URL http://kits.ai/glossary/synth-preset.

[55] Harvard Labs. Machine learning systems - audio feature engineering, May 2024. URL http://harvard-edge.github.io/cs249r_book/contents/kws_feature_eng/kws_feature_eng.html#:~:text=High%20Dimensionality%3A%20Audio%20signals%2C%20especially.

[56] Landr. Synth bass: 7 bass types and how to build them, Jan 2021. URL http://blog.landr.com/synth-bass.

[57] Siddique Latif, Aun Zaidi, Heriberto Cuayahuitl, Fahad Shamshad, Moazzam Shoukat, and Junaid Qadir. Transformers in Speech Processing: a survey, 2023. URL http://dx.doi.org/10.48550/arXiv.2303.11607.

[58] Alex Lavoie. The 5 best royalty-free sample libraries and how to use them, Dec 2022. URL http://blog.landr.com/sample-library.

[59] Jongpil Lee, Taejun Kim, Jiyoung Park, and Juhan Nam. Raw waveform-based audio classification using sample-level CNN architectures. *CoRR*, abs/1712.00866, 2017. doi: 10.48550/arXiv.1712.00866. URL https://doi.org/10.48550/arXiv.1712.00866.

[60] Linkedinnagarjun. Crafting a Quality Training Dataset for Machine Learning: Best Practices and Considerations, Feb 2024. URL http://medium.com/@linkedinnagarjun/crafting-a-quality-training-dataset-for-machine-learning-best-practices-and-considerati

[61] Computer Music Specials 06 May 2013. What is bass music?, May 2013. URL http://musicradar.com/news/tech/what-is-bass-music-574712.

[62] Melanie. The importance of cross validation, Sep 2023. URL http://datascientest.com/en/the-importance-of-cross-validation.

[63] Dusti Miraglia. Synth bass: 6 different bass types and how to expertly use them, Dec 2023. URL http://unison.audio/synth-bass.

[64] Mozilla. Promise, Mar 2019. URL http://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise.

[65] Mozilla. Eventtarget.addeventlistener(), May 2024. URL http://developer.mozilla.org/en-US/docs/Web/API/EventTarget/addEventListener.

[66] Loris Nanni, Gianluca Maguolo, and Michelangelo Paci. Data augmentation approaches for improving animal audio classification. *Ecological Informatics*, 57:101084, 2020. doi: 10.48550/arXiv.1912.07756. URL http://doi.org/10.48550/arXiv.1912.07756.

[67] Jakob Nielsen. Powers of 10: time scales in user experience, Oct 2009. URL http://nngroup.com/articles/powers-of-10-time-scales-in-ux.

[68] Node.js. File system: node.js v17.8.0 documentation, May 2024. URL http://nodejs.org/api/fs.html#file-system.

[69] Node.js. Path: node.js v22.0.0 documentation, May 2024. URL http://nodejs.org/api/path.html#path.

[70] OpenAI. Openai api, May 2024. URL http://platform.openai.com/docs/guides/fine-tuning.

[71] Sida Peng, Eirini Kalliamvakou, Peter Cihon, and Mert Demirer. The Impact of AI on Developer Productivity: evidence from github copilot, 2023. URL http://dx.doi.org/10.48550/arXiv.2302.06590.

[72] Aden Russell. The producer's guide to file and sample organization, Oct 2019. URL http://edmprod.com/file-sample-organization.

[73] Maximilian Schmitt and Björn Schuller. End-to-end Audio Classification with Small Datasets – making it work. In *2019 27th European Signal Processing Conference (EUSIPCO)*, pages 1–5, 2019. doi: 10.23919/EUSIPCO.2019.8902712. URL http://dx.doi.org/10.23919/EUSIPCO.2019.8902712.

[74] ScienceDirect. Nyquist Theorem: - an overview - sciencedirect topics, May 2024. URL http://sciencedirect.com/topics/engineering/nyquist-theorem#:~:text=The%20Nyquist%20theorem%20specifies%20that.

[75] Amazon Web Services. What is hyperparameter tuning? - hyperparameter tuning methods explained - aws, May 2024. URL http://aws.amazon.com/what-is/hyperparameter-tuning.

[76] FL Studio. Effect plugins, May 2024. URL http://image-line.com/fl-studio-learning/fl-studio-online-manual/html/effects_plugins.htm.

[77] Syntorial. Synth quickie – fm bass syntorial, Apr 2019. URL http://syntorial.com/tutorials/synth-quickie-fm-bass.

[78] Evidently AI Team. How to explain the roc auc score and roc curve?, May 2024. URL http://evidentlyai.com/classification-metrics/explain-roc-curve.

[79] Sundeep Teki. Knowledge Distillation: Principles, algorithms, applications, Jul 2022. URL http://neptune.ai/blog/knowledge-distillation.

[80] TensorFlow. vggish/readme.md, May 2024. URL http://github.com/tensorflow/models/blob/master/research/audioset/vggish/README.md.

[81] TensorFlow. Transfer learning with yamnet for environmental sound classification - tensorflow core, May 2024. URL http://tensorflow.org/tutorials/audio/transfer_learning_audio.

[82] Unison. How To Manage Your Sample Library: 7 essential tips to ensure organization - unison, Dec 2021. URL http://unison.audio/sample-library.

[83] Verbit. Understanding WAV Files: The ultimate guide to audio quality and file management, May 2024. URL http://verbit.ai/understanding-wav-files-the-ultimate-guide-to-audio-quality-and-file-management/#:~:text=Introduction%20to%20WAV%20Files%20and.

[84] Prateek Verma and Jonathan Berger. Audio Transformers: transformer architectures for large scale audio understanding. adieu convolutions. *CoRR*, abs/2105.00335, 2021. doi: 10.48550/arXiv.2105.00335. URL https://doi.org/10.48550/arXiv.2105.00335.

[85] James Walker and J.S. Don. *Mathematics and Music: Composition, Perception, and Performance*. Chapman and Hall/CRC, 04 2013. ISBN 9780429159695. doi: 10.1201/b14784. URL http://dx.doi.org/10.1201/b14784.