WARWICK

THE UNIVERSITY OF WARWICK

CS310

Computer Science

Deep Content-Based Music Curation

Author:
Sylvester Cardorelle

Supervisor:
Dr Victor Sanchez

Student Number: 1510373

Abstract

As digital music platforms continue to grow exponentially, the automated curation of music has become a significant problem in the last decade. Current state-of-the-art recommender systems depend on the collaborative filtering model. Nevertheless, these systems experience the cold start problem; they break down when no historical data is available and as a result they cannot recommend new and unpopular songs. In this report, the author proposes a new recommendation model that uses music audio and deep learning to produce playlists based on a given query song. The system is evaluated qualitatively using a human evaluation study on the free music archive dataset. The results show that the model produces recommendations of a sensible nature. To conclude, the project shows that use of deep learning in a music recommendation setting has the potential to outperform traditional approaches.

Keywords and Abbreviations

MIR	Music information retrieval
CF	Collaborative Filtering
CNN	Convolutional Neural Network
FFT	Fast Fourier Transform
TF	Tensorflow
FMA	Free Music Archive
GTZAN	George Tzanetakis's genre classification dataset
MSD	Million Song Dataset

Acknowledgements

I would like to thank Dr Victor Sanchez for supervising my project and helping me throughout the development of the project. I would like to extend thanks to Sander Dieleman whose work at Spotify inspired me to undertake this project. Finally, I would like to extend my gratitude to all the participants who partook in my study and also to James Martin who widened my understanding of key musical theory concepts and for motivating me during the project.

Contents

1 Introduction **4**

1.1 Problem analysis 5

1.2 Objectives 6

2 Background Research **7**

2.1 Music Information Retrieval 7

2.2 Recommendation techniques 8

2.2.1 Collaborative Filtering 8

2.2.2 Natural Language Processing 8

2.2.3 Content-Based Recommendation 9

2.2.4 Genre Recognition 9

2.3 Feature Representation 10

2.3.1 The spectrogram 11

2.3.2 The chromagram 13

2.4 Machine Learning 14

2.4.1 Supervised Learning 15

2.4.2 Classification 15

2.4.3 Validation 16

2.5 Deep Learning 17

2.5.1 Neural networks 17

2.5.2 Training 18

2.5.3 Convolutional Neural Networks 19

3 Design **22**

3.1 Recommendation System 22

3.2 Dataset 23

3.3 Convolutional Neural Network 26

3.4 Playlist Generator 27

3.5 User Interface 28

4 Implementation **29**

4.1 Tools and Programs Used 29

4.2 Feature Preprocessing 30

4.3 Convolutional Neural Network 33

4.4 Playlist Generator 41

4.5 User Interface 43

5 Testing **45**

5.1 CNN training 45

5.2 Playlist generation 47

5.3 Human evaluation study 50

6 Author's Assessment of the Project **54**

6.1 Technical challenges 54

6.2 Project management 56

6.3 Project contributions 57

6.4 Project achievements 57

7 Conclusion **58**

Chapter 1

Introduction

The consumption of music has been transformed in the last decade, an increasing amount of songs are sold in a digital format instead of a physical one. Digital libraries containing millions of songs are now within our grasp through digital music platforms such as iTunes, Pandora and Spotify. This growth in digital distribution has presented many areas for automation.

In particular, music recommendation has become a significant feature for these platforms. The purpose of music recommendation is to recommend new artists and songs to the listener based on their preferences. Not only does it enrich a listener's experience by allowing them to discover new music but it simultaneously helps upcoming artists find potential listeners.

Recommender systems have become very popular in recent years and are applied to a variety of different settings. In addition to music, recommender systems also exist for movies, books and other products (e.g. Amazon Marketplace) [1].

Generally, music listeners do not usually listen to a single song in isolation. Rather, their listening sessions tend to persist over a sequence of songs known as a playlist [2]. Therefore, recommending playlists is a key way that streaming platforms can interact with their user's experience and is currently a competitive research area.

1.1 Problem analysis

A popular technique used in music recommendation is known as Collaborative Filtering ('CF') [3]. CF determines user preferences based on historical usage data. This data can be obtained by listeners rating songs or marking them as favourites (explicit feedback), or by analysing their listening patterns and other behaviours (implicit feedback).

The use of implicit feedback is more common in practice because it does not require any effort from the listener. As a result, CF is content-agnostic because it does not regard the information carried within a song. This feature is also its biggest weakness as it solely relies on usage patterns. Meaning that popular songs are easier to recommend, than new and unpopular ones which have less available historical data.

Due to this popularity bias, CF recommendations can often be perceived to be boring and predictable. If there is no usage data on a song, the collaborative filtering approach breaks down, this is known as the 'cold start' problem [4]. New and unpopular songs that have not been consumed before cannot be recommended. Items that are only of interest to a niche audience are more difficult to recommend because usage data is scarce.

In the last five years, streaming services such as Spotify and researchers in the field of Music Information Retrieval (MIR) have been exploring different content based approaches to mitigate this cold start problem. Content based techniques use the information associated with songs such as meta data (artist information, lyrics, genre labels, tags), or the audio content of the songs [5]. Thus, content-based approaches do not suffer from the cold start problem because they do not rely on historical data. This is why research into content-based recommendation has become increasingly worthwhile.

The audio signal is undoubtedly the most difficult feature of a song to use effectively. As there is a large semantic gap between the music audio and the various aspects of it that dictate listener preferences [6]. Extracting high-level properties such as genre, mood, instrumentation and lyrical themes from audio signals requires powerful models that are capable of capturing the complex hierarchical structure of music. Despite this, in recent years use of deep learning has been shown to produce better results than traditional recommendation approaches using music audio as data.

1.2 Objectives

This project aims to solve the cold start problem by developing a deep learning content based recommendation system. By using a content based approach, the system will be able to recommend new and unpopular songs which collaborative filtering models struggle to do.

