L.AI.GHTs

Automating Visualisation and Lighting for DJ Sets

Felix Cohen, Mark Jegorovas, Dain Jung, Oli Wing

Name	Page Numbers	Page Count
Felix Cohen	16-19, 52-72, 106-108, 116-117	30
Mark Jegorovas	8-15, 86-103, 105, 113-115	30
Dain Jung	5-7, 20-31, 73-85, 118-119	30
Oli Wing	1-4, 32-51, 104, 109-112, 120	30

Contents

1 Overview

1.1 Introduction - Oli Wing

The purpose of this project is to outline the design and implementation of an intelligent lighting and visualisation system that utilises machine learning (ML) to control lighting and create dynamic visuals in real-time for DJs. L.AI.GHTs aims to make live music more immersive for audiences, providing DJs with an interesting and synchronised visual accompaniment to their sets. L.AI.GHTs will enhance performances, engage audiences, and unlock new creative possibilities for DJs.

Visuals accompanying electronic dance music (EDM) take two main forms - visualisers and physical lighting. DJs often use live visuals to enhance their performances. DJs play EDM to live audiences by blending songs together to create a continuous flow of music. These live performances are known as 'sets'. When properly synchronised, visuals can emphasise the impact of certain musical moments, drops, or transitions, amplifying the emotional connection between the music and visuals. Visual accompaniment can also reinforce a DJ's branding and identity.

Although software for visualisation and lighting control exists, it is limited in various ways. Visualisers and lighting controllers are able to create diverse visuals but they cannot be automated, meaning that creating these visuals requires the knowledge and operation of a lighting or visualiser specialist which amateur DJs do not have access to. Some DJ software such as Rekordbox offer a live visualisation, but these are generic and unable to reflect the emotive content of the music, thus falling short of professional visuals.

To solve this, we propose L.AI.GHTs, a fully automated visualisation and lighting software application that creates reactive and dynamic visuals for DJs during live performances. Our software provides two forms of visual synthesis - visualisation and lighting control.

To produce convincing visuals of the same calibre as a lighting designer, the visual synthesiser must be able to identify qualities of the song being visualised. These include musical features (tempo and downbeat location) and descriptive features (energy and valence). This is possible using machine learning methods to analyse the music, then using the extracted information to determine the parameters and behaviour of the visuals.

We chose to analyse the song at three levels of detail: whole song, individual segments, and beats. This allowed us to focus on four avenues of Music Information Retrieval (MIR) most important for pro-

ducing a visualisation: Genre Recognition (whole song), Structural Segmentation (segments), Emotion Recognition (segments), and Downbeat Detection (beats). The genre classification allows us to determine the colour of the visuals, complementing the nature of the music. Determining the structure of the track allows us to create dynamic transitions between song sections, as well as analysing the emotion of sections individually to make the visuals more reactive. Downbeat detection allows for the visualiser and lighting to synchronise with the music.

Once the MIR analysis has been completed, we use this information to inform the creation of the visuals. For the visualiser, this is done using live visual synthesis software based on the principles of modular synthesis, allowing for variations in parameters and behaviour. This means every song produces a unique and distinctive visualisation that reflects its musical qualities. For the lighting, the visualisation can then be segmented and mapped onto the lighting control scheme to create a unique but interrelated behaviour. The downbeats allow us to change lighting parameters synchronously with the audio.

Figure 1.1 illustrates the design of our product and the process from selecting a song to producing a visualisation and lighting scheme. The MIR stage encompasses pre-processing the song into a useful form for the machine learning algorithms, then using these algorithms to obtain the downbeat locations, the structural segment boundaries, and emotion and genre information. The visual synthesis stage comprises the creation of the visualisation and the lighting control sequence using the information obtained from MIR.

We developed a comprehensive business plan with ethical considerations, risk analysis and realistic financial projections for our product. Within this model, L.AI.GHTs retails at $99 for the base visualiser and $199 for the visualiser with lighting control.

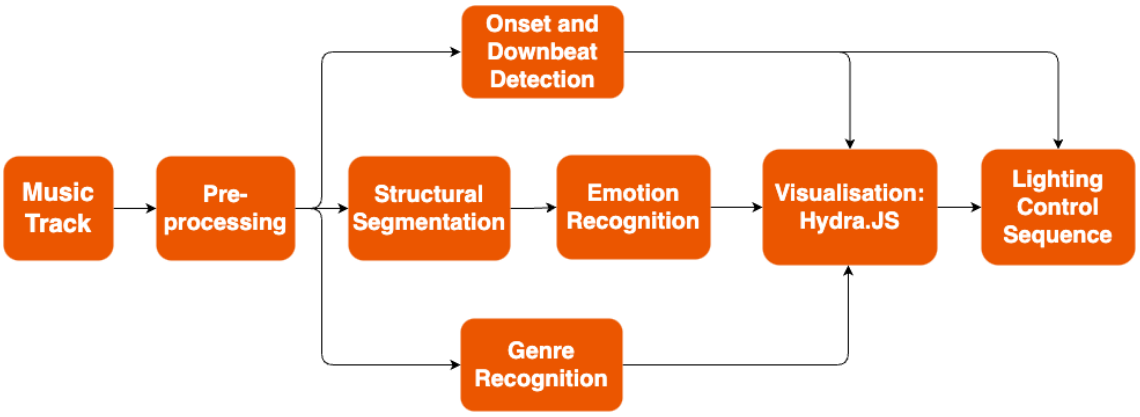


Figure 1.1: Flow chart of the steps from MIR to visual synthesis.

1.2 Business Plan - Dain Jung

1.2.1 Business Drivers

The key drivers impacting L.AI.GHTs include the increasing popularity of live DJ performances and the growing demand for advanced visualisation and lighting solutions. The trend towards digitisation and the use of machine learning to enhance live performances is also a significant driver. Additionally, the increased affordability of smart lighting and projection systems, and the rising expectations of audiences for immersive and interactive experiences, will further drive demand for our product.

1.2.2 Market Analysis

Target Market and Customer Segmentation: Our primary target market ranges from amateur to semi-professional DJs and music venues needing automated visualisation and lighting solutions for live performances. We have identified our primary customer base as music enthusiasts aged 18-40, proficient in technology. Other markets include event organisers, lighting designers, and audio-visual equipment retailers. Amateur DJs, who only have access to projection systems for small local events, will likely favour our base visualiser product. Semi-professional DJs and larger venues with lighting infrastructure will prefer our premium visualiser and lighting control product. Figure 1.2 shows the customer segmentation in the target market.

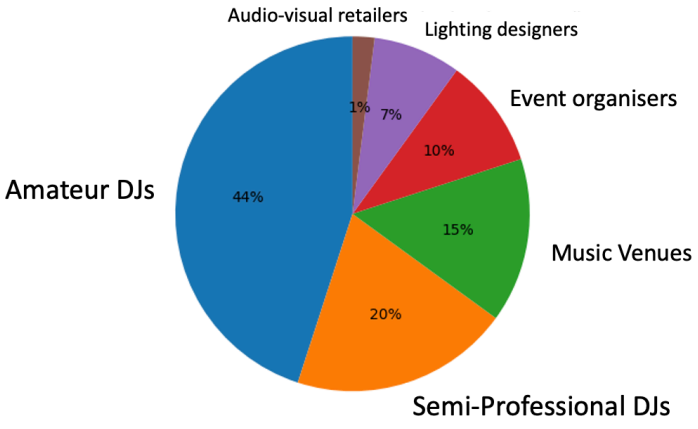


Figure 1.2: Customer segmentation.

Market Potential and Competitive Advantage: The market for live music performances and DJ events is experiencing significant growth [1]. The details are further discussed in Section 11.3.1. This presents a demand for innovative solutions such as L.AI.GHTs. Machine learning integration in lighting systems

distinguishes L.AI.GHTs from traditional designs, directly catering to our customer's needs with dynamic, automated lighting that enhances performances. The ability to run L.AI.GHTs on multiple operation systems allows for the largest potential customer base. We aim to invest in R&D to refine our algorithms, audio analysis, and visualisation techniques. This ensures that we consistently meet and anticipate customer needs.

Marketing and Distribution: The marketing strategy will leverage online platforms, industry events, and partnerships to showcase the product and its unique features. Strategic partnerships with music equipment retailers, both online and offline, will be explored to maximise product reach and visibility. The distribution will be primarily direct-to-consumer via our website, with plans to expand to retail partnerships.

1.2.3 IP Strategy

To protect the unique technical aspects of L.AI.GHTs, we will be proactive in developing a strong IP strategy. This includes applying for patents for our unique algorithms and lighting control systems, registering trademarks for our brand name and logo, and using copyright to protect our software code. In addition, we will protect our trade secrets, such as specific machine learning techniques and business strategies, through non-disclosure agreements and other legal measures.

In addition, DRM technology will protect our online product from piracy and unauthorised use. We will use a DRM Controller IP to generate and deliver a one time encrypted license key to our customers, which will activate the product on their devices.

1.2.4 Growth and Future Developments

The business model is designed for long-term, with revenue from software sales, subscriptions, and value-added services providing a steady income stream. Continuous R&D effort will keep the product competitive by introducing new features and improvements based on customer feedback and development in technologies. Future versions of the product could include advanced features such as voice control, integration with other DJ equipment, and an expanded library of visualisation patterns and effects. The product can also be used to create music videos using the visualisation.

1.3 Product Specifications - Dain Jung

Following our business plan, it is important to outline the specific features and functionalities that our product should offer to meet the expectations of our target market. These specifications guide the development process and ensure that our product delivers the desired performance and user experience. The objectives are categorised as either objective (O) or subjective (S).

S1: Real-time Visualisation: The product should provide a dynamic, real-time response to changing audio parameters with a latency of less than 100 ms. (O)

S2: Adaptive to Different Music Genres: The product should be capable of interpreting a wide range of genres with at least 80% classification accuracy. (O)

S3: Operate in Various Audio Quality Conditions: The product should be robust to noise, maintaining consistent performance for both high-quality and low-quality audio files. (S)

S4: Customisable Visualisation and Lighting Effects: The product should offer at least five customisable visualisation options and presets, and it should allow DJs to fine-tune the lighting effects, providing flexibility to create a unique atmosphere for each performance. (O)

S5: User-friendly, Intuitive User Interface and Documentation: The product should feature a user-friendly interface. User manuals and documentation should be provided for ease of use. (S)

S6: Audio Reactive Lighting Configuration and Synchronisation: The product should have a lighting configuration reactive to the audio track content and automatically synchronise lighting effects with the music, enhancing the overall DJ performance. (O)

S7: Machine Learning for Audio Analysis: The product should apply machine learning to perform real-time audio analysis, accurately capturing beat, tempo, and energy. (O)

S8: Dynamic Lighting Effects: The product should generate dynamic lighting effects, colours, patterns, and movements, based on real-time audio analysis for a unique visual experience. (S)

S9: Competitive Pricing and Usability: The product should be available in two versions - the base visualiser and the premium visualiser with lighting control, competitively priced at $99 and $199 respectively. Its pricing should position it strongly against competitors. (O)

2 Lighting And Visualisation Control - Mark Jegorovas

L.AI.GHTs contains two main visual aspects that can be controlled during a live performance: lighting and visualisation. In this section, we will discuss the attributes of the lighting that can be changed, how to control the lighting, and how to project the visualisation. We will then discuss how to map lighting parameters to audio parameters from the song.

2.1 Lighting Attributes

The first visual aspect of the L.AI.GHTs project is creating a unique lighting sequence. It is important to understand how we can vary each light source and the fixture that contains it. Lighting fixtures have many different attributes and controllable factors. There are four main attributes: light intensity, light colour, fixture distribution, and fixture movement [2]. The first two describe how the light source inside the fixture can be varied, while the latter two describe how the fixture itself can be varied. It is important to note that different lighting fixtures will contain a different combination of controllable factors, with the factors usually determined by the purpose of the light [3].

2.1.1 Light Intensity and Colour

Light intensity describes the transmitted power from the light source, which can be increased (brightening) or decreased (fading). Brightness and intensity are different parameters, with brightness being a perception of intensity which is related to the components of colour. If R, G, B are the intensities of the red, green, and blue channels respectively, the lighting parameters are defined as:

$$\text{Intensity} = \frac{R+G+B}{3} \quad \text{and} \quad \text{Brightness} = \max R, G, B \tag{1}$$

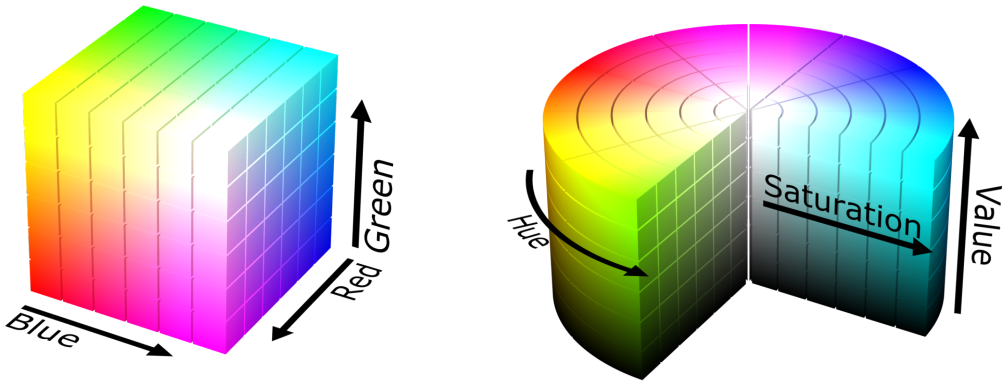
Fixtures may also have shutters that completely block the light source or gobos, which are objects placed in front of the fixture. The shutters can be periodically opened and closed to create a strobe, which is described by its frequency and duty cycle (the proportion of time the shutter stays open).

The lighting colour is described by the distribution of wavelengths of light leaving the fixture. This is typically achieved through additive or subtractive colour mixing. Additive colour mixing involves an array of red, blue, and green (RGB) LEDs, in which a specific combination of RGB intensities creates the desired colour. Subtractive colour mixing involves a white light source behind three dichroic filters (cyan,

magenta, and yellow), which absorb a proportion of a range of wavelengths [4]. The filters can rotate to absorb different proportions of intensity for various wavelengths.

There are two ways to describe colour: red-green-blue-alpha (RGBA) and hue-saturation-value (HSV). The RGBA colour system is represented by a Cartesian coordinate system, with colours created by a linear combination of RGB. It is commonly used in digital systems such as DMX channels, as it is widely supported by software. The alpha value represents the transparency of the image, which is not necessary for lighting purposes (this is expanded further when referring to visualisation in section 9.3.2). The RGBA colour system is shown in Figure 2.1a.

On the other hand, the HSV colour space matches people's visual perceptions of colour more intuitively than RGBA [5]. The hue represents pure colours in an angular range, the saturation represents the amount of that colour, and the value is related to the brightness (closer to the luminescence). It is represented as a cylindrical coordinate system [6]. The HSV colour space is shown in Figure 2.1b.



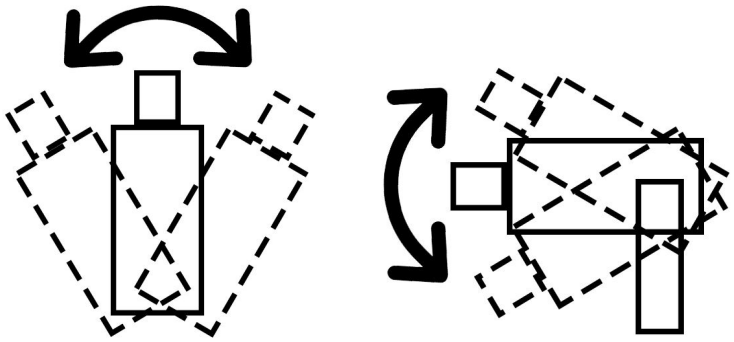
(a) RGB colour system. (b) HSV colour space.

Figure 2.1: Figures showing ways to describe colour [6].

2.1.2 Fixture Distribution and Movement

Fixture distribution describes the position and angle of the fixture. The position refers to where the fixture is located in physical space. Fixtures can be front lights or back lights, where the fixtures are placed in the front or back of the stage, respectively [2]. Figure 2.2 shows how to describe the fixture rotation, where the fixture angle can be described by panning (Figure 2.2a) and tilting (Figure 2.2b).

Lastly, lighting movement involves the relationship between a fixture and the other fixtures, for example,



(a) Fixture panning. (b) Fixture tilting.

Figure 2.2: Figures showing types of fixture rotation.

one light brightening and another fading (split-fading). Movement also involves the fixture speed, which is how quickly the transition between parameter changes happens.

2.2 Lighting Control

To automate the real-time adjustment of these lighting attributes, a control system is necessary. We will discuss the industry standards for digital control (DALI) and analogue control (DMX) of lighting, as well as how to configure the control system to enable software control of the lighting.

2.2.1 Digital Addressable Lighting Interface

For digital control of lighting systems, the industry standard is the Digital Addressable Lighting Interface (DALI). The network comprises a bus power supply, controllers (including sensors), and lighting components, with all devices connected to a bi-directional bus. This means that devices are capable of both transmitting and receiving control signals. All lighting components should be LED and support the DALI protocol [7]. The only factor that can be changed is the light intensity, which is mapped logarithmically using a byte of data. Each device in the network has a unique 6-bit address, allowing for the connection of up to 64 devices. An example network topology is shown in Figure 2.3.

2.2.2 Digital Multiplex

For analogue control of lighting systems, the industry standard is the Digital Multiplex with 512 channels (DMX-512), where each channel corresponds to a byte (8 bits) of data [8]. A DMX universe consists of these 512 channels. In a DMX universe, a controller is connected to each fixture using a multi-drop bus

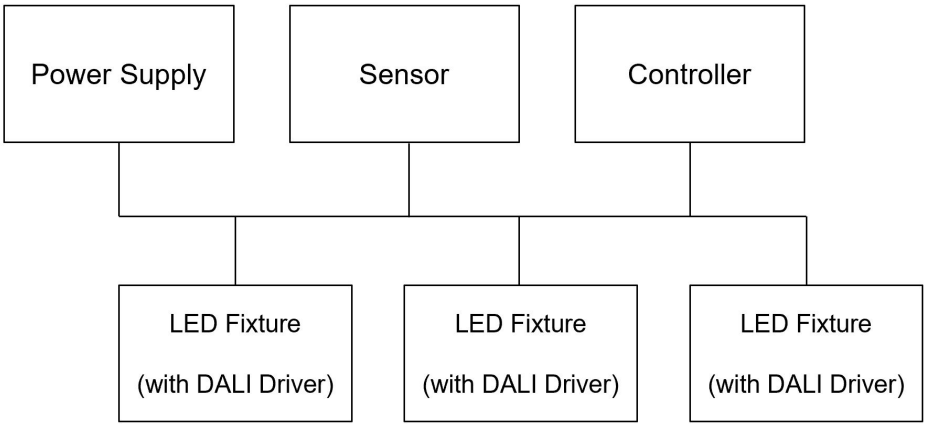


Figure 2.3: Example DALI network topology: 3 fixtures, a sensor, and a controller all connected via a bus.

topology known as a daisy chain. Each fixture reads the data outputted from the controller and extracts the information required to control its respective fixture [9]. The controller contains sliders which can set up the lighting parameters in a certain way. For example, a simple 5-channel fixture may use 2 channels for panning, 2 channels for tilting, and 1 channel for intensity. A universe can then control up to 102 5-channel fixtures. This means for each DMX channel, the fixture number and the fixture attribute should be mapped. If more channels are required, multiple universes can be connected to a single controller. Figure 2.4 shows the daisy chain bus topology of the DMX-512 universe.

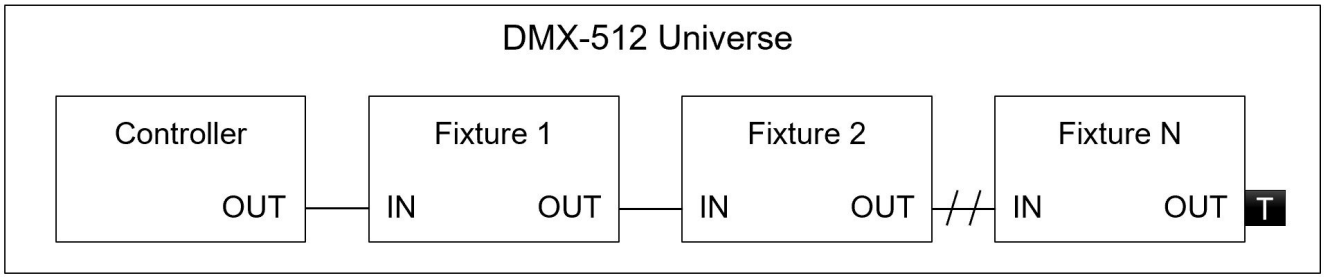


Figure 2.4: Network topology of an N fixture DMX-512 Universe. T represents a terminating resistor that matches the characteristic impedance of the cables to prevent reflections. Each cable is a 5-pin XLR.

As we do not assume the user's available lighting options, DMX is preferred over DALI. DALI can only be used for LEDs, whereas DMX can accommodate any DMX lighting fixture. Additionally, DALI has slower control with a delay time of 2 seconds [7], while DMX can achieve a maximum packet send time of 23 ms [10] (equivalent to a refresh rate of 44 Hz, although some fixtures may not process changes that fast, so lower refresh rates are used). The low refresh rate of DALI can be extremely detrimental as the lighting changes need to synchronise with the music. A shorter delay allows for more responsive lighting adjustments, resulting in an improved final outcome.

While DALI can connect to multiple controllers, only the user's device should be used for lighting control. Although DALI has a standardised light intensity mapping, it is limited to adjusting only the intensity of lighting sources, whilst DMX provides the capability to control all four controllable factors discussed in Section 2.1.

In conclusion, to create a unique lighting scheme, it is necessary to adjust multiple controllable factors beyond just light intensity. A reasonable delay when changing parameters is also important to closely match the audio inputted. Furthermore, considering that not all fixtures are LED, DMX is the better lighting control scheme for our project.

2.2.3 Lighting Control Preparation

To control the lighting, we need to prepare the console to correctly map the DMX channels to the fixture parameters and enable the DMX console to accept external inputs. The input is the user's device running the L.AI.GHTs software.

Different lighting fixtures contain a different combinations of controllable attributes. This means each lighting fixture requires a unique number of DMX channels. DMX channels can be mapped to the fixture parameters using a patch. A patch is acquired from a fixture library that maps each channel to the corresponding fixture and attribute. Multiple fixtures can also be grouped together, sharing the same set of channels.

Once a universe is set up, changing between different sets of parameters can be done using a cuelist. A cue is a single set of parameters for a group of fixtures, and a cuelist is a set of cues [3]. When changing attribute parameters through cues, the console uses a "latest takes priority" precedence, except for intensity, which follows a "highest takes priority" precedence. This combines into the control structure for lighting as shown in Figure 2.5.

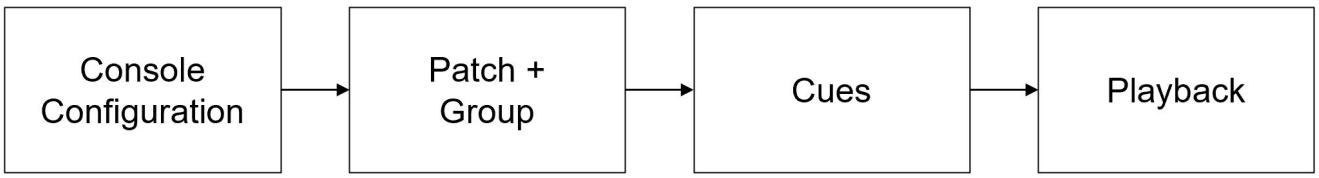


Figure 2.5: Control flow for a DMX system.

In order to control the console remotely, DMX-512 controllers accept MIDI in two formats: MIDI timecode (MTC) and MIDI show control (MSC). MTC uses a time and a cue for execution. The time is split into hours, minutes, seconds, and frames. In context, MTC is used primarily for pre-recorded or rehearsed

musical performances [10]. On the other hand, MSC uses a set of commands that are transferred between consoles/controllers. Each lighting fixture is assigned a unique ID that can be included in these commands. Since the DJ set is not pre-recorded or rehearsed, MTC cannot be used, and thus MSC is the suitable choice for controlling the final DMX-512 system. The user's device can output MSC to change the fixture parameters through the DMX console.

As mentioned in section 2.2.2, there is a limit to the frequency of MSC packets sent due to the maximum packet send time in the universe. The refresh rate of DMX is typically lower (between 25-30 Hz), so the MSC packet send frequency should be adjusted accordingly. The console control structure is shown in Figure 2.6.

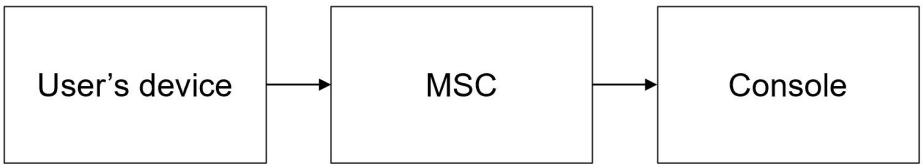


Figure 2.6: Control flow for the lighting system.

To connect the user's device to the console, an adapter from USB to DMX is required. This introduces some initial cost to the product. The device can either be connected directly to the console or act as the controller for the fixtures (essentially making the user's device the controller in the universe, as shown in Figure 2.4). Another solution is to connect to the console by DMX over Ethernet using a protocol like Art-Net [11].

2.3 Visualisation Control

The second visual aspect of the project involves a projection of a visualiser. There are two methods for producing the projection: using an LCD screen or Digital Light Processing (DLP) [3].

LCD screens work on a pixel basis, with each pixel being a liquid crystal director between two polarising filters at 90° to each other, and two transparent tin oxide electrodes. When the LCD pixel is off, the crystal rotates the light passing through the filter by 90°, allowing it to pass through the second polarising filter unhindered. When a voltage is applied between the two electrodes, the LCD is turned on. The crystals align due to the electric field created by the voltage difference, preventing the rotation of ambient light and absorbing it through the second polarising filter. As a result, the pixels that are on appear dark [12]. This mechanism is shown in Figure 2.7.

DLP uses digital mirror devices (DMDs) arranged in an array that matches the resolution of the projected

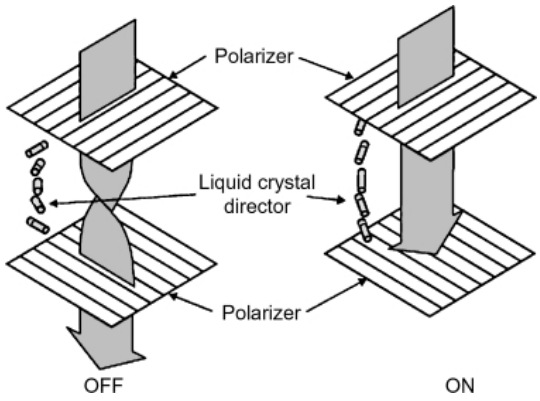
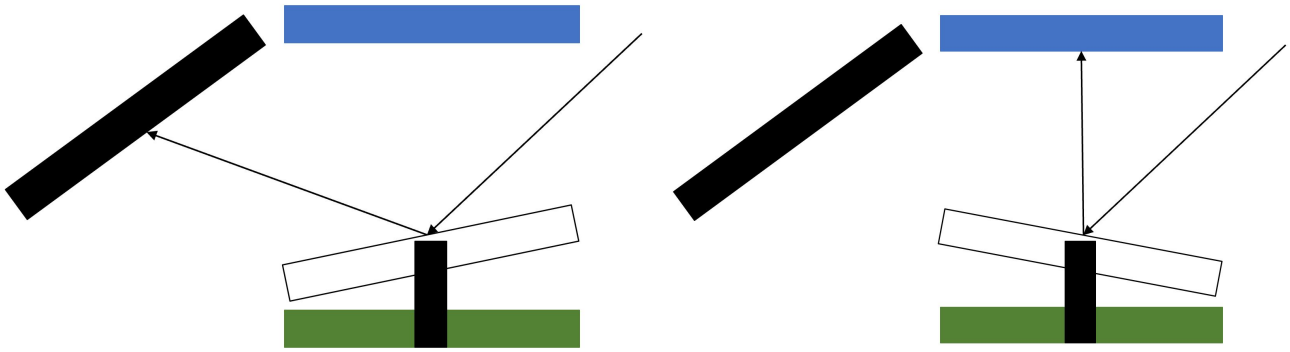


Figure 2.7: LCD pixel structure and mechanism [12].

image, with each DMD serving as an optical pixel. When light is shined onto the mirror, it can be reflected onto a black light absorber or through to the objective, depending on the angle of the mirror. The mirror is tilted electrostatically using a CMOS substrate, with different angles deciding whether the incoming light is projected [13]. This mechanism and the structure of the DMD is shown in Figure 2.8.



(a) Mirror rotated so incoming light is absorbed.

(b) Mirror rotated so incoming light is projected.

Figure 2.8: Figures showing DMD composition and workings. CMOS substrate is green, projection system is blue, light absorber is black, the mirror is white. The arrows dictate the path of the light.

To control the projection via DMX, a media server is needed. This is a powerful computer that can run a software application capable of storing and playing back video files, graphics, and live video with real-time effects [10]. A DMX address mapping for the lighting system is created by the server, which creates a patch for the projection. Media servers can have multiple layers (pieces of content) streamed through them, with DMX layer attributes being assigned to them in the same format as described in section 2.2.2. Having too many layers can lead to poor playback performance [14], but this is not a concern as the combination of multiple layers can be achieved through the visualiser as discussed in section 9.3.3. Therefore, we can combine projection control and lighting control within the network shown in Figure 2.9.

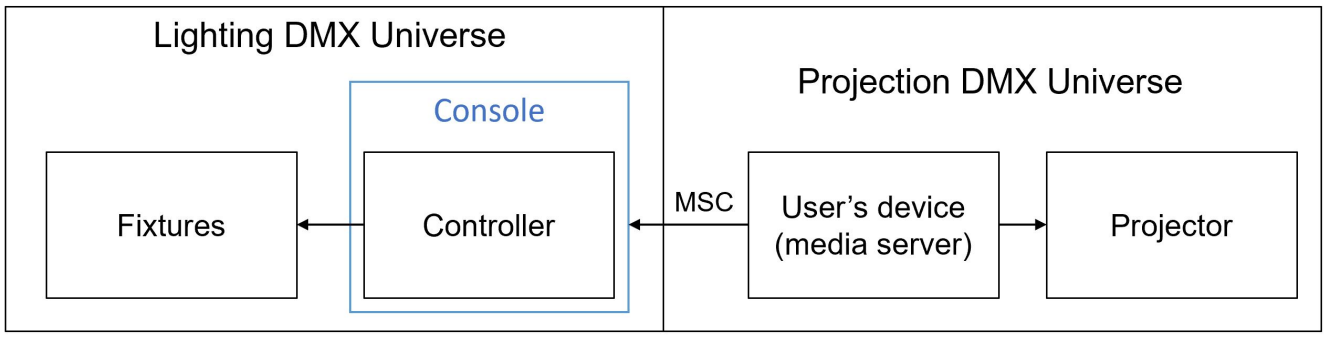


Figure 2.9: Complete lighting network.

2.4 Mapping Audio Parameters To Lighting Attributes

Later we discuss the MIR process that precedes the visual synthesis. These audio parameters influence the lighting and visualisation. Table 1 shows a mapping between the audio parameters of the song and the lighting parameters to align with the product specification.

Audio Parameter	Lighting Parameter	Section
Genre	Lighting Colour	4, 8
Downbeat	Strobe Frequency and Minor Changes	5
Segmentation	Significant Changes	6
Arousal (Energy)	Fixture Movement	7
Valence (Happiness)	Lighting Intensity	7

Table 1: Mapping between audio and the lighting parameters.

The downbeats are used to emphasise the pulse of the music with strobe lighting and minor changes (e.g., pulsing intensity), while also synchronising the lighting with the audio. The genre is used to create a colour mapping for the lighting. The arousal and valence factors describe the emotion of the music. A high arousal and valence phrase corresponds to fast fixture movement and high intensity respectively. Segments correspond to significant changes in the lighting scheme, such as transitions between verses and choruses.

3 Audio Pre-Processing - Felix Cohen

An issue common to many MIR tasks in machine learning is that a limited amount of data is available to train the algorithms on [15]. This makes it hard to learn robust patterns in the data effectively without overfitting to the specific noise of the dataset. This problem can be alleviated by pre-processing the data [16]. Pre-processing is a technique that uses prior knowledge and expectations to transform the audio data into features that capture relevant and informative aspects of the music. With no pre-processing, our initial input features are the amplitude values of the audio track over time. Starting with a specific combination of these features that reflects our prior knowledge of the problem domain will decrease the size of the search space for a solution. This means our algorithms are more likely to start at a point in the search space that is closer to an optimal solution. In this section, we will discuss various pre-processing methods that we will consider using as inputs to our MIR algorithms.

3.1 Pre-Processing Methods

An audio clip's spectral parameters can be extracted over time using the Short Time Fourier Transform (STFT). The STFT is a technique designed for non-stationary signals whose properties change over time such as music. Time-localised frequency information is extracted by dividing the audio track into time series of length N and computing their Fast Fourier Transform. The Fourier Transform decomposes a signal into its frequency components by projecting it onto a basis of complex sinusoids. To avoid spectral leakage these time series should be pre-multiplied by a hamming window and overlapped to avoid subsequent information loss. Taking the magnitude of these outputs and concatenating them in the time dimension will result in a spectrogram output, a matrix where element x_{ij} corresponds to the amplitude of frequency bin j at time frame i.

For a discrete time series of length N, a standard Discrete Fourier Transform calculation premultiplies the series by an $N \times N$ DFT matrix. This is of time complexity $\mathcal{O}(n^2)$. The Fast Fourier Transform improves upon this by recursively rearranging the time series by interlaced decomposition so that the DFT matrix can be decomposed into a sequence of sparse matrices that are less computationally intensive to multiply. This is of time complexity $\mathcal{O}(n \log(n))$. Due to the recursive nature of this technique, the best results are achieved for a time series of 2^n, therefore the input time series should be zero-padded so it is the appropriate length.

The FFT offers clear time savings over the DFT however, increasing the frame window size by a factor

of N will still lead to an increase in run-time. The number of FFT calculations required will only decrease linearly whilst the FFT calculation time increases non-linearly. Although decreasing frame size will lead to a larger output to analyse, which could increase run-time, this can be remedied by max pooling results along the time axis as shown by Grill and Schulter [17]. Spectrograms will be needed as input for many different information retrieval models that will be used in our product. Therefore to minimise the computational expenditure of the combined MIR model, it is beneficial to choose a time series length N and an overlap percentage that will lead to a good performance for all. This way the spectrogram of an audio track only has to be calculated once thus reducing run-time.

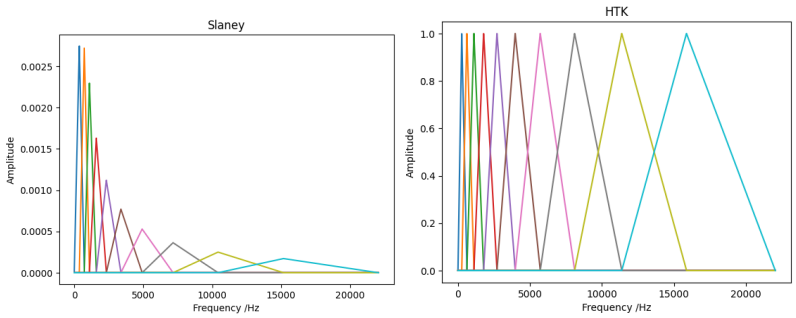
In order to meet the product specification it is vital to be examining parameters that correspond to human perception. As humans perceive frequency non-linearly, the frequency scale must be mapped to a new frequency scale such that a linear increase in this new frequency domain corresponds to a perceived linear increase in pitch. This frequency scale is known as the Mel scale and there are multiple methods for its calculation. The Librosa python music analysis library [18], a library designed for MIR, offers the choice of two methods: the Slaney Tool Kit implementation [19] and the HTK Hidden Markov Model Tool Kit implementation [20].