It will use a Convolutional Neural Network (CNN) to classify songs into different genres based on the audio signal. Then by creating a high-dimensional genre space, similar songs can be found in this space to generate playlists for the listener. The system will be evaluated in a music recommendation setting using a human evaluation study. To achieve this aim, the following objectives were set:

1. Find a suitable dataset that provides music audio
   (a) Extract 30 second audio sample from the middle of each song
   (b) Choose an appropriate feature representation of the audio
   (c) Apply feature engineering to each audio signal
2. Build a convolutional neural network that will classify songs into a high-dimensional genre space.
   (a) Design a CNN architecture
   (b) Train the CNN using the dataset(s)
   (c) Evaluate the CNN performance in prediction phase
3. Create an algorithm which generates a playlist of similar songs found in the genre space
   (a) Develop a similarity metric to measure song similarity
   (b) Evaluate the playlists in a recommendation setting

Chapter 2

Background Research

2.1 Music Information Retrieval

Music information retrieval (MIR) is an interdisciplinary field incorporating aspects of music theory, Computer science, Psychology, Neuroscience and Machine learning. It researches ways to retrieve useful information from music using computational techniques that can be applied in real world applications [7]. An interesting area of research in MIR is the extracting of information from the audio waveforms of a song. The current challenge is the semantic gap between an audio waveform and the music it represents. Some features of a song are easier to extract compared to others. It is easy to determine the instruments present in an audio sample by analysing the low level features such as timbre. On the other hand, it is harder to determine the genre of a song based on the audio due to temporal dynamics and the fact that the idea of a genre is not as well-defined as the class of an instrument which leads to ambiguity in classification.

This semantic gap has been tackled in the past using human participation. For example, Pandora's Music Genome Project uses a team of trained musical experts to manually tag songs based on hundreds of attributes such as genre and mood [8]. The resulting data is fed into a recommendation system that uses a complex matching algorithm to produce lists of similar songs based on their attributes. However, this approach has many short comings, it is time consuming and human-labour intensive which substantially reduces the scalability of the system. The project commenced in 1999 and is still ongoing with a library that consists of almost 2 million songs. Competitors such as Spotify and Apple music boast libraries containing over 30 million songs. As a result, automated music recommendation has become crucial for streaming services such as Pandora and which is why content-based MIR approaches have become useful.

2.2 Recommendation techniques

2.2.1 Collaborative Filtering

Collaborative filtering uses a model-based approach [9]. It creates a large matrix R (fig.1) where each row represents a song and each column represents a user. The matrix is then populated with binary values, where a one represents a song that has been streamed by particular user or a zero if it hasn't been streamed.

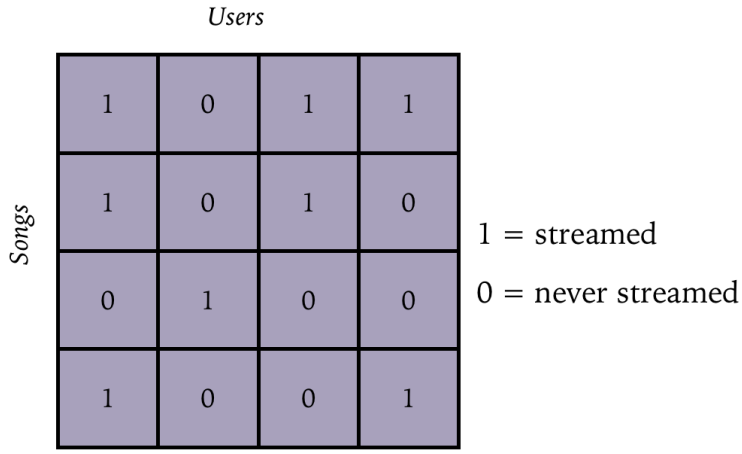

Figure 2.1: Matrix R

Implicit matrix factorisation is then applied to this matrix R using the below equation, producing two vectors X and Y.

$$\sum_{u=song} \sum_{i=user} C_{ui}(R_{ui} - X_u Y_i) + \lambda(||X||^2 + ||Y||^2) \tag{2.1}$$

Where C is the confidence matrix derived from R. Y is a user vector which represents each user's taste with k latent factors and X is a song vector expressing each song's profile with k latent factors also. The algorithm will then compare each row in the user vector to find similar users and will do the same with the song vectors. An example of CF is shown in the figure below.

2.2.2 Natural Language Processing

Natural language processing techniques can be used in a music recommendation setting. It is known that Spotify use an NLP technique known as Word2Vec. Word2Vec is a group of machine learning models that take large bodies of text and encode each word into a mathematical representation, a vector. These vectors can be embedded in a high dimensional space such that words of similar context will be located in close proximity in this space.

It is known that Spotify's NLP algorithm will take playlists and treat them as paragraphs, treating each song name as a single word [10]. This results in a better representation of a song such that song similarity can be determined. The algorithm also crawls the web constantly looking for blog posts and other written texts about music and tries to associate what adjectives and language is frequently used to describe these songs.

2.2.3 Content-Based Recommendation

A very influential paper in the content-based music recommendation area is *Deep content-based Music recommendation* [11]. Where two MIR researchers Sander Dieleman and Aaron Van Den Oord at Ghent University applied deep learning to music recommendation for the first time in 2013. They used a convolutional neural network to predict latent factors from a song's audio, to provide missing data in a collaborative filtering setting.

Their network consisted of two convolutional layers with 128 feature maps spanning the entire frequency range, yielding one-dimensional convolutions. A max-pooling layer followed each hidden layer, the first having a pooling window of size 4, and the second one with a pooling window of size 5. A fully connected hidden layer with 400 units was stacked on top of this, and finally the output layer had a number of units equaling the number of latent factors they were trying to predict.

This approach was able to provide sensible recommendations, however it still relied on the collaborative filtering model. Their results also showed that deep learning translates very well to a music recommendation setting and the use of neural networks significantly outperformed the traditional approach of using a bag of words representation of audio signals.

2.2.4 Genre Recognition

Another related task in the MIR field that attempts to bridge the semantic gap between audio and high-level features, is the genre recognition task using song audio. In 2014, Tao Feng a researcher at the University of Illinois, applied deep learning to predict genres on a dataset of 1000 songs [12]. Tao's paper published results for 2,3 and 4 genre classification.

He was able to reach a staggering 70% accuracy with 3-Genre classification task using a deep belief neural network which is another class of neural networks. After comparing Tao's approach to Dieleman et al's approach in the previous section, this led the author to think that CNNs could be applied to this genre classification task.

This idea led to the discovery of the DeepSound project developed by a group of students from the University of Warsaw (Piotr K, Jakub K, Łukasz M and Bartosz M) for the Braincode 2016 Hackathon competition [13]. They used Convolutional-Recurrent Neural Networks for Live Music Genre Recognition. Given a query song,

the system would give a real-time chart of probabilities describing the network's current belief of the song's genre. They were able to use the probability distribution output from the neural network in a creative and unusual way.