The general functionality of both methods is to apply triangular filter banks that are equally spaced on the Mel-frequency scale to a frequency spectrum. These filters are applied with the aim of recreating the bandpass filtering theorised to occur in the ear [21]. HTK creates M unit height triangular filter banks, with their peaks being equally spaced in the Mel domain and the bandwidth of each bank being lower and upper bounded by the proceeding and succeeding peak locations respectively. The location of these peaks in the Mel-frequency domain f_{mel} are then mapped to the standard frequency domain f by:

$$f = 700(10^{f_{mel}/2595} - 1) \tag{2}$$

The mapped filters can now be applied to the signal. The Slaney Tool Kit differs from this in how it decides upon peak locations, being linearly spaced up to 1000Hz, after which they are spaced logarithmically. The filter-banks are also equalised to have unit area rather than unit height. An example of 10 Slaney filter-banks and 10 HTK filter-banks for a time series of length 1s sampled at 44100Hz have been plotted in Figure 3.1.

Ganchev [22] compared the effectiveness of these methods and discovered that the Slaney toolkit returned slightly superior results. This, combined with the fact that this is Mcfee et al's choice of default for



(a) Slaney. (b) HTK.

Figure 3.1: Slaney and HTK Mel Filter Banks.

the Librosa library [18], has led us to decide to use the Slaney Tool Kit.

In order to normalise the Mel-spectrogram, each spectrogram element x should be mapped as so:

$$x_{new} = \log_{10}(x + c) \tag{3}$$

where c is a parameter to be decided upon. The value of c prevents errors caused by taking $\log_{10}(0)$ and determines the extent to which low amplitude frequencies are magnified. An example log Mel spectrogram is pictured in Figure 3.2. The log Mel-spectrogram can be used as an input to MIR algorithms, for example, it will serve as input for our downbeat tracking algorithm in Section 5.2.1. Alternatively, the log Mel-spectrogram can be used to extract further parameters.

One such set of parameters is the Mel Frequency Cepstral Coefficients (MFCC), these preserve coarse

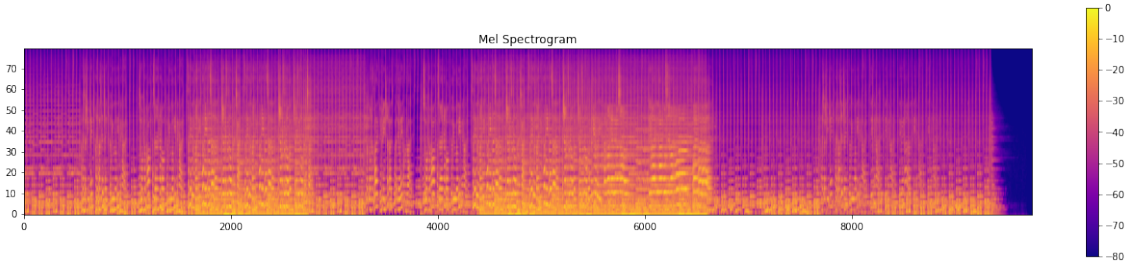


Figure 3.2: Log Mel spectrum of Yumi Zouma's rendition of Oasis's Champagne Supernova using 80 Mel Filter-banks.

spectral shape whilst discarding fine harmonic structure due to pitch [23]. To extract MFCCs the type II Discrete Cosine Transform (DCT) is taken for each time frame's Log Mel Spectrum. For a time series of length N, the DCT will return N coefficients that correspond to the basis functions $y_n[k]$:

$$y_n[k] = \cos\left(\frac{\pi(2n+1)k}{2N}\right) \qquad k \in [0, N-1] \quad n \in [0, N-1] \tag{4}$$

The first basis function is a constant and therefore only corresponds to the average log energy of a signal, as a result, this is often omitted in analysis [24]. The lower frequency basis functions control the broad shape of a spectrum whilst the higher frequency functions correspond to fine spectral detail.

An alternative set of coefficients are the Pitch Class Profiles (PCPs), these capture the harmonic content of an audio signal by mapping the energy of a frequency spectrum to an octave-dependent histogram [25].

A relatively new filter-bank method incorporates the filter-banks into the front end of a given neural network such that their parameters will be learnt over time. H sets of filter-banks are each applied to an input spectrogram of time-length T. The h-th set has filter-bank peaks centred at the h-th harmonic of a set of F centre frequencies $h \in [1, H]$. These outputs are stacked forming a harmonic-frequency-time tensor of size $H \times F \times T$. This provides harmonic pitch information without sacrificing temporal-spectral locality as the PCPs do [16]. The equation for the h-th harmonic filterbank Λ_h of center frequency f_c is shown below:

$$\Lambda_h = g\left(1 - \frac{2 - |f - h \cdot f_c|}{(h \cdot \alpha f_c + \beta)/Q}\right) \tag{5}$$

The parameters α, β, Q are learnt values that control the bandwidth. All centre frequencies f_c are also learnt values and $g(\cdot)$ is a rectified linear function. The number of centre frequencies alongside the number of harmonic filterbanks is chosen by the user. One benefit of this is that both harmonic and timbral information is preserved. Comparatively, MFCCs preserve only timbral information, and PCPs preserve only harmonic information. Another benefit is that it can learn and optimise the performance of the task at hand.

3.2 Conclusion

Overall, each pre-processing method makes a different assumption about the problem domain and therefore encourages the algorithm to tend to different areas in the solution search space. Each input's strengths can be leveraged to varying effects depending on the given task. For example, a song's harmonic features may contain information that is relevant to extracting its emotive content but not its downbeats. Choosing the right pre-processing method is therefore a vital first step in designing our MIR algorithms.

4 Music Genre Classification - Dain Jung

Music genre classification is the practice of categorising music tracks based on their distinct musical content. Machine learning enables the automation and the simplification of this procedure, which reduces manual labour and enhances the user experience.

4.1 Literature Review

4.1.1 Importance of Music Genre Classification

Today, modern music encompasses diverse styles that are fundamentally different from traditional music, which leads to a more ambiguous concept of music genre. The lack of consensus on genre categories nowadays makes users rely on search phrases like ”more like this” by providing examples of music in a particular style [26]. Despite this trend, some people still search for music based on genres, demonstrating the continued importance of music genre classification. According to a study conducted by Lee et al. [27], when searching for music, people use genre and other metadata in addition to collective knowledge such as reviews and recommendations. The relevance and necessity of music genre classification are further emphasised by interactive tools like *Musicmap* [28]. By providing a visual history of popular music genres, *Musicmap* facilitates students' comprehension of the intricate relationships and evolution of different genres over time.

Music genre classification plays an important role in creating engaging visualisations. Accurate genre classification considers multiple genre-specific features, such as tempo, harmony, and texture [29]. By identifying the correct genre, a visualiser can make visual representations that are tailored to each genre.

Music visualisation with accurate genre prediction also aids in music structure analysis [30], and promotes comprehension of the relationships between different genres [31]. It makes it simple for users to comprehend their musical preferences in greater depth. This can contribute to the development of more personalised and effective music recommendation systems, hence improving the user's experience [32].

In Hsiao's study [33], stage lighting technicians were asked to select a colour of light that matched the features of the music. Most professionals who took part in the study agreed that each genre of music has its own ”emotion-colour map” for lighting. This highlights the importance of accurate genre classification when creating immersive music visualisations.

4.1.2 Machine Learning Models for Genre Classification

Various machine learning models have been investigated for the purpose of the music genre categorisation problem. Tzanetakis and Cook [34] were one of the first to use supervised machine learning methods to classify music genres. They experimented with Gaussian Mixture models and k-nearest neighbour classifiers to find the best way to classify music genres effectively.

Support Vector Machines (SVM) have also been used for music genre classification. Changsheng Xu et al. [35] compared the effect of using SVM with different distance metrics. Convolutional Neural Networks (CNN) have been more popular in recent years, due to their success in image classification tasks. Wyse [36] trained CNN to classify genres by using spectrograms, which are pictures that show the frequency content of a signal. Li et al. [37] also developed a CNN that could predict music genres by using the raw MFCC matrix as input. Hidden Markov Models (HMM) are widely used in speech recognition tasks. Scaringella and Zoia [38] and Soltau et al. [39] applied HMM to music to classify genres based on their temporal structure and achieved promising results.

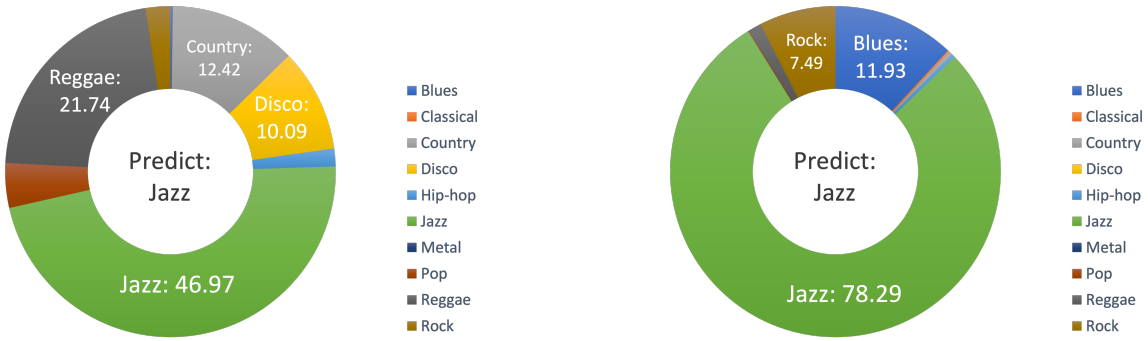
There have been attempts to classify genres in a short time. Bisharad et al. [40] used Residual Networks (ResNet) to train a model with 3-second audio clips taken from the GTZAN dataset. They considered overlapping features of different genres and achieved 94% accuracy. Chillara et al. [41] conducted a study comparing various pre-existing models to find the best machine learning algorithm for music genre classification. The models were trained on MFCCs and other features of the songs. The study found that deep learning techniques, especially CNNs, can achieve the highest accuracy in classifying genres.

4.2 Using Multi-Class Probabilistic Classifier

Music genre classification is a multi-class classification problem where a data point can belong to more than one class. It is a complicated task, as tracks from different genres can share similar properties, and there is often variation in the musical features within a single genre. However, this can be solved effectively with a multi-class probabilistic classifier.

Music tracks often show characteristics of more than one genre, which makes it difficult to put them into a single genre. A multi-class probabilistic classifier not only predicts the label for each input, but also the probability that the input belongs to each class. This enables us to view music genres as a continuum as opposed to discrete categories.

Figure 4.1 demonstrates the probability distribution of genres assigned by the multi-class probabilistic classifier. This helps audiences to understand the complex relationships between genres more intuitively. The example tracks are represented as a pie chart, with colours corresponding to the different genres and their respective probabilities. In Figure 4.1, the two different tracks are both predicted to be a jazz music. In 4.1a, Track 1 is predicted to be 47% jazz and 22% reggae, which indicates that the jazz music also has some characteristics of a reggae genre. It has a very different mood and music features compared to Track 2, a music track with characteristics of 78% jazz and 12% blues in 4.1b. An effective visualisation should present each of them differently. However, a traditional classification model would treat these two tracks similarly because the model's classification process only involves assigning the music track to the genre with the highest probability.



(a) Track 1. (b) Track 2.

Figure 4.1: Probability distribution of genres for example tracks.

4.3 Process

The flowchart presented in Figure 4.2 provides a systematic overview of the genre classification process. Beginning with dataset selection and preparation, the process advances through feature extraction, model selection, and training. The final stage involves evaluating the chosen models' performance.

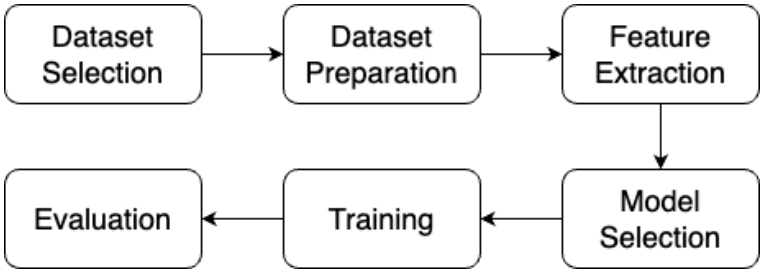


Figure 4.2: Flow chart of the genre classification process.

4.3.1 Dataset Selection

To obtain reliable result, it is essential to select an appropriate dataset. There are several datasets publicly available for researchers, each with its own characteristics and benefits. After reviewing the literature, we compared the key features and limitations of each dataset, as shown in Table 2.

Dataset	Main Features	Limitations
Million Song Dataset[42]	Large collection of metadata for million music tracks	No audio files, no genre labels
FMA[43]	106,574 tracks from 161 genres and sub-genres	Non-commercial music only
Subset of FMA	8,000 audio clips from FMA, evenly distributed across 8 genres	Non-commercial music only
ISMIR Genre[44]	1,000 audio clips of 10 seconds each, evenly distributed across 6 genres	Mostly Western music
MagnaTagATune[45]	26,000 audio clips with user-provided tags for genres and moods	Subjective and inconsistent tags
DSD100[46]	100 complete songs with separated original music sources	Small number of songs
Audio Set[47]	Human annotated database of over 2 million samples of audio events	Focuses on sound events, not music-related contents
GTZAN[48]	1,000 audio tracks, evenly distributed across 10 genres	Mostly Western music

Table 2: Comparison of music datasets for genre classification.

Based on the features, GTZAN dataset was chosen for our project. Before using this dataset, we also considered its limitations and characteristics, as described in Sturm's studies [49]. This dataset was suitable because it has a balanced number of genres from various recording environments.

4.3.2 Dataset Preparation

Before training the model, we have to ensure that the audio signals in the GTZAN dataset are properly pre-processed. Raw audio data often contains unwanted noise or artifacts. To solve these problems, pre-processing techniques such as noise removal or filtering are applied.

Noise removal helps eliminate background noise or other irrelevant sounds from the audio signal. We used a spectral subtraction method [50], which reduces noise by approximating the average noise magnitude in the noise-only section and subtracting it from the entire signal.

Filtering techniques can put more or less emphasis on certain frequency components of a signal, making

related features stand out more. We applied a bandpass filter to focus on the frequency range of human hearing (20Hz-20kHz). This filtering helps eliminate high-frequency and low-frequency parts that are irrelevant to the genre.

After pre-processing, the data is split into training and test sets. This process is essential, as it reviews the performance of the model in real situations with the unseen data. We chose to assign 80% to the training set and reserve the remaining 20% for the testing set. This ensures that we have a large amount of training data to construct an accurate model, with a sufficient size of test set for evaluation. We used stratified sampling to make sure that each class is well-represented in both the training and test sets.

4.3.3 Feature Extraction

Raw audio data is difficult to analyse and interpret directly due to its complexity and high dimensionality. Feature extraction converts the raw audio data into a set of meaningful and distinctive features. These features simplify the analysis process by providing a more concise and interpretable representation of the essential information in the signal. The quality of the extracted features has a strong effect on the model's ability to accurately differentiate between various music genres.

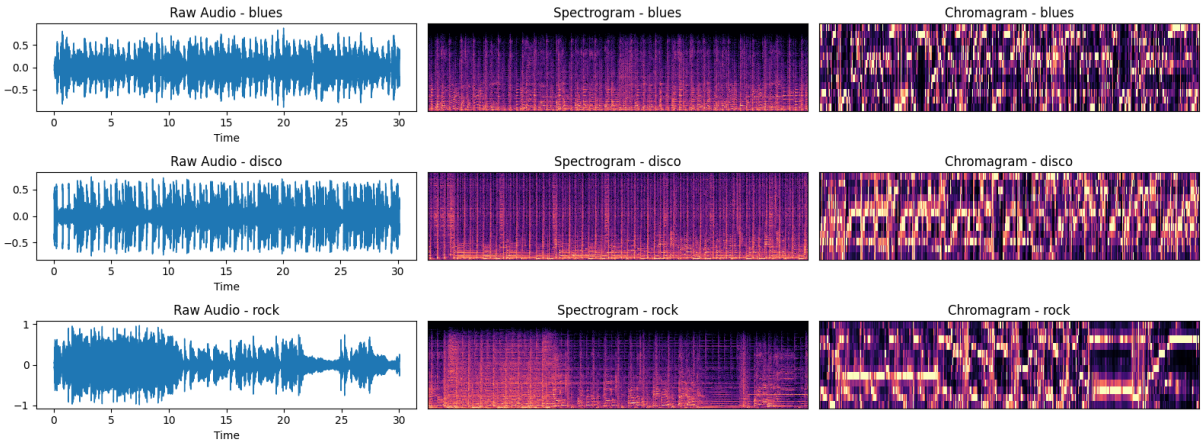


Figure 4.3: Raw audio signals and their spectral features.

We used *Librosa* for feature extraction [18]. *Librosa* is a Python library specifically designed for audio and music analysis tasks. It provides a wide range of feature extraction functions and utilities. A set of essential features that capture the characteristics of each music genre was extracted. These extracted features can be grouped into two categories: spectral and temporal features. The spectral features include Chromagram, Mel-Frequency Cepstral Coefficients (MFCCs), Root Mean Square (RMS) Energy, spectral centroid, spectral roll-off frequency, and zero-crossing rate. Figure 4.3 illustrates the information

captured by the feature extraction process, highlighting the differences in features across genres. On the other hand, the temporal features include rhythm and tempo.

A custom function was used to perform the feature extraction and to compute the various statistics such as mean and variance. Each music file was given as an argument to the function. After feature extraction, the features were normalised by removing the mean and scaling to unit variance to ensure that all features contributed equally to classification. This step is necessary because some features might have significantly larger or smaller values than others, leading to biased classification results.

4.3.4 Model Selection

It is essential to carefully evaluate various machine learning models and choose the ones that meet the objectives of the project. This section discusses the process of model selection, including the rationale behind our choices and a comparison of the different models.

Softmax Regression, Support Vector Machines (SVM) with Platt Scaling, and Convolutional Neural Networks (CNN) with Softmax were identified as potentially applicable machine learning models for our project. These models were chosen based on the decision matrix shown in Table 3. The criteria considered were their ability to generate probability estimates for each class, the required computational resources, and their ability to handle multi-class problems.

Model	Multi-class (0.2)	Probability (0.3)	Complex Patterns (0.2)	Computing Time (0.3)	Score	Selected
Softmax Regression	0.8	1	0.6	1	0.88	Yes
SVM with Platt Scaling	0.8	1	1	0.7	0.87	Yes
CNN with Softmax	1	1	1	0.5	0.85	Yes
RNN with Softmax	1	1	0.9	0.2	0.74	No
Random Forest	0.9	0.3	0.7	0.6	0.71	No
k-NN	0.8	0.7	0.5	0.6	0.65	No

Table 3: Decision matrix for machine learning models for music genre classification.

Softmax Regression Softmax regression is a logistic regression for multi-class problems. This method assumes that each data belongs to precisely one class, which may not be accurate for all music genres (0.8). Softmax function, which transforms an input vector into a probability distribution, models the probability of each class (1). Softmax regression is simple, and it performs well in linearly separable problems. However, it often lacks flexibility due to its linearity. Therefore, it is inadequate for complex problems

(0.6). Despite this limitation, its low computational complexity makes it suitable for large datasets and real-time applications (0.8).

SVM with Platt Scaling Support Vector Machine (SVM) is known for showing good performance at non-linear problems. SVM is designed for binary classification problems but can be extended to solve multi-class problems using one-vs-one or one-vs-rest techniques (0.9). Furthermore, the addition of Platt Scaling to SVM transforms the output into a probability distribution (1). SVM with Platt Scaling shows high accuracy and robustness in complex multi-class problems (1). However, it can be computationally expensive, particularly when large datasets or high-dimensional features are involved (0.7).

CNN with Softmax Activation Function Convolutional Neural Networks (CNN) can effectively handle multi-class classification problems (1). They have achieved high performance in dealing with data that has spatial or temporal structure, such as images or audio signals. When a Softmax activation function is used in the final layer, they can generate probability estimates for each class (1). CNNs can be computationally expensive and require large amount of training data (0.5). They may also require specialised hardware like GPUs. However, they are exceptionally good at discovering complex patterns in large-scale data (1).

RNN with Softmax Activation Function On the other hand, Recurrent Neural Networks (RNN) is another type of deep learning model for multi-class classification problems (1), suitable for sequential data. Like CNN, they can also output class probabilities with a softmax layer (1). RNN is also good at finding complex patterns (0.9). However, training RNN can be computationally demanding compared to CNN, especially when dealing with long sequential data (0.2). Moreover, RNNs are also more prone to issues like vanishing and exploding gradients.

For our project, CNN was preferred over RNN because our audio features are fixed-length representations rather than sequences with variable lengths. CNNs show better performance than RNNs at finding local patterns and structures in input data with a fixed size.

Random Forest Random Forest is a model that constructs multiple decision trees and combines their output to make predictions. It shows high performance when dealing with multi-class problems (0.9). It can handle complex non-linear relationships without the risk of overfitting, but it might not be as effective as CNN at capturing complex patterns (0.7). We did not choose Random Forest mainly because it cannot

produce probability estimates directly (0.3).

K-Nearest Neighbours K-Nearest Neighbours (k-NN) is a simple, instance-based algorithm. It classifies samples based on their similarity to the k nearest samples in the training set. It can handle multi-class problems (0.8). Although k-NN can provide probability estimates, it does this based on the proportion of neighbours that belong to each class, which is sometimes unreliable (0.7) [51]. It shows a disappointing performance when the classes are not separated clearly, or the feature space has many dimensions (0.4). K-NN only needs a little processing power during the training phase, but it requires calculating the distance between the query point and all training points when using the model (0.6). Because of this, it might not be the best choice for our project.

4.3.5 Training

We trained the selected models, Softmax Regression, SVM with Platt Scaling, and CNN with Softmax, using categorical cross-entropy as the loss function, which effectively handles multiple class labels. The categorical cross-entropy loss function is defined as follows:

$$L(y, \hat{y}) = -\sum_{i=1}^{C} y_i \log(\hat{y}_i) \tag{6}$$

where C is the number of classes, y_i is the true probability of class i, and $\hat{y}_i$ is the predicted probability of class i. This function measures the dissimilarity between the predicted probability distribution of class labels and the true probability distribution, penalising incorrect predictions more heavily when the model's confidence is high.

4.4 Evaluation

In this section, we evaluate the performance of our trained models, Softmax Regression, SVM with Platt Scaling, and CNN with Softmax Activation Function, on the validation set by comparing the predicted probabilities to the true genre labels. We used evaluation metrics such as log loss, Brier score, and accuracy to assess the model's performance.

4.4.1 Log Loss and Brier Score

Log loss and Brier score assess the quality of probabilistic predictions made by a classifier. Both metrics penalise incorrect predictions more heavily when the model's confidence is high.

Log loss is defined as:

$$L(y, \hat{y}) = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{C} y_{ij} \log(\hat{y}_{ij}) \tag{7}$$

where N is the number of samples, C is the number of classes, y_{ij} is the true probability of sample i belonging to class j, and $\hat{y}_{ij}$ is the predicted probability of sample i belonging to class j. A lower log loss indicates better performance.

Brier score is defined as:

$$BS(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{C} (y_{ij} - \hat{y}_{ij})^2 \tag{8}$$

A lower Brier score indicates better performance. Log loss is usually more sensitive to extreme probabilities than Brier score.

4.4.2 Accuracy

Training and test accuracy are evaluation metrics that measure the ratio of the number of correct predictions to the total number of predictions made. Comparing the test and training accuracy allows us to assess the model's generalisation ability and detect overfitting. In our models, accuracy was computed by comparing the predicted genre labels with the ground truth labels.

4.4.3 Evaluation Metrics Comparison

To compare the performance of the three chosen models, we compared the evaluation metrics (log loss, Brier score, test accuracy, and train accuracy) of the three chosen models.

Table 4 summarises the evaluation metrics for each model. The CNN with Softmax model has the lowest log loss and Brier score, suggesting better probabilistic predictions. On the other hand, the SVM with Platt Scaling model has the highest train accuracy, which may indicate overfitting, as its test accuracy is similar to the other models. The CNN with Softmax model provides a good balance of performance and generalisation, with the highest weighted average precision and comparable evaluation metrics to

the other models.

Metric	Softmax Regression	SVM with Platt Scaling	CNN with Softmax
Log Loss	0.834	0.706	0.551
Brier Score	0.0389	0.0342	0.0261
Test Accuracy	71.7%	76.3%	81.5%
Train Accuracy	74.5%	80.3%	91.5%

Table 4: Evaluation metrics for Softmax Regression, SVM with Platt Scaling, and CNN with Softmax Activation Function.

4.4.4 Classification Report Comparison

We analysed the classification reports of the three models, which include precision, recall, and F-1 score for each genre. Precision measures the proportion of true positive predictions among all positive predictions, and recall measures the proportion of true positive predictions among all actual positive instances.

Imagine the model is trying to identify jazz songs. Precision looks at how many songs the model labelled as jazz is actually jazz. For example, if the model labels 10 songs as jazz, but only 7 of them are jazz, then the precision is 70%. Higher precision means the model is reliable when it predicts a song is a jazz. In contrast, recall is the total number of jazz songs that the model was able to find. For example, if there are 20 jazz songs in total, and the model correctly identifies only 7 of them, the recall is 35%. A higher recall indicates that the model is capable of detecting jazz songs. In order to get a balanced view of these metrics, we employed the F-1 score, the harmonic mean of precision and recall [52]. More details about the methods of comparison is explained later in Section 6.1. As shown in Table 5, CNN outperforms other models in every metrics.

Metric	Softmax Regression	SVM with Platt Scaling	CNN with Softmax
Precision	0.72	0.76	0.82
Recall	0.72	0.76	0.81
F-1 score	0.72	0.76	0.81

Table 5: Classification Report Metrics for Each Model.

4.4.5 Confusion Matrix

Table 6 shows the confusion matrices for each model. Each row represents the actual class, while each column represents the predicted class. The diagonal entries of the matrix represent the number of correct predictions, while the off-diagonal elements show misclassifications. The higher the diagonal values of

the confusion matrix, the better, indicating many correct predictions.

	blues	**classical**	**country**	**disco**	**hiphop**	**jazz**	**metal**	**pop**	**reggae**	**rock**
blues	140	1	12	3	2	12	13	0	9	8
classical	2	189	1	1	0	5	0	0	0	1
country	17	1	118	9	2	14	3	6	5	24
disco	1	4	5	132	6	0	10	7	10	25
hiphop	7	1	3	16	121	0	6	12	33	1
jazz	12	8	6	2	0	168	0	1	0	3
metal	4	0	3	3	5	0	170	0	3	12
pop	0	3	6	13	6	2	0	154	7	9
reggae	8	1	9	3	13	3	4	12	139	8
rock	19	1	22	26	6	10	5	2	6	103

(a) Softmax Regression.

	blues	**classical**	**country**	**disco**	**hiphop**	**jazz**	**metal**	**pop**	**reggae**	**rock**
blues	149	0	9	7	2	11	8	0	6	8
classical	1	190	0	0	0	5	0	0	0	3
country	16	0	129	6	2	10	4	4	5	23
disco	0	2	4	138	8	1	7	8	7	25
hiphop	4	1	3	15	138	0	3	9	26	1
jazz	5	9	3	1	0	180	0	0	1	1
metal	1	0	1	1	3	0	180	0	2	12
pop	0	3	8	11	6	1	0	160	7	4
reggae	8	1	7	2	14	1	0	10	148	9
rock	19	0	17	17	6	4	3	2	10	122

(b) SVM with Platt Scalng.

	blues	**classical**	**country**	**disco**	**hiphop**	**jazz**	**metal**	**pop**	**reggae**	**rock**
blues	170	1	3	2	2	8	6	0	4	4
classical	0	192	0	0	0	7	0	0	0	0
country	16	3	136	3	1	13	3	4	7	13
disco	2	1	3	141	11	2	9	5	8	18
hip-hop	1	2	3	3	157	0	4	2	23	5
jazz	4	8	3	0	0	185	0	0	0	0
metal	2	0	1	2	2	0	186	0	2	5
pop	0	2	7	9	5	1	2	156	9	9
reggae	8	2	7	2	8	3	0	0	163	7
rock	8	2	13	7	5	6	6	2	9	142

(c) CNN with Softmax Activation.

Table 6: Confusion matrix of machine learning models.

The confusion matrices show that the CNN with Softmax model has the highest diagonal values, meaning it made the most correct predictions among all three models. For example, in the Softmax Regression model, 140 blues songs were correctly classified, while in the CNN with Softmax model, the number of correctly classified blues songs increased to 170.

The confusion matrices also show the types of misclassifications made by each model. For instance, the Softmax Regression model often confuses country with blues, with 17 country songs being misclassified as blues. The SVM with Platt Scaling model has a similar issue but with fewer misclassifications. The

CNN with Softmax model shows a significant reduction in this type of misclassification, with only 16 country songs misclassified as blues.

Table 7 shows an example of an extended confusion matrix with prediction probabilities. The first elements represent the correct predictions, while the second and third columns show the probabilities of the classifier guessing the correct genre in the second and third chance. The updated confusion matrix helps us understand the model's confidence in its predictions.

	Correct	2nd	3rd
blues	70	6.5	6
classical	94.97	2.51	1.01
country	59.3	12.06	8.54
disco	66	12.5	5
hiphop	60.5	16.5	8
jazz	84	6	4
metal	85	6	2.5
pop	77	6.5	4.5
reggae	69.5	6.5	6
rock	51.5	13	11

(a) Softmax Regression.

	Correct	2nd	3rd
blues	74.5	5.5	4.5
classical	95.48	2.51	1.51
country	64.82	11.56	8.04
disco	69	12.5	4
hiphop	69	13	7.5
jazz	90	4.5	2.5
metal	90	6	1.5
pop	80	5.5	4
reggae	74	7	5
rock	61	9.5	8.5

(b) SVM with Platt Scaling.

	Correct	2nd	3rd
blues	85	4	3
classical	96.48	3.52	0
country	68.34	8.04	6.53
disco	70.5	9	5.5
hip-hop	78.5	11.5	2.5
jazz	92.5	4	2
metal	93	2.5	1
pop	78	4.5	4.5
reggae	81.5	4	4
rock	71	6.5	4.5

(c) CNN with Softmax Activation.

Table 7: Updated Confusion Matrix of Machine Learning Models.

4.5 Conclusion

Following the analysis of evaluation metrics, the classification report, and the extended confusion matrix, it is clear that CNN with Softmax Activation outperforms other models in music genre classification. This model provides robust performance and high F-1 score across the majority of musical genres. It also shows a nuanced understanding of the genre spectrum, which makes it the best model for this task.

However, the model has difficulty when classifying 'blues, 'country' and 'rock' genres. These genres share common characteristics that may blur the model's decision boundaries. Future work could focus on improving the model's performance in these specific genre classifications. This could be done by incorporating more genre-specific features or using more targeted data augmentation techniques to help the model distinguish these genres more clearly within the genre spectrum.

5 Downbeat Tracking - Oli Wing

5.1 Background

Metric structure is essential to dance music; repeating patterns of strong and weak beats supported by an unchanging pulse let the listener dance to the music [53]. The pulse is a series of uniformly spaced beats, either audible or implicit that sets the tempo and contextualises the rhythm. In contrast to rhythm which can be complex and difficult to match, almost any listener can instinctively follow the pulse of song by tapping at a constant tempo despite variations in the rhythm of the sounds on top of the pulse. Pulses that are accented at constant intervals are known as downbeats, and pulses in between downbeats are beats [54]. Meter is the measurement of the number of beats between regularly occurring downbeats.

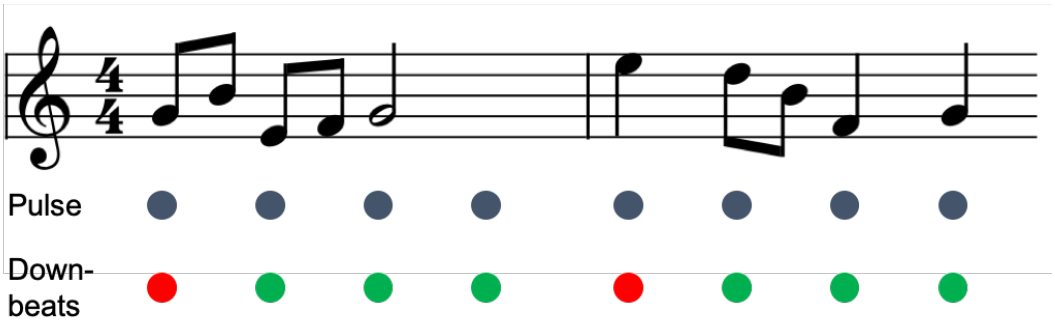


Figure 5.1: Example of the downbeats on a short piece of music.

Typically, electronic dance music is temporally quantised into bars 4 beats long with a downbeat always occurring on the first beat of the bar. It also uses repetition of musical ideas, over phrases multiple bars long. These predictable features allow the listener to follow the song and dance to it more easily.

Visualisers and lighting schemes play an important role in enhancing listening to dance music; they complement the music by reflecting the musical and emotional features of the audio. Because downbeats have more energy than regular beats and underpin the rhythmic structure of the song, they must be emphasised to create an effective visual representation of a song. For lighting, this is done by synchronising the strobe frequency to the tempo and phase of the audio, and increasing the intensity of the light on downbeats. For a visualisation, this is done by using a sequence that changes certain parameter values (e.g colour) on every downbeat as illustrated below in figure 5.2. The capabilities of the visualiser are explained further in section 9.

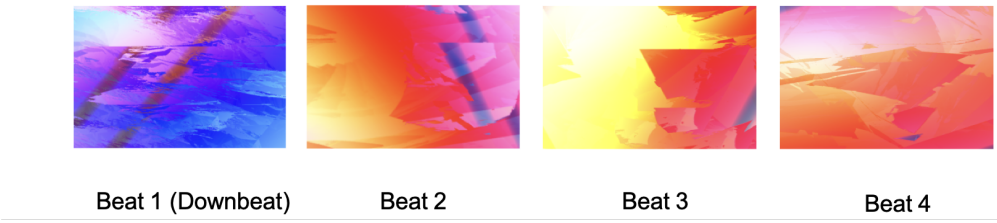


Figure 5.2: Example visualisation showing how a distinct visual change (changing the colour to blue) occurring on the downbeat emphasises the downbeat and creates energy and movement.

Thus, it is necessary to know the position of downbeats in the audio for the visualisation and lighting to be fully synchronised, which is crucial in creating a cohesive and immersive experience for the audience. After finding the location of the downbeats and assuming 4 beats per bar, it is simple to calculate a songs tempo. Tempo is needed by other features of our software; it is a parameter that is used to classify genres (explained further in section 4). Downbeat locations are also useful for structural segmentation as segment boundaries always occur on downbeats.

5.2 Current Models

Within the field of music information retrieval, automatic inference of metric structure (beat tracking) is a fundamental problem. Although tracking the beat of a song is intuitive for humans, automating this process for a machine is much more challenging. Nevertheless, beat tracking algorithms have improved dramatically in accuracy in the past decade. At their core, most methods follow a similar process to that in the diagram shown in figure 5.3. Initially, some features such as onset, amplitude envelopes, spectral information or chord changes are extracted from the signal. The next stage involves determining periodicities (length of recurrent sections of the signal) to estimate tempo and using the phase of the periodic signal to estimate the beat locations. This can be done using autocorrelation.