2.3 Feature Representation

Audio signals are a complex mixture of different sound components. A first step in a better understanding of a given signal is to transform it so that it's components are better accessible for subsequent processing steps in a music recommendation setting.

In the MIR domain, researchers are concerned with extracting high-level features from the audio using a variety of sound representations and how we can apply them with machine learning techniques to bridge the semantic gap. The most notable representations being the spectrogram and chromagram which are discussed in this section.

Naturally when a song is represented digitally, it is given a waveform representation that shows changes in amplitude over time. Amplitude describes the size of a wave which directly dictates the loudness of that wave. However, this visualisation is hardly informative about the musical content within a song. This section will discuss the spectrogram and chroma feature representations which are industry standards in MIR [14].

2.3.1 The spectrogram

The spectrogram is a time-frequency representation of sound, that shows the frequency content over time and uses a colour scale to indicate the intensities of frequencies. This frequency domain representation is more informative of a song's musical properties such as tones, chord progressions and melodies [15]. In particular, the spectrogram describes the timbre of a sound. Timbre is the characteristic of a sound that allows it to be distinct from its pitch and intensity.

In recent years, researchers in the MIR community have found significantly better results using a variant of the spectrogram known as the mel-spectrogram [16]. The difference between them is a mel-spectrogram uses a mel frequency scale is used instead of a linear one along the frequency axis. The magnitudes in a mel-spectrogram are scaled logarithmically which represents the sound more closely how humans perceive sound and consequently reduces the dimensionality of the input.

The spectrogram is generated using the discrete Fourier transform (DFT) which converts a signal from the time domain to the frequency domain. The DFT is the finite-resolution version of the Fourier transform that is extremely useful for sampled signals such as audio clips [17]. The DFT is computed using an algorithm known as the fast Fourier transform (FFT) [18]. The spectrogram algorithm can be seen below:

1. The waveform is divided into chunks and Fourier transformed to calculate the magnitude of the frequency spectrum for each chunk.
2. Each chunk then corresponds to a vertical line in the image; a measurement of magnitude against frequency for a given time.
3. This sequence of spectral vectors are appended to form the spectrogram.

This process is essentially equivalent to computing the squared magnitude of the short-time Fourier transform (STFT) of the audio clip.

$$spectrogram(m, w) = |STFT(m, w)|^2 \tag{2.2}$$

Where m is *time* and w is *frequency*.

An example mel-spectrogram can be seen in the figure below. It is extracted from a classical song (piece No.79 from GTZAN dataset's classical section).

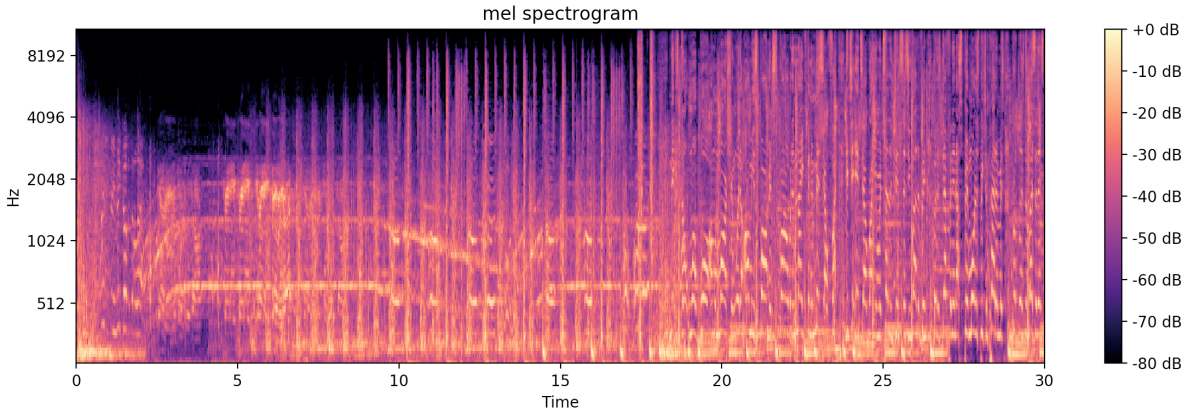

Figure 2.2: Mel-Spectrogram visualised

2.3.2 The chromagram

The chromagram is a time-chroma representation of sound. It projects the entire sound spectrum onto 12 pitch classes (known as chroma) representing the 12 distinct semitones of the musical octave [19]. Therefore, it can provide information on the pitch content of a song.

The generation of a chroma feature is similar to that of a spectrogram, as seen below:

1. Decide how to quantise the chroma range ; 12 pitch classes is commonly used in accordance with the equal temperament chromatic scale.
2. Generate a spectrogram using the squared magnitude of the STFT
3. For each time-step in the spectrogram
   (a) Find the chroma bin for each frequency
   (b) Sum the amplitude of all frequencies with the same chroma bin
   (c) Place that value in the corresponding chroma bin

An example chromagram can be seen in the figure below. It is extracted from a hip-hop song (piece No.20 from GTZAN dataset's hip-hop section).

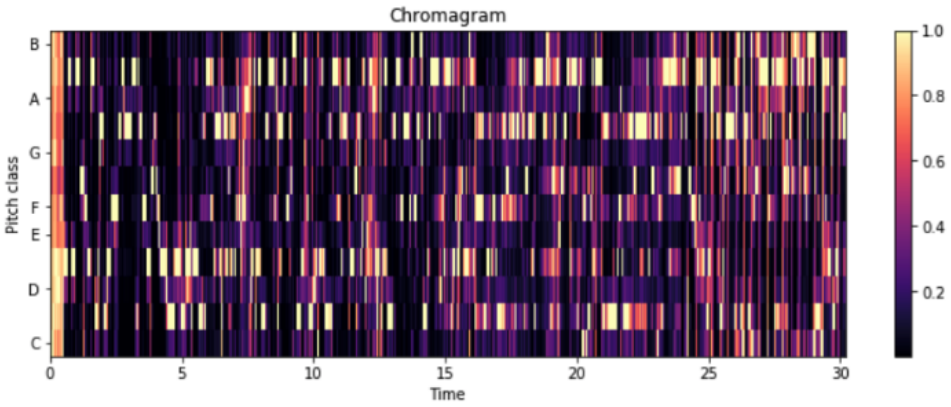

Figure 2.3: Chromagram visualised

2.4 Machine Learning

Machine learning is the study of computer systems that are able to learn from examples without being explicitly programmed [20]. This can be achieved by constructing a parameterised model and then calculating the optimal set of parameters by minimising an objective function that measures how well the model fits to the data (i.e. an error measure). The end goal is to tune the model to fit the example data well, so that it can may cast predictions about it, classify it or generate new examples.