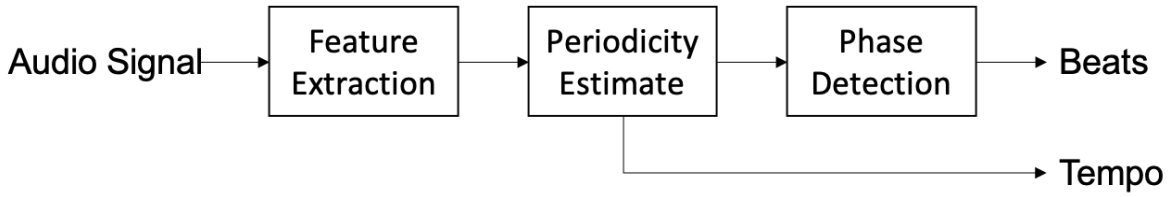


Figure 5.3: Flowchart of typical beat tracking algorithm.

Identifying downbeats add another layer of complexity to the problem. The above method is unable to distinguish regular beats from downbeats. Currently, the best performing algorithm with software imple-

mentation [55], and the one chosen for this project, uses a more advanced version of this template. First, the signal is pre-processed to extract relevant features. Then, a bi-directional Long Short Term Memory network calculates activation functions estimating the probability that a given point in the signal is a downbeat, a beat or no beat. Lastly, a Dynamic Bayesian Network (DBNs) is used to directly infer both tempo and beat position from the activation functions produced by the network. The process is shown in figure 5.4.

Figure 5.4: Flowchart of downbeat tracking algorithm.

We discuss how this algorithm works as it is used in [55], and what modifications we chose to make it more suitable for visual synthesis. Then we explain the current software implementation of this algorithm and how it would fit with the rest of our design. Finally, we briefly cover the limitations and discuss how downbeat tracking in our software could be improved in the future with a superior algorithm.

5.2.1 Pre-Processing

The downbeat detection algorithm uses some of the pre-processing techniques described in section 3 but with additional steps. First a raw pulse code modulated (PCM) signal sampled at 44.1kHz is used as an input. To simplify the computational process, stereo signals are converted to a monophonic signal by taking the average of the two channels. To ensure the network is able to capture information from the signal precisely in both time and frequency, this discrete signal is then segmented into overlapping frames of length 1024, 2048 and 4096 samples long. For each segmented signal a standard Hann window $w(l)$ of the same length is applied to the frames to reduce spectral leakage [56]. This increases the dynamic range as it prevents signal components of similar magnitude and frequency from getting swamped. A Short-Term Fourier Transform (STFT) is then used to calculate the complex spectrogram $X(n,k)$ with:

$$X(n,k) = \sum_{l=-\frac{W}{2}}^{\frac{W}{2}-1} w(l) * x(l+nh) * e^{-2\pi jlk/w} \tag{9}$$

where n Is the frame index, k is the frequency bin index and h the time shift in samples between adjacent frames. The complex spectrogram is converted into the power spectrogram $S(n, k)$ using the equation:

$$S(n, k) = |X(n, k)|^2 \tag{10}$$

Psychoacoustic knowledge is used to reduce the dimensionality of the power spectra. This is achieved with the use of a filterbank of 20 triangular filters placed evenly on the Mel scale which transforms the spectrogram into the Mel spectrogram $M(n, m)$. By grouping the thousands of frequency bins into 20 frequency bands, the dimensionality is significantly reduced while still being high enough resolution to capture the behaviour of the signal across the whole spectrum. As noted in section 3, humans do not perceive volume equally across all frequencies [57]. To reflect this, a logarithmic scale is chosen and the spectrum is converted using the formula below:

$$M(n, m) = log\ (S(n, k) * F(m, k)^T + 1.0) \tag{11}$$

1.0 is added to the magnitude values to ensure they are positive before taking the logarithm. When classifying downbeats it is important that human loudness perception is accounted for as downbeats are a psychoacoustic phenomenon and a product of the way the human brain processes rhythmic audio. Using a logarithmic scale means the algorithm “hears” the song the same way a human would and can more accurately estimate the position of the downbeats.

Finally, to aid the network in training, the positive median first order difference (effectively the gradient) is provided to the network along with $M(n, m)$ as [58] found empirically that this improved the performance of the network at classifying downbeats. Figure 5.5 and figure 5.6 below shows a visual representation of the Mel spectrogram and the first order difference as they are presented to the neural network.

5.3 Neural Networks

Once the signal has been pre-processed, it is sent to a Long Short-term Memory (LTSM) network. LTSM networks are a specific type of recurrent neural network (RNN). To explain these concepts, it is necessary to explain the underlying principles of neural networks.

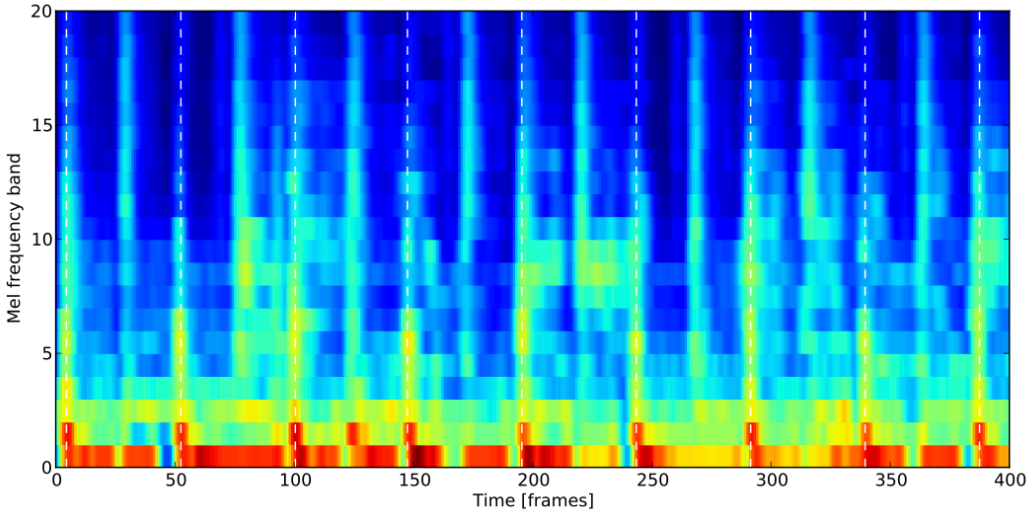


Figure 5.5: Mel Spectrogram. Image from [58].

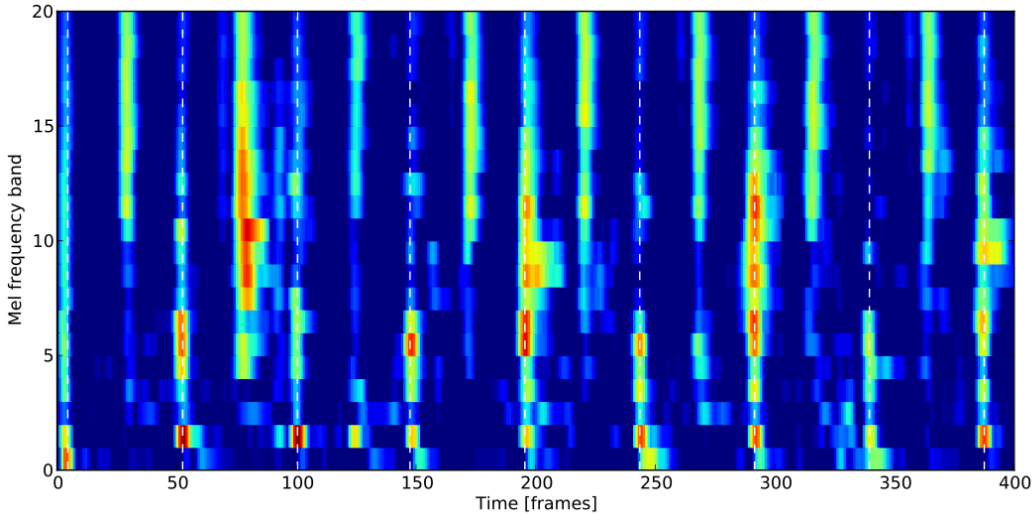


Figure 5.6: Positive Median First Order Difference. Image from [58].

5.3.1 Basics

The most basic type of neural network is a type of feed forward neural network (FNN) called a multi-level perceptron (MLP) which consists of an input layer, an output layer and 1 or more hidden layers [59][60]. In an FNN the neurons in the hidden layers only have connections to the layers before and after it, never to themselves or other neurons in the same layer (known as cyclic connections). The network is strictly causal meaning the output is computed directly from present input values, independent of past or future values.

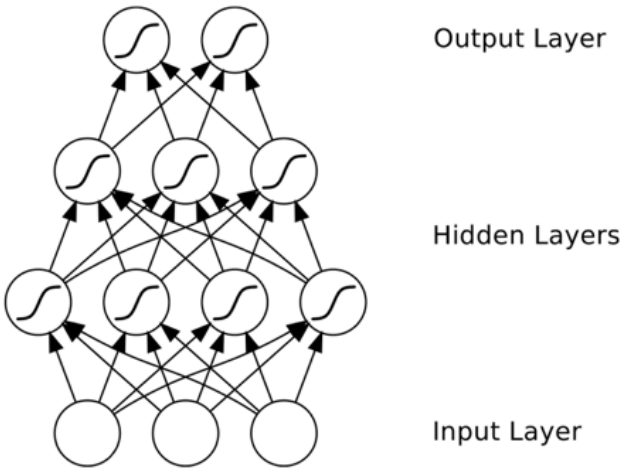


Figure 5.7: Multi-layer perceptron. Image from [59].

Each neuron in the hidden layer takes a sum of the weighted input connections (weights w input x), denoted a_h. The activation function θ is applied and the final activation of the neuron is given as b_h.

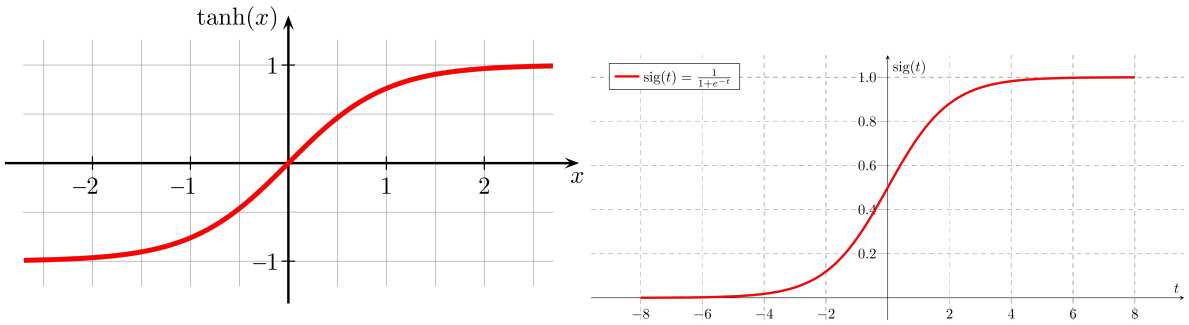
$$a_h = \sum_{i=1}^{I} w_{ih} x_i \tag{12}$$

$$b_h = \theta_h * a_h \tag{13}$$

5.3.2 Activation Functions

The two most common choices for activation functions are the hyperbolic tangent and logistic sigmoid. This is due to their ability to map any input value to a value between -1 and 1 and 0 and 1 respectively (see figures 5.8a) and 5.8b), as well as their non-linearity. Any combinations of linear functions are always linear, meaning a multi-layered MLP with linear activation functions can be reduced to an equivalent

network with a single layer. In contrast, non-linear functions are significantly more powerful as layers can be stacked to re-represent input data [61].



(a) Hyperbolic Tangent. (b) Logistic Sigmoid.

Figure 5.8: Non-linear activation functions. (a) from [62], (b) from [63].

Both the tanh and logistic sigmoid functions are differentiable, a property that plays a central role in training the network.

5.3.3 Loss Functions

Neural networks use loss functions to calculate the error between the estimation that the network outputs and the true value. There are several functions that can perform this task depending on the purpose of the network. For classification tasks such as downbeat tracking, cross-entropy loss is used [64]. The estimated probability of each class is compared to the known desired output (either 0 or 1 corresponding to true or false respectively) and a loss value is calculated that penalises based on how far the estimate is from the true value. Cross entropy is defined as the following:

$$\mathcal{L} = -\sum_{i=1}^{n} t_i log(p_i) \tag{14}$$

for n classes where t_i is the truth label of the class and p_i is the softmax probability. As the function is logarithmic, it yields a large value for differences close to 1 (a bad estimate) and low values for differences close to 0 (a good estimate). The objective of the network thus becomes to minimise the loss function.

For example, if a given frame gives the softmax probabilities $[0.90, 0.08, 0.02]$ for downbeat, regular beat and no beat respectively with the truth labels $[1, 0, 0]$ (Downbeat), $[0, 1, 0]$ (Regular Beat), and $[0, 0, 1]$ (No

Beat) , the downbeat activation loss value $\mathcal{L}$ for that frame is:

$$-1 * log(0.90) - 0 * log(0.08) - 0 * log(0.02) = -log(0.90) = 0.046$$

0.046 is close to 0 as expected as the softmax probability of the frame being a downbeat was 0.90, which is close to 1. This is explained further in section 5.3.10.

5.3.4 Backwards Pass

As neural networks are, by design, differential operators, they can be trained to minimise any differentiable loss function using gradient descent. The basic concept is taking the derivative of the loss function with respect to each of the networks weights and adjust the weights in the direction of the negative slope, thus minimising the loss function. To calculate the loss function gradient at each stage of the network, a technique known as a backwards pass is used [60][64]. The first step is to differentiate the loss function with respect to network outputs. Then, working backwards through the hidden layers with repeated application of chain rule algebraically derives the equation below describing the derivative of the loss function with respect to the network weights:

$$\frac{\partial \mathcal{L}(x,z)}{\partial w_{ij}} = \frac{\partial \mathcal{L}(x,z)}{\partial a_j} b_i \tag{15}$$

$\mathcal{L}(x,z)$ is the loss function, w_{ij} is the weight of the connection between node i and node j, a_j is the weighted sum of input connections to node j, and b_j is the final activation value of node i.

5.3.5 Training

Although it can be demonstrated algebraically how neural networks can be trained using gradient descent, it is important to discuss the practical implementation of this with a gradient descent algorithm. The simplest method is known as steepest gradient or just gradient descent; small fixed sized steps are repeatedly taken in the direction of the negative slope of the loss function.

$$\Delta w^n = -\alpha \frac{\partial \mathcal{L}}{\partial w^n} \tag{16}$$

Δw^n is the nth weight update, w^n is the weight before Δw^n is applied and $\alpha \in [0,1]$ is the *learning rate*.The learning rate determines how much to change the model in response to the estimated error each time

the model updates. Too small a value and the network will take too long to train since a smaller change in weight with each update requires more epochs; too large a value and the network is trained faster but may converge on a sub-optimal value. This process is repeated until some *stopping criteria* is met, such as a failure to reduce loss for a given number of epochs. A significant downside to gradient descent algorithm in its current form is that it can easily become stuck in local minima. This problem is solved with the addition of a *momentum* term.

$$\Delta w^n = -\alpha \frac{\partial \mathcal{L}}{\partial w^n} + m\Delta w^{n-1} \tag{17}$$

$m \in [0, 1]$ is the momentum parameter and Δw^{n-1} is the change in weight of the previous epoch. This term effectively gives inertia to the motion of the algorithm as it moves along the loss function, increasing convergence speed and bypassing local minima [65]. This is illustrated in figure 5.9.

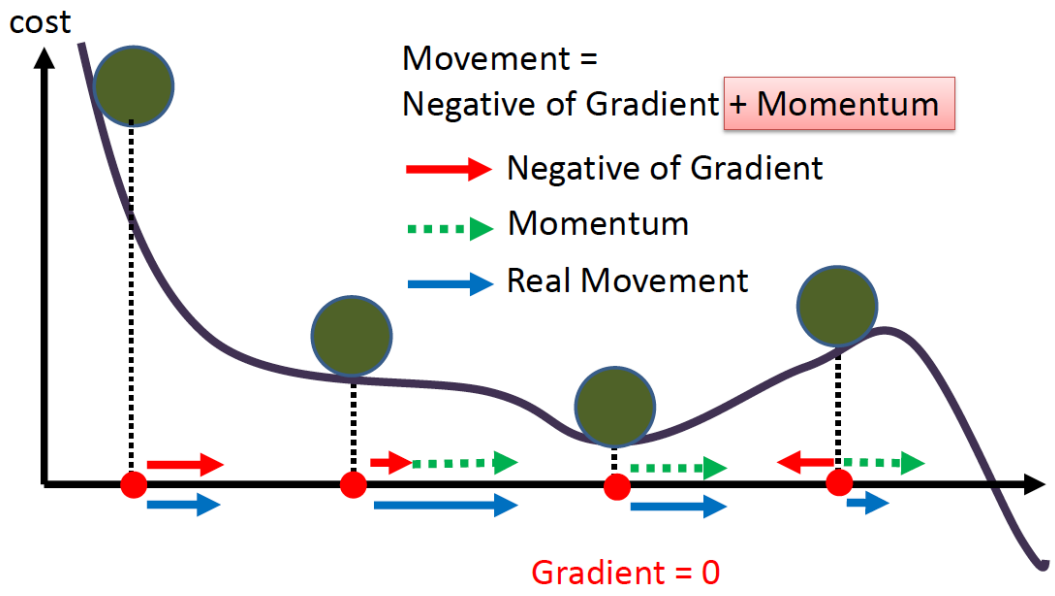


Figure 5.9: Movement of the loss function with momentum component. Momentum prevents the loss function from getting stuck in the local minima. Image from [66].

5.3.6 Stochastic Gradient Descent

A variation of this algorithm is Stochastic Gradient Descent (SGD) which differs slightly in how the loss function is calculated and the network is updated. For regular gradient descent, the network weights are not updated until the entire training dataset has been inputted to the network. The mean gradient of the dataset is calculated and the network weights are updated in the negative direction of this gradient.

This is in contrast to stochastic gradient descent where the network is updated after a single instance from the dataset (in random order) is processed. This method has important benefits such a significantly increased learning speed due to not having to process the entire dataset through the network every epoch. The drawback of this method is a reduction in effectiveness at converging to the exact minimum due to the higher variance of each update causing large fluctuations in the loss function. However, by slowly decreasing the learning rate, it has been shown that SGD exhibits identical convergence behaviour as gradient descent that uses the entire dataset each epoch. The higher fluctuation also improves the algorithms ability to avoid local minima as the loss function is different for each training example.

5.3.7 Recurrent Neural Networks

If cyclic connections within the hidden layers are allowed, the network is considered recurrent. While FNNs are only capable of mapping the last input to each output, RNNs can in theory map the entire history of all inputs to each output. The recurrent connections allow a 'memory' of previous inputs to persist in the networks internal state and therefore influence the output. This is through a technique known as Backpropogation Through Time (BPTT) [67]. It is a modification of a backwards pass where each neuron with a cyclic connection is represented in past states of time as well as the usual interactions in an FFN. Like regular backpropogation, repetition of the chain rule backwards through the network from output to input is used to adjust weights to minimise the loss function.

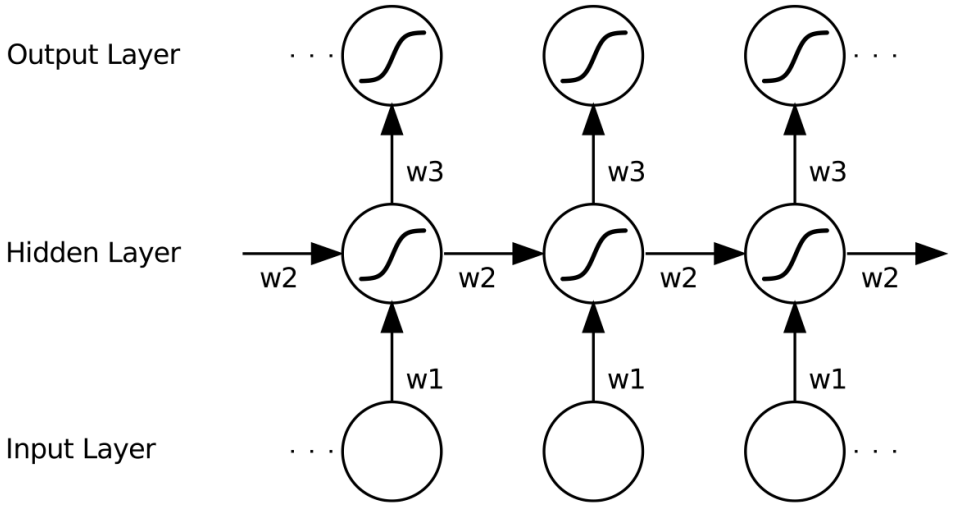


Figure 5.10: An RNN in unfolded form. Each section represents a cyclic node at a single timestep. Image from [59].

However, cyclic connections introduce a new problem. Inputs to the network either decay to zero or blow

up exponentially as they continually loop through the cyclic connections. Exploding gradient values can be prevented by using small initial weights for the network but this does not account for decaying values in which the gradient tends to 0. This phenomenon is referred to as the vanishing gradient problem. The gradient decays to 0 as the algorithm moves temporally backwards through the network, repeatedly multiplying gradients together. This is due to the nature of the sigmoid function always having a derivative less than 0.25. Consequently, the weights across the entire network become inaccurate; certain weights are barely updated as the gradient of the loss function at that point in the network is close to 0.

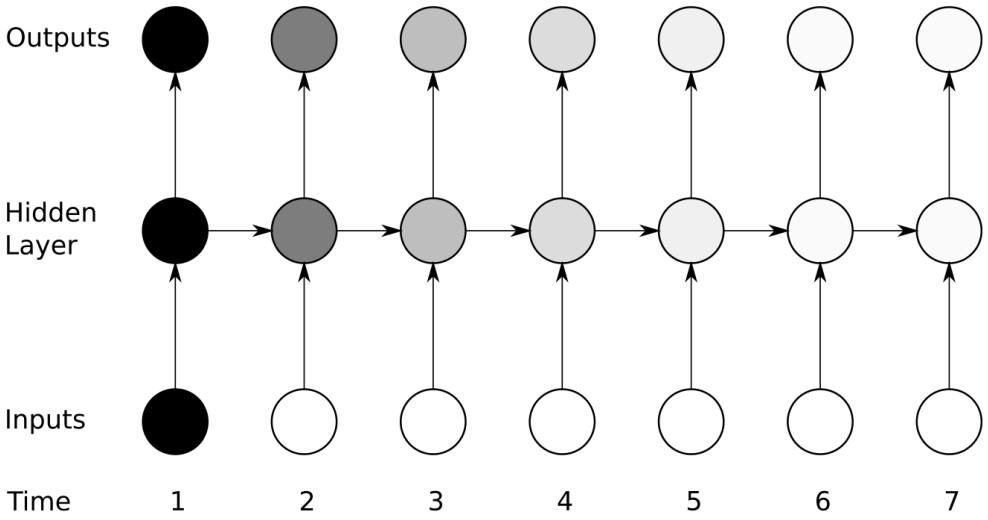


Figure 5.11: Vanishing gradient of an RNN. Lighter nodes indicate a gradient closer to 0. Image from [59].

5.3.8 Long Short Term Memory

Conveniently, there are several solutions to the vanishing gradient problem in RNNS, one of which is known as Long Short Term Memory (LSTM). A LSTM network is the same as a standard RNN but with memory blocks replacing the non-linear operators in the hidden layers.

Memory blocks contain an input, output and forget gate. Each gate is a non-linear summation unit that takes activations from inside and outside the block, and controls the activation of the block via multiplications. All other connections within the block have a fixed weight of 1. The multiplicative gates allow LTSM blocks to store and access information over long periods of time which negates the vanishing gradient problem. For example, if an input gate remains closed (has an activation close to 0) then that block cannot be overridden with new inputs arriving in the network and can be made available to the network

later in time by opening the output gate. This preserves gradient information over the entire duration of training [59].

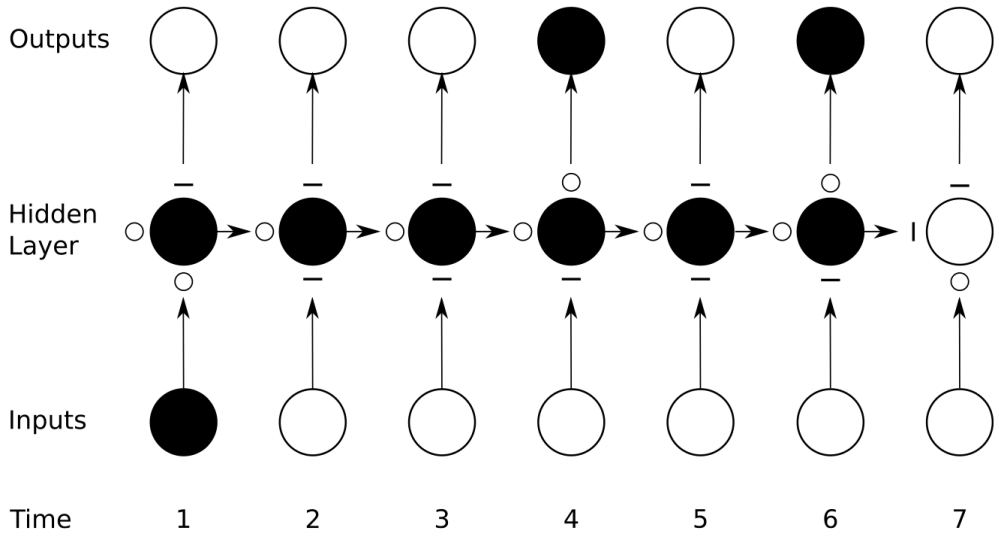


Figure 5.12: Preservation of gradient information in LSTM network. Image from [59].

5.3.9 Bi-Directional LSTMs

For many sequence labelling tasks including downbeat detection where causality is unnecessary, it is beneficial to have access to both past and future inputs. Bidirectional networks allow this by presenting each training sequence in both forward and reverse order to 2 separate recurrent hidden layers, with both sets of hidden layers being combined at the same output layer. Doing so provides the output with the complete past and future context of every point in the input sequence; useful for classifying downbeats as downbeat location depend on the entire past and future audio signal, not just the signal up to a certain point in time. Bidirectional recurrent neural networks (BRNNs) [68] have been shown to consistently outperform unidirectional RNNs for sequence labelling tasks.

5.3.10 Output Layer

Downbeat detection is essentially a classification problem; the model estimates the probability that a given frame is either a beat but no downbeat, downbeat, or no beat. To achieve this, the output layer uses 3 neurons each with a softmax activation function. The softmax activation function is a generalisation of the logarithmic sigmoid function for more than 2 classes [69]. The softmax function is chosen over a sigmoid function as softmax functions account for the fact that the probability values for each class are

not independent and more importantly because it will always classify an input into a single class. This is necessary for the subsequent DBN to infer the meter and downbeat positions more easily.

$$softmax(z_i) = \frac{exp(z_i)}{\sum_j exp(z_j)} \tag{18}$$

The softmax function effectively compares the relative probabilities between the output neurons and normalises the activation values so the total sums to 1.

Thus, the output of the neural network are 3 activation functions b_k, d_k, n_k, the probability of frame k being a beat but no downbeat, downbeat, or no beat respectively. Classification is simple - for a given frame, the class corresponding to the neuron with the highest activation value is chosen.

5.4 Dynamic Bayesian Networks

Once the processed signal has been fed through the bidirectional LSTM network and the 3 activation functions for beat, downbeat and no beat have been obtained, it is possible to determine the tempo and beat/downbeat locations. This is done most effectively using a Dynamic Bayesian Network (DBN) to jointly infer the tempo and beat/downbeat positions by exploiting the fact that these 2 properties of the audio are interrelated. Compared with a threshold and peak picking approach, DBNs are better at dealing with ambiguous RNN results and are shown to consistently produce more accurate results [55]. Before explaining how DBNs are utilised for the specific task of tracking downbeats, it is helpful to provide a brief overview of how they function.

5.4.1 Overview

DBNs are used to model probabilistic systems that change with time. A sequence of observable variables $\mathbf{Y} = [y_0, y_1 \ldots y_k]$ is modelled by assuming that each observation is dependent on a hidden state $\mathbf{X} = [x_0, x_1 \ldots x_k]$ [70]. The sequence of hidden states **X** behave as a first order Markov model: the state at time t is dependent only on its immediate past (the state at time $t-1$) [71].

The joint probability of the hidden and observable states can be expressed as follows:

$$P(X, Y) = \prod_{t=1}^{T-1} P(x_t|x_{t-1}) \prod_{t=1}^{T-1} P(y_t|x_t) P(x_0) \tag{19}$$

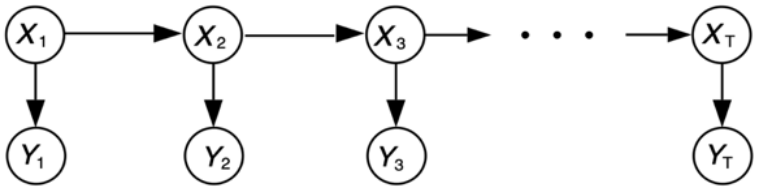


Figure 5.13: Hidden states **X** and observations **Y** of a Dynamic Bayesian Network. Image from [70].

The right-hand side of the expression contains 3 sets of probability density functions (pdf). First, the state transition pdfs $P(x_t|x_{t-1})$ - the time dependencies between hidden states at time t and $t-1$. Then the observation pdfs $P(y_t|x_t)$ – the dependencies of observation nodes with respect to hidden states at time t. Lastly the initial state distribution $P(x_0)$ – the pdf at time $t=0$. Unless prior information is known then a uniform distribution is typically used.

In the downbeat tracking algorithm used, the basic principle is to use the 3 activation functions with the activation values at every frame as the set of observations, each with a corresponding hidden state. Using a process called inference, the values of each hidden state can be calculated. By designing the hidden state space appropriately, this reveals the position of the beats, downbeats and tempo simultaneously.

5.4.2 State Space

The hidden state space **X** is divided into 3 discrete states. These states $s(\phi, \dot{\phi}, r)$ exist in a 3-dimensional space indexed by the bar position state ϕ, the tempo state $\dot{\phi}$ and the time signature state r. States that fall on bar position $\phi = 1$ are considered downbeats and states that fall on a beat position are considered beats. The number of bar position states for a given song is a function of the tempo. This is necessary to keep resolution for different tempos consistent; if a fixed number of bar positions is used then songs with lower tempo lose resolution as a longer section of signal is divided into the same number of cells as a song at higher tempo with shorter bars [72]. Combined with the fact there are more features extracted per bar (activation function frames per bar) at lower tempo as features are extracted at constant rate, this causes a mismatch between the time resolution of the activation function and the resolution of the discretised bar position. To avoid this the number of bar positions can be set equal to the number of frames per bars using:

$$M(T) = round\left(\frac{r * 60}{\dot{\phi} * \Delta}\right) \tag{20}$$

where $\dot{\phi}$ is the tempo in beats per minute (bpm), r is the number of beats per bar and Δ being the audio frame length. The number of tempos depends on the tempo range accounted for, and is distributed logarithmically between the minimum and maximum tempos. This is to preserve resolution at lower tempos, as if tempos were distributed linearly then the percentage change in tempo would increase as the tempo got slower.

5.4.3 Observation Model

The activations of each frame can be converted into observation distributions $P(o_k|s_k)$ by

$$P(o_k|s_k) = \begin{cases} d_k & s_k \in \mathcal{D} \\ b_k & s_k \in \mathcal{B} \\ \frac{n_k}{\lambda_0 - 1} & otherwise \end{cases} \tag{21}$$

where $s_k \in \mathcal{D}$ and $s_k \in \mathcal{B}$ are all frames belonging to the sets of downbeats and beats respectively. These are frames of the activation function with a value greater than 0.5 as the activation functions d_k and b_k outputted by the neural network are limited to the range $[0, 1]$ and show high values at beat/downbeat positions and low values elsewhere. λ_0 is a parameter relevant to refining beat and downbeat selection (see section 5.4.5).

5.4.4 Inference

The goal of the DBN is to find the sequence of hidden states $\mathbf{s}_{1:K}$ that maximises the posterior probability of the hidden states given the observations (activations of the network). In other words, finding the most probable downbeat and beat positions given the activation functions provided by the neural network. The maximum posterior state sequence $\mathbf{s}^*_{1:K}$ is given by

$$\mathbf{s}^*_{1:K} = argmax(P(\mathbf{s}_{1:K}|o_{1:K})) \tag{22}$$

The argmax function can be computed efficiently with the Viterbi algorithm [73].

5.4.5 Refining Selection

As explained early, the sequence of beat and downbeat times is determined by the set of time frames assigned to a beat or downbeat state based on activation function value. Having decided on a sequence, the exact times are refined by searching for the highest activation value inside a window of size $\frac{\Phi}{\lambda_0}$. Böck et al. [55] found that $\lambda_0 = 16$ achieved the best results. This final step refines beats and downbeat locations to a single frame which is a more useful output.

5.5 Improvements

Although the down beat tracking algorithm proposed by Böck et al. is a good basis for our product, there are several improvements to be made. Most of these improvements take advantage of the fact that our software is only used to analyse electronic dance music (rather than all genres of dance music as it was used for in [55]), so simplifications can be made as the inputs to the algorithm are more homogenous. This improves both computation speed and overall accuracy. Additionally, we will briefly discuss a recent paper proposing a downbeat tracking algorithm specifically designed for Drum and Bass [74] that marginally outperforms [55] but is currently without implementation.

5.5.1 Neural Network Improvements

The current network used in madmom is trained on a very wide dataset of which electronic dance music is only a small subset. By retraining the network on a dataset consisting of majority EDM songs, it will be able to identify beat and downbeat locations more reliably. As the network will be more familiar with the input as with more EDM songs in the training dataset, the chances that the network has been presented with a similar song are higher which will make classification more accurate. As well as this, the activation threshold for the output nodes would be increased. It is low in the current implementation because in [55] the algorithm was also used to analyse pieces with overall low activation values (choral works). However, as nearly all the songs being analysed by our software have clear beat locations so will have high activation values, pieces with low activation need not be accounted for. This means that musically irrelevant intro sections that negatively affect the performance of the algorithm (as they have little to no transient information) are more likely to be ignored when determining beat and downbeat locations.

5.5.2 Dynamic Bayesian Network Improvements

In its current implementation, the DBN uses a 3-dimensional hidden state space with a state r representing tempo. For our software, this is redundant as nearly all EDM uses 4 beats per bar. With this assumption, the hidden state space can be reduced to 2 dimensions (bar position and tempo), greatly simplifying the computation thus increasing the efficiency of the algorithm.

Because tempo is modelled as a variable, there are slight inaccuracies in the location of beats (that are not downbeats) within bars. This does not prevent our software from functioning as the set of downbeats is still sufficiently accurate and can be used to infer the location of the rest of the beats. Assuming 4 beats per bar, the set of beats calculated by the DBN can be ignored and instead estimated as 3 beats at constant intervals between downbeats. Despite not using the beat set calculated by the DBN, it is still important to use a beat activation function (for beats not downbeats) in the DBNs observations $o_{1:K}$ as in [55] as downbeat tracking performance is hindered without it. Assuming it is constant, the tempo of the song can be easily calculated easily using the following formula:

$$\dot{\phi} = (average\left(\frac{(d_{k+1} - d_k) * l}{4}\right))^{-1} \tag{23}$$

where $\dot{\phi}$ is tempo and $(d_{k+1} - d_k) * l$ is the time between adjacent downbeats in seconds.

5.5.3 Alternative Algorithm

Even with optimisations, the current open source algorithm has shortcomings. The main issue is computation time; the DBN is still computationally intensive with 2 dimensions. Although it currently has no open source implementation, the method proposed in [74] solves this issue as their approach does not require a DBN to locate downbeats. Here we provide a brief overview of this method.