To solve prediction and classification tasks without the need to explicitly write a program to do so is extremely valuable, because many of these tasks are difficult to capture in a single algorithm. This is especially true for certain tasks that are very easy to do for humans, such as identifying objects in an image, or speech recognition. Machine learning makes it possible for computers to perform these tasks without relying on a manually engineered solution. This significantly reduces development costs and ultimately leads to better performance and more robust solutions.

Moreover, Machine learning can be cast as an optimisation problem; given a dataset consisting of training examples (input), a machine learning model is trained by optimising an objective function. However, it differs from typical optimisation problems because the goal is to achieve high performance on new, previously unseen data, instead of maximising performance on the available training data.

This is an extremely important distinction because if the parameters of a model are simply optimised to minimise the training objective, this will likely result in overfitting. Overfitting is the scenario when a model will exhibit good performance on the training data but show poor generalisation performance. It is a consequence of the fact that training data for a machine learning model is usually noisy. Powerful models may be able to fit this noise exactly, allowing them to memorise each training example. To prevent this from happening, the complexity of the model needs to be controlled by regularisation.

A variety of problems can be solved by function approximation. In this case, the training examples for a supervised machine learning model would be a collection of pairs (x, t) , where x represents an example and t represents the desired output of the function. The output of the model $y = f(x)$ should then approximate t as close as possible. This setting is known as supervised learning. Examples of supervised learning include classification, regression (predicting one or more continuous values) and ranking.

In the case of unsupervised learning, the model should instead discover structure in the training data. Examples of unsupervised learning are clustering, dimensionality reduction (mapping the examples into a lower-dimensional space), feature learning (learning useful representations for use in supervised learning problems) and density modelling (approximating the probability distribution of the training data). This section is intended to provide an overview of the field of machine learning, with a focus on the concepts and models that has been used in the project.

2.4.1 Supervised Learning

Let the vector x_n be a data point from a dataset D consisting of N examples, and t_n its associated target vector.

$$D = \{(x_n, t_n), n = 0, ..., N-1\} \tag{2.3}$$

We will represent both the data point and the target as vectors for convenience, but they could have any dimensionality. Both the data points and targets can be discrete or continuous. Let $f(x|\theta)$ be a parameterised function representing the model, with parameter vector θ. Once the model is trained, we can obtain a prediction y_n for each corresponding data point x_n.

$$y_n = f(x_n|\theta) \tag{2.4}$$

The end goal of supervised learning is to produce the predictions y_n which approximate the targets t_n as accurately as possible. We will need to define an objective function $L(D, \theta)$ that measures the quality of the approximation across all $(x_n, t_n) \in D$. Lastly, an alogrithim is used to optimise function L.

2.4.2 Classification

In a classification setting, each data point x_n is belongs to a subset of K classes, and the goal for the model is to be able to identify which of these classes each data point belongs to [21]. Usually, the classes will be mutually exclusive, so that each data point belongs to one class only, but they do not necessarily have to be. We will assume that they are in this example. The K classes can be represented by indices ranging from 0 to $K-1$. We will use these indices for the targets t_n and the corresponding predictions y_n, which are both discrete in this case. A classification model will output a score $s_{nk}, k = 0, ..., K-1$ to each class, given a data example x_n. The predicted class is then the one with the highest score:

$$y_n = argmax(s_{nk}) \tag{2.5}$$

Classification is commonly approached with a probabilistic interpretation. The scores s_{nk} can be constrained to form a discrete probability distribution across the K classes:

$$0 \leq s_{nk} \leq 1 \ \forall n, k \tag{2.6}$$

$$\sum_{k=0}^{K-1} s_{nk} = 1 \ \forall n \tag{2.7}$$

and we then define $p(y_n = k) = s_{nk}$. Then by using the conditional log likelihood $p(y|x)$ of the model predictions given the data points, as the following objective function:

$$L(D, \theta) = \sum_{n=0}^{N-1} \log p(y_n = t_n|x_n) \tag{2.8}$$

This objective should be maximised, because we wish to maximise the likelihood of the data under the model. By convention, a minus sign is often added in front to get the negative log-likelihood (NLL), transforming it into a minimisation problem. The scores can then be constrained to form a probability distribution by using the softmax function in the model:

$$s_{nk} = \frac{exp(a_{nk})}{\sum_{k=0}^{K-1} exp(a_{nk})} \tag{2.9}$$

where a_{nk} are real numbers that come from the model.

2.4.3 Validation

When evaluating the performance of a model, we are not interested in the performance of the model on training data but rather how it generalises to new, Previously unseen data [22]. This is why it is important to validate models on a holdout dataset.

The training data and the holdout set are commonly referred to as the training set and validation set respectively. Apart from the learnable parameters, Machine learning models have several hyper parameters that influence the model and training procedure. Finding optimal values for these parameters can be done by determining how each parameter choice affects the performance on the validation set. This is crucial because hyper parameters affect the generalisation capabilities of the model, therefore it is not possible to measure the influence on the training set at all.

Also, another set of data is usually reserved to evaluate the model performance after the parameters have been optimised, which is known as the test set. The test set allows us to obtain an unbiased estimate of the real-world performance of the model. This three-way split of data is important because it allows us to compare models in an unbiased manner.

Machine learning models are known to be data hungry. In many cases data is often scarce, splitting it into three sets and only using one for the training may not always be feasible. It may actually have a detrimental affect on the performance because it reduces the amount of available data for the training phase. To combat this a common approach is to split off only the test data, the remainder set is then split iteratively into different training and validation sets several times over. Each split is known as a fold. When the are K such folds, this technique is termed K-Fold cross validation. The resulting performance metrics are then averaged across each fold combination, to compare to different models.

2.5 Deep Learning

Deep learning is an area of research that falls under the field of Machine learning, it is concerned with models known as deep neural networks which are related to information processing and communication patterns of the biological nervous system [23]. Deep learning includes architectures such as deep feed-forward neural networks, deep belief networks and convolutional neural networks.

These architectures consist of several layers of processing that form a hierarchy: each subsequent layer extracts a progressively more abstract representation of the input data and builds upon the representation from the previous layer, typically by computing a non linear transformation of its input. The parameters of these transformations are optimised by training the model on a dataset.