The first step is to locate beat positions. This algorithm assumes each song has a constant tempo meaning beat locations are defined by 2 parameters, period (or equivalently tempo) and phase (the time difference between the start of the audio signal and the first beat). The basis of this method is that repetition of musical onsets (transients) occur after an integer number of bars (in EDM 4) and the loudest onsets typically occur on beat locations. To locate onsets, an onset detection function (ODF) [75] is used which has a high value at onset locations and a low value elsewhere. The ODF works by splitting the signal into frequency bins, calculating the change in energy level of each bin with respect to time

(effectively the gradient of the signal), and using an argmax function to find values of period and phase that result in the highest autocorrelation of the ODF.

From the beat positions, the downbeats can be determined. To begin the audio is trimmed by removing the first and last sections with energy levels beneath a threshold (some fraction of the RMS energy of the song) to improve the performance of the downbeat classifier which struggles with inexpressive sections of audio. Then features including loudness [76] and energy level across the frequency spectrum are extracted from the beat segments. A logistic regression classifier is used to assign probability values (that a beat is the first, second, third or fourth measure of the bar) to each segment by comparing it to nearby segments. These predictions are then aggregated over entire song by summing each of the 4 possible trajectories and choosing the largest sum (most likely trajectory).

The result is an algorithm that is more accurate and significantly faster than the current madmom (see section 5.6) implementation. According to [74], the average analysis time is 4.5s per song with an average accuracy of 98.1 compared to madmoms 35.4s and 87.5 accuracy when tested on 160 songs. Despite outperforming on both speed and accuracy, the lack of open source software implementation means it is still beneficial to use the madmom downbeat tracker for the first iteration of our product as designing the product around existing software is much more efficient. However, it would be feasible to implement the algorithm [74] in the future when a software implementation becomes available due to the modular design of our software; each component is dependent only on the output of the other components and not their inner workings. It can be noted that although [74] was designed specifically to analyse drum and bass songs, the method applies to all forms of EDM and can be adapted by simply increasing the tempo range with negligible increase in computation time.

5.6 madmom

madmom is an open source Python library with a multitude of music information retrieval functions [77]. These are known as processors; one of which being a downbeat tracker utilising the algorithm described earlier. Each processor is a stand-alone program meaning the downbeat tracker can easily be implemented into our software. Other than the Python standard library, madmom requires only three additional libraries being NumPy, SciPy and Cython. Minimising the number of dependencies is important to make our software accessible to as many users as possible.

The processor DBNDownbeatTracker [78] uses the algorithm explained earlier 5.2, and will be the basis

for our downbeat tracker with the necessary adjustments made 5.5. The processor takes an audio file in any common format and outputs the timestamps of every downbeat and their relative position within the bar. Equation 23 can then be used to determine the tempo of the song.

Timestamps (s)	Beat Number
0.020	1
0.480	2
0.920	3
1.360	4
1.800	1
2.240	2
2.680	3
3.120	4
3.560	1
4.000	2
4.440	3

Table 8: Example output from DBNDownbeatTracker. Values on the left indicate temporal position of beats in seconds. Values on the right show the relative position of the beat within the bar. Downbeats occur on beat number 1.

Once a song has been analysed, these values can be stored as an array. This means that when the DJ selects that song to play, the array of downbeat and beat positions as well as the tempo can be passed quickly to the visualiser without need for the song to be analysed again.

5.7 Utilising Downbeats For Visual Synthesis

Downbeats are needed for both visualisation and lighting schemes. This is for three main reasons. The first is synchronisation. By accurately identifying downbeats, the visualiser and lighting can be synchronised with the underlying pulse of the music, improving the overall experience. This means that any repeating patterns of visuals in the visualiser (section 9.4) or lighting (strobe) occur at the same tempo as the music and are aligned in phase.

The second reason is visual emphasis. Downbeats carry more energy compared to regular beats in a song. Recognising the downbeats allows emphasis on these strong beats. This could be expressed through a lighting scheme by making the fixture movement faster or increasing the intensity of the light at timestamps on which downbeats occur. For the visualiser, one example is to create a sequence of 4 RGB values (determined by genre analysis 8) each mapped to a number 1-4, then using the sequence to change the colour of the visualiser by changing to the corresponding colour at the each timestamp (see

figure 5.14). This highlights the rhythmic structure and gives the downbeats more impact, making the visualisation more dynamic and engaging. A downbeat synchronised sequence could be used to change any number of parameters to create immersive visualisations and lighting schemes.

Lastly, downbeat locations are used for segmentation. This is because in EDM, segments (sections of the song with similar musical features) always have boundaries that occur on downbeats. Combining the segmentation analysis explained in section 6 with downbeat locations improves the ability to accurately segment the song, as timestamps for section boundaries are rounded to the nearest downbeat.

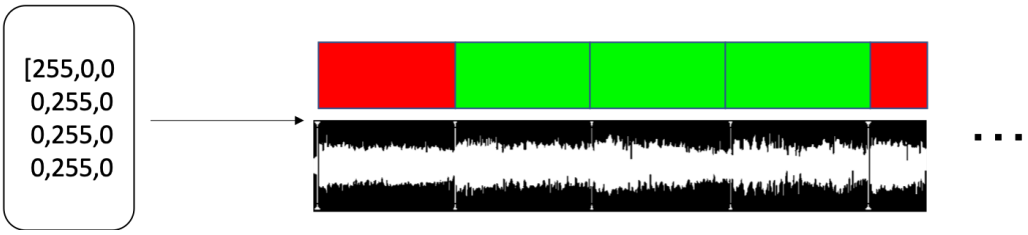


Figure 5.14: Diagram showing how a sequence of RGB values can be used to create a visualisation that changes background colour to emphasise the downbeats of a song.

6 Segmentation - Felix Cohen

A compositional structure is a feature that all songs possess. Any song can be naturally divided into different segments such as verse or chorus. By using an algorithm to locate boundaries between song segments, these segments can be analysed individually as separate entities, allowing for a unique lighting configuration for each segment. However, locating these boundaries is a non-trivial task. In this section, we will first discuss the principal ideas boundary calculation is based on, then our choice of measure of a segmentation algorithm's quality and then the choices of representation of an audio track. Finally, we will compare automated methods of analysis that can robustly interpret these representations. Our decisions will be based on the key criteria which impact the user experience: the quality of music information retrieval and the expected processing run-time of the algorithm.

Boundary location calculation relies on the assumption that audio frames that belong to the same song segment will be relatively similar whilst being adequately different to frames belonging to different segments [79]. Following this assumption, as one approaches a segment boundary, the level of novelty between frames will increase, peaking at the boundary location. Novelty can be understood as a measure of dissimilarity between consecutive audio frames. Therefore, the boundary location problem can be broken down into two sections: firstly analysing the parameters and representations at our disposal to output a measure of novelty throughout a song and secondly applying a peak picking algorithm to this novelty curve to decide upon these boundary locations. An example novelty (also known as boundary activation) curve is depicted below in Figure 6.1.

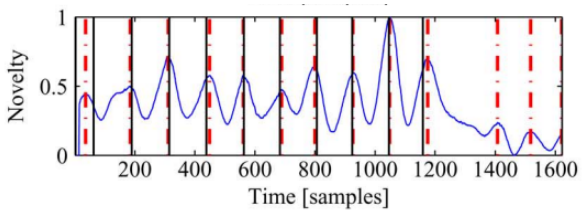
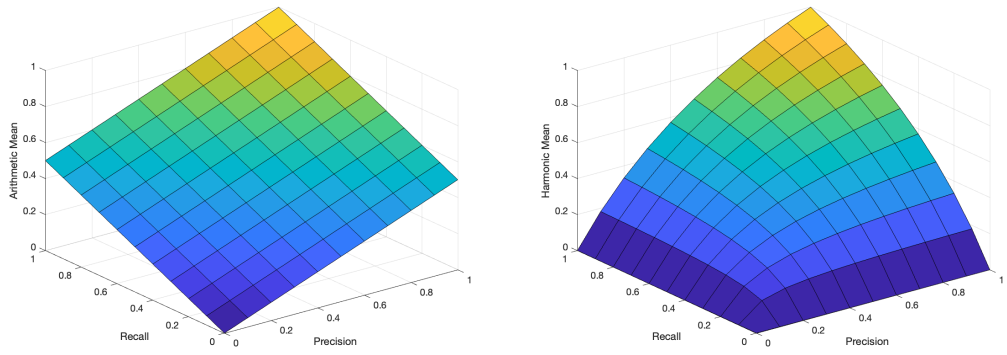


Figure 6.1: Example Novelty Curve taken from [25], bold black lines correspond to ground truth whilst red dashed lines are outputs of the peak-picking algorithm.

6.1 Methods of Comparison

In order to choose an appropriate segmentation algorithm we must be able to quantitatively measure and compare them. One method commonly used in the literature is the F-1 measure [15][24][80]. This evalu-

ates information retrieval algorithms' performance by calculating the harmonic mean of their precision and recall. As stated in Section 4, precision is the ratio of correct positive classifications to total positive classifications whilst recall is the ratio of correct positive classifications to all possible positive classifications. Both of these attributes are needed to truly characterise an information retrieval algorithm's performance. Consider a boundary classification algorithm that correctly classifies one boundary but makes no other classifications. Although this will return a precision score of 100%, it is clear that its true performance is quite poor. The harmonic mean is used as we are averaging over a common numerator. This has the added benefit of heavily penalising a poor performance in either precision or recall as a high value in both is needed to return an adequate score. The arithmetic and harmonic means have been plotted in Figure 6.2 to demonstrate this effect. For an algorithm with a precision of 1 and recall of 0, the arithmetic mean returns 0.5 whilst the harmonic mean adequately penalises this combination, returning 0.



(a) Arithmetic Mean (b) Harmonic Mean

Figure 6.2: Example plots of the arithmetic and harmonic mean of precision and recall

Nieto et al. [81] performed an experiment to determine whether the F-1 Measure effectively predicted a listener's preference in segmentation algorithm. This was achieved by using both a characteristically high precision algorithm and a characteristically high recall algorithm to annotate song segment boundaries for one-minute excerpts. The results were then filtered so that each algorithm's F-1 measure performance on an excerpt was within 5% but their precision and recall performance was greater than 10% apart. The quality of each algorithm's boundaries was then rated by participants with an average of 2.8 years of musical training. Logistic Regression Analysis found that the F-Measure could not predict listener preference at a statistically significant level and that precision was more perceptually relevant. This provides sufficient evidence that an additional composite measure of precision and recall is needed.

The F-1 measure is derived from the more generalisable F-α measure:

$$F_\alpha = (1+\alpha^2)(\frac{P}{\alpha^2 P + R}) \tag{24}$$

$$\alpha = \frac{R}{P} \mid \frac{\partial F_\alpha}{\partial P} = \frac{\partial F_\alpha}{\partial R} \tag{25}$$

Equation 5 demonstrates that α is a weighting parameter that controls the relative contributions of precision P and recall R to the overall F-α Measure. As demonstrated in Figure 6.3, it is only optimal to increase an algorithm's recall performance over precision up to the point that $R = \alpha P$. The area of the graph to the right of the line $R = \alpha P$ is the Recall-Precision space where it is beneficial to improve recall over precision. Nieto et al. [81] swept through the values of α from 0 to 1 and performed Logistic Regression analysis. This revealed that the F-measure's predictive power increased as α decreased. However, as there was no data gathered for large differences in precision performance and recall, it was speculated that this positive correlation between P-R and preference would in fact not hold at large differences. To ensure a contribution from an algorithm's recall performance such that false negatives are still punished, the largest value of alpha was chosen such that the Hosmer and Lemeshow test [82] returned a p-value less than 0.01 (a lower p-value indicates superior predictive power), this α value was 0.58.

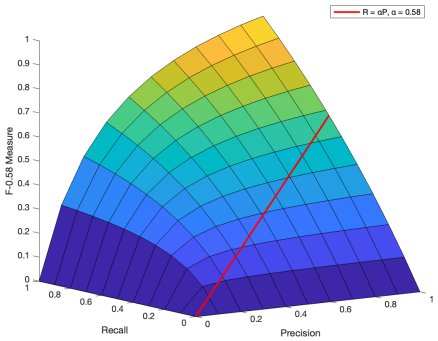


Figure 6.3: Plot of the F-0.58 Measure and the line segment $R = \alpha P$ (in red) at which it is equally beneficial to increase recall and precision. In this example $\alpha = 0.58$.

However, as stated by Grill et al [24], the F-0.58 measure is not very well established and therefore should be used in conjunction with the standard F-1 measure. Furthermore, there are various criticisms of the Hosmer Lemeshow test being able to output multiple results depending on how a researcher tailors its parameters [83]. Overall, despite the clear need for further research into this measure, we believe that there is enough evidence indicating that the F-0.58 measure is statistically relevant and should at least

be considered when benchmarking segmentation algorithms.

Another decision is the level of tolerance to use when deciding if a classification is correct. Common tolerances are either a 0.5s or 3s region around the boundary [84]. Whilst employing a strict criterion of 0.5s is more useful in distinguishing between two algorithms' performance at identifying boundaries to the nearest beat, the more lenient tolerance of ±3s is helpful in assessing an algorithm's ability to differentiate between broad sections in a song for scenarios where exact timing is less critical. Depending on the genre of the live performance, either could be a more useful measure, a song with a simpler compositional structure may not need the fine precision of ±0.5 second tolerance. Overall, we believe that a ±0.5s tolerance should be used as it will be more discriminative. However, future research endeavours should focus on developing a collection of algorithms, each trained to meet specific performance metrics or cater to a particular niche, such as a tolerance measure of ±3s.

Another important factor to consider when interpreting a structural segmentation algorithm's F-Measure results is the feasible range of values it can take. An algorithm that returns equally spaced boundary guesses will return an F-measure higher than 0. Therefore a lower bound to the F-measure can be established via the use of an algorithm in which this spacing has been chosen optimally [85]. Similarly, the inherent subjectivity of where a boundary falls in a song leads to an upper bound on an algorithm's best possible performance. This upper bound can be calculated by selecting all items in a data set that have been annotated multiple times and using the difference between these annotations to calculate a maximum F-measure. F-measure results should be compared considering these upper and lower bounds. Grill et al.[24][17] have calculated multiple bounds depending on their specific testing data set. When comparing algorithms in Section 6.5 we will use their values that had been calculated on the largest data set [17]. On this note, there is evidence that the F-measure may have a larger explorable range than the F-0.58 measure. The upper and lower bound of the F-1 and F-0.58 measures calculated from Grill et al.'s dataset [24] demonstrated that while they share an upper bound of 0.72, the F-1 measure has a smaller lower bound of 0.14 (compared to 0.19) and therefore may be both more sensitive and precise. Overall, given the product specification is to create a lighting system that is perceived to react to music, we have chosen to give the F-0.58 Measure higher precedence over the F-1 Measure. Algorithms that perform highly in this more perceptually relevant measure will more precisely satisfy this specification. However, it is important to also include the F-1 measure as some papers may not record the F-0.58 performance of their model.

6.2 Self Similarity Matrices

There is a multitude of indicators for a change in song segment that a person can easily and intuitively recognise such as changes in timbre, pitch, or song dynamics. To create an effective segmentation model, it is important to choose a representation of an audio track that adequately captures these indicators.
A commonly used representation in structural segmentation is the Self Similarity Matrix depicted in Figure 6.4. Element x_{ij} of an audio track's SSM is a measure of the similarity between frame i and frame j of its spectrogram. This measure allows us to visualise the recurrence of similar frames throughout an audio track and locate areas of low similarity where it is very likely for a change in song segment to reside. Foote [23] initially calculated this similarity by parameterising audio frames using MFCCs (described in Section 3) and computing the cosine angle between these two feature vectors v_1 and v_2. Other methods use the cosine distance $\langle v_1, v_2 \rangle = 1 - \frac{v_1 \cdot v_2}{||v_1|| ||v_2||}$ [85]. These normalised distance measures are robust to spectral energy, not discriminating against highly similar frames with low energy levels.

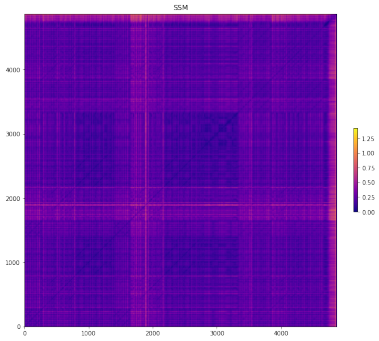


Figure 6.4: SSM matrix of Yumi Zouma's rendition of Oasis's Champagne Supernova.

On the other hand, the Self Similarity Lag Matrix (Figure 6.5) is of size $T \times L$ (time$\times$ lag) and contains elements x_{tl} that measure the degree of similarity between a frame at time t and a frame lagging behind it by l frames $l \in [0, L]$. This is a more computationally efficient method than the SSM. Even if one were to exploit the symmetry of an SSM, $\frac{N^2}{2}$ calculations are required for a time series of N frames, whilst only NL are required for the SSLM. For the first L frames of a track, there are not enough preceding frames available to compare to. This issue is addressed by circularly extending the input spectrogram, copying its final L frames and appending them to the beginning. While padding with white noise may seem like a reasonable alternative to circular padding, it has been shown to lead to poor performance in Mel Log Spectrograms. Ullrich et al. [85] found that white noise has dissimilar properties to the background noise in natural recordings, which can negatively impact performance. Therefore, circular padding is preferred as it maintains the underlying characteristics of the song and therefore does not introduce additional

information that could obscure patterns in the data. The SSLM provides a more fine-grained view of the structure and repetition in a frame's local environment that may be overlooked in the global viewpoint of the SSM. The extent of this locality is determined by the lag value L.

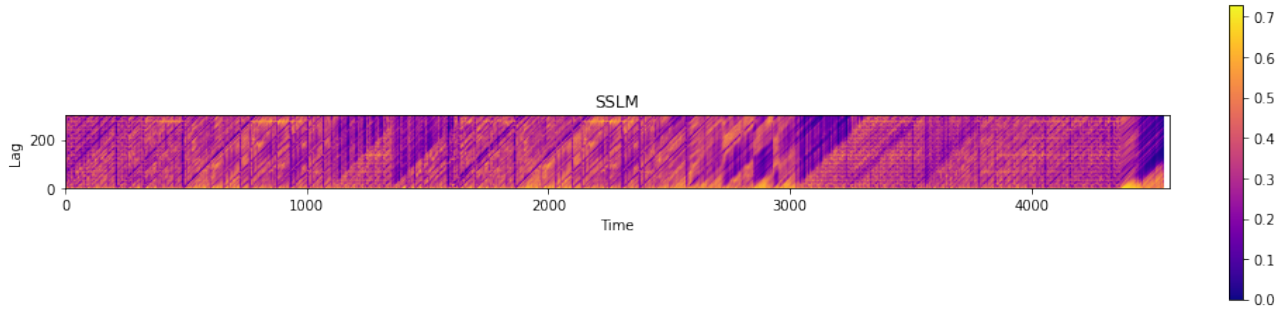


Figure 6.5: Circularly padded SSLM matrix of Yumi Zouma's rendition of Oasis's Champagne Supernova with a lag context of 14 seconds.

6.2.1 Delay Embedding

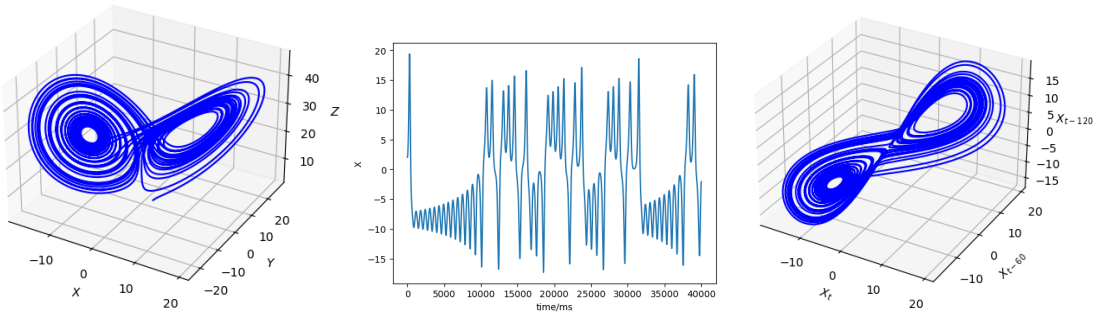
The method of calculating an SSM was improved upon by Serra et al [25]. Firstly, a delay embedded time series $[\hat{\mathbf{x}}_0, .., \hat{\mathbf{x}}_{N-m\tau}]$ is used rather than the time series of frame feature vectors $[\mathbf{x}_0, .., \mathbf{x}_N]$ where:

$$\hat{\mathbf{x}}_i = [\mathbf{x}_i^T \, \mathbf{x}_{i-\tau}^T \, ... \, \mathbf{x}_{i-(m-1)\tau}^T]^T \tag{26}$$

The embedding dimension m and time delay τ are parameters that need to be decided. This method originated in non-linear time series analysis. Consider an n-dimensional non-linear system in which we only have access to the measurements of one state. As stated by Bradley et al. [86], if one embeds enough delay dimensions, the plotted trajectory in this embedding space is guaranteed to be topologically similar to the true trajectory of the system, revealing underlying dynamics that were originally difficult to detect from the one state. An example of this technique is shown in Figure 6.6 where the Lorenz system trajectory of Equation 27 is reconstructed using only the delay embedding of the X state. Just as more information has been revealed about the true state dynamics via the construction of a topologically similar state trajectory, in theory, more information will be extracted from the audio time-series through delay embedding. However, one needs to be careful when applying this technique. Given that a song is not a dynamical system that can be expressed by an equation of motion, this technique should be used pragmatically [87]. Grill et al. [24] also note the trade-off between the reduction in noise and resultant temporal blurring when increasing the amount of embedding dimensions. Overall, as Serra et

al. [25] discovered that delay embedding had a minor positive effect on the effectiveness of boundary classification and as it is still a widely used technique [24] [17] [80], We believe this should be used in self-similarity calculation if it leads to minimal increase in algorithm run-time.

$$\frac{dx}{dt} = 10\,(y - x),\ \frac{dy}{dt} = x\,(28 - z) - y),\ \frac{dz}{dt} = x\,y - \frac{8\,z}{3} \tag{27}$$



(a) True Trajectory. (b) X state. (c) Reconstructed Trajectory.

Figure 6.6: Plots of true state trajectory $[x, y, z]$ of equation 3 with starting conditions $[1, 1, 1]$, state $[x]$ against time and a plot of trajectory $[x[i], x[i - \tau], x[i - 2\tau]]$ respectively with $\tau = 60ms, i = 120, 121, ...40000ms$.

6.2.2 Thresholding

In order to emphasise salient recurrences and diminish noise, we can consider using a thresholding function. Serra et al. [25] choose to employ a K-Nearest-Neighbours algorithm in which SSM matrix element x_{ij} is classified as 1 if both frame i and j are respective neighbours of each other. For a set of frames $[f_1, ..f_i, ...f_n]$, the nearest neighbours of $[f_i]$ are classified by sorting subset $[f_1, ..f_{i-1}, f_{i+1}, ...f_n]$ in regards to their distance from f_i and selecting the K nearest frames. If frame i and frame j are not mutual neighbours of each other, element x_{ij} is classified as 0. Serra et al. [25] found that this restrictive requirement was most effective at minimising noise. This SSM is then reformatted into an SSLM. This resultant recurrence matrix is treated as a sample of a bi-variate probability-mass function depicting the probability of time lag recurrences. An estimate of the underlying distribution is then obtained by convolution with a gaussian kernel, this will interpolate probabilities across discrete times and lags by weighting the contribution of observations from the sample based on their distance in time and lag away from the current lag time element in the sample we are inspecting.

Alternatively, Grill et al. [24][17] employed an SSLM thresholding method in which SSLM element x_{tl} is calculated using the sigmoid function:

$$x_{tl} = \sigma(1 - \frac{D_{t,l}}{\epsilon_{t,l}}) \tag{28}$$

where $D_{t,l}$ is the cosine distance between frames f_t and f_l and $\epsilon_{i,l}$ is an adaptive threshold equal to the 0.1-quantile of the set of distances within the lag contexts (the preceding L frames) of both frame t and frame $t - l$. This adaptive threshold serves a similar purpose to the K-Nearest-Neighbours restriction as a value greater than 0.5 will only be returned if $D_{i,l}$ is within the 0.1-quantile of these distances, a similar concept to being a nearest neighbour. The largest difference between the two methods is that the method using the sigmoid function immediately returns a continuous output rather than a binary output that then has to be interpreted as a sample from a probability distribution. We have decided to move forward with this thresholding process for SSLM methods as it is more computationally efficient. The SSLM is immediately calculated which as discussed earlier, will reduce the number of distance calculations required. The time complexity for calculating the 0.1-quantile will be $\mathcal{O}(2L \log(2L))$ as it is an issue of sorting a set of $2L$ distances, comparatively, Serra et al.'s [25] method will have to sort a set of N every time to calculate the nearest neighbours. Typically $L \ll N$.

From the SSLM a novelty curve can be extracted by computing the distance between consecutive feature vectors along its time axis [25].

6.3 Supervised Machine-Learning Methods

6.3.1 Choice of Dataset for Supervised Machine-Learning Methods

Before discussing the use of supervised machine learning models, we must first consider the decision of which dataset to train the models on. A popular dataset of over 1400 song's structural annotations is the SALAMI dataset [88], having been used to train many segmentation models [17] [80]. One advantage of this dataset is its large size, having access to more data for training will hopefully lead to a more robust model. Evidence for this effect is seen in [89], who created a very similar model to the state-of-the-art but achieved subpar results which they assign to their use of a smaller dataset. Similarly, Hernandez et al. [80] built an identical model as Grill et al. [17] yet achieved a lower F-Measure score which they attributed to Grill et al. having access to a private dataset of 733 songs alongside the SALAMI dataset. This further establishes the relationship between dataset size and song segmentation performance.

Many songs in the SALAMI dataset have been annotated by two people or more. Working under the

assumption that there is an underlying ground truth boundary location, if one was to view each person's annotation as outputs from a sensor with a variance value unique to the person, more annotations will lead to a sample mean distribution with less variation around the ground truth value due to the central limit theorem. The creation of a dataset with each song annotated by multiple people may also improve the theoretical upper bound of a segmentation algorithm's F-measure performance as we will be training the model on outputs from a theoretical sensor with less variance.

Following this logic, it would be preferable to train on a more reliable data-set in which each song has been annotated by as many people as possible. The SPAM data-set [90] has had each audio clip annotated by 5 people, making it on of the most reliable data-sets available, however, it only consists of 50 tracks. This trade-off between the number of annotations and a data-set's size is something to consider. An option that bypasses this trade-off is combining multiple data-sets. Wang et al. [15] incorporated a multitude of data-sets into their training scheme which led to currently the best results in the field. Although this success is most likely due to their choice of machine learning model, Wang et al then added yet another data-set to the training set which increased boundary detection performance by 3% [91]. This verifies the positive impact of using more data. This combination method was also used by Ullrich et al.[85], combining a private data-set with the SALAMI data-set. The difference, in this case, is that the private data-set was annotated according to SALAMI guidelines, this ensures the added samples are from the same population.

Another strategy we will employ is data augmentation such as noise injection or high/low pass filter application [15]. This will increase the size and variety of our dataset, allowing our models to better handle likely distributional shifts in real-world input data such as variations in noise.

Overall, we believe there is substantial evidence that segmentation models are significantly impeded by dataset size, the success seen by Wang et al. [91] also indicates combining datasets is a viable option, therefore we will employ the same method. We have also decided to include the SPAM dataset on top of the ones used by Wang et al. [91] due to its large number of annotators per audio track.

6.3.2 Labelling Training Data

There are various factors that complicate the decision of how to label the training and testing data used for a structural segmentation neural network. Ostensibly this is a binary classification task, every frame in an audio track should be analysed and a probability should be returned predicting whether or not this is a segment boundary frame. Therefore, one option is to naively label boundary frames as positive and

every other frame as negative and use a cross-entropy loss function. However, there is an issue with labelling frames near the segment boundary as negative. As discussed in Section 6.1, there is an inherent subjectivity as to where a segment boundary occurs, therefore these frames could be considered to be the true boundary location and punishing positive classifications of these frames would lead to a poor model.

In order to account for this inaccuracy of ground truth labels, a smearing effect can be applied, in which all frames $\pm E$ away from the boundary frame are classified as positive results during training [85]. The positive weighting of the frames is subject to a Gaussian taper centred at the boundary frame as demonstrated in Figure 6.7. A similar smearing technique was also used by Wang et al. [15], this popularity indicates there are grounds for the use of this method.

In order to take into account the context of surrounding frames, the input to neural networks is commonly an excerpt of N frames centred upon the subject frame, therefore the inputs for neighbouring frames will be broadly similar, differing by only a single frame out of N. If we were to use a naive labelling system, the positive classification of frames neighbouring the boundary (which are almost identical to boundary frames) would be discouraged. Penalising the classification of such similar excerpts may lead to the model struggling to learn relevant patterns. This is especially important for the use of CNNs as they have an inherent inductive bias of translational equivariance.

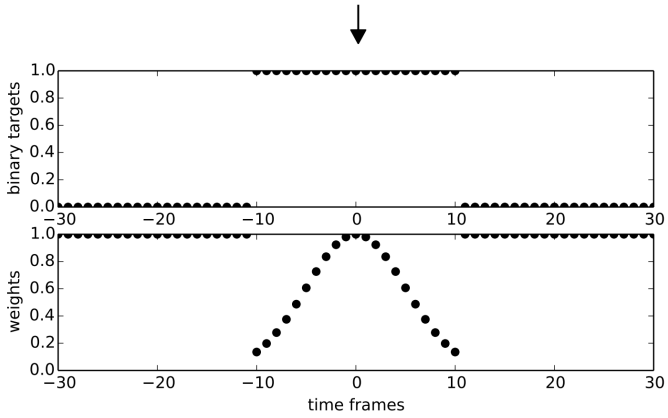


Figure 6.7: Positive classification weighting taken from [85], arrow corresponds to 'true' boundary frame.

An alternative approach to this single output methodology has been raised in which the input remains the same as above but the output is a boundary probability curve for all frames in the excerpt. It is theorised this may allow the network to learn the contextual dependency of previous and future boundary probabilities [15]. Moreover, an advantage of the multiple-output model is that it reduces the run-time when

analysing a whole song because it needs fewer inputs to cover the same number of frames.

6.3.3 Convolutional Neural Networks

Perhaps the most well-explored method of calculating audio novelty is the use of Convolutional Neural Networks (described in Section 4.3.4). Excerpts of N frames from a certain audio representation are inputted into the network and a single probability of boundary likelihood is outputted. In regards to inputs, log Mel-spectrograms have been used [85], but the best results have been achieved by inputting both SSLMs and log Mel-spectrograms into two separate CNNs, fusing the kernels midway through the process. One has to ensure that both inputs are of the same temporal context and resolution otherwise when the kernels are combined, the combination of high-level features that correspond to different time segments renders the information meaningless [24]. Due to the fixed size input constraint of CNNs, the MLS must be padded with pink noise [85] to enable the analysis of edge frames. The SSLM should be circularly padded as discussed in Section 6.2.

6.3.4 Transformer Models

A recent development in the field is the application of a Transformer in Transformer model, specifically, SpecTNT [92]. This section briefly describes the mechanisms behind SpecTNT.
The driving force behind this architecture is the multi-head self-attention mechanism. For a series of input vectors, self-attention allows the model to consider the relationship between inputs and account for the effect of ones with high relevance. Each vector is mapped to a query vector and key-value vector pair via a learned linear transformation. The query and key vectors are size $1 \times d_k$ and the value vector $1 \times d_v$. The keys, queries and values are stacked respectively to create 3 matrices:

$$K \in \mathbb{R}^{n_{inputs} \times d_k}, \quad Q \in \mathbb{R}^{n_{inputs} \times d_k}, \quad V \in \mathbb{R}^{n_{inputs} \times d_v} \tag{29}$$

The dot product of the query and key vectors is taken to create an attention matrix. The output is formed by premultiplying the value matrix with this attention matrix such that only values associated with keys highly correlated to the query are carried forward. This equation is shown below:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{30}$$

$$softmax(x_i) = \frac{e^{x_i}}{\sum_{j=0}^{K} e^{x_j}} \qquad i \in [0, K] \tag{31}$$

This is analogous to a search mechanism, each input vector asks a question and the most relevant answers are returned and fed forward into the network. If the key vector is close to parallel to the query vector, its associated value is highly relevant. The *softmax* function is computed along each of the attention matrix's rows. This normalises each row so that each element represents the relative importance of each value for that row's corresponding query. The $\sqrt{d_k}$ factor is to prevent large dimensional dot products from outputting large values. Large value inputs to softmax would make it hard to distinguish between its outputs as this function flattens at large values [93].

Multi-head self-attention simply applies this mechanism to the inputs h times, using different learned linear mappings. This allows the network to attend to information from different representation subspaces [93]. These outputs are concatenated and linearly projected to the desired output dimension. This output is then fed into a feed-forward network, which is a series of two fully connected linear networks with ReLu activation functions following them. Residual connections are implemented, these skip the processing of each block, summing the inputs of the block to its outputs. This prevents loss of information, a similar idea to preventing the vanishing gradient problem commonly associated with RNNs is touched on in section 5.3.7. This aggregate sum is then layer normalised, layer normalisation stabilises hidden state dynamics without imposing restrictions on the input amount or batch size [94]. This overall Multi-Head Self-Attention into Feed-Forward Network block can be stacked to form a Transformer encoder.

One benefit of this encoder mechanism is that the input vectors can be processed in parallel. Being able to process in parallel leads to vast time savings and the ability to take advantage of GPUs' parallel processing. However, this parallel processing means that the positional information of each input needs to be injected into the vector via a positional embedding. This is a summation with a positional vector, the type of vector used depends on the task, every possible position must have a unique identifier and the magnitude of the vector should be small enough that it doesn't displace the original vector to a different semantic space.

The input to SpecTNT is a Harmonic Frequency Time tensor of size $H \times F \times T$ described in section 3. This is first inputted into a stack of convolutional layers, outputting a tensor of size $\hat{H} \times \hat{F} \times \hat{T}$. This is then inputted into a hierarchical Transformer architecture in which there is one encoder type for spectral information and another for temporal information. Each $\hat{H} \times \hat{F}$ matrix in the tensor is treated as separate frequency series inputs to $\hat{T}$ separate spectral encoders, each frequency axis is concatenated with a

”Frequency Class Token” vector that will attend to information from the other frequency bins as it passes through the MHSA blocks. This vector acts as a conduit for information to be passed between the spectral and temporal encoders. By only passing one vector, the size of the model is greatly reduced, improving the model’s performance on small datasets. The frequency bins are positionally embedded through the summation of a learned matrix.

There is also a learned temporal embedding matrix $E = [e_0, .., e_t, .., e_T]$ of size $D \times \hat{T}$ where the temporal embedding dimension D is a value to be decided. Each e_i vector corresponds to a timeframe and is projected to $\mathbb{R}^{\hat{H}}$ via a linear layer and summed to its corresponding frequency series FCT. This transfers temporal information to the FCT. After the spectral encoder, each FCT is projected up to $\mathbb{R}^D$, the FCT information has now been distributed across the time axis [92]. These are now summed with the original temporal embedding vectors and inputted into the temporal encoder.

Overall, this model is an examination of information at two levels of granularity, both harmonic and timbral features are first finely analysed by the spectral transformers, before being summarised by the FCTs and examined at a more macro scale by the temporal transformers.

Figure 6.8 illustrates the process of one SpecTNT block. The left figure demonstrates the appending of the FCT vector, and the right figure shows the hierarchical transformer structure, there are many spectral transformer encoders which each project their aggregate FCT vectors up to the singular temporal encoder. These SpecTNT blocks can be stacked using the output of the final temporal encoder as input into a fully connected network.