2.5.1 Neural networks

A feed-forward neural network is a traditional neural network, where each layer consists of a number of units (or neurons) that compute a weighted linear combination of the layer input, followed by an elementwise non-linearity [24]. These weights constitute the model parameters. Let the vector $xn - 1$ be the input to layer n, W_n be a matrix of weights, and b_n be a vector of biases. Then the output of layer n can be represented as the vector

$$x_n = f(W_n x_{n-1} + b_n), \tag{2.10}$$

where f is the activation function, an elementwise non-linear function. Popular choices for the activation function are linear rectification $f(x) = max(x, 0)$, which gives rise to rectified linear units (ReLUs), or a sigmoidal function $f(x) = (1+e^{-x})^{-1}$. If we consider a network with N layers. The network input is represented by x_0 , and its output by x_N.

A schematic representation of a feed-forward neural network is shown in the figure below. The network computes a function of the input x_0. The output x_N of this function is a prediction of one or more quantities of interest. Where t is the desired output (target) corresponding to the network input x_0. The final layer of the network is referred to as the output layer. All the layers below it are known as hidden layers.

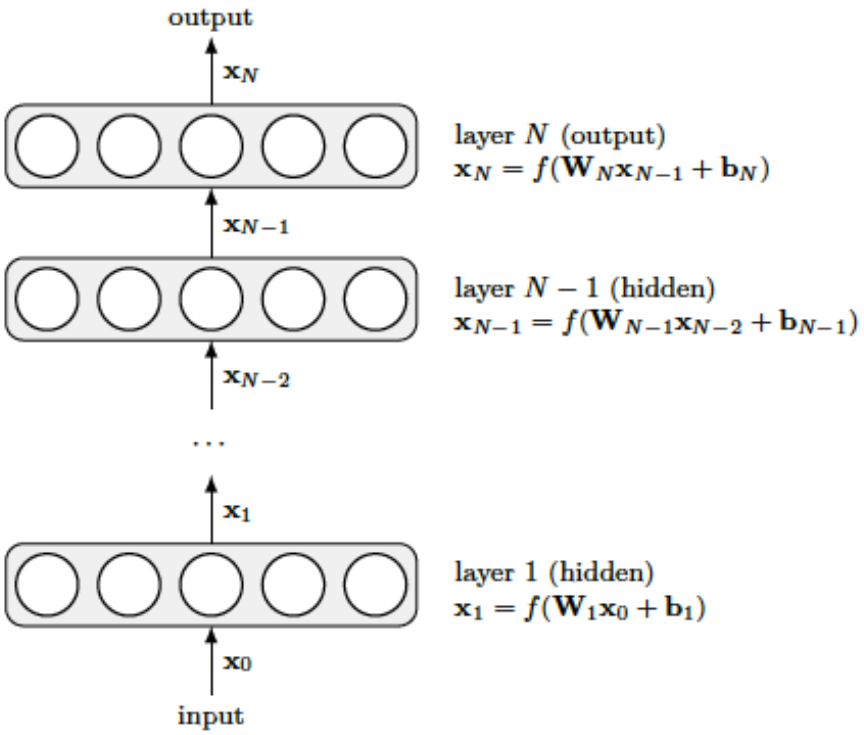

Figure 2.4: Schematic of typical Feed Forward Network

2.5.2 Training

When training a neural network, the parameters of each layer of the network are jointly optimised to produce the output x_N which approximates the target output t as close as possible. The prediction error is quantified using an error measure $e(x_N, t)$. Consequently, the hidden layers will learn to compute representations of the input data that are useful for the given task and the output layer will learn to predict the target output from these representations [25].

To determine how the parameters should be altered to reduce the prediction error, *gradient descent* is traditionally used. The gradient of $e(x_N, t)$ is computed with respect to the model parameters W_n , b_n for $n = 1...N$. The parameter values of each layer are then changed by iteratively taking small steps in the direction opposite to the gradient.

$$W_n \leftarrow W_n - \lambda \frac{\partial e(x_N, t)}{\partial W_n} \tag{2.11}$$

$$b_n \leftarrow b_n - \lambda \frac{\partial e(x_N, t)}{\partial b_n} \tag{2.12}$$

Where, λ is the *learning rate*, a hyper parameter controlling the step size.

2.5.3 Convolutional Neural Networks

Convolutional neural networks (CNNs) are a subset of neural networks with restricted connectivity patterns between layers [26]. Their use is recommended when the input data exhibits some kind of topological structure such as an image containing a grid of pixels or the temporal structure of an audio signal. As a result, CNNs are commonly used in image recognition tasks. They typically contain two types of layers with restricted connectivity: convolutional layers and pooling layers.

A convolutional layer (hidden layer) takes an array of feature maps (array size is dependent on number of image channels) as input and convolves over these maps with a set of learnable kernels to produce an output of feature maps. It is common practice to sub sample the feature maps after each convolutional layer, this is achieved by inserting pooling layers.

A pooling layer reduces the dimensionality of a feature map by calculating an aggregation function which is usually the maximum across a local region of the input, as seen in the image below [27]. This allows the higher layers of the network to model correlations across a larger part of the input. Pooling is useful because it reduces the computational load of a CNN by reducing the dimensionality of the features maps.

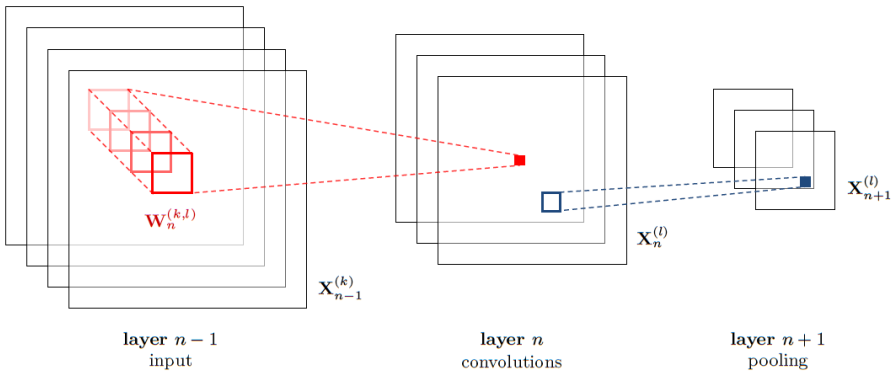

Figure 2.5: Visual overview of CNN structure

An important aspect of CNNs is the convolutions, which they are famously known for. Convolution is a mathematical concept that is commonly used in Digital Signal Processing, when dealing with signals that take the form of a time series. Essentially, convolution is a mechanism to combine two functions of time in a coherent manner. It can be mathematically described as seen below, where the signal is a 2D image $f(x, y)$, (x, y) is the position of a pixel and $g(x, y)$ is the kernel.