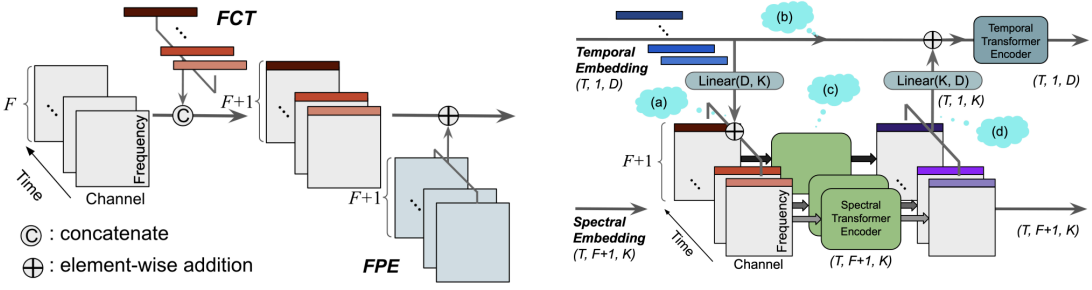


Figure 6.8: SpecTNT architecture taken from [92].

Wang et al. [15] implemented SpecTNT to achieve both boundary detection and segment-type recognition. The input is an N frame excerpt along the time dimension of an audio track’s HFT tensor representation, the output is a boundary activation value for each frame. This has achieved F-1 measure values higher than all competing models. Whether this is due to the use of a combination of 4 datasets

or the model itself is uncertain. However, the addition of another dataset of 18,843 music excerpts only increased boundary performance by 3% [91]. Therefore it is likely the novel model architecture played a significant role.

6.4 Peak Picking Algorithm

Another choice we have to make is which peak-picking algorithm to use. Ullrich et al. [85] state that the choice of peak-picking algorithm led to no major effect on algorithm performance. Therefore we have chosen to employ their algorithm. A boundary value is a candidate if it has not been surpassed in ±6 seconds. To account for long-term trends, the average boundary activation of the past 12 and future 6 seconds should be subtracted from each candidate. Various thresholding methods can now be applied such as requiring a candidate to be larger than a minimum value. This algorithm has also been used by Wang et al. [15], further confirming the validity of our choice of peak-picking algorithm.

6.5 Implementation

To choose the appropriate audio representation and analysis method for our product, we will now evaluate them based on information retrieval quality and run-time. All time measurements were taken using a Google Colab base-level cloud computer, with a 2.20GHz Intel Xeon CPU and 13GB DDR4 RAM. As discussed in section 6.1 we will compare the quality of two algorithms using the F-1 measure and F-0.58 measure and all comparisons will be made with reference to each measure's explorable range. This will be achieved via linear interpolation so that the lower bound corresponds to 0 and the upper bound to 1. Table 9 depicts the respective run times of calculating various audio representations. This reveals the high computation cost of calculating SSLMs when compared to the MLS. The SSLM+E+T input takes 394.2 times as long as the calculation of MLS. Due to this large time difference, we should heavily favour algorithms that do not require an SSLM input.

	MLS	**SSLM**	**SSLM + E**	**SSLM + T**	**SSLM + E+T**
Time /s	0.53	34.45	33.702	207.69	208.95

Table 9: Time taken to calculate the Log Mel Spectrogram, SSLM, SSLM with embedding, SSLM with thresholding, and SSLM with embedding and thresholding. Averaged over 5 trials for an audio track of 231s.

Table 10 compares Grill et al.'s state-of-the-art CNN's [24][85], Serra et al.'s unsupervised analysis of the SSLM [25] and Wang et al.'s novel Transformer in Transformer architecture [15]. Unsupervised SSLM

analysis both underperforms and requires an SSLM input. Therefore it is not suitable for our product.
The MLS CNN architecture proposed is relatively simple, with only two convolutional layers and two linear layers [85], recreating and timing this revealed it would take 4.53ms on average to analyse a sample. This will take 7.51s on average to analyse a 231s audio track as for this model, 7.18 frames correspond to 1s of the audio track [85]. On the other hand, due to the SSLM input, choosing the combined SSLM MLS CNN model would increase processing time from under 10s for an average song to over 3.5 minutes. Although this leads to a 24.9% increase in F-0.58 measure performance relative to the F-0.58 measure's bounds, this increase in processing time is a significant drawback.

Model	F-1 Measure	F-0.58 Measure
Upper Bound (estimate)	0.74	0.74
SSLM [25]	0.246	0.251
CNN MLS input [85]	0.438	0.482
CNN MLS+SSLM input [24]	0.523	0.572
SpecTNT [15]	0.623	-
SpecTNT trained with additional dataset [91]	**0.643**	-
Lower Bound (estimate)	0.15	0.21

Table 10: 0.5s tolerance F-1 and F-0.58 measures calculated on the MIREX-10 RWC-pop dataset. Results available at the MIREX website and in [15].

Regarding the SpecTNT model, although its F-0.58 measure performance was not given, the relationship between the F-0.58 measure and F-1 measure for all the other models in which F-0.58 slightly outclasses F-1 implies that there will not be a drastic drop in F-0.58 measure performance for SpecTNT. This model outclasses all other models discussed and its F-1 measure is the highest value recorded. Relative to the F-1 measure's upper and lower bounds, this model's F-1 measure is a 71% improvement over the MLS CNN's and a 32% increase over the MLS+SSLM CNN. The input to this model is a computationally inexpensive HFT tensor, as described in section 3. This a tensor in which H filterbanks have been applied to a spectrogram, therefore it will take at most H times as long as MLS calculation. Nieto et al. [79] discovered performance peaked at $H = 6$. Therefore, this remains a computationally inexpensive pre-processing procedure.
One possible issue could be the run-time of the SpecTNT model itself rather than the feature pre-processing. The time complexity of an encoder's self-attention mechanism is $\mathcal{O}(n^2)$, every input vector has to attend to information from every other input vector. Fortunately, testing the model with the optimal input tensor size stated by Wang et al. [15] of $\hat{H} \times \hat{F} \times \hat{T} = 6 \times 128 \times 125$, revealed an average time of 0.107s per sample. This surprisingly low run-time can be attributed to the fact that the input is split into

merely a 125x6 size tensor per spectral encoder. At first glance, SpecTNT seems to be a slower model than the MLS CNN architecture. However, the output is a boundary activation curve for a 24-second segment [15], not a single frame. This means it takes 1.07s on average to analyse a 231s audio track. Future work should involve examining if F-1 and F-0.58 measure results decrease with distance from the central input frame. This knowledge could influence whether we choose to overlap input segments, which would increase run-time as a by-product.

We have decided to move forward with the SpecTNT model, due to its high performance and low run-time. This also has the benefit of leaving us with room to improve the product in the future. The SpecTNT segmentation architecture was originally created to segment and classify the sections of a song such as verse or chorus [15]. Therefore in the future, it would be possible to append this functionality to the product for little cost by simply repurposing and reusing code from the current segmentation model. Taking inspiration from maximum a posteriori estimation, this section categorisation could act as a prior in order to improve the choice of lighting parameters, the knowledge that a particular section is a pre-chorus could lead to a higher probability that lighting brightness increases throughout the section for example.

6.6 Conclusion

Overall, this segmentation module accepts an audio track as an input, and outputs timestamps that correspond to the boundaries between a song's segments, such as from verse to chorus. These timestamps can then be used in conjunction with the downbeat tracker to send cues to the DMX console to change the lighting scheme significantly between segments. As stated in Section 5, musical phrases typically start on the downbeat, therefore these segmentation timestamps will be used to choose the nearest downbeat timestamp, the lighting configuration change will occur at this point. This will ensure the configuration changes are synchronous with the music. These timestamps will also be used to segment the audio track so that each segment can be separately inputted into the Music Emotion Recognition model as described in the section 7, which will therefore return a unique Arousal and Valence value for each segment. These values can be used to determine a unique lighting configuration per song segment. This achieves the required product specification of being able to create a dynamic lighting scheme that is perceived to react to the music. In order to optimise user experience we compared various segmentation algorithms and chose the highest performance lowest run-time algorithm.

7 Music Emotion Recognition - Felix Cohen

The relationship between music and emotion is thoroughly important. In an analysis of creative professionals' queries when searching for music to pair with video (e.g. an advert), 80% were found to contain emotional descriptors [95]. This demonstrates the significance of music in art as the emotional response it facilitates. Therefore, in order to create a visual that viewers perceive as reacting to the music, it is vital to be able to extract a song's emotive content. In this section, we will first discuss the available models to describe emotion before then choosing an appropriate training data-set and Music Emotion Retrieval (MER) machine learning algorithm. We will then discuss the functionality of this module in the context of the whole project.

7.1 Emotional Models

A musical excerpt's mood or emotional content can be characterised either using a set of discrete entities or a multi-dimensional continuous space. The most common model used in the literature is Thayer's Model (Figure 7.1), which describes emotion via two independent dimensions: arousal and valence. Arousal has been described as the "physical activation of the autonomic nervous system." This can be viewed as a description of how energetic an emotional response is [96]. Conversely, valence describes the extent to which a response is pleasant.

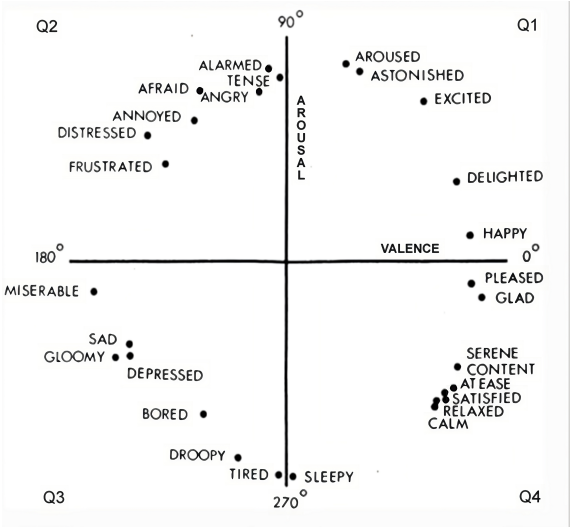


Figure 7.1: Thayer's Dimensional Arousal-Valence Model taken from [97]

We have decided to proceed with Thayer's model as we believe that this arousal-valence space may be better suited for capturing subtle nuances in a musical excerpt's emotional content. For example, there is

evidence of the existence of emotions uniquely triggered by music, such as aesthetic awe [96]. This would be very hard to account for in a categorical model. Another reason to choose Thayer's model is that it is widely accepted in the MER community, giving us the freedom to choose from multiple compatible MER models for our product [97][98]. This arousal-valence space can be separated into four quadrants to train a music emotion classifier (see Q1-4 in Figure 7.1). However, we have decided to create a music emotion regression model that returns a point on the A-V space. This takes advantage of Thayer's dimensional model fully and also prevents the issue of misclassifying edge cases. For example, classifying a piece with arousal of 0.01 and valence of 0.5 as a Q4 class rather than a Q2 class.

7.2 Choosing a Dataset

Two common Music Emotion Recognition tasks explored in the literature are static and dynamic recognition. Static recognition determines the emotional content of longer-form audio clips of 15-20s, while dynamic recognition determines how the emotional content of an audio clip changes over time via a sliding window approach, returning a value every 0.5-1s [97]. Unfortunately, neither of these approaches fits the design specification. If the algorithm could only alter its lighting composition every 20s, it would miss the start or end of any segment that is shorter than 20s. Conversely, using a sliding window to track the emotional content of an entire song will likely remove the possibility of a distinct lighting change occurring at a boundary, as each segment will not be treated as its own entity. Therefore, we must create a technique that takes into account these boundary locations. As stated by Grekow [97], 6s was the minimum amount of time needed for a person to detect the emotional content of an audio segment; it is also stated by Bachorik et al. [99] that the average time required is 8.31s. This implies that a score for the emotional content of a 1s excerpt is meaningless without considering its surrounding context. This further discourages the use of dynamic emotion recognition. Therefore our chosen data-set should consist of clips within the range of 6-10s.

Out of all the MER data-sets available, there are two main candidates that fit the discussed specification: the curated static data-set of 324 6s excerpts used by Grekow [97] or the DEAM dataset [100]. This data-set is incredibly large, containing 58 full-length songs and 1744 45s excerpts. Although the DEAM data-set has dynamic annotations, meaning it will be less suited to the specific needs of this task, we have decided to use it due to its size. We have opted to create a static 6s data-set by dividing each item into non-overlapping 6s excerpts and averaging the arousal and valence values of each excerpt, resulting in a final data-set size of over 12614 items. We have chosen the segment size to be 6s as this is the min-

imum time needed to perceive the emotional content of a song and this segment size is guaranteed to fit inside any given song segment due to the nature of our peak-picking algorithm discussed in section 6.4. Each song segment can now be analysed individually, fitting the required design specification. For song segments over 6s in duration, a sliding window approach within the segment can be employed. Then we can calculate the mean and variance of the A-V values for the segment. As there was no statistical correlation between an excerpt's arousal and valence value in Grekow's dataset [97], the development of a multi-output regressive model may unnecessarily complicate the task for no improvement in predictive performance. Training individual arousal and valence recognition models will likely lead to smaller, more computationally efficient networks that can be trained in parallel. Therefore, two separate arousal and valence datasets should be created.

Aljanaki et al. [100] also showed that multi-output regressive models may not generalize well across datasets with different emotion patterns. They found that models trained on data with high arousal-valence correlation performed poorly on valence prediction when tested on data with low correlation. This suggests that the models failed to learn valence-specific cues from the training data. This further indicates that we should create two separate models.

7.3 Use of Recurrent Neural Networks

Due to the fact our product is specifically crafted for the dance genre, in which the presence of lyrics is sparse, We will focus this discussion on neural networks that do not take advantage of lyrical analysis. However, the finetuning of a pre-trained large language model for lyric sentiment analysis could be a possible avenue to pursue in the future once we have established a Minimum Viable Product.

Although the majority of the literature agrees that the use of RNNs currently achieves optimal results in regards to MER [98] [97] [101], there is much variation in how to preprocess the input data. Similarly, although most of the literature uses Thayer's model to describe emotion, there is a split between models that classify segments into one of four A-V categories [98][102] and those that output dimensional values [97][101]. Due to the use of different evaluation metrics for classification and regression tasks making it hard to directly compare the two, we will now focus on and favour models that have proven successful in the regressive domain.

One such model created by Grekow [97] firstly extracts 529 spectral, tonal and rhythmic features from the audio track using an audio analysis library called Essentia. A full list of the extracted features is available at the Essentia website [103]. A time series of these features is then inputted into an LSTM (described

in section 5.3.8). An LSTM's MER performance noticeably changes based on the quality of feature input [97]. Feature input can be improved by pre-training a single-layer linear neural network on a similar task to act as a feature selector. Instead of the feature vectors, the linear layer's neuron activations are now inputted into the LSTM. To avoid overfitting in the complete model whilst allowing for the extraction of transferrable features, a similar task is used rather than an identical one. Using an identical task would be no different from adding a linear layer to the front end and making the model larger, which could lead to overfitting. A different feature selection method based on information theory that removed highly correlated features was applied by Hizlisoy et al. [98]. However, when the two were compared by Huang et al.[102], Grekow's model outperformed this.

A different feature-extracting approach by Orjesek et al. [101] applies 1-dimensional convolutional filters directly to the audio track itself. Dielmann et al. [104] have shown that these learned filters are capable of frequency selection, picking out frequencies in a similar fashion to the Slaney filterbanks described in Section 3, being linear up to about 1000Hz. These features are then input into an "Iterative Reconstruction" layer. This passes the features repeatedly between a linear "encoder" layer and a linear "decoder" layer in order to enhance important features and suppress irrelevant ones. This is achieved through 3 loss functions. Alongside a standard loss function comparing network output to the true value, a loss function that compares each decoder output and the original feature input is used along with another that compares the final encoder output to a rounded version of itself (e.g. all values are either [-1,0,1]). The former encourages the encoder output to remain as similar to the feature input as possible whilst the latter encourages the desirable property of enhancing and suppressing features [101].

This is then inputted into a BiGRU network, and then finally, a fully connected layer outputs valence and arousal values. The BiGRU is an RNN that is very similar in design philosophy to the BiLSTM network discussed in Section 5.3.9. The GRU block has fewer parameters than the LSTM block because it does not have an output gate. The output gate in the LSTM block controls how much of the cell state is exposed to other units in the network. Comparatively, the GRU block does not have this capability, exposing the entire hidden state. This leads to a simpler network structure that can be trained in less time; the trade-off being that the network has less control over information flow [105].

7.4 Choice of Model

Both of these approaches are incredibly similar, employing feature extractors before inputting the results into a RNN. Table 11 compares the performances of the two as reported by Grekow and Ojesek. As a

consequence of the triangle inequality, for a given set of values, their Root Mean Square Error (RMSE) will always be greater than or equal to their Mean Absolute Error (MAE):

$$|r_1|^2 + ... + |r_n|^2 \geq (|r_1| + ... + |r_n|)^2 \quad \therefore \quad RMSE^2 \geq MAE^2 \tag{32}$$

Therefore, Table 11 demonstrates that the Iterative Reconstruction model has a slightly higher performance than the Linear Feature Selector model. However, as the models were tested on different datasets, choosing one model over the other based on this comparison is tenuous. Furthermore, the performance difference is very small. We have decided to move forward with Grekow's model as this is designed to accept static 6s inputs and consequently, more precisely fits our design specification.

	Linear Feature Selector LSTM /MAE	**Iterative Reconstruction BiGRU /RMSE**
Valence	0.12	0.114
Arousal	0.11	0.105

Table 11: RMSE results taken from [101] and MAE results taken from [97]

7.5 Conclusion

To summarise, it was vital that the MER algorithm could analyse audio tracks of different lengths as the structural segmentation algorithm (discussed in Section 6) outputs audio segments of varying lengths that must be individually analysed. This was achieved through the choice of an algorithm that accepted input of the same length as the smallest possible segment length. A sliding window is used to extract the emotive content of larger segments. In order to optimise user experience, we chose the highest-performing algorithm that simultaneously met these requirements. This design allows us to return unique arousal and valence parameters per segment. The arousal value will control fixture movement, this will allow the lighting to correlate to how exciting the audio track is. The relationship between emotion and colour was considered, colour stimuli were found to evoke emotional responses however, the nature of the response was contingent on individual colour preference [106]. Therefore, we instead decided to use the valence value to control the lighting intensity. This allows for a unique lighting configuration per song segment that reflects its emotive content, meeting the design specification of a reactive lighting system.

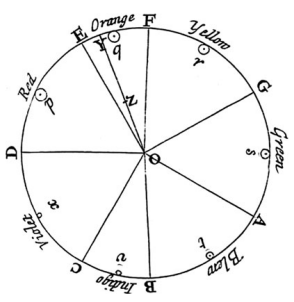
8 Genre-Specific Colour Mapping - Dain Jung

Colour is one of the most important aspects of visual synthesis. It can evoke emotional responses and contribute to the immersion of DJ sets. In this section, we will discuss the associated challenges, the objectives, and the design process of creating a genre-specific colour map.

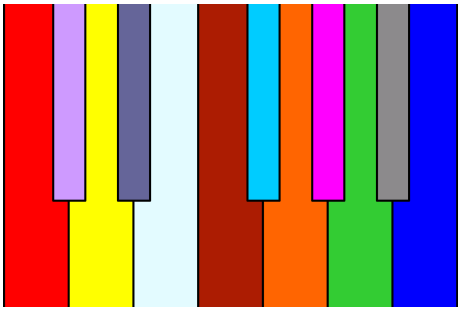
8.1 Literature Review

8.1.1 Chromesthesia

Chromesthesia is a type of synesthesia in which people sense colours in addition to auditory experiences when listening to music [107]. The interconnection of colour, light, and sound has been studied continuously. Several renowned scholars have proposed various theories to explain this connection.



(a) Newton's colour wheel [108].



(b) Scriabin's colour organ [109].

Figure 8.1: Visual representations of chromesthesia.

Johann Wolfgang von Goethe [110] believed light and sound were interrelated. He developed a colour theory based on the idea that light and sound originate from a shared fundamental principle. Similarly, Isaac Newton [108] viewed sound and light's relationship as sharing common frequencies. Newton's theory states that different light frequencies result in varying colours while different sound wave frequencies produce varying sounds based on the same mathematical relationship. Figure 8.1a demonstrates Newton's concept, where the seven colours in the rainbow correspond to the seven musical notes on a music scale (DEFGABC). Carl Jung [111] mentioned "colour hearing" in his book "Symbols of Transformation" as an example of synesthesia.

A musician who may be the most famous for having chromesthesia is Scriabin. Figure 8.1b shows the instrument that Scriabin invented, which uses lights of different hues given by his ability to express the different pitches. Although there have been ongoing debates about Scriabin's chromesthesia [112], this

shows that there have been attempts to use colours to enhance the effect of music.

8.1.2 Existing Colour Mappings and Their Applications

Having explored the implications of chromesthesia, we now examine its practical applications in music through the use of colour mappings. Effective mappings of music to colour have been studied in various research and business applications. The colour mapping of musical genres can enrich the listener's experience by providing emotional associations and aiding in understanding music [113]. The colours associated with a particular genre may trigger a multi-sensory response [114]. When applied to lighting design, the colours can keep the audience enthusiastic by reinforcing the genre's atmosphere. Colours also have cultural and symbolic meanings that can strengthen the story or themes in a musical genre. Those meanings can work as an additional layer to the music.

Existing colour mappings can give insight about potential improvements for our project. Hubbard's study [115] suggests that people associate lighter colours with high frequencies and darker colours with low frequencies. The model for our colour map could quantify this theory by analysing the relationship between the lightness value and the pitch. Voong's Colour Player [116] allows users to assign a colour to a music track based on the emotions that a music track evokes. The proposed system creates a playlist with a colour tag, which helps organise the music files. Finally, Pampalk's MusicRainbow [117] arranges artists on a circular rainbow where colours encode different music styles, allowing similar artists to be located closer to each other.

8.1.3 Colour Theory

The explored mappings in existing applications are underpinned by colour theory. Colour theory is a set of art principles, which can be used as a theoretical framework for creating eye-catching visualisations.

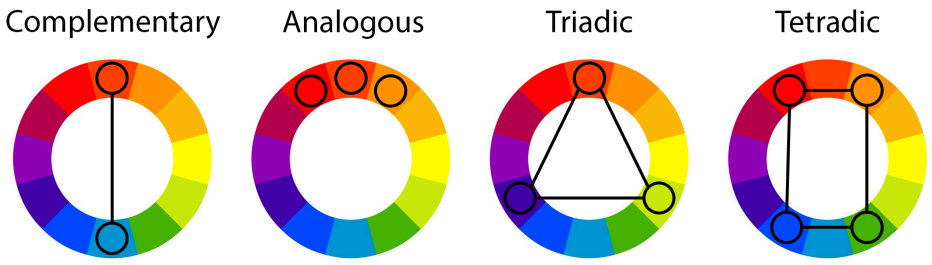


Figure 8.2: Different types of relationship between colours in the colour wheel.

As depicted in Figure 8.2, a key principle of colour theory is the use of complementary and analogous colours. Complementary colours are those that are opposite each other on the colour wheel. When placed next to each other, they stimulate different parts of the eye, creating a strong contrast. Therefore, they could be used to emphasise the difference between genres with contrasting characteristics. On the other hand, analogous colours are adjacent on the colour wheel. They create harmonious and peaceful effects, making them ideal for visualising similar genres. In addition, they could be used to represent different sub-genres within a broader genre, such as different types of pop music.

Colour theory also highlights the significance of colour harmony, which can be achieved by implementing triadic and tetradic colour schemes [118]. A triadic colour scheme consists of three equidistant colours on the wheel, while a tetradic colour scheme comprises two sets of complementary colours. These colour schemes further support the harmony principles in analogous colours. Harmony of scale is achieved by selecting colours with similar lightness but the same hue, while harmony of hues involves selecting colours with the same lightness but slightly different hues. [119]

The relationship between two colours can be quantified by converting them into a perceptually uniform colour space and using a suitable colour difference formula, such as the delta E formula, like CIEDE2000 [120]. The ultimate goal is to ensure that the colour palette successfully demonstrates the characteristics of each music genre. Therefore, using only the colours that add meaningful insight is essential. Meaningless colours that may cause confusion or be overwhelming should be avoided because the users would try to understand what it means [121].

8.2 Challenges and Limitations

Colour mapping for music visualisation presents several challenges and limitations. The initial problem we encountered during the project was the lack of publicly accessible datasets with colour labels for music. To address this, we devised effective colour mapping strategies based on existing literature.

8.2.1 Individual Preferences

One significant challenge is the subjective nature of colour perception, where different individuals perceive colours differently due to factors such as age, culture, and personal preferences. An online questionnaire exploring colour mappings to musical attributes revealed the absence of a universally accepted colour-genre mapping [122]. For instance, red can evoke different emotions; it can symbolise love and

warmth to some, while others may associate it with danger or anger [106].

8.2.2 Cultural Differences

Colour	Western Cultures	Eastern Cultures	Middle Eastern Cultures
Red	Love, Passion	Prosperity, Good Fortune	Courage, Sacrifice
Blue	Calm, Stability	Immortality, Life	Protection, Spirituality
Green	Nature, Growth	Harmony, Freshness	Fertility, Luck
Yellow	Happiness, Energy	Imperial, Nobility	Wisdom, Learning

Table 12: Cultural colour associations in eastern and western culture.

In addition to individual preferences, cultural differences also contribute to the subjectivity of colour perception. Differences in colour preference arise from distinct cultural social norms governing colour usage [123]. As Table 12 illustrates, colours symbolise different concepts across cultures. Significantly, colour preference shows different patterns in non-industrialised societies, barely influenced by consumerism [124]. Therefore, considering the audience's cultural background is important in designing a colour mapping scheme. A potential strategy is to develop region-specific colour mapping algorithms based on cultural colour associations.

8.2.3 Multidimensional Nature of Music and Colour Perception

Another challenge is the complexity of establishing a one-to-one mapping between musical features and colours due to the multidimensional nature of music and colour perception. For example, different musical features might be associated with multiple colours; conversely, a single colour could represent different musical features. This characteristic could make it challenging to design an effective colour mapping scheme that accurately represents the musical features.

8.3 Objectives

This section develops a music visualisation tool that uses colour to represent different aspects of the music. Cytowic [125] argued that non-synesthetes unconsciously engage in synesthesia. This suggests that although most audiences do not directly sense colour while listening to music, they have a basic understanding of the harmony between music and colour.

A simple experiment showed how two non-syntheses associate colours and music from different genres.

Figure 8.3: Colour-genre associations for two non-synesthetes.

The subjects were asked to listen to songs that achieved more than 90% possibility of belonging to each genre in Section 4, and picked a colour that best describes the genre. Figure 8.3 shows that people from completely different backgrounds could share some common thoughts about the connection between genre and colour. By visualising the music like how individuals with chromesthesia experience it, we can enhance the emotional impact and enjoyment of music for all listeners [126]. These tools can potentially transform how people experience and appreciate music, leading to exciting new developments in music technology.

Here are the detailed objectives of the colour mapping for the music visualisation product:

1. **Accurate Colour Mapping**: The colour maps should accurately depict the distinct characteristics of each music genre, including energy level, mood, and tempo. Audiences can better understand and appreciate the nuances of different genres by ensuring a meaningful visual representation of the music.

2. **Aesthetically Pleasing**: The colour maps should be pleasing to the eye. A more beautiful and colourful visualisation is more engaging. An aesthetic colour combination can lead to a more memorable experience. Colour theory principles should be incorporated, and attention to detail in colour transitions and gradients is also essential.

3. **Customisable and Adaptable**: The colour maps should be customisable and adaptable. It makes the system more user-friendly, leading to a better retention rate [127].

4. **Cultural Sensitivity**: The colour maps should consider different cultural norms and colour preferences, in order to cater to the need of people worldwide. Cultural sensitivity ensures that the visualisations appeal to users from various backgrounds and helps avoid misunderstandings.

5. **Real-time Colour Mapping**: The system should continuously update the colour mapping in real-time based on user feedback and interactions. This can be done by asking users to rate the accuracy of the colour mapping or by asking them to provide feedback on the colours being used.

8.4 Colour Mapping Design

The colour map defines how the audio features extracted from the music tracks are visually represented. The design process consists of three main steps: selection of primary colours, selection of colour palettes for each genre, and sensitivity analysis.

8.4.1 Primary Colour Selection for Genre

The initial step is selecting the right colour palettes for each music genre. We are using the GTZAN dataset used in Section 4. As mentioned above, every music genre has unique spectral characteristics and associated emotions. To create an effective colour mapping, it is important to select colours that accurately reflect these attributes and resonate with listeners' perceptions of the respective genres.

Data Collection To create the default colour palette, we use big data to construct our own dataset. We can gather data that captures people's perceptions of colours and the associated emotions. It could be obtained from various sources, such as web scraping, surveys, expert opinion, and existing studies.

Web scraping gathers data from music review websites, social media, and other online platforms where people discuss and describe music genres. This data shows how the public views each genre. This process could be done by using *BeautifulSoup* library in Python.

Surveys could be done to generate a more analysed dataframe. We could use websites like Google Forms to ask people for more direct labels. Experts like musicologists and music therapists can also give additional insights to complement the data obtained from web scraping and surveys. Existing studies on colour-music associations and chromesthesia can also be used as a theoretical foundation.

Since surveys and expert opinion require additional processes, we decided to use web scraping. Several websites host a global user base and contain rich data that can be mined for sentiment analysis. However, we should take account that the use of web scraping should always respect the site's Terms of Service and the privacy of its users.

Table 13 outlines various data sources and their respective policies for data scraping. Among the five options mentioned, Twitter and Reddit are the most feasible choices for this project. Both platforms offer APIs for data access and have relatively more permissive policies regarding data scraping. However, we must still comply with respective policy and not use the data for commercial purposes without permission.

Data Source	Description	Policy	API
Twitter	Popular social media platform with diverse opinions on music.	Allows scraping but restricts commercial use without permission.	O
Reddit	Discussion website with subreddits for various music genres.	Use data in compliance with Reddit API terms of service.	O
Spotify Community	Spotify's community forums for gathering user opinions on music genres.	Prohibits unauthorized automated access.	X
Music Review Blogs	Personal music review websites with user comments.	Varies by site; scraping may require permission or be prohibited.	X
YouTube	Vast source of music-related content with user comments.	Scraping explicitly not allowed; use API for structured data access.	O

Table 13: Data sources and policies for sentiment analysis.

Once the data is collected, it needs to be preprocessed to prepare it for analysis. This step involves several tasks, such as removing irrelevant information, handling missing data, tokenisation (breaking up text into individual words), and removing stopwords (common words like ”the”, ”and”, ”is”) that are usually not meaningful. *NLTK*, the Python libary for Natural Language Processing, is used for this task.

Genre	Characteristics	Proposed Colour	Colour Examples
Blues	Emotional, soulful, slow	Dark bluish colours	indigo, blue-violet
Classical	Elegant, soothing, complex	Soft pastel colours	light blue, pale green
Country	Upbeat, storytelling, acoustic	Earthy tones	brown, orange, yellow
Disco	Energetic, danceable, repetitive	Bright, saturated colours	hot pink, neon green
Hip hop	Rhythmic, urban, socially conscious	Bold, contrasting colours	red, black, white
Jazz	Improvisational, sophisticated, smooth	Deep, rich colours	burgundy, navy, gold
Metal	Aggressive, intense, loud	Strong, dark colours	black, metallic shades
Pop	Catchy, upbeat, mainstream	Bright, cheerful colours	pink, sky blue, yellow
Reggae	Laid-back, rhythmic, socially conscious	Vibrant, warm colours	red, yellow, green
Rock	Energetic, rebellious, guitar-driven	Bold, intense colours	red, orange, dark grey

Table 14: Genre characteristics and proposed colour mapping schemes.

Sentiment Analysis The proposed example colour mapping schemes for each music genre are presented in Table 14 and Figure 8.4. The colours were selected based on each genre's energy level, mood, tempo, and cultural and historical context. For instance, energetic and lively genres such as pop and rock are represented by warm colours like red and yellow. In contrast, soothing and calming genres like classical and jazz are represented by cool colours like blue and green. However, it is important to note that this

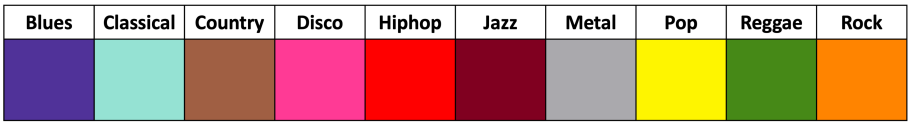


Figure 8.4: Suggested initial primary colour mapping for different genre.

colour map may not be universally applicable to every individual. The final colour mappings would also involve subjective judgment, creativity, and consideration of practical constraints such as colour contrast and visibility.

Modeling Dimensionality reduction compresses the original spectral data to a lower-dimensional one while preserving as much meaningful information as possible. This simplifies the process and reduces computational complexity. It helps us identify essential features that distinguish different music genres, which can then be used to guide our colour mapping process.

Clustering is a technique used to group similar data points together. To cluster similar audio features, we use algorithms such as k-means or hierarchical clustering. Since each genre forms a category, the number of clusters is set to match the number of genres in our dataset, which is 10. Once the clusters are formed, each cluster is allocated a colour from the palette based on the dominant features within it.

Several different colour space models can be used, including HSL, HSV, or CIELAB. We chose the HSL colour space, characterised by hue, saturation, and lightness levels to express the 'colour' label of each data point [128]. HSL descriptions are more intuitive and directly related to human perception of colours.

Feedback Loop The colour of a cluster is determined by the corresponding values. This, however, is not a fixed assignment. The colour palette has been designed to be adaptive and iterative, reacting to user feedback. This is done in a loop that is repeated until the user is satisfied with the colour palette.

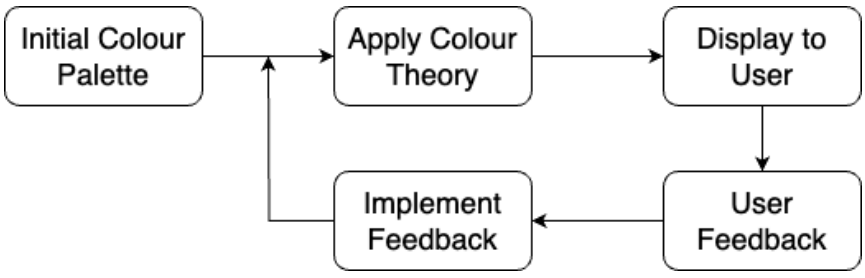


Figure 8.5: Feedback loop process.

Figure 8.5 shows the process of iterative adjustments based on user feedback. The loop has two major

parts: applying colour theory and adjusting to user feedback. As stated in Section 8.2, aesthetics and human perception should be considered. In each iteration, the previous palette is fine-tuned using colour theory. Colour theory ensures that the chosen colour combination is harmonious.

Figure 8.6: Tweakpane colour picker.

Then, the revised palette is displayed to the user. This allows the user to visualise the palette and verify if it is satisfactory or requires additional improvement. Subsequently, the user is prompted to provide feedback on the colour palette for each cluster. The feedback is collected using the Javascript library *Tweakpane*. Figure 8.6 shows the user interface for feedback. If the user is dissatisfied with the colour palette for a certain cluster, they can express their preference or alter the HSL values for that cluster. This opinion is incorporated into the next iteration.

The iterative method enables the user to manually fine-tune the colour palette based on their personal preferences and special needs. The final colour palette is designed to be both aesthetically appealing and intuitively indicative of the music genres. It combines machine learning, colour theory, and user feedback to develop colour mappings for music visualisations.

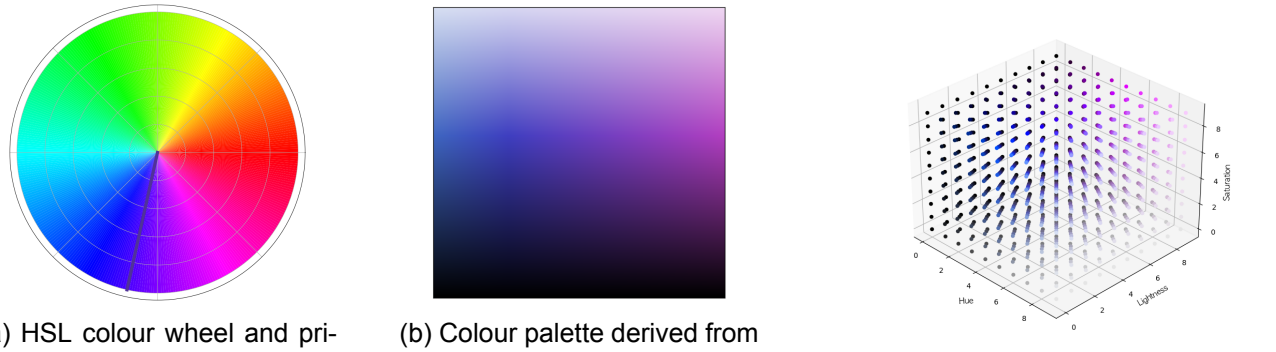
8.4.2 Specific Colour Selection for Music Tracks: Hue and Saturation Distribution

Audio Feature Extraction After determining the primary colours for each genre, the distribution of colours and saturation levels among the audio features is decided. In other words, we develop a more specific palette for each music genre. Given the added complexity, we begin by identifying the key audio elements to extract. Since our product aims to provide real-time visualisation, we require features that can be retrieved from short music clips of around 3 seconds. The Mel Frequency Cepstral Coefficients (MFCCs), tempo, and Energy-Valence values are useful options as they quickly and accurately capture key elements of music. At this stage, sliding window technique used in Section 7 could be employed.

Mapping Features to Colour Space Next, the extracted features is mapped to the colour space. This is done using normalised feature values to define colours in an HSL colour space. Starting with the primary colour for each genre mentioned in Section 8.4.1, a range of colours will be created using colour theory principles. For instance, if the primary colour for a genre is blue, the palette could include various shades of blue, complementary colours, and analogous colours.

The appropriate number of colours for each genre needs to be determined, as different genres may require varying amounts of colours. This could be accomplished by using machine learning algorithms, such as k-means clustering, to identify clusters within the audio feature data for each genre.

Creating the Colour Palette A custom function is defined to generate variations of the primary colour by manipulating its HSL values. This creates a colour palette centred around the primary colour, providing consistency within each genre while still allowing for variations based on specific audio features. For instance, different shades of the primary colour could be used to represent different levels of energy in the music, while variations in hue could represent different moods or emotions.



(a) HSL colour wheel and primary colour.

(b) Colour palette derived from primary colour.

(c) 3D colour palette in HSL space.

Figure 8.7: Visualisations of the colour space and derived colour palette.

Figure 8.7 illustrates the process of generating a colour palette using the primary colour of blues (251, 46, 35) from Section 8.4.1. The HSL Colour Wheel in Figure 8.7a illustrates the position of the primary colour in the spectrum. Figure 8.7b then displays variations in hues and lightness as a 2D representation. These variations represent distinct audio features within the genre. Figure 8.7c expands this view, showing the full range of HSL values that are used to represent the multi-dimensionality of the data.

Training the Multi-Layer Perceptron Once the palette is established, the next step is to map the audio features to the colours in the palette. We use a machine learning model called a Multi-Layer Perceptron (MLP). The MLP is trained on our music data, learning to map the spectral features to HSL values. We

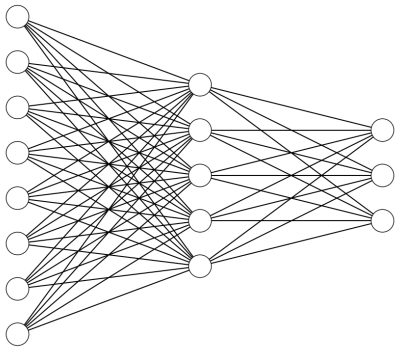


Figure 8.8: Structure of Multi-Layer Perceptron (MLP) with One Hidden Layer.

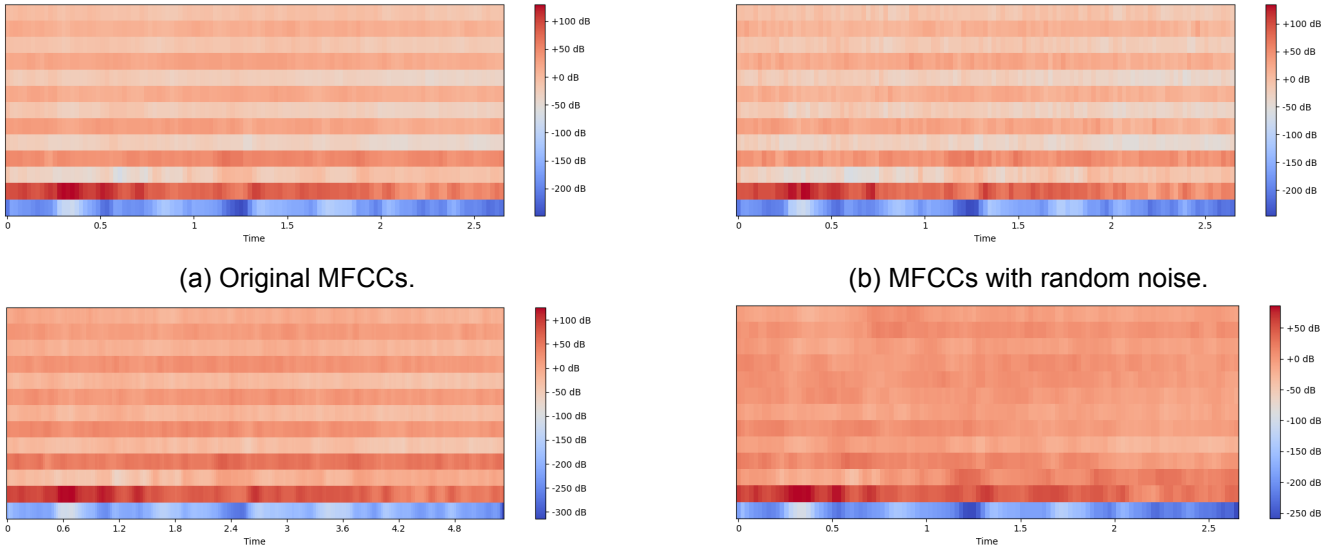
choose MLP over a Convolutional Neural Network (CNN) because MLP is generally faster than CNN - it is a more efficient choice for our real-time application [129]. Figure 8.8 demonstrates the structure of MLP by showing input, hidden, and output layers. Once the MLP is trained, it can predict the colour (in HSL space) for any given set of spectral features. This way, we have a mapping function that can generate a unique colour for any short segment of music based on its spectral content.

Colour Blending Based on Spectral Features Music is often a blend of multiple spectral features, each contributing to the overall auditory experience. Some users might prefer blending colours based on the importance of multiple audio features. To account for this, a function that takes a list of colours and a corresponding list of weights as input is used. While each colour corresponds to an important audio feature, the weight corresponds to the importance of that feature. These weights could be learned using a machine learning technique, such as a Support Vector Machine or a Random Forest, or they could be set manually by the user. The function computes a weighted sum of the HSL values of the colours, ensuring that more important features contribute more to the final colour. The weighted sum gives us the blended colour in the HSL space.

Iterative Feedback Loop Similarly to Section 8.4.1, the users are asked to evaluate the palette and its representation of the mood and emotion of the music through an iterative feedback loop. The colour mapping and the blending functions are adjusted based on the feedback received. Through this iterative process, the performance of our system could be improved continuously until it reaches satisfactory state.

8.4.3 Sensitivity Analysis

Finally, to ensure that the genre-specific colour mapping is robust and performs well across a variety of music tracks and genres, it is important to perform sensitivity analysis. This involves evaluating the performance of the colour mapping with respect to small perturbations in the audio feature data and assessing its ability to maintain meaningful and informative spectrograms in the presence of noise and variability.



(a) Original MFCCs.

(b) MFCCs with random noise.

(c) MFCCs with slower tempo.

(d) MFCCs with higher pitch.

Figure 8.9: Comparison of MFCCs under different conditions.

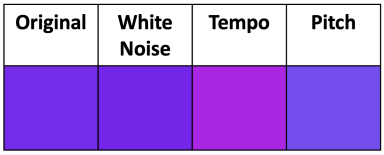


Figure 8.10: Change in colour mapping when perturbation is applied.

1. Perturbation Analysis: One approach to sensitivity analysis is to introduce small perturbations in the audio feature data and evaluate the resulting changes in the colour mapping [130]. This demonstrates the stability of the mapping function and its ability to maintain distinguishable colours for similar audio features. Figure 8.9 shows the changed MFCC when a noise is added or tempo or pitch is varied slightly. Figure 8.10 shows the corresponding change of observed colour mapping. We should assess if the colour gradations still accurately reflect the genre-specific characteristics of the music despite these perturbations.

2. Cross-validation: Cross-validation is another important part of sensitivity analysis. It evaluates how well colour mapping works on a wide range of music tracks of all kinds. This aids in identifying potential flaws in the colour mapping, such as overfitting to a specific genre or underperformance with specific music track types. For example, colour mapping could be assessed using a wide range of tracks from different genres, subgenres, recording environments, and audio quality.

3. Robustness Metrics: To quantify the robustness of the colour mapping, we can define robustness metrics that capture the stability and performance of the mapping function. This can include measures such as the average perturbation distance in the colour space, the classification accuracy of the spectrograms, and the perceptual similarity between perturbed and unperturbed spectrograms.

8.5 Conclusion

Our genre-specific colour mapping for music visualisation has been successfully designed and implemented using a combination of deep learning and clustering techniques.

We aimed to make the design abstract and easy to modify, so that it can later include more advanced techniques for the initial colour palette selection or the feedback loop. Combining the three basic principles of data-driven techniques, colour theory, and user feedback in an iterative process brings the machine learning and the artistic sensibility together. It acknowledges the subjective nature of colour perception while accurately representing the objective characteristics of different music genres.

Later in Section 9, the allocated colours are incorporated as key input parameters into the final program. This allows users to experience dynamic and attractive visuals that correspond to the genre being played.

9 Visualisation - Mark Jegorovas

In this section we will discuss the visualisation of audio, which is considered the first aspect of the visual synthesis of L.AI.GHTs. We will discuss the visualisers available, how the functionality of the visualiser we chose, and then the implementation of our visualiser.

9.1 Literature Review

Visualisation is the process in which a musical performance is converted into a static or dynamic image [131]. An effective visualisation is unique and is dynamic, meaning the visuals change when the audio changes. Visualisations are typically real-time, but can be pre-processed to get the best results. They should also have visual parameters that follow the audio parameters as closely as possible, this is called visual correlation. However, the effectiveness of a visualisation is very subjective. Biswas and Ghattamaraju [132] use two visualisations: a simple bar spectrogram to show visual correlation to an audio feature space, and a sample visual space involving a set of moving, differently sized and coloured spheres. This has a unique background with camera movement depending on the audio feature space. Although this is more of a proof of concept, they suggest even the simple visual space effectively displays visual correlation.

Lima et al. [133] conducted a survey on 51 papers related to music visualisation. Although most of these papers goals are academically orientated, some interesting observations for effective visualisations are noted. They note that interaction in visualisation is extremely important. They also note that sonic events should be simple to notice, and that the temporal logic in the audio should match the spacial logic in the visuals. Many visualisations use shapes and colour to primarily show change in audio features, with a majority of people finding these visualisations as interesting.

9.2 Visualiser Comparison

In order to create a visualisation, we can use and modify an existing visualiser. Although there are many types of visualisers, we have only focused on 5. Hydra and VEDA are WebGL visualisers utilising JavaScript (JS). Cathodemer and Lumen are software applications. Renderforest is a cloud-based service that contains music visualisers. For our project L.AI.GHTS we will need to consider visualiser customisability, scalability, cost, and responsivity. These are chosen in relation to product specification outlined in section 1.3.

Customisability is related to the product specification **S4**. It entails how many ways the visualiser can be changed to get a unique visual of the music. Good customisability is important as it allows for high visual correlation without looking generic. This means that the visualiser needs to create completely different visual designs based on factors like genre, downbeats, and emotion as discussed previously. A good visualiser should also be intuitive for the user to customise and interact with as discussed in section 9.1. This is the most important aspect as our visuals should be unique and have high visual correlation.

Scalability entails how many audio signals can be inputted and how many visuals can be outputted and therefore combined. As DJ sets involve playing multiple songs, having high scalability allows for multiple songs to be mixed simultaneously with each having a unique visualisation that can be combined, rather than one visualiser that is affected by all the songs. Multiple outputs also allows for a more modular approach which will also allow for more customisability. This aspect is the least important as one input with one output will still allow for a viable product.

Responsivity is related the product specification **S6**. It involves how quickly the visualiser changes in response to changes in audio. This includes changing the visuals with extracted audio features. Good responsivity means the visualiser should not have a large enough delay to be noticeable in the final product. A responsive system should also be easy to alter by code, such that we can can automatically change parameters without manually changing each setting.

Cost is related to the product specification **S9**. A visualiser with an upfront payment or subscription has high cost. Cost also includes if the software can be used for commercial purposes and for multiple operating systems (OS). A low cost visualiser is beneficial to reduce the initial payment for L.AI.GHTS and to have the widest audience base.

9.2.1 Hydra

Hydra is a live-codeable visual synth that is free and open source (and available for commercial use) [134]. It works like an analogue synth where filters/transformations are chained and applied to a video source to create a patch. As there are many ways to combine these filters and sources, the uniqueness of the visualisation is high. It is written in JS and runs on any browser with WebGL. The main advantages are that it is available for commercial use and can be scaled by coding in JS. As it assumes some knowledge in programming, user's may not find it easy to adjust the visualisation. It can also be combined with other JS modules such as P5.js and Meyda.js which allow for more customisability. As Hydra is live-codeable, this means we can code changes parameters of the visualisation in real time with no noticeable delay,

allowing for high responsivity. It allows for as many audio inputs and visual outputs required meaning the scalability is high. Hydra is free and works with any OS that has a browser that can run WebGL and JS. An example visual is shown in Figure 9.1.

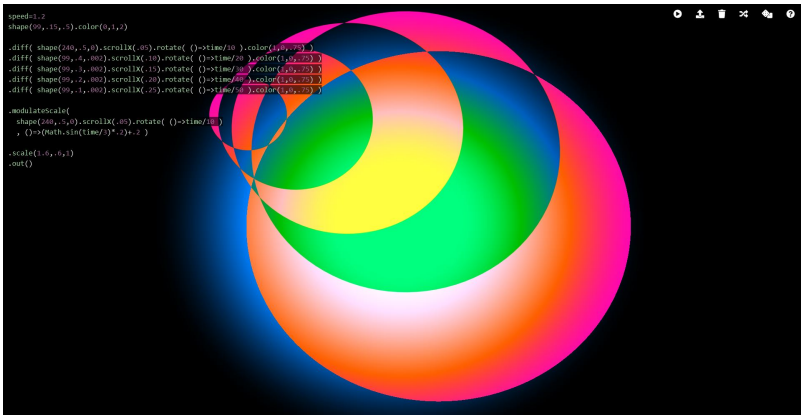


Figure 9.1: Example of Hydra output [134].

9.2.2 VEDA

VEDA is an OpenGL Shading Language (GLSL) runtime environment for Atom [135]. There is a separate JS module that allows for live coding and runtime in browsers with WebGL. All sources, filters, and effects must be manually coded meaning the customisability is high but users need to have an inherent knowledge of GLSL to have that capability. This means that VEDA is less easily codeable than Hydra which results in less responsivity. There is only one input and output, meaning scalability is low. VEDA is free and open source that is available for commercial use, meaning the cost is low.

9.2.3 Lumen

Lumen is an analogue style visual synth available for MacOS only [136]. It is semi-modular, meaning you cannot add new modules. Lumen only has oscillators and webcam footage as a visual source, and there are visual filters and effects that can be applied after the sources. This means the overall customisability is good. Lumen is a premium software, with a one time cost of $129, and is only available for MacOS. This means the cost of Lumen is quite high. There are only 3 sources available, meaning the scalability is lower than other options. There is an intuitive GUI for Lumen which allows users to customise the visuals easily. Lumen has parameters that can be changed in real time but is not easily codeable as it is software, therefore it has bad responsivity. An example of the GUI of Lumen is shown in Figure 9.2.

Figure 9.2: Example of Lumen output and GUI [136].

9.2.4 Cathodemer

Cathodemer is a pixel-based video synthesizer and CRT display simulator software[137]. It allows only 1 audio or MIDI input and there is one output which is not very scalable for our application. There are many filters and sources allowing for a good level of customisability. There is a UI for Cathodemer meaning users can easily change parameters and therefore the output. It has the ability to change parameters in real time but is not easily codeable, meaning responsivity is low. The cost is also low (currently £21) and it works for both Windows and Mac. An example visual is shown in Figure 9.3.

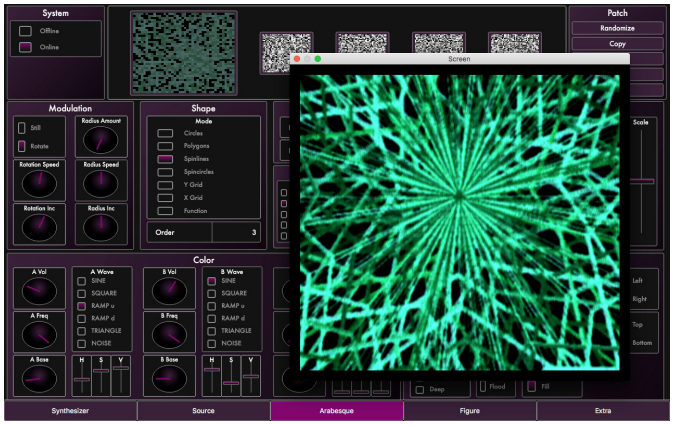


Figure 9.3: Example of Cathodemer output and UI [137].

9.2.5 Renderforest

Renderforest is a cloud-based online platform that provides services for branding [138]. It has preset visualisers where audio can be uploaded and a video can be rendered. It currently has 93 presets that cannot be customised easily, with no way for the user to create unique visualisations meaning customisability

is low. It is scalable as multiple audio sources can be uploaded and separate videos can be outputted but they cannot be combined in the service. There is no way to change the visualisation if the audio changes as the output is a video meaning there is no responsivity. The cost is extremely high with a 'PRO' subscription costing £25 a month however the service is not primarily for audio visualisation.

9.2.6 Visualiser Choice

In order to establish the best visualiser for our project, a decision matrix with weighted criteria for each of our aforementioned factors was created. A weighting for customisability, scalability, responsivity, and cost was decided as 0.5, 0.1, 0.2, and 0.2 respectively based on the discussion in Section 9.2. The resultant decision matrix is shown in table 15. Hydra is chosen as the most suitable visualiser for our project due to the ease at which parameters can be changed manually and by code, the highly modular nature, the ability to have multiple audio inputs and visual outputs, and the open-source nature of the visualiser.

	Weight	Hydra	Lumen	Renderforest	Cathodemer	VEDA
Customisability	0.5	8	5	2	7	8
Scalability	0.1	10	7	3	5	3
Responsivity	0.2	8	4	0	7	6
Cost	0.2	10	5	1	7	10
Total		8.6	5	1.5	6.8	7.5

Table 15: Decision matrix of different visualisers.

9.3 Hydra Background

We will now discuss the functionality of Hydra. Hydra works under the principle of modular synthesis, in which arbitrary changes to a visual is achieved by discrete components (modules), called patching [139]. The output is of the patch is rendered and shown in a window when rendered. All of these modules are standalone JS functions in Hydra. They can be applied to visual sources, or to each other. Modularity is introduced by allowing outputs from functions to be patched into inputs to other functions, which reconfigures the visual between the set of modules.

In Hydra, there are 5 sets of functions: sources, geometry, colour, blending, and modulation.

9.3.1 Sources

In Hydra, the signal is a visual that is generated from a source, this can be webcam footage, a video, or generated by JS code using GLSL. The sources used for L.AI.GHTS are oscillators and shapes.

Oscillators are sinusoids in which the intensity varies in the x-direction. They have three parameters: frequency, sync speed, and colour offset. The frequency indicates the number of sinusoids displayed in the output, with a frequency of 2π being one sinusoid. The sync speed is how the sinusoid changes with respect to time. Combining these, the argument for the sinusoid is $f(x+ct)$, where f is the frequency, x is the distance in the x-direction (in pixels) divided by the window width, c is the sync speed, and t is the time. The colour offset dictates how the RGB intensities varies across the sinusoid, with red being an offset behind, blue being unchanged, and green being an offset ahead. We use oscillators in L.AI.GHTs based off of Lumen [136], which creates visualisations exclusively with oscillators, therefore there is enough variety to use oscillators to create a unique visualisation. An example oscillator is shown in figure 9.4.

Figure 9.4: Example of an oscillator with frequency $10 * 2\pi$, colour offset $\pi/3$.

Shapes are polygons with three parameters: sides, radius and smoothing. Sides and radius sets the number of sides and the size of the polygon respectively. Smoothing changes the gradient of the edges of the shape, meaning how shallow the gradient of the step function that defines each edge is, with 0 being a normal step function. Shapes are used as Biswas and Ghattamaraju [132] use a sphere as discussed in Section 9.1. They easily show visual correlation to the audio and the mappings between musical features extracted and the parameters for the shapes are fairly intuitive. An example of the shapes function is shown in Figure 9.5.

Hydra can also use a webcam or camera as a source, which could be used to increase the interactivity of the visualiser. This is discussed in Section 9.4.4.

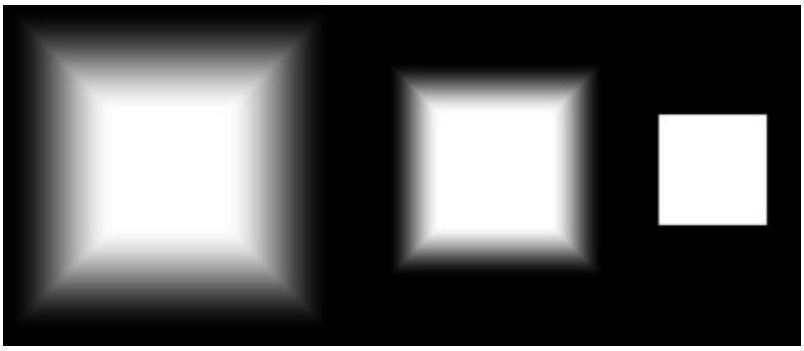
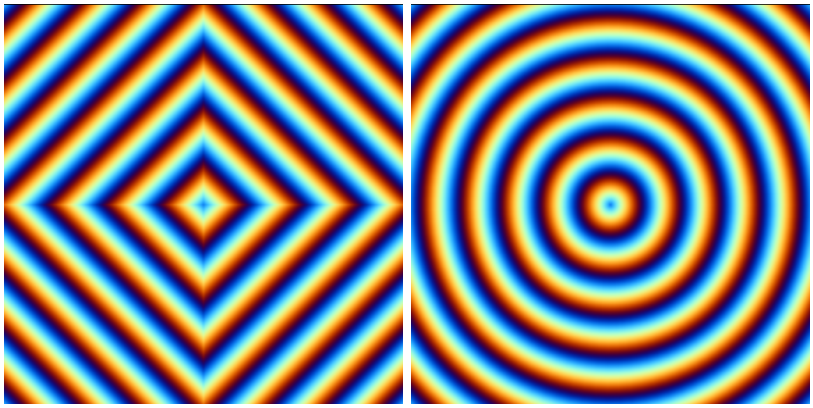


Figure 9.5: Example of the shapes source: squares with decreasing smoothing from left to right.

9.3.2 Geometry and Colour

In order to modify the visuals created by sources, we can change the spatial aspects by using geometry functions, and the colour using colour functions. The geometry functions are important as Lima et al. [133] states that the temporal aspect should match the spatial aspect of a visualisation. In Hydra, visuals can be rotated at various speeds, scaled, or moved. The most visually interesting geometry function is *kaleid*, which creates a kaleidoscopic effect for a certain number of sides. Examples of which can be seen in Figure 9.6. Figure 9.6bshows that a large number of sides creates a circular oscillator, which will be a component to the L.AI.GHTs visualiser.



(a) Kaleid with 4 sides. (b) Kaleid with 99 sides.

Figure 9.6: Example of Kaleid function on Figure 9.4.

Colour functions affect the colour characteristics of individual pixels in the source. Hydra allows both the RGBA and HSV values to be changed, the difference between them being discussed in Section 2.1.1. The alpha variable is how transparent that pixel is for that layer. This means for a low alpha layer, the layer underneath will be seen clearly. Hydra also allows values above a certain threshold to be displayed to create square oscillators. This effect can be produced by two separate function, Luma and Thresh.

Both these functions have a threshold value that the pixel intensity must be larger than, and a tolerance which dictates the gradient of the step function which is shown in Figure 9.7.

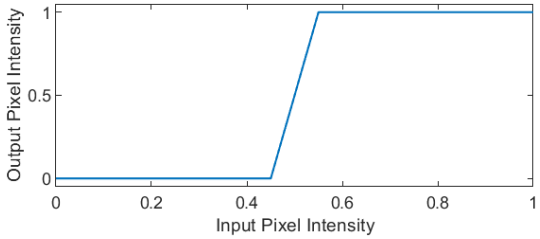
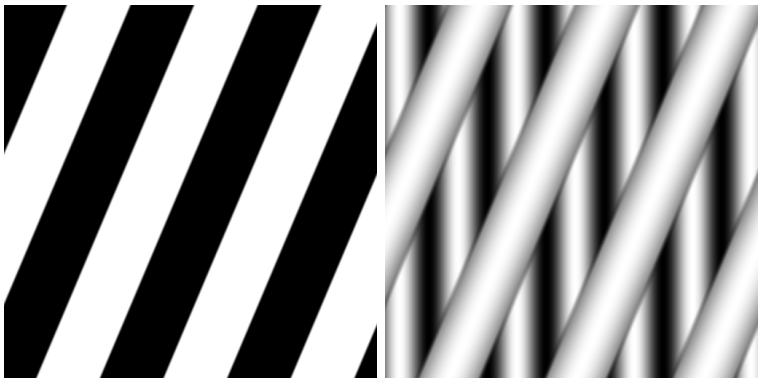


Figure 9.7: Threshold curve with threshold of 0.5 and tolerance of 0.05.

The Luma function preserves the RGB values of the input source, whilst Thresh maps all RGB values to the same value (equal to the output intensity). A comparison of these functions are shown in Figure 9.8. The figures show two oscillators layered on top of each other, with the functions applied to the top layer with the same threshold and tolerance. In Figure 9.8a the Thresh function is applied to the top layer, however as it is not transparent, the bottom layer cannot be seen. It is important to note that the alpha is still preserved and that the output is all mapped to the max pixel intensity, creating a square wave. In Figure 9.8b, the Luma function is applied to the top layer and as it is transparent, the bottom layer can be seen. In addition, the colour of the top layer is preserved, with the sinusoidal variation in pixel intensity being preserved.



(a) Thresh applied to top layer. (b) Luma applied to top layer.

Figure 9.8: Comparison of the Thresh and Luma function.

9.3.3 Blending and Modulation

An important aspect to L.AI.GHTs is the ability to create a unique visualisation for each song playing in a DJ set and combining them. This can be achieved either by blending or modulation.

Blending involves the direct combination of pixel values between two textures. The RGB values of corresponding pixels can be added or multiplied together, subtracted from each other, or the difference between them can be used as the output pixel value. All of which provide a unique effect. Outputs can also be blended together, where a weighted average of RGB values between pixels are taken. Two outputs can be layered on top of each other which requires outputs to be transparent (which is where alpha is useful).

Modulation involves taking an input (a source) and a modulator (another source), and changing the geometry or colour aspects of each pixel of the input depending on the RGB values of the corresponding pixel of the modulator. The modulator distorts the input with respect to its own pattern. Each modulate function has two parameters, the modulator source and the amount of modulation. The amount is a linear modifier to the modulation, with 0 being no modulation.

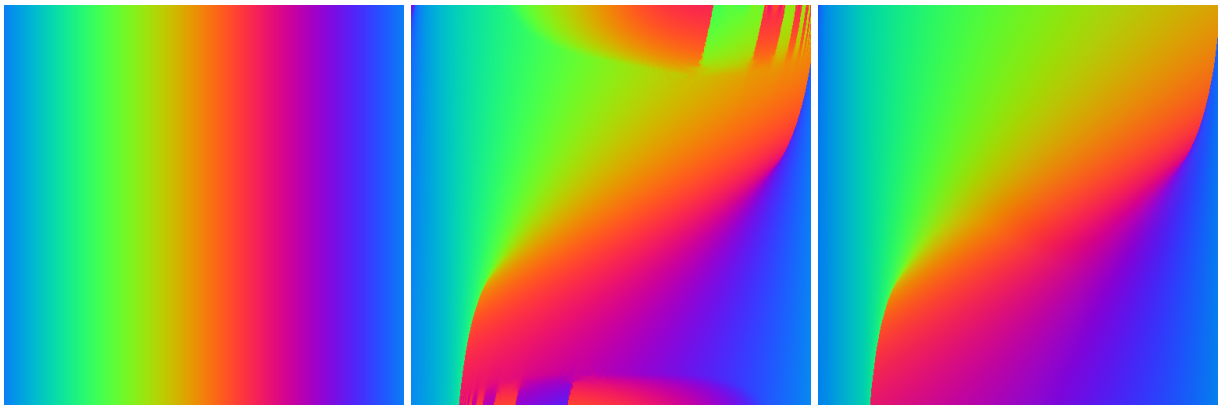
The pseudocode for the base modulate function is shown below. Each pixel in the output is a shifted pixel from the input, with the shift in the x and y direction being proportional to red and green channels of the modulator respectively.

```
Input;                                  /* Input: Input Frame (2d matrix of pixels) */
Modulator; Amount;          /* Parameters: Modulator Frame, Amount of Modulation */
Output;                                              /* Output: Output Frame */
y ← 0; x ← 0;
while y < Window Height do
    while x < Window Width do
        ModPixel ← Modulator [y] [x];
        yNew ← y + ModPixel.Green × Amount;
        xNew ← x + ModPixel.Red × Amount;
        Output [y] [x] ← Input [yNew] [xNew];
        x ← x + 1;
    end
    y ← y + 1;
end
```

The issue of modulation using this algorithm is that $yNew$ and $xNew$ can be larger than window height and width respectively. The way Hydra prevents this is by modding $yNew$ and $xNew$ by the window height and width respectively. However, there are usually large discontinuities in pixel values between the top and bottom of the window, causing discontinuities in colour for the output.

An example of this issue is demonstrated in Figure 9.9. The input source is shown in Figure 9.9a, which is then modulated by itself. This output is shown in Figure 9.9b. There are discontinuities in colour making the output visually unappealing. This effect is more apparent around the top and bottom edge of the

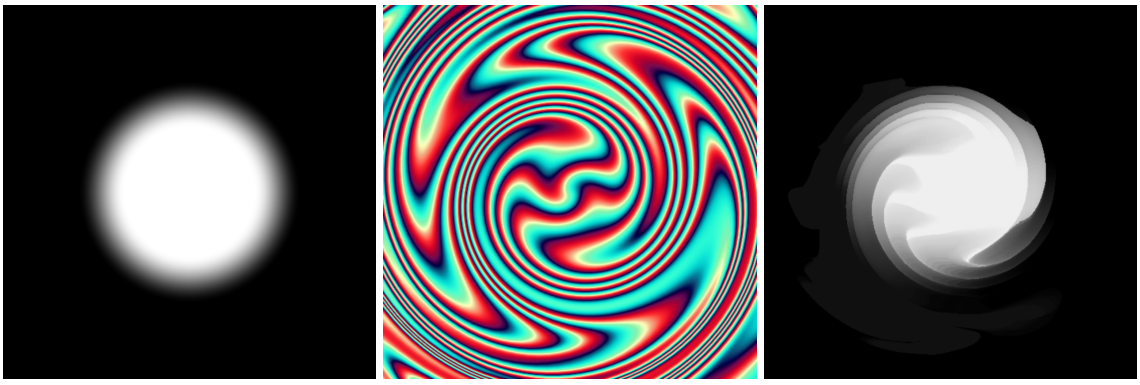
window. This can be corrected by blending the output with itself. The corrected output is shown in Figure 9.9c.



(a) Input and Modulator. (b) No Blending. (c) Corrected output.

Figure 9.9: Example of correcting a modulated output.

Most of the geometry/colour functions discussed in Section 9.3.2 have their own respective modulate function, with each using a certain number of RGB channels to modulate. This is due to each function only having a certain number of arguments, so as the base modulate function shifts in the x and y direction, two arguments and therefore two channels are required (which are selected from left to right in with respect to the acronym RGB). Other modulate functions such as modulating the scale only have one argument and thus requires only the red channel [140]. An example of hue modulation is shown in Figure 9.10.



(a) Input source. (b) Modulator. (c) Output.

Figure 9.10: Example of hue modulation.

An important aspect to creating unique visualisations is self-patching, in which the output of a function is used as the input of the same function, creating a visual feedback loop. With self-patching, analogue systems and digital systems behave differently, with analogue behaviour being approximated as a time

delay in the digital system. Blending and modulation is also the main way to implement self patching, with the texture or modulator being the output of the patch. This is achieved by buffers, which are the final outputs of a patch. They are typically numbered starting from 0. Each of these buffers can then be inputted into the modulation function as the modulator. This is shown in Figure 9.11.

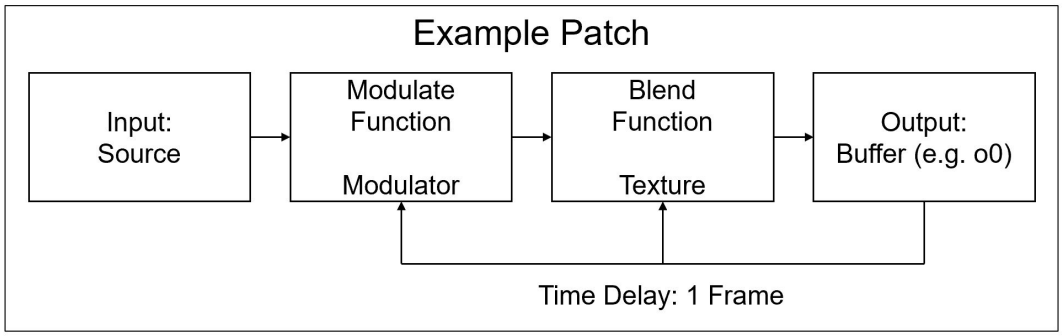


Figure 9.11: Example of self patching, where the output is the modulator.

A visual example of this patch is shown in Figure 9.9, with the input shown in Figure 9.9a, and the output shown in Figure 9.9c.

The ability to create multiple buffers that all have different patches allows us to feedback, modulate, and blend multiple outputs created from different songs, allowing for extremely unique visualisations.

9.3.4 Creating Dynamic Visuals

In order to create a visualisation, we must start with a source, with each function (geometry, colour, blending or modulation) being 'chained' onto that source. This is called a patch. A simple example patch is shown as a flow chart in Figure 9.11. The modulate function has been chained onto the input source. In Hydra, if the source was an oscillator, this would be implemented as:

```
osc(10, 1, 1).modulate(o0).blend(o0).out(o0);
```

The *osc* function is the oscillator source with a frequency of 10, scroll speed of 1, and colour offset of 1. This is chained by the function *modulate* which modulates the source by the 0th buffer by the default amount, and the function *blend*, which takes an average of each pixel value. Then the function *out* stores the output visual to the 0th buffer. The buffer can then be outputted onto a window using:

```
render(o0);
```

Which displays the resultant visual stored in the 0th buffer in the window. If there are multiple buffers being used, we can choose which buffer to output onto our window by changing the parameter to the

desired buffer. This means we can have multiple visualisations running in the background, and that the final output can be changed between them during a live performance.