$$(f * g)(x, y) = \sum_{u=-\infty}^{+\infty} \sum_{v=-\infty}^{+\infty} f(u, v)g(x - u, y - v) \tag{2.13}$$

CNNs use this concept and apply a modified version to the input image, known as cross-correlation, in which one of the inputs is time-reversed.

$$y[n] = x[n] * h[n] = \sum_{k} x[k] * h[n + k] \quad where\ k \in [-\infty, +\infty] \tag{2.14}$$

The filters (or kernel) used in the convolutional layers are an integral component of the layered architecture. Generally, it refers to an operator applied to the entirety of an image such that it transforms the information encoded in the pixels e.g. high-pass filter [28]. In CNNs however, a kernel is a smaller-sized matrix e.g.4x4 in comparison to the input dimensions of the image, that consists of randomly initialised values. The filters are then convolved with the input (applying a dot product) to obtain the so-called 'activation maps' (or feature maps). Activation maps indicate 'activated' regions in the input, regions where features specific to the filter have been detected in the input. The real values of the filter matrix change with each learning iteration over the training set, indicating that the network is learning to identify which regions are of significance for extracting features from the data. This process is demonstrated below, using a 4x4 input image and 3x3 filter.

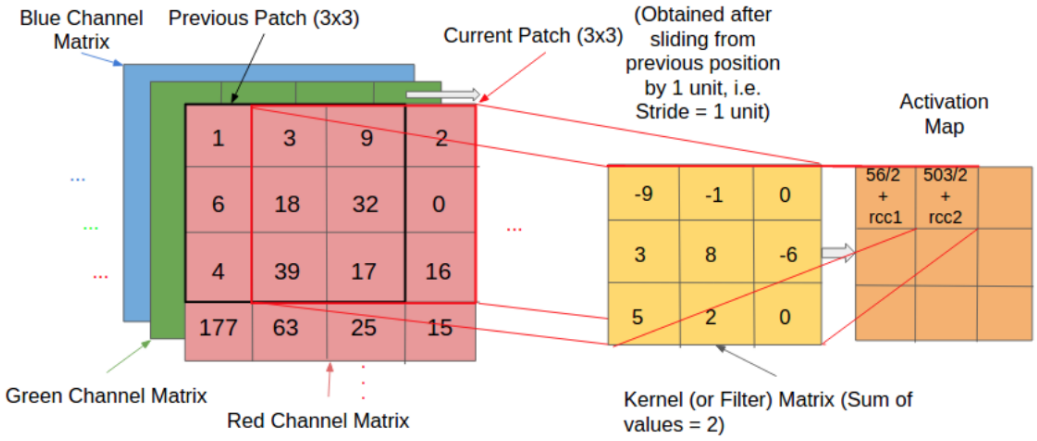

Figure 2.6: Example operation of convolution kernel on RGB image

Zero-padding is another common process of symmetrically adding zeroes to the input matrix. It allows the size of the input to be adjusted to our requirement. It is mostly used in convolutional layers when the dimensions of the input volume need to be preserved in the output volume.

Furthermore, CNNs require a large amount of hyper parameters to be configured. Firstly, the learning rate which controls how large the changes are to the neuron weights and biases in the optimisation algorithm. A fixed learning rate can be used or a gradually decreasing learning rate depending on the choice of optimiser such as SGD, Adam or RMSProp [29]. Other important hyper parameters include *the number of epochs* (the number of times the CNN is trained on the entire dataset), the *stride length* for the kernels and the number of layers that are used.

Chapter 3

Design

This section outlines the design process undertaken by the author to create a content based recommendation system and ultimately provide a solution to the cold start problem. Firstly, the overall architecture of the recommendation system was designed to ensure feasibility and provide a system overview to the author. Secondly, planning the dataset structure was the next step, as it was crucial for the training of the CNN. This involved deciding on a feature representation of the audio signal which was the mel-spectrogram and it's relevant parameters. Thirdly, the structure of the the CNN was designed whilst taking careful consideration of it's hyper parameters. Lastly, the playlist generation algorithm and user interface was designed.

3.1 Recommendation System

As stated in the project objectives, the main aim of the system is to be given a query song and then produce a recommendation playlist consisting of 10 similar songs.

The system begins with receiving a query song from the user in MP3 or WAV format. A 30 second sample is taken from the middle of the song. This is to ensure that the sample contains musical information that is representative of the song, compared to sampling the intro or ending. The audio sample is then converted into a mel-spectrogram before it is input into the trained CNN engine (trained using the dataset). The CNN will then output an n-dimensional probability vector from the softmax layer, where n is the number of genres. This will be used to create a high dimensional genre space, in which the query song will be plotted along with a library of songs from the dataset. The playlist generation algorithm will then use a similarity metric to find and rank the top 10 most similar songs, which will be output to the user. The system overview can be seen in the diagram below.

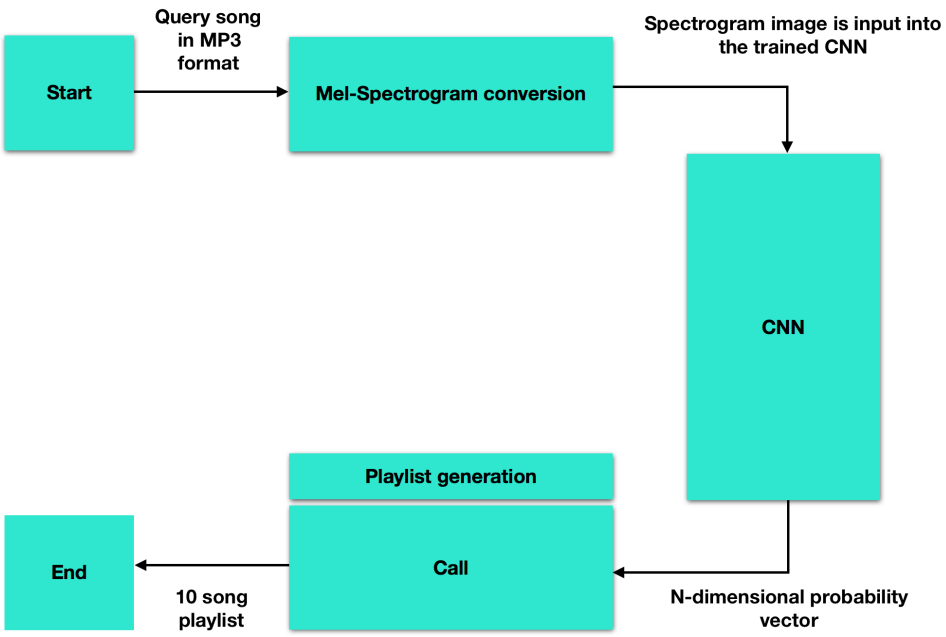

Figure 3.1: A dataflow diagram of the recommendation system

3.2 Dataset

As discussed in the introduction, the type of data needed for the content-based recommendation system is the audio signal. The dataset is needed to train the convolutional neural network to classify songs into their genres, so that the genre probability scores output from the network can be used to create a high dimensional genre space. The dataset is crucial to the performance of the system because it is needed to train the CNN.

The task of finding a dataset of songs containing their audio was ultimately a very time consuming task. Working with audio can be problematic; each file being approximately 7-15 MB, file sizes can quickly become a problem when constrained by storage space. There are various qualities and parameters of song recordings e.g. Sampling rate, Bits per second, etc. which varies between different datasets. However, the largest issue with obtaining audio files is copyright. Traditionally, research in music information retrieval (MIR) on large scale datasets was limited to industry level, because large collections of music audio cannot be published easily due to licensing issues.

The first dataset the author discovered was the Million Song Dataset (MSD), a freely available dataset of a million contemporary song's metadata [30]. The metadata consists of precomputed features for each song e.g. BPM, MFCC values and tags. Unfortunately, the dataset does not contain the actual audio samples needed to train the neural network. A solution to this problem was documented by researcher Sander Dieleman who circumvented the problem by obtaining audio samples for each song in the dataset from the 7Digital API [11]. However, 7Digital no longer

provides this service, so the author had to look for other suitable datasets.

An alternative option was the GTZAN dataset, a popular dataset in the MIR field [31]. The dataset is quite small, as it consists of only 1000 audio tracks of 10 genres (100 songs for each genre), each audio track is 30 seconds long. This was the dataset used in Tao Feng's genre classification task [12]. This amount of data is insufficient for a convolutional neural network, a known data hungry structure because it needs to learn a large number of parameters.

Eventually, the Free Music Archive (FMA) dataset was found. It contains 106,574 songs of 161 genres [32]. The full dataset boasts a size of 917GB however due to storage constraints, a subset consisting of 8000 songs (30 second samples) was used instead (approximately 7.2GB). The author then decided to use the FMA dataset to train the CNN and use the GTZAN dataset to evaluate the CNN's accuracy in the genre classification task.

For the supervised training and evaluation of the CNN, the target genre tags (e.g. Blues, Hip-Hop) provided for each song would be used to fit the model and indicate the accuracy of the network. To allow it to learn which features dictate a genre of a song, these labels need to be encoded in a numerical format that is accepted by the neural network. This format is known as one hot encoding[33]. It is the process that converts categorical data into a numerical form which is easier for the neural network to process. It allows the use of binary values to represent the known genre category of a song. An example of one hot encoding is shown below, using three music genres.

Genre
Rock
Hip-Hop
Classical

Rock	Hip-Hop	Classical
1	0	0
0	1	0
0	0	1

Figure 3.2: Each genre is encoded into a 1-D binary one-hot vector

The next design problem faced by the author was how to apply feature engineering to the music audio in the dataset(s). The raw data within .MP3 or .WAV files, is the pure audio signal. As discussed in the feature representation section, the time-amplitude representation is not very informative of the tonal characteristics and timbre that dictates the user preference. By applying mathematical techniques such as the Fourier transform, we can represent the audio in frequency domain which provides much more information about the harmonic content of the audio.

The industry standard of feature representation in MIR is the spectrogram, a time-frequency representation. The spectrogram is discussed in detail in the feature representation section. A spectrogram is a mid-level representation of the digital audio signal. It describes the energies of the signal in various frequency bands, as they vary over time [34]. The spectrogram can be post-processed further to match the properties of human perception. This is achieved by converting the linear frequency axis to a logarithmic scale because humans perceive sound in a logarithmic manner. This step results in the mel-spectrogram being formed. Therefore, each song in the dataset needs to be converted into a mel-spectrogram. This also allows a fixed-size representation to be used as input to the neural network.

The following parameters for the mel-spectrogram conversion were decided based on the preprocessing steps carried out by Dieleman et al [11].

- Number of frequency bins - 128
- FFT Window size - 1024
- Hop length - 512

The number of frequency bins specifies the number of intervals along the frequency axis, therefore the more intervals the more detailed the harmonic content will be. The window size refers to how many samples are taken during each fast Fourier transform analysis window and the hop length denotes the number of samples between each successive frame.

3.3 Convolutional Neural Network

The architecture of the CNN consists of three hidden layers, as seen in the figure below. Each hidden layer is followed by a subsequent max-pooling layer with a pooling window of size 4, 2 and 2 respectively. The output feature maps from the third hidden layer are then fed into a fully connected layer with 1024 rectified linear units (neurons). The final layer is the softmax classifier that will then predict the genre classification of the given song. It will transform the fully connected layer's output into an n-dimensional vector of probabilities in the range [0,1] , where n is the number of genres.

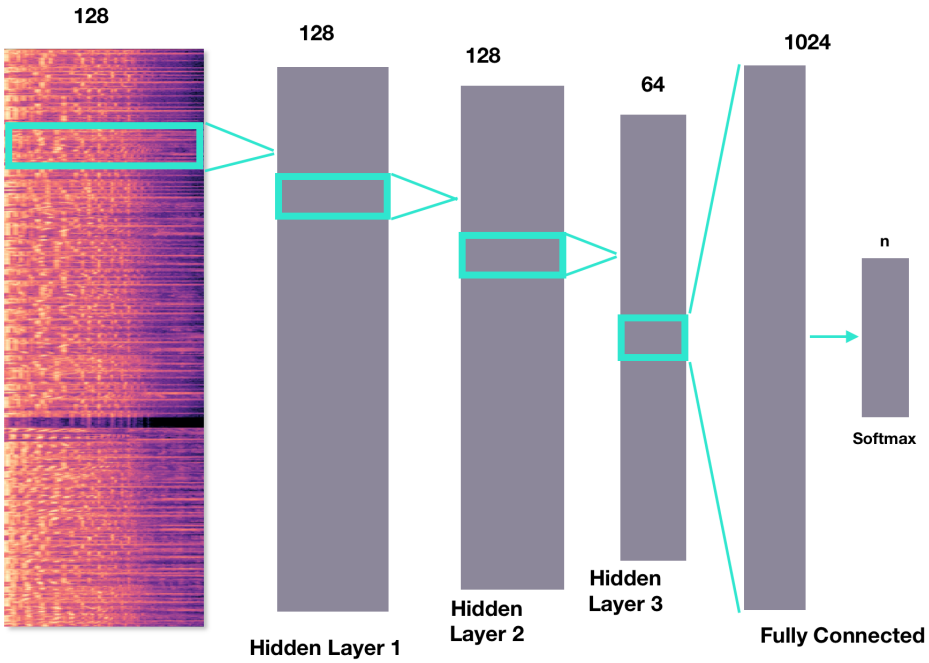

Figure 3.3: the CNN architecture used

As neural network architectures are problem dependent, there is no set architecture to apply to this problem.The structure was inspired by the CNN (discussed in the research section) used by Dieleman et al [11]. Their CNN model was slightly smaller as it only used two hidden layers. This was influenced by their decision to train the network on 3 second audio clips rather than 30 seconds, which would speed the up the training process. The author decided to increase the number of hidden layer's to allow more features to be extracted from the data, allowing more complex features to be learnt thus creating a 'deeper' network.

3.4 Playlist Generator

The author's idea to create a genre space to represent a song's musicality was inspired by the latent factor space used in the collaborative filtering model, used to plot songs using their latent factors. By using genres as a meaningful measure of music rather than simple categories, a new way to find similar songs is possible with this genre space. The playlist generator will use the following algorithm devised by the Author.

```
1. For each song in the library
    1.1 Read the softmax vector output
    1.2 Create a high dimensional genre space
    1.3 Plot each song in this space using their vectors
2. Plot the user's query song in the space
    2.1 Using a distance based similarity metric output
    the top 10 closest songs to the query song

```

Figure 3.4: Pseudocode for the playlist generation logic

There are many different types of distance based similarity metrics. The most common and instinctive one being the Euclidean distance metric, it is the most intuitive and when data is dense or continuous it is usually the best choice [35]. Another popular similarity measure, is cosine similarity (eq. 3.1). It uses the normalised product of two vectors to determine similarity. This metric is particularly known to be efficient when handling sparse vectors. It is normally used in positive space where the data is neatly bounded between [0,1], which the softmax vectors conveniently satisfy [36].

$$Sim(A, B) = \cos\theta = \frac{A \cdot B}{||A||||B||} \tag{3.1}$$

Other similarity measures were also considered, such as the Manhattan distance (L1 norm) which uses the absolute difference of the Cartesian coordinates of two points. However, after careful consideration it's use seemed less appropriate than the aforementioned measures.

3.5 User Interface

The aim of the user interface was to provide a simple way to demo the system and for proof of concept. As recommendation systems are usually incorporated into the back end of a streaming platform, an interface of this nature was not feasible given the project timeline. The planned interface would consist of the user choosing a song from the available library and upon selection, they would receive the recommended playlist. This interface would also include a simple audio player, allowing the user to easily experience the music curated. To adhere to the overall aim of the interface, the following usability principles were included in the design [37].

- Simplicity - The interface should be easy to use and require minimal user guidance
- Consistency - The interface should use consistent notation and standard web conventions and present information in a logical manner
- Memorability - It should be easy for the user to proficiently use the interface even if use is infrequent
- Satisfaction - The user should enjoy the look and feel of the interface design

Chapter 4

Implementation

This section will give an in-depth description of how the different components were implemented in the programming language Python. The development of the project was done initially using a Mac OSX machine and in the later stages an Ubuntu server machine with 16GB RAM and 160GB disk space [38]. The author decided to use a server machine because it provided more processing power needed for the CNN training and more storage space for the dataset. Code was written using the using Atom text editor, which is an open source text editor developed by Github. It was chosen because it provides various package that enhance the programming experience. To keep track of the various projects iterations, the version control software Github was used.

4.1 Tools and Programs Used

To implement the designs discussed in the previous section, various tools were incorporated into each of the Python script components. This section will detail the tools and libraries used throughout development.

The external tools used:

Librosa - An open-source Python library used for music and audio analysis [39]. It is commonly used in music information retrieval systems. It provides the necessary functions needed to process the raw audio signals into a mel-spectrogram.

Tensorflow - A computational framework used to build Machine learning models made by Google. It provides many tools that allows model construction at different levels of abstraction [40]. These tools contain reusable functions for common model components, such as the hidden layer. It is uses data flow graphs to build neural networks and it's flexible architecture allows it to be used across a variety platforms (CPUs,GPUs,TPUs). It provides the necessary tools needed to build and train a CNN in Python.

Pickle - A Python module that contains useful functions needed for python object serialisation and deserialisation [41]. Pickle is used to save processed spectrogram arrays to disk space and then accessed by other Python scripts through a read/write process known as 'pickling'. This allows the system to be modularised with key components separated into different Python scripts.

Numpy - A Python library that provides useful data structures such as the n-dimensional arrays needed to hold input and output data used in the convolutional neural network.

Plotly - A python library that provides interactive and publication quality graphs e.g. pie charts and scatter graphs [42]. It was used for data visualisation in evaluation stages of the project. It is useful because after compiling the graphs in Python, it can upload them to Plotly servers allowing them to be viewed online. It also supports conversion to JSON objects which allows graphs to be viewed in the user interface using JavaScript.

The internal tools used:

Os, Glob - These Python modules provided the use of operating system dependent functionality and allowed filesystem traversal and pattern matching. These modules were used in the dataset traversal task, in which each audio file had to read by the spectrogram conversion script, to produce the corresponding mel-spectrograms.

4.2 Feature Preprocessing

The first system component that was implemented was the feature engineering script which performs the essential mel-spectrogram conversion step. It began with creating a for loop to iterate through the dataset saved in the file system space, this was achieved using the mentioned Os and glob tools. When given the file path of the dataset and the format of the desired files e.g.MP3. The for loop will apply a regular expression to collect all the files which are matched. Then another inner loop is placed to iterate through the collected files to apply the spectrogram conversion. The snippet below shows this nested loop being initialised.

```
1        # this function will iterate through each file in the dataset
2        def extract_features(basedir,extension='.au'):
3            features=[]
4            labels=[]
5        # iterate over all files in all subdirectories of the base
     directory
6        for root, dirs, files in os.walk(basedir):
7            files = glob.glob