In the current state, the above patch only has constants as parameters. This means that as the song changes, the visualiser will not react, causing low visual correlation. This means we will want to change parameters of certain functions as the songs progress. For example, we may want the scroll speed to be proportional to the tempo of the song, changing the example scroll speed to a value other than 1.

A naive way to change parameters is to input a new constant into the function and to re-render the output onto the window. The main issue being that the whole window will be re-rendered, meaning there will be some delay as the new patch is processed and subsequently rendered as a whole new instance of the buffer will be created rather than keeping the same buffer. The delay introduced is approximately 200ms for large patches, which violates product specification **S1**. This causes visible stuttering when changing parameters which severely affects the performance of the visualiser. Another issue is that the visual will be reset to its default state, meaning there will be a discontinuity of geometry and colour. This reduces the smoothness of the visualiser, making it more unappealing to look at. However, this method of re-rendering has to be used when changing the actual patch itself, such as adding or removing functions onto the patch.

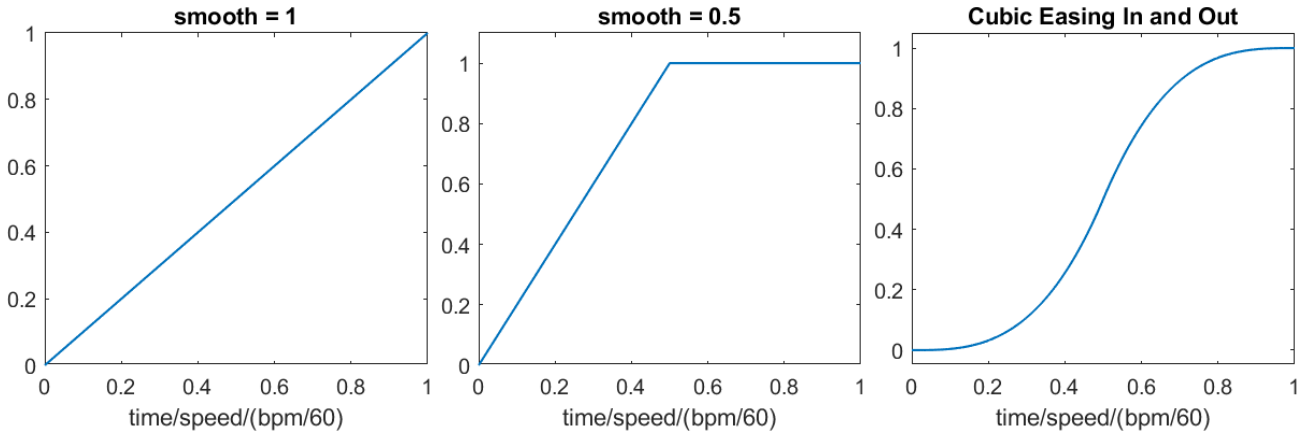
The second way to change parameters is to input an array as a parameter. Hydra will then switch the parameter after a set amount of time, the algorithm is shown below:

```
index = time * speed * (bpm / 60);
value = array[ floor(index % array.length) ];
```

The two variables dictating how quickly the output value changes are *speed* and *bpm*. The *speed* is a local variable in the scope of the array, whilst the *bpm* is a global variable, with its default being 30. The bpm variable allows us to align the changes of the visualiser with the beats of the song as discussed in Section 5. The index is then modded to the array's length to make sure it is in an indexable range for the array and the floor is found so the output can be extracted.

There are other methods built into the Hydra array that can change how the parameter varies with time. We can also choose to interpolate between array values. This easing function is by default linear with a gradient dictated by the value of *smooth*, with the value being the proportion of time spent increasing. The difference between *smooth* values is shown in Figure 9.12a and 9.12b. We can also choose the type of easing function. These typically involve a polynomial with easing 'in' to the next value and/or 'out' of

the next value. An example of an 'in' and 'out' cubic easing function is shown in Figure 9.12c.



(a) *smooth* is 1. (b) *smooth* is 0.5. (c) Cubic Easing Function.

Figure 9.12: Different smoothing functions, y axis is proportion of next array value.

This allows us to work with changing parameters without completely recreating the buffer, and as Hydra re-evaluates parameters every frame, creates continuity with the geometry and colour. This way of changing parameters works well when we have preprocessed values for these parameters that we can change every 'frame' or once every beat such that we can create the array with the easing function and speed desired, and then input it into the parameter of the function.

The final way to change parameters is to input a function as the parameter. Hydra will then evaluate the function every frame. For example, if we want a sinusoidal variation in the oscillator frequency from 30 to 40 with a time period of 20 seconds we can achieve this as follow:

```
let lfo = () => 30 + 5 * Math.sin(time * 2 * Math.PI / 20);
osc(lfo,0,0).out();
```

This is the same as applying a low frequency oscillator (LFO) that modulates the oscillator frequency. The arrow function syntax for JS allows for concise, local, anonymous functions that make this method of changing parameters best when we do not need to directly synchronise a parameter with the song. For example, we can evaluate audio's loudness every frame, and change the size of a shape accordingly. Using a function instead of an array means we can have data structure consisting of the time and the corresponding parameter value, rather than a large array with parameter values that are padded in order for the parameter change to happen at the correct time.

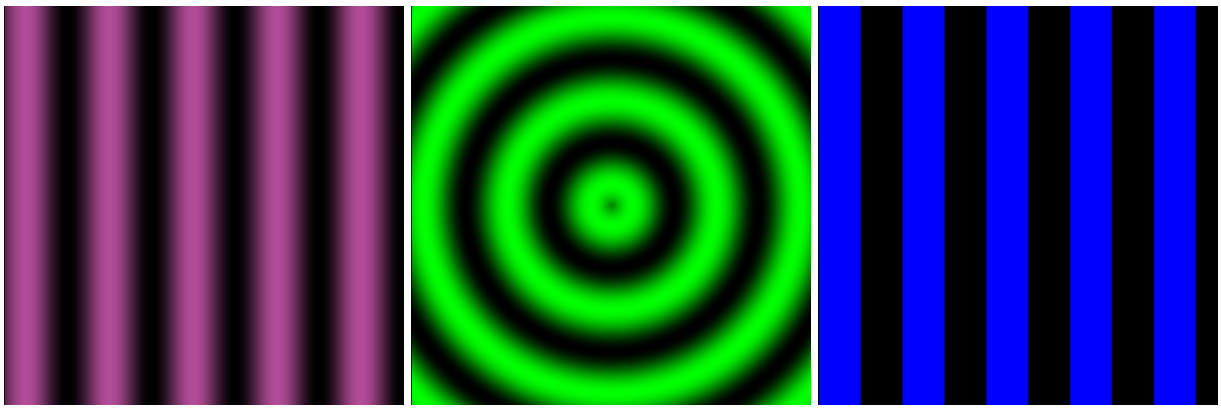
9.4 Implementation

The final visual involves layering a background and foreground visual. The background involves combining a selection of patches for a set of oscillators. This is based on Lumen [136], which creates a visualisation with a set of 3 oscillators, as described in section 9.2.3. We can work under the assumption that a set of oscillators can create a unique and varied visualisation.

The foreground involves a sphere that has texture mapped onto it. This is based on the implementation by Biswas and Ghattamaraju [132], which is described in 9.1. The use of a sphere means we can easily map the audio feature space to the visual space. Hence, we can combine these two aspects and use a set of oscillators as a background, with a set of spheres as the foreground of the visualisation.

9.4.1 Background Visual

The background visual involves combining a selection of patches for a set of oscillators. In order to create a customisable set of oscillators, we will need to create a set of patches that the each oscillator can default to. Each patch will be initialised by an oscillator source, in which the frequency and scroll speed can be changed. These can be synchronised to the beats obtained (as discussed in Section 5). The colour offset does not need to be set as we can directly select the colour we want with respect to the colour mapping as discussed in Section 8. An example of 3 different patches are shown in Figure 9.13. Figure 9.13a shows a purple oscillator, Figure 9.13b shows a green circular oscillator, and Figure 9.13c shows a blue square wave oscillator.



(a) Purple Oscillator. (b) Green Circular Oscillator. (c) Blue Square Oscillator.

Figure 9.13: Example of 3 oscillator patches.

These oscillators can then be outputted onto their respective buffers and combined together in a final patch using blending and modulation as described in Section 9.3.3. This final patch can be outputted as

a separate buffer and rendered to use as our final output visualisation. The use of buffers also means that we can create patches that use feedback to create interesting visualisations. Two example outputs combining the presets in Figure 9.13 are shown in Figure 9.14.

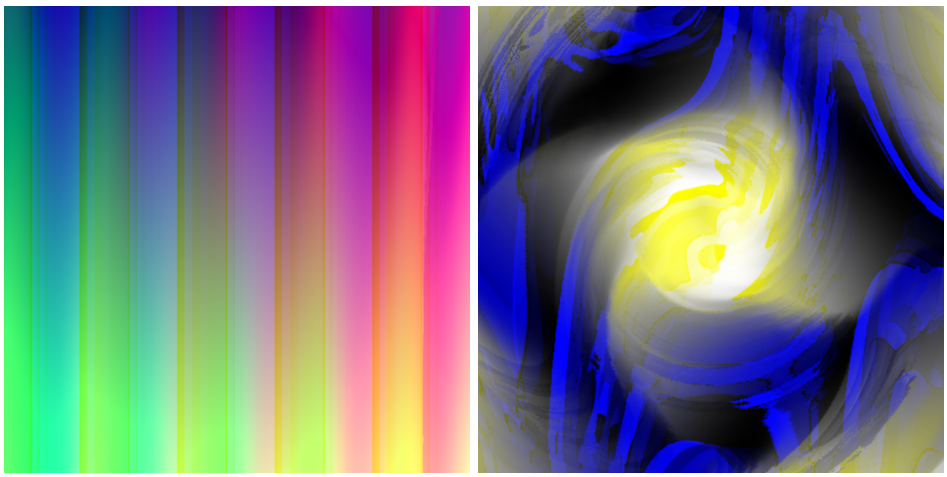


Figure 9.14: Examples of final outputs created by combining the presets in Figure 9.13.

We can also map audio parameters to visual parameters easily, this is discussed more in Section 9.4.3.

9.4.2 Foreground Visual

The foreground involves a sphere that has texture mapped onto it. As Hydra does not provide an implementation to create a sphere, we will need to generate a sphere in JS by either using Three.js [141], or by creating a custom Hydra function.

The first method utilises the Three.js library, which already has a GLSL implementation to generate a sphere. We can apply different materials, make the background transparent, and layer the spheres onto our background generated in Hydra. We can also add lighting, for which different materials will look different. Figure 9.15 shows two spheres generated using Three.js. The left and right sphere use the Lambert and Phong materials respectively.

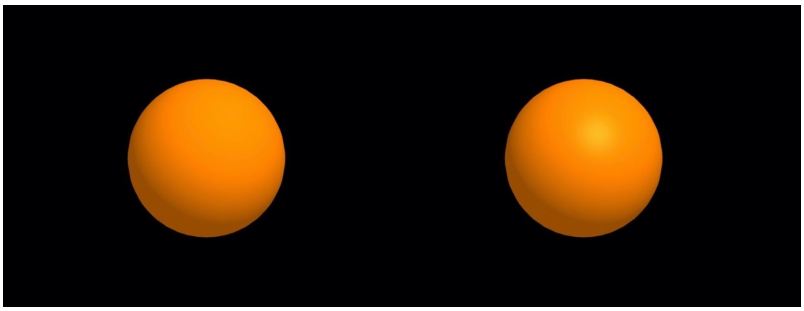
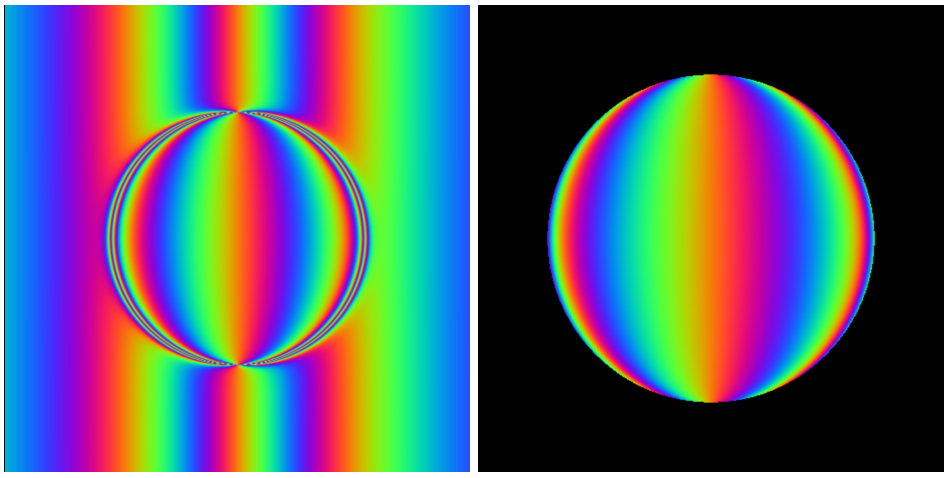


Figure 9.15: Example of two spheres with different materials generated using Three.js.

The main benefit of this method is that we can add new materials, textures, and shaders to each sphere creating a unique look. We can also create many unique spheres that are independent of each other.

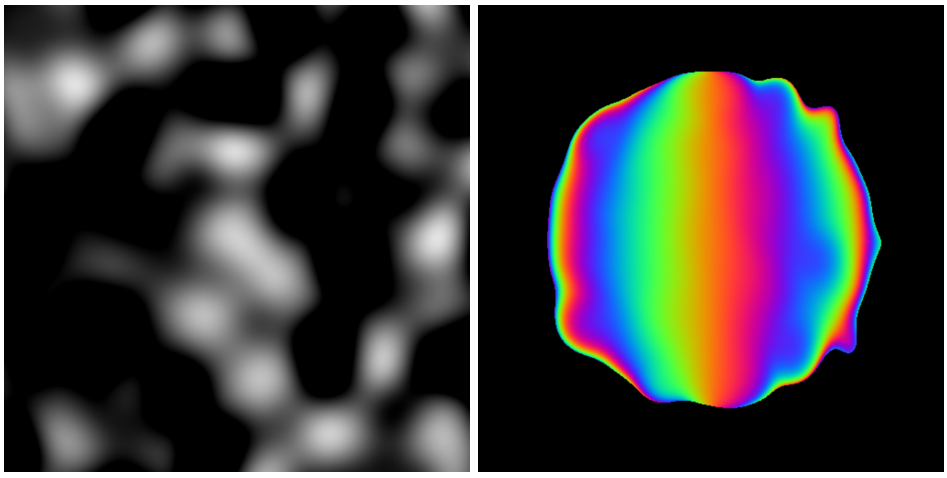
The second method involves creating a custom Hydra function to generate a sphere-like texture. Hieda [140] provides an implementation that creates a sphere by displacing the input texture it is chained onto. An example of the displacement of the input texture is shown in Figure 9.16a. We can remove the background by setting any pixels outside the radius of the sphere to be transparent (0 alpha). An example of the sphere without the background is shown in Figure 9.16b.



(a) Sphere with background. (b) Sphere without background.

Figure 9.16: Examples of spheres generated by a custom Hydra function.

This function can input another texture separate from the input as a modulator. This modulates the radius of the sphere based on the colour channel of the modulating texture. An example of the modulator is shown in Figure 9.17a, and the output sphere in Figure 9.17b.



(a) Modulator. (b) Sphere with modulated radius.

Figure 9.17: Example of a modulated sphere.

The main benefit of using this method is that this function can be easily implemented into our codebase and included into a patch. This method also allows use of the oscillators described in Section 9.4.1 as the input or modulating texture. This means that the Hydra implementation is better than the Three.js implementation.

In conclusion, to generate the sphere in the foreground we will use a custom Hydra function. This is because implementation is simple as it does not require an external library. In addition, it provides lots of customisability by allowing the input and modulating texture to be changed.

9.4.3 Outcome

The foreground visual can be layered onto background visual to create our final visualisation. An example of which is shown in Figure 9.18.

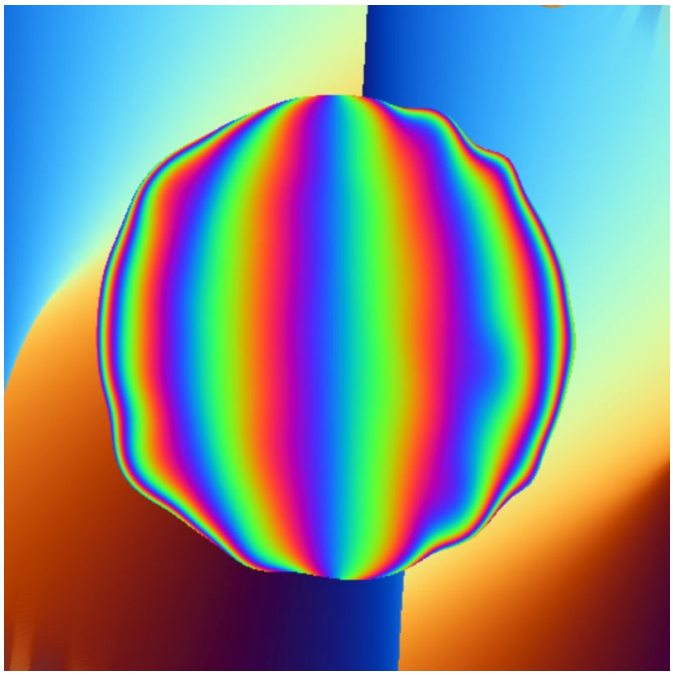


Figure 9.18: Example of the final visualisation.

We can then utilise audio parameters as described in Section 3 to make the visualisation adaptive in real time. This can be achieved by utilising the library Meyda.js [142]. This library is already included in Hydra, and therefore we can extend its functionality to change visual parameters in real time. We can achieve this by the arrow function as described in Section 9.3.4. As an example, we can change the radius of the sphere in the foreground relative to the loudness of the song. The loudness of the song can be set as a variable using Meyda.js. We then have an arrow function that evaluates that variable, such that every

frame, the radius changes proportional to the loudness of the sphere. This will allow us to achieve good visual correlation in our visualiser.

Once we have our visualisation, we can use this to inform aspects of our lighting sequence. For example, if we have 8 groups of fixtures, we can extract the average colour information, the overall intensity, and the rate of change of these values from each of the 8 segments of the visualisation as shown in Figure 9.19. This can then be used to create a unique mapping for each of the groups of fixtures.

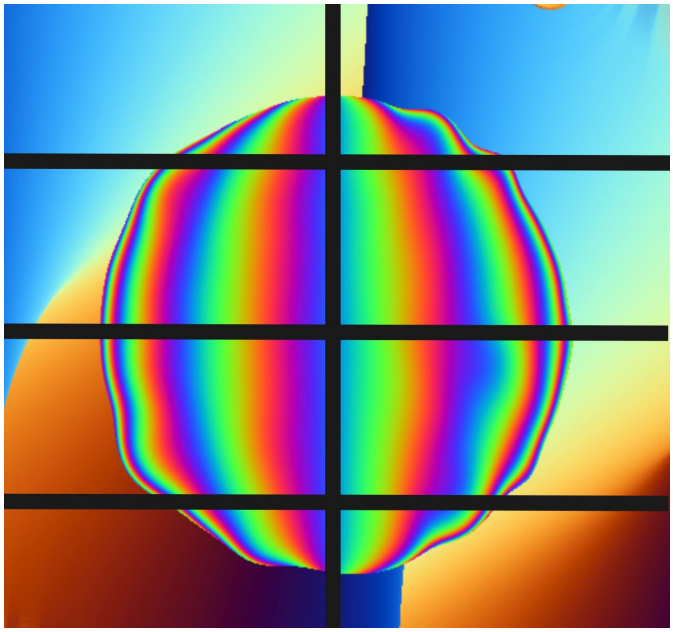


Figure 9.19: Example of a visualisation segmented into 8.

9.4.4 Interactivity

As an extension to the L.AI.GHTs project, we could add interactive elements to improve the dynamics of the visualisation. Hydra can use mouse parameters, webcam footage, or a screen recording as a visual source. We can use these visual sources to enhance the visual performance, as these provide interactivity.

Bonde et al. [143] use a Kinect to generate visual emotional cues due to changes in the crowd's behaviour in a live music setting. We could then use the emotional cues generated and output the changing background onto the camera recording the crowd to generate in interesting visual, with crowd interactivity. As noted by Lima et al. [133], interactivity is important for interesting visualisations, therefore including crowd interactivity as well as DJ interactivity which could generate an even more interesting visual result.

10 Implementation - Oli Wing

Our software is designed to run in real time as a DJ performs their set. This means it needs access to the song being played to generate the visual patterns, and the playback speed and the playback position to synchronise the visuals to the audio correctly. This information is available within the DJ software used to mix the audio. Therefore, it is logical to design our software as a plugin that operates inside DJ software. This dramatically improves user experience compared to a standalone application that would require each song to be input twice (once to the DJ software and once to the visualiser). An increasing number of popular DJ programs offer plugin support making this the most viable option.

Audio plugins have several formats, the most widely supported of which being VST [144]. Nearly all VST plugin are written in C++; the single most important aspect of an audio plugin necessary for a good user experience is speed making C++ the most effective choice. This is because C++ is a compiled language so is fewer abstractions from machine code. As C++ has become the standard for audio plugins, existing libraries have been constructed to facilitate plugin development. One example is JUCE [145], an open source framework that is the used in a large range of successful audio software products (including several DJ mixing programs). An advantage of using a well-known framework is it makes software development easier as many developers will be familiar with using JUCE.

Developing the software in C++ seems as though it presents a dilemma; almost all of the functions in our product require either Python (downbeat tracking, mood analysis) or Javascript (visualiser). Fortunately, this issue is circumvented by embedding the Python and Javascript components within C++ as the individual components of the visualiser only require the outputs of other components and do not need to access internal values. For Python, this is done using a C++/Python API [146].

We interface the Python needed for the MIR functions and the JS needed for the visualiser by exporting the output from each of the MIR processes as a JSON. Using JSON as an input to the visualiser means despite requiring different programming languages, separate components of our software are easily integrated.

11 Engineering In Society

In this section we will contextualise the ethical implications, evaluate the risk, and provide a forecast of the finances of L.AI.GHTs.

11.1 Ethics - Mark Jegorovas

When considering the ethical implications of our project, we can look at the code of ethics for software engineering provided by the IEEE [147].

When considering the public, one key issue is making the visualisation and lighting accessible for people with disabilities. An example of which is a fast strobe light causing seizures in people with photosensitive epilepsy, with frequencies between 3 to up to 60 Hz [148] commonly causing seizures. Therefore, an option to either switch off, or reduce the frequency of strobe to below 3 Hz should be available. In addition, light and dark contrasting patterns may cause seizures. Therefore, allowing a patch for the visualiser to use more muted colours with less contrast and with oscillations being less tightly packed should be available.

Another issue is collecting user data for use in our algorithms, especially when developing a colour map as discussed in section 8.4.1. A key aspect of visual synthesis are that they are subjective, and users may see different visualisations and lighting schemes as more or less desirable. This means user feedback is essential in selecting which aspects of the visuals to exemplify or to damper. The feedback data will need to be collected from voluntary participants with informed consent that their feedback and use of the product will be used to improve the product. All feedback will also need to be anonymous, confidential, and adhere to the GDPR [149]. Ethical use of data collected will minimise the potential for legal liability for our project, and will minimise social and psychological harm for users.

When considering colleagues in the performance arts, the main ethical issue is the replacement of lighting designers by software. There were over 10k lighting designer jobs with 11% belonging to media industry in the US in 2022 [150]. Our software effectively phases out the need for DJs to hire lighting designers. However, our market demographic focuses on amateur DJs, therefore it is important to note that many of them may not be able to afford to hire lighting designers for their sets, therefore our project will provide a good alternative to create immersive and responsive lighting. In comparison, professional DJs may want the visual aspects of their live performance to be more bespoke and may choose to not utilise software such as L.AI.GHTs. Therefore, the ethical implications of our project in this regard is minimised.

11.2 Project Risk - Felix Cohen

An integral factor in determining a project's success is how well its scope is defined. The scope of a project is the work that needs to be done in order to create a product that satisfies the decided-upon product specification. The PERIL database [151] has been contributed to by hundreds of project leaders, listing typical past project problems and the extent of their impact. Over half of the impact data was represented by scope-related risks. Moreover, Leffingwell [152] argues that due to the intangible nature of a software product, it is harder to relate to its form and function. This applies to both team members and customers: in regard to surveys revealing market preference, customers may only understand what they truly desire when presented with a prototype they can interact with. Consequently, the risk of a flawed understanding being used to derive product specification, which therefore leads to a poor product scope, is high. A drastic change of scope late into a project will inevitably lead to the waste of resources and the delay of the project (which itself will lead to even more resource loss via overheads and increased person-hours). In a worst-case scenario, this could completely deplete the project's budget leading to its failure. It is of the utmost importance that this high-impact-high-probability risk is mitigated.

This can be achieved by forgoing the traditional waterfall method of management and adopting an agile management methodology, in which a perfect understanding of the project specification at the beginning of the project is not assumed. Instead, rapid prototyping followed by meetings with stakeholders and customers leads to the iterative refinement of the product specification, this is known as a sprint.

Another scope-related risk is an external change in the industry, such as the release of new technology, forcing a scope change upon the project. This is especially relevant due to Artificial Intelligence being such a rapidly advancing field. Krenn [153] reported that the number of papers published per month in the AI category is growing exponentially, doubling every 23 months. Agile development is more suited to such a rapidly evolving environment as the constant project scope re-evaluation prevents excessive investment into technologies that could be outdated at the end of the longer traditional waterfall cycle.

The 16th annual State of Agile Report [154] discovered that over 87% of its 3220 respondents used Agile Scrum methodology. The largest population in this survey was companies in the technology sector. Therefore we propose that our team adopts the agile scrum methodology. A scrum team consists of three archetypes:

1. The Product Owner: This role is responsible for maintaining the project backlog and prioritising its items. The project backlog is a list of the project's requirements and objectives. The Product Owner

defines the vision of the project whilst managing the expectations of different stakeholders [155].

2. The Scrum Master: This role has authority over the scrum process, for example being able to change sprint duration. The Scrum Master ensures the team is applying scrum principles effectively.

3. The Development Team: This is an inter-functional self-organising team responsible for delivering a potentially shippable product at the end of each sprint.

Figure 11.1 describes the Scrum process, during a sprint meeting it is decided which of the product functionalities in the project backlog should be moved to the sprint backlog, the list of items the team will deliver at the end of the sprint. During the sprint, every day, there will be a short meeting known as the ”Daily Scrum” where every team member articulates what they have completed since the last meeting, what they aim to complete before the next and what stands in their way. This allows the team to communicate and plan their immediate work. The sprint retrospective occurs after the sprint has finished and is when the shippable product is presented to all relevant stakeholders, allowing them to offer their feedback. This allows the project owner to inspect and re-prioritise items on the project backlog. The final step is the project review which is an opportunity for the team to discuss what areas they can improve upon. Overall, the Scrum methodology's self-organising nature empowers teams and the sprint reviews act as a feedback loop determining the optimal project specification and providing the flexibility to react to changes in the market.

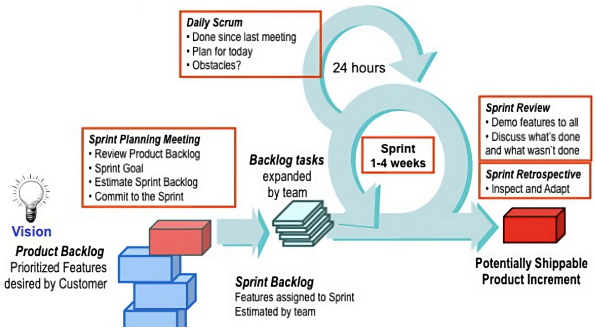


Figure 11.1: Agile Scrum Cycle taken from [156]

One potential downside that is amplified by this constant incremental shifting of product specification is the risk that a change to the code implementing a new functionality may lead to the loss of another essential functionality. Establishing a robust framework to combat this risk now will have numerous advantages as this can be adopted and used during the continual improvement of the software after it has been released. As stated by Humphrey [157], small code changes and maintenance may be 40 times

more error-prone than initial development.

One common strategy to combat this is Test-Driven-Development, each project requirement must be defined by creating a test that the current project code will fail, the objective now is to adjust the code, doing whatever is necessary, until the test is passed. The code can now be improved and refactored, eliminating all duplication you may have created in the previous stage [158]. This gives developers confidence that their peers' work will be reliable and free of bugs as any change that affects the functionality of the product will be immediately made apparent. This also decreases the time taken to debug code as the test results report what exactly is broken and what its expected output is; there is no need to spend time searching for the issue. In 2017 the technology sector had the highest turnover rate out of any sector at 13.2% [159], therefore it is very likely employees will have to examine and alter code written by someone other than themselves. Minimising the effort required for debugging is therefore of particular importance. It could be argued that the increase in time taken to create a rigorous testing scheme in which no test is coupled and every test runs as quickly as possible may not be worth the benefits test-driven development brings. In order to quantify the impact of adopting TDD, Table 16 records the resultant shift in defect density between teams that adopted TDD and a team within the same organisation that had not adopted this strategy. Management estimates of the resultant development time increase were also recorded. Although the use of TDD leads to a consistent increase in time taken, all the teams agreed that this was offset by the reduced maintenance costs due to the decrease in defect density [160].

	IBM: Drivers	**Microsoft: Windows**	**Microsoft: MSN**	**Microsoft: VS**
Decrease in Defect Density	39%	62%	76%	91%
Increase in Time Taken	15-20%	25-35%	15%	0-25%

Table 16: Impact of adopting TDD for 4 teams at Microsoft and IBM [160].

Table 17 summarises these discussed risks and the proposed methods for combating them.

Risk	**Likelihood**	**Impact**	**Preventative Measure**
Poorly defined initial project scope	High	High	Agile Scrum Project Management
Change in external environment forces scope change	Medium	High	Agile Scrum Project Management
Project delays due to poor quality code and debugging	High	Medium	Test-Driven Development

Table 17: Risk Assessment.

11.3 Project Finance - Oli Wing

We have created financial projections for months 0-12 of our business as shown in Table 18. Taking a bottom up approach to our financial analysis, our projections rely on estimations of direct and indirect costs. These estimations are based on information about the necessary expenses for our business. However, since our product is extremely niche, there is almost no financial information available on similar companies to compare with and it must be accepted our projections have considerable uncertainty. Despite this we have attempted to make as realistic estimations as possible. As our product is targeting a global market, all prices are given in USD.

11.3.1 Turnover

Our product will have 2 versions that will be sold at separate prices. First will be the base visualiser. This will cost $99. This is cheaper than our closest competitors for visualisation software (Lumen - $108 [136] and Visual Synthesizer - $129 [161]). Unlike our competitors, our visualiser is fully automated and requires no prior knowledge of video synthesis to produce professional visuals. This means it appeals to a wider audience as the market for amateur DJs is large and many would wish to enhance their performances with automatic visuals that can still look good with inexpensive equipment (projectors). The wider market means we can set a price lower than the competition while still being profitable which makes our product highly viable. Based on the size of the global DJ equipment market [162][163], we predict 555 sales of the $99 version in the first 12 months, with sales per month increasing towards the end of the year (global software sales rose to 13% above the yearly average in Q4 2022 [164]).

The second version will be the full visualiser with the ability to create a DMX output to automate lighting. This will be priced at $199. This version is priced higher to reflect the smaller target demographic - small venue owners who do not want to pay for lighting specialists (as well as some DJs). This is in the same price range as other lighting control software [165] and has the same advantage of the visualiser, being fully automated and having no barrier for entry. Based on the size of the stage lighting market [166] and number of music venues worldwide, we predict 125 sales of the full $199 version in the first 12 months, with 10-15 average monthly sales.

The turnover for our business is the revenue generated from visualiser sales. We expect 0 turnover for the first 3 months while the software is developed.

11.3.2 Direct Costs

The main direct cost of running a software start-up is software development. It is difficult to accurately predict how much our plugin will cost to develop as no direct competitors exist and it is hard to extrapolate from the little data available on other similar products (audio VSTs). This cost is almost entirely from paying the salaries of the software engineers. Therefore it is logical to outsource the development of our product to a country with a large pool of software developers offering lower rates, the most ideal choice being India. Average hourly pay is $25 [167] and 3 of the most popular programming languages in India are Java, Python and C++ which conveniently are the three main languages our software requires. We assumed 2 developers on $25/hour working 37.5 hour weeks for 12 weeks, resulting in a direct monthly cost of $7500 for the first 3 months.

Once the initial software is made, it requires maintenance. This will consist of bug fixes and adding features keep our product relevant and competitive. It is indefinite. Again, this is hard to estimate for our product as almost no public information exists on plugin maintenance costs. We assumed 1 developer on $25/hour working 8 hour weeks, resulting in $1500 monthly for the rest of the year.

11.3.3 Indirect Costs

It is recommended that small start-ups spend around 10-15% of revenue on marketing [168]. As a real-time AI visualiser like ours currently do not exist, it is important that we inform potential customers of our product as they would not be searching for it otherwise. Therefore for months 4-6 we will spend 20% of turnover on advertising to establish our visualiser within the market, then 15% after that. Since our product is targeting a fairly niche market, small scale targeted advertising (in DJ magazines such as DJ Mag) will be most effective as it is usually cheaper than mass marketing and has a higher conversion rate. Another necessary cost is creating and maintaining a website to distribute the visualiser from. We assumed 1 developer on $25/hour working 30 hour weeks for 10 weeks to create a website and distribution infrastructure whilst our software is developed, and $40 monthly on upkeep/hosting after.

11.3.4 Business Loan

To fund the cost of developing our visualiser software and setting up the infrastructure to sell it, we will need a loan. The most suitable for our company is a term loan - these provide businesses with an amount of money to be paid back over a fixed period [169]. The nature of software development means the direct

costs are disproportionately high for the first few months but decrease significantly once the software has been developed. Thus a term loan is ideal as a loan is only necessary for the initial development of the software; once this is completed, revenue generated from sales will pay for maintenance and overheads. To cover the initial costs of software and web development, a loan of $50,000 is sufficient. Using a modified version of the amortisation formula shown below, the amount to be repaid each month can be calculated.

$$r' = \frac{P((1+i)^{1/12} - 1)}{1 - (1+i)^{-N}} \tag{33}$$

P is the initial loan, i is the annual interest rate, N is the time over which the loan is payed off in years and r' is the monthly repayment amount. Taking the initial loan as $50,000, interest rate to be 8 and the repayment period as 1 year (approximations based on typical values for short term loans as exact loan conditions vary case by case), the monthly repayment is $4342.

	Q1			**Q2**			**Q3**			**Q4**		
Month	1	2	3	4	5	6	7	8	9	10	11	12
Turnover	0	0	0	3965	5950	6940	7930	8925	9920	10910	12395	12885
Direct	7500	7500	7500	1500	1500	1500	1500	1500	1500	1500	1500	1500
Indirect	3000	3000	1500	833	1230	1428	1230	1379	1528	1677	1899	1973
Profit	-10500	-10500	-9000	1632	3220	4012	5201	6046	6892	7734	8996	9412
Loan	50,000	0	0	0	0	0	0	0	0	0	0	0
Repayment	4342	4342	4342	4342	4342	4342	4342	4342	4342	4342	4342	4342
Balance	35158	20316	6974	4264	3142	2812	3671	5375	7925	11316	15970	21040

Table 18: Financial projections for the first 12 months. All values given in USD ($).

11.3.5 Growth

We predict steady long term growth in the number of potential customers beyond 12 months. The duality of our product means it is able to capitalise on both the expanding DJ market and stage lighting market. The best available indicator of industry growth is compound annual growth rate (CAGR) [170], of which the DJ equipment market is expected to grow between 6.6% [163] and 9.5% [171] and the programmable stage lighting market between 7.0% [172] and 9.1% [166] from now until 2030. Additionally, with the main expense of developing the initial software not a factor, we would have the option of lowering the price of our visualiser after year 1 and still remaining profitable. This would increase sales.

As figure 11.3 clearly shows, once the software is developed our model predicts a steady increase in profit each month, resulting in an exponentially increasing cash balance 11.2. As advertising expenses

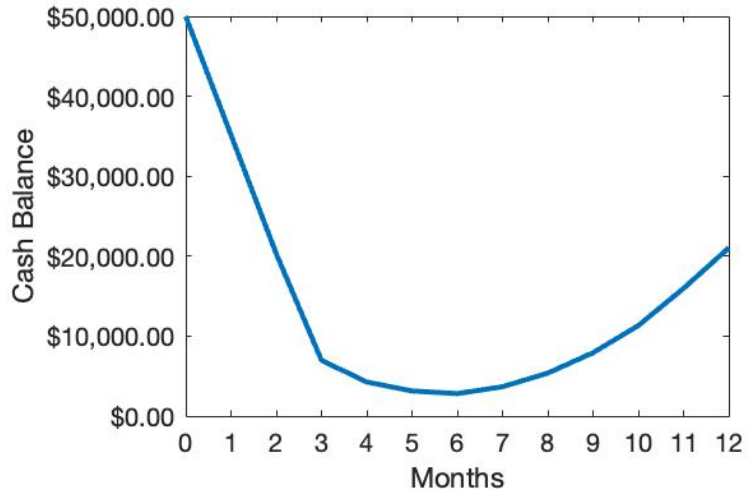


Figure 11.2: Graph of projected cash balance for months 0-12.

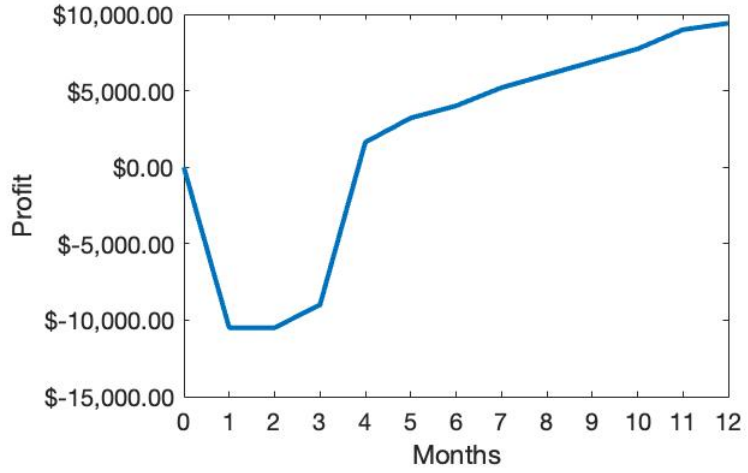


Figure 11.3: Graph of projected profit for months 0-12. Profitable from month 4 onwards with profit steadily increasing.

are proportional to revenue, increasing sales means an increase in marketing and exposure, creating a positive feedback loop as this in turn brings in more sales.

We also have the option of charging additional fees for major updates (a similar model to many music software products such as Ableton). This allows us to generate revenue from existing customers, who are likely to pay for an update that provides new features if they purchased our product once already.

Despite the uncertainty of these predictions due to the many assumptions made, we believe the combination of the projected cash balance and future growth factors demonstrate the potential of our visualiser and that our business is financially viable in the long term.

12 Summary - Mark Jegorovas

L.AI.GHTs aimed to create a fully automated real-time visualiser and lighting scheme by utilising MIR with modern ML techniques. We aimed to create an affordable product that provides amateur DJs and small venue owners with professional quality visuals. These visuals were intended to react to the music and reflect its emotional qualities.

We achieved this first by evaluating the macroscopic feature of the songs - genre. This allowed us to create a colour mapping unique to each genre, such that the lighting configuration reflected the nature of the song. We created variation of the lighting scheme throughout the song by establishing segments to enable different lighting schemes for different parts of the song. Each section was then individually analysed by a Music Emotion Retrieval algorithm, allowing distinction between sections of different energy and valence. Lastly, we increased the reactivity of our lighting scheme by locating the downbeats, allowing us to synchronise aspects of our lighting scheme to the songs being played.

The visualisation of the music was achieved in two ways. First, a visualiser was produced using Hydra. The modular nature of Hydra allowed for complex and dynamic visualisations determined by the parameters obtained from MIR. From segmenting the visualiser, a DMX lighting control sequence was derived. Both the visualiser and the lighting were fully reactive, unique and representative of the music.

We considered the ethical implications of seizures, collecting user data, and replacing workers. We identified common risks of running a software company and selected suitable project management strategies to mitigate them. Using data about the software industry, and the DJ equipment and stage lighting markets, we produced financial projections, demonstrating the financial viability of our product.

We can extend the final design by adding to the product's Music Information Retrieval capabilities, a segment classification or lyric analysis feature could lead to a more customisable and personalisable lighting scheme. A more ambitious endeavour would be to invest in the creation of a generative diffusion model for video that accepts extracted MIR parameters as an input. This will allow us to extend into the market of music video creation.

References

[1] Martinette Kruger and Melville Saayman. “Listen to your heart: Motives for attending Roxette live”. In: *Journal of Convention & Event Tourism*. Vol. 13. 3. Taylor & Francis. 2012, pp. 181–202.

[2] Richard E Dunham. *Stage Lighting Second Edition: The Fundamentals*. Routledge, 2018.

[3] Brad Schiller. *The automated lighting programmer’s handbook*. Routledge, 2021.

[4] Carl Nave. *Subtractive Color Mixing: Filters*. 2017. URL: http://hyperphysics.phy-astr.gsu.edu/hbase/vision/filter.html (visited on 03/21/2023).

[5] Mohammad Kazem Moghimi and Hossein Pourghassem. “Shadow detection based on combinations of HSV color space and orthogonal transformation in surveillance videos”. In: *2014 Iranian Conference on Intelligent Systems (ICIS)*. IEEE. 2014, pp. 1–6.

[6] Felippe Schmoeller da Roza et al. “Modular robot used as a beach cleaner.” In: *INGENIARE-Revista Chilena de Ingenieŕia* 24.4 (2016).

[7] Yvette Brown. *What Is Digital Addressable Lighting Interface (DALI)?* 2022. URL: https://www.comfortlights.com/digital-addressable-lighting-interface/ (visited on 03/22/2023).

[8] James Eade. *The DMX512-A Handbook: Design and Implementation of DMX Enabled Products and Networks*. Entertainment Technology Press, 2013.

[9] *DMX-512*. 2013. URL: https://www.thedmxwiki.com/dmx_definitions/dmx512#dmx-512 (visited on 03/21/2023).

[10] Richard Cadena. *Automated lighting: The art and science of moving light in theatre, live performance, and entertainment*. Taylor & Francis, 2010.

[11] *Art-Net*. URL: https://art-net.org.uk/ (visited on 04/17/2023).

[12] Dogan Ibrahim. *Using LEDs, LCDs and GLCDs in microcontroller projects*. John Wiley & Sons, 2012.

[13] Tommy Gmünder. *DMP using digital micromirror devices: a primer*. Society of Photo-optical Instrumentation Engineers, 2016.

[14] Vickie Claiborne. *Media Servers for Lighting Programmers: A Comprehensive Guide to Working with Digital Lighting*. Routledge, 2021.

[15] Ju-Chiang Wang, Yun-Ning Hung, and Jordan BL Smith. “To catch a chorus, verse, intro, or anything else: Analyzing a song with structural functions”. In: *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2022, pp. 416–420.

[16] Minz Won et al. “Data-driven harmonic filters for audio representation learning”. In: *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2020, pp. 536–540.

[17] Thomas Grill and Jan Schlüter. “Music Boundary Detection Using Neural Networks on Combined Features and Two-Level Annotations.” In: *ISMIR*. 2015, pp. 531–537.

[18] Brian McFee et al. “librosa: Audio and music signal analysis in python”. In: *Proceedings of the 14th python in science conference*. Vol. 8. 2015, pp. 18–25.

[19] Malcolm Slaney. “Auditory toolbox”. In: *Interval Research Corporation, Tech. Rep* 10.1998 (1998), p. 1194.

[20] Steve Young et al. “The HTK book”. In: *Cambridge university engineering department* 3.175 (2002), p. 12.

[21] Vibha Tiwari. “MFCC and its applications in speaker recognition”. In: *International journal on emerging technologies* 1.1 (2010), pp. 19–22.

[22] Todor Ganchev, Nikos Fakotakis, and George Kokkinakis. “Comparative evaluation of various MFCC implementations on the speaker verification task”. In: *Proceedings of the SPECOM*. Vol. 1. 2005. Citeseer. 2005, pp. 191–194.

[23] Jonathan Foote. “Automatic audio segmentation using a measure of audio novelty”. In: *2000 ieee international conference on multimedia and expo. icme2000. proceedings. latest advances in the fast changing world of multimedia (cat. no. 00th8532)*. Vol. 1. IEEE. 2000, pp. 452–455.

[24] Thomas Grill and Jan Schluter. “Music boundary detection using neural networks on spectrograms and self-similarity lag matrices”. In: *2015 23rd European Signal Processing Conference (EUSIPCO)*. IEEE. 2015, pp. 1296–1300.

[25] Joan Serra et al. “Unsupervised music structure annotation by time series structure features and segment similarity”. In: *IEEE Transactions on Multimedia* 16.5 (2014), pp. 1229–1240.

[26] David Bainbridge, Sally Cunningham, and J. Downie. “How People Describe Their Music Information Needs: A Grounded Theory Analysis Of Music Queries”. In: (Jan. 2003).

[27] Jin Ha Lee and J. S. Downie. “Survey Of Music Information Needs, Uses, And Seeking Behaviours: Preliminary Findings”. In: *International Society for Music Information Retrieval Conference*. 2004.

[28] *Musicmap: The Genealogy and History of Popular Music Genres from Origin till Present (1870-2016)*. 2016. URL: `https://musicmap.info/`.

[29] Tuomas Eerola. “Are the emotions expressed in music genre-specific? An audio-based evaluation of datasets spanning classical, film, pop and mixed genres”. In: *Journal of New Music Research* 40.4 (2011), pp. 349–366.

[30] Ho Hsiang Wu and Juan P Bello. “Audio-based music visualization for music structure analysis”. In: *7th Sound and Music Computing Conference, SMC 2010*. Sound and music Computing network. 2010, p. 11.

[31] Masahiro Hamasaki, Masataka Goto, and Tomoyasu Nakano. “Songrium: a music browsing assistance service with interactive visualization and exploration of protect a web of music”. In: *Proceedings of the 23rd International Conference on World Wide Web*. 2014, pp. 523–528.

[32] Ahmet Elbir et al. “Music genre classification and recommendation by using machine learning techniques”. In: *2018 Innovations in intelligent systems and applications conference (ASYU)*. IEEE. 2018, pp. 1–5.

[33] Shih-Wen Hsiao, Shih-Kai Chen, and Chu-Hsuan Lee. “Methodology for Stage Lighting Control Based on Music Emotions”. In: *Information Sciences* 412 (May 2017).

[34] George Tzanetakis and Perry Cook. “Musical genre classification of audio signals”. In: *IEEE Transactions on speech and audio processing* 10.5 (2002), pp. 293–302.

[35] Changsheng Xu et al. “Musical genre classification using support vector machines”. In: *2003 IEEE International Conference on Acoustics, Speech, and Signal Processing, 2003. Proceedings.(ICASSP’03).* Vol. 5. IEEE. 2003, pp. V–429.

[36] Lonce Wyse. “Audio spectrogram representations for processing with convolutional neural networks”. In: *arXiv preprint arXiv:1706.09559* (2017).

[37] TL Li, Antoni B Chan, and AH Chun. “Automatic musical pattern feature extraction using convolutional neural network”. In: *Genre* 10.2010 (2010), p. 1x1.

[38] Nicolas Scaringella and Giorgio Zoia. “On the Modeling of Time Information for Automatic Genre Recognition Systems in Audio Signals.” In: *ISMIR*. 2005, pp. 666–671.

[39] Hagen Soltau et al. “Recognition of music types”. In: *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP’98 (Cat. No. 98CH36181)*. Vol. 2. IEEE. 1998, pp. 1137–1140.

[40] Dipjyoti Bisharad and Rabul Hussain Laskar. “Music genre recognition using residual neural networks”. In: *TENCON 2019-2019 IEEE Region 10 Conference (TENCON)*. IEEE. 2019, pp. 2063–2068.

[41] Snigdha Chillara et al. “Music genre classification using machine learning algorithms: a comparison”. In: *Int Res J Eng Technol* 6.5 (2019), pp. 851–858.

[42] Thierry Bertin-Mahieux et al. “The Million Song Dataset.” In: Jan. 2011, pp. 591–596.

[43] Michaël Defferrard et al. “FMA: A Dataset for Music Analysis”. In: *18th International Society for Music Information Retrieval Conference (ISMIR)*. 2017.

[44] Pedro Cano et al. “ISMIR 2004 audio description contest”. In: *Music Technology Group of the Universitat Pompeu Fabra, Tech. Rep* (2006).

[45] Edith Law et al. “Evaluation of algorithms using games: The case of music tagging.” In: *ISMIR*. Citeseer. 2009, pp. 387–392.

[46] Antoine Liutkus et al. “The 2016 Signal Separation Evaluation Campaign”. In: *Latent Variable Analysis and Signal Separation - 12th International Conference, LVA/ICA 2015, Liberec, Czech Republic, August 25-28, 2015, Proceedings*. Ed. by Petr Tichavský et al. Cham: Springer International Publishing, 2017, pp. 323–332.

[47] Jort F. Gemmeke et al. “Audio Set: An ontology and human-labeled dataset for audio events”. In: *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2017, pp. 776–780.

[48] Bob L. Sturm. “An Analysis of the GTZAN Music Genre Dataset”. In: *Proceedings of the Second International ACM Workshop on Music Information Retrieval with User-Centered and Multimodal Strategies*. MIRUM ’12. Nara, Japan: Association for Computing Machinery, 2012, pp. 7–12.

[49] Bob L Sturm. “The GTZAN dataset: Its contents, its faults, their effects on evaluation, and its future use”. In: *arXiv preprint arXiv:1306.1461* (2013).

[50] Steven Boll. “Suppression of acoustic noise in speech using spectral subtraction”. In: *IEEE Transactions on acoustics, speech, and signal processing* 27.2 (1979), pp. 113–120.

[51] Trevor Hastie et al. *The elements of statistical learning: data mining, inference, and prediction*. Vol. 2. Springer, 2009.

[52] Marina Sokolova and Guy Lapalme. “A systematic analysis of performance measures for classification tasks”. In: *Information processing & management* 45.4 (2009), pp. 427–437.

[53] Bruce Benward. *Music in Theory and Practice vol. 2*. London, United States: McGraw Hill Higher Education, 2018.

[54] Grosvenor Cooper and Leonard B Meyer. “The Rhythmic Structure of Music. Chicago: University of Chicago”. In: *CooperThe Rhythmic Structure of Music1960* (1960).

[55] Sebastian Böck, Florian Krebs, and Gerhard Widmer. “Joint Beat and Downbeat Tracking with Recurrent Neural Networks.” In: *ISMIR*. New York City. 2016, pp. 255–261.

[56] SG Braun et al. “Encyclopedia of Vibration: Volumes 1, 2, and 3”. In: *Appl. Mech. Rev.* 55.3 (2002), B45–B45.

[57] Lav R Varshney and John Z Sun. “Why do we perceive logarithmically?” In: *Significance* 10.1 (2013), pp. 28–31.

[58] Sebastian Böck and Markus Schedl. “Enhanced beat tracking with context-aware neural networks”. In: *Proc. Int. Conf. Digital Audio Effects*. 2011, pp. 135–139.

[59] Kazuya Kawakami. “Supervised sequence labelling with recurrent neural networks”. PhD thesis. Technical University of Munich, 2008.

[60] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. *Learning internal representations by error propagation*. Tech. rep. California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[61] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. “A fast learning algorithm for deep belief nets”. In: *Neural computation* 18.7 (2006), pp. 1527–1554.

[62] *Hyperbolic Tangent*. URL: `https://commons.wikimedia.org/wiki/File:Hyperbolic_Tangent.svg` (visited on 04/09/2023).

[63] *Derivative of Sigmoid Function*. URL: `https://towardsdatascience.com/derivative-of-the-sigmoid-function-536880cf918e` (visited on 04/09/2023).

[64] Russell Reed and Robert J MarksII. *Neural smithing: supervised learning in feedforward artificial neural networks*. Mit Press, 1999.

[65] David C Plaut et al. “Experiments on Learning by Back Propagation.” In: (1986).

[66] *Neural Networks 101*. 2019. URL: `https://jameskle.com/writes/neural-networks-101` (visited on 04/09/2023).

[67] Ronald J Williams and David Zipser. “Gradient-based learning algorithms for recurrent”. In: *Backpropagation: Theory, architectures, and applications* 433 (1995), p. 17.

[68] Mike Schuster and Kuldip K Paliwal. “Bidirectional recurrent neural networks”. In: *IEEE transactions on Signal Processing* 45.11 (1997), pp. 2673–2681.

[69] John S Bridle. “Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition”. In: *Neurocomputing: Algorithms, architectures and applications*. Springer. 1990, pp. 227–236.

[70] V Mihajlovic and Milan Petkovic. “Dynamic bayesian networks: A state of the art”. In: *University of Twente Document Repository* (2001).

[71] Zoubin Ghahramani. “Learning dynamic Bayesian networks”. In: *Adaptive Processing of Sequences and Data Structures: International Summer School on Neural Networks “ER Caianiello” Vietri sul Mare, Salerno, Italy September 6–13, 1997 Tutorial Lectures* (2006), pp. 168–197.

[72] Florian Krebs, Sebastian Böck, and Gerhard Widmer. “An Efficient State-Space Model for Joint Tempo and Meter Tracking.” In: *ISMIR*. 2015, pp. 72–78.

[73] Andrew Viterbi. “Error bounds for convolutional codes and an asymptotically optimum decoding algorithm”. In: *IEEE transactions on Information Theory* 13.2 (1967), pp. 260–269.

[74] Len Vande Veire and Tijl De Bie. “From raw audio to a seamless mix: creating an automated DJ system for Drum and Bass”. In: *EURASIP Journal on Audio, Speech, and Music Processing* 2018.1 (2018), pp. 1–21.

[75] Daniel PW Ellis. “Beat tracking by dynamic programming”. In: *Journal of New Music Research* 36.1 (2007), pp. 51–60.

[76] Stanley S Stevens. “The measurement of loudness”. In: *The Journal of the Acoustical Society of America* 27.5 (1955), pp. 815–829.

[77] Sebastian Böck et al. “Madmom: A new python audio and music signal processing library”. In: *Proceedings of the 24th ACM international conference on Multimedia*. 2016, pp. 1174–1178.

[78] *madmom Downbeat Features*. URL: `https://madmom.readthedocs.io/en/v0.16.1/modules/features/downbeats.html` (visited on 04/09/2023).

[79] Oriol Nieto et al. “Audio-based music structure analysis: Current trends, open challenges, and applications”. In: *Transactions of the International Society for Music Information Retrieval* 3.1 (2020).

[80] Carlos Hernandez-Olivan, Jose R Beltran, and David Diaz-Guerra. “Music Boundary Detection using Convolutional Neural Networks: A comparative analysis of combined input features”. In: *arXiv preprint arXiv:2008.07527* (2020).

[81] Oriol Nieto et al. “Perceptual analysis of the f-measure for evaluating section boundaries in music”. In: *Proceedings of the 15th International Society for Music Information Retrieval Conference (ISMIR 2014)*. 2014, pp. 265–270.

[82] David W Hosmer Jr, Stanley Lemeshow, and Rodney X Sturdivant. *Applied logistic regression*. Vol. 398. John Wiley & Sons, 2013.

[83] Oliver Kuss. “Global goodness-of-fit tests in logistic regression with sparse data”. In: *Statistics in medicine* 21.24 (2002), pp. 3789–3801.

[84] Gabriel Sargent, Frédéric Bimbot, and Emmanuel Vincent. “Estimating the structural segmentation of popular music pieces under regularity constraints”. In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 25.2 (2016), pp. 344–358.

[85] Karen Ullrich, Jan Schlüter, and Thomas Grill. “Boundary Detection in Music Structure Analysis using Convolutional Neural Networks.” In: *ISMIR*. 2014, pp. 417–422.

[86] Elizabeth Bradley and Holger Kantz. “Nonlinear time-series analysis revisited”. In: *Chaos: An Interdisciplinary Journal of Nonlinear Science* 25.9 (2015), p. 097610.

[87] Joan Serra, Xavier Serra, and Ralph G Andrzejak. “Cross recurrence quantification for cover song identification”. In: *New Journal of Physics* 11.9 (2009), p. 093017.

[88] Jordan Bennett Louis Smith et al. “Design and creation of a large-scale database of structural annotations.” In: *ISMIR*. Vol. 11. Miami, FL. 2011, pp. 555–560.

[89] Alice Cohen-Hadria and Geoffroy Peeters. “Music structure boundaries estimation using multiple self-similarity matrices as input depth of convolutional neural networks”. In: *Audio Engineering Society Conference: 2017 AES International Conference on Semantic Audio*. Audio Engineering Society. 2017.

[90] Oriol Nieto and Juan Pablo Bello. “Systematic exploration of computational music structure research.” In: *ISMIR*. 2016, pp. 547–553.

[91] Ju-Chiang Wang, Jordan BL Smith, and Yun-Ning Hung. “MuSFA: Improving Music Structural Function Analysis with Partially Labeled Data”. In: *arXiv preprint arXiv:2211.15787* (2022).

[92] Wei-Tsung Lu et al. “SpecTNT: A time-frequency transformer for music audio”. In: *arXiv preprint arXiv:2110.09127* (2021).

[93] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017).

[94] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. “Layer normalization”. In: *arXiv preprint arXiv:1607.06450* (2016).

[95] Charles Inskip, Andy Macfarlane, and Pauline Rafferty. “Towards the disintermediation of creative music search: analysing queries to determine important facets”. In: *International Journal on Digital Libraries* 12 (2012), pp. 137–147.

[96] Patrik N Juslin. “From everyday emotions to aesthetic emotions: Towards a unified theory of musical emotions”. In: *Physics of life reviews* 10.3 (2013), pp. 235–266.

[97] Jacek Grekow. “Music emotion recognition using recurrent neural networks and pretrained models”. In: *Journal of Intelligent Information Systems* 57.3 (2021), pp. 531–546.

[98] Serhat Hizlisoy, Serdar Yildirim, and Zekeriya Tufekci. “Music emotion recognition using convolutional long short term memory deep neural networks”. In: *Engineering Science and Technology, an International Journal* 24.3 (2021), pp. 760–767.

[99] Justin Pierre Bachorik et al. “Emotion in motion: Investigating the time-course of emotional judgments of musical stimuli”. In: *Music Perception* 26.4 (2009), pp. 355–364.

[100] Anna Aljanaki, Yi-Hsuan Yang, and Mohammad Soleymani. “Developing a benchmark for emotional analysis of music”. In: *PloS one* 12.3 (2017), e0173392.

[101] Richard Orjesek, Roman Jarina, and Michal Chmulik. “End-to-end music emotion variation detection using iteratively reconstructed deep features”. In: *Multimedia Tools and Applications* 81.4 (2022), pp. 5017–5031.

[102] I-Sheng Huang et al. “A generative adversarial network model based on intelligent data analytics for music emotion recognition under IoT”. In: *Mobile Information Systems* 2021 (2021), pp. 1–8.

[103] *Essentia Documentation*. URL: `http://essentia.upf.edu/documentation/algorithms_reference.html` (visited on 05/14/2023).

[104] Sander Dieleman and Benjamin Schrauwen. “End-to-end learning for music audio”. In: *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2014, pp. 6964–6968.

[105] Roberto Cahuantzi, Xinye Chen, and Stefan Güttel. “A comparison of LSTM and GRU networks for learning symbolic sequences”. In: *arXiv preprint arXiv:2107.02248* (2021).

[106] Naz Kaya and Helen H Epps. “Relationship between color and emotion: A study of college students”. In: *College student journal* 38.3 (2004), pp. 396–405.

[107] Jamie Ward. “Synesthesia”. In: *Annual review of psychology* 64 (2013), pp. 49–75.

[108] I Newton. “Opticks, book II, part I, observation 17”. In: *Smith and Watford, London* 1704 (1704).

[109] *Tone-to-color mapping of Scriabin’s Clavier à lumières*. 2005. URL: `https://commons.wikimedia.org/wiki/File:Scriabin_keyboard.png`.

[110] Johann Wolfgang Von Goethe. *Theory of colours*. 3. Mit Press, 1970.

[111] Carl Gustav Jung. “Symbols of transformation”. In: (1956).

[112] Michael Kaykov. “Scriabin’s Synesthesia, Demystified”. In: *ICONI* (2021), pp. 22–28.

[113] Alexander Schindler and Andreas Rauber. “An audio-visual approach to music genre classification through affective color features”. In: *Advances in Information Retrieval: 37th European Conference on IR Research, ECIR 2015, Vienna, Austria, March 29-April 2, 2015. Proceedings 37*. Springer. 2015, pp. 61–67.

[114] Jana Machajdik and Allan Hanbury. “Affective image classification using features inspired by psychology and art theory”. In: *Proceedings of the 18th ACM international conference on Multimedia*. 2010, pp. 83–92.

[115] Timothy L Hubbard. “Synesthesia-like mappings of lightness, pitch, and melodic interval”. In: *The American journal of psychology* (1996), pp. 219–238.

[116] Michael Voong and Russell Beale. “Music organisation using colour synaesthesia”. In: *CHI’07 extended abstracts on Human Factors in Computing Systems*. 2007, pp. 1869–1874.

[117] Elias Pampalk, Masataka Goto, et al. “MusicRainbow: A New User Interface to Discover Artists Using Audio-based Similarity and Web-based Labeling.” In: *ISMIR*. 2006, pp. 367–370.

[118] Kenneth E Burchett. “Color harmony”. In: *Color Research & Application: Endorsed by Inter-Society Color Council, The Colour Group (Great Britain), Canadian Society for Color, Color Science Association of Japan, Dutch Society for the Study of Color, The Swedish Colour Centre Foundation, Colour Society of Australia, Centre Français de la Couleur* 27.1 (2002), pp. 28–31.

[119] Johannes Itten. *The elements of color*. Vol. 4. John Wiley & Sons, 1970.

[120] M Ronnier Luo, Guihua Cui, and Bryan Rigg. “The development of the CIE 2000 colour-difference formula: CIEDE2000”. In: *Color Research & Application: Endorsed by Inter-Society Color Council, The Colour Group (Great Britain), Canadian Society for Color, Color Science Association of Japan, Dutch Society for the Study of Color, The Swedish Colour Centre Foundation, Colour Society of Australia, Centre Français de la Couleur* 26.5 (2001), pp. 340–350.

[121] Lindsay W MacDonald. “Using color effectively in computer graphics”. In: *IEEE Computer Graphics and Applications* 19.4 (1999), pp. 20–35.

[122] Jukka Holm, Antti Aaltonen, and Harri Siirtola. “Associating Colours with Musical Genres”. In: *Journal of New Music Research* 38 (2009), pp. 100–87.

[123] Amitava Chattopadhyay, Peter R. Darke, and Gerald J. Gorn. “Roses are Red and Violets are Blue - Everywhere? Cultural Differences and Universals in Color Preference and Choice Among Consumers and Marketing Managers”. In: 2002.

[124] Chloe Taylor, Alexandra Clifford, and Anna Franklin. “Color preferences are not universal.” In: *Journal of Experimental Psychology: General* 142.4 (2013), p. 1015.

[125] Richard E Cytowic. “Synesthesia and mapping of subjective sensory dimensions”. In: *Neurology* 39.6 (1989), pp. 849–850.

[126] Eric J Isaacson. “What You See Is What You Get: on Visualizing Music.” In: *ISMIR*. 2005, pp. 389–395.

[127] Kar Yan Tam and Shuk Ying Ho. “Understanding the impact of web personalization on user information processing and decision outcomes”. In: *MIS quarterly* (2006), pp. 865–890.

[128] Alvy Ray Smith. “Color gamut transform pairs”. In: *ACM Siggraph Computer Graphics* 12.3 (1978), pp. 12–19.

[129] S Ben Driss et al. “A comparison study between MLP and convolutional neural network models for character recognition”. In: *Real-Time Image and Video Processing 2017*. Vol. 10223. SPIE. 2017, pp. 32–42.

[130] Yu-Chi Larry Ho and Xi-Ren Cao. *Perturbation analysis of discrete event dynamic systems*. Vol. 145. Springer Science & Business Media, 2012.

[131] Rumi Hiraga, Fumiko Watanabe, and Issei Fujishiro. “Music learning through visualization”. In: *Second International Conference on Web Delivering of Music, 2002. WEDELMUSIC 2002. Proceedings.* IEEE. 2002, pp. 101–108.

[132] Rohit Biswas and Nischay Ghattamaraju. “An effective analysis of deep learning based approaches for audio based feature extraction and its visualization”. In: *Multimedia Tools and Applications* 78 (2019), 23:949–972.

[133] Hugo B Lima, Carlos GR Dos Santos, and Bianchi S Meiguins. “A survey of music visualization techniques”. In: *ACM Computing Surveys (CSUR)* 54.7 (2021), pp. 1–29.

[134] *Hydra*. URL: https://hydra.ojack.xyz (visited on 03/23/2023).

[135] *VEDA*. URL: https://veda.gl/vedajs/ (visited on 03/23/2023).

[136] *Lumen*. URL: https://lumen-app.com/ (visited on 03/23/2023).

[137] *Cathodemer*. 2022. URL: https://hypertonal.net/cathodemer/index.html (visited on 03/23/2023).

[138] *Renderforest*. URL: https://www.renderforest.com/ (visited on 03/23/2023).

[139] Michael Palumbo, Alexander Zonta, and Graham Wakefield. “Modular reality: Analogues of patching in immersive space”. In: *Journal of New Music Research* 49.1 (2020), pp. 8–23.

[140] Naoto Hieda. *Hydra Book*. URL: https://hydra-book.glitch.me/#/ (visited on 03/30/2023).

[141] *Three.js*. URL: https://threejs.org/ (visited on 04/09/2023).

[142] *Meyda.js*. URL: https://meyda.js.org/ (visited on 04/09/2023).

[143] Esben Oxholm Bonde, Ellen Kathrine Hansen, and Georgios Triantafyllidis. “Auditory and Visual based Intelligent Lighting Design for Music Concerts”. In: *Eai Endrosed Trasactions on Creative Technologies* 5.15 (2018), e2.

[144] *Virtual Studio Technology*. URL: https://en.wikipedia.org/wiki/Virtual_Studio_Technology (visited on 04/09/2023).

[145] *JUCE*. URL: https://juce.com (visited on 04/09/2023).

[146] *Embed Python in C++*. URL: https://docs.python.org/3/extending/embedding.html (visited on 04/09/2023).

[147] *IEEE Computer Society: Code of ethics*. 1999. URL: https://www.computer.org/education/code-of-ethics (visited on 04/09/2023).

[148] *Epilepsy Society: Photosensitive Epilepsy*. 2019. URL: https://epilepsysociety.org.uk/about-epilepsy/epileptic-seizures/seizure-triggers/photosensitive-epilepsy (visited on 04/09/2023).

[149] GOV.UK. *Data Ethics Framework*. 2020. URL: https://www.gov.uk/government/publications/data-ethics-framework/data-ethics-framework-2020 (visited on 04/09/2023).

[150] *Lighting Designer Demographics and Statistics In The US*. 2022. URL: https://www.zippia.com/lighting-designer-jobs/demographics/ (visited on 04/09/2023).

[151] Tom Kendrick. “Overcoming Project Risk: Lessons from the PERIL Database”. In: *Program Manager, Hewlett-Packard Company* (2003).

[152] Dean Leffingwell. *Scaling software agility: best practices for large enterprises*. Pearson Education, 2007.

[153] Mario Krenn et al. “Predicting the Future of AI with AI: High-quality link prediction in an exponentially growing knowledge network”. In: *arXiv preprint arXiv:2210.00881* (2022).

[154] *Digital.ai 16th Annual State of Agile Report*. https://info.digital.ai/rs/981-LQX-968/images/AR-SA-2022-16th-Annual-State-Of-Agile-Report.pdf. Accessed: 2023-04-25. Dec. 2022.

[155] Hrafnhildur Sif Sverrisdottir, Helgi Thor Ingason, and Haukur Ingi Jonasson. “The role of the product owner in scrum-comparison between theory and practices”. In: *Procedia-Social and Behavioral Sciences* 119 (2014), pp. 257–267.

[156] Valpadasu Hema et al. “Scrum: An effective software development agile tool”. In: *IOP Conference Series: Materials Science and Engineering*. Vol. 981. 2. IOP Publishing. 2020, p. 022060.

[157] Watts S Humphrey. *Managing the software process*. Addison-Wesley Longman Publishing Co., Inc., 1989.

[158] Kent Beck. *Test-driven development: by example*. Addison-Wesley Professional, 2003.

[159] *LinkedIn reveals the latest talent turnover trends*. https://news.linkedin.com/2018/3/linkedin-data-analysis-reveals-the-latest-talent-turnover-trends. Accessed: 2023-04-08. Mar. 2018.

[160] Nachiappan Nagappan et al. “Realizing quality improvement through test driven development: results and experiences of four industrial teams”. In: *Empirical Software Engineering* 13 (2008), pp. 289–302.

[161] *Visual Synthesizer*. URL: https://www.imaginando.pt/products/vs-visual-synthesizer (visited on 04/09/2023).

[162] *DJ Report 2018*. URL: https://musicbiz.org/wp-content/uploads/2019/09/DJ-REPORT-Updated-1.pdf (visited on 04/09/2023).

[163] *DJ Equipment Industry Report*. URL: https://www.emergenresearch.com/industry-report/dj-equipment-market (visited on 04/09/2023).

[164] *End of Year Trends for SaaS 2022*. URL: https://fastspring.com/blog/2022-saas-and-software-holiday-spend-report/#:~:text=Global%5C%20End-Of-Year%5C%20Trends%5C%20For%5C%20SaaS%5C%20And%5C%20Software%5C%20Purchases&text=Once%5C%20again%5C%2C%5C%20when%5C%20we%5C%20looked,higher%5C%20on%5C%20a%5C%20global%5C%20scale. (visited on 04/09/2023).

[165] *enttec Software*. URL: https://www.enttec.co.uk/range/controls/dmx-lighting-control-software/ (visited on 04/09/2023).

[166] *Global Stage Lighting Market Report*. URL: https://straitsresearch.com/report/programmable-stage-lighting-market (visited on 04/09/2023).

[167] *Outsourcing Software Development*. URL: https://sloboda-studio.com/blog/best-countries-to-outsource-software-development/ (visited on 04/09/2023).

[168] *Startup Marketing Budget*. URL: https://nituno.com/sales-and-marketing-for-startups/ (visited on 04/09/2023).

[169] *Business Loan Terms*. URL: https://www.forbes.com/advisor/business-loans/small-business-loan-terms/ (visited on 04/09/2023).

[170] *Investopedia - CAGR*. URL: https://www.investopedia.com/terms/c/cagr.asp (visited on 04/09/2023).

[171] *Global DJ Equipment Market Report*. URL: https://www.businessresearchinsights.com/market-reports/dj-equipment-market-102781 (visited on 04/09/2023).

[172] *Programmable Stage Lighting Market Report*. URL: https://dataintelo.com/report/programmable-stage-lighting-market/ (visited on 04/09/2023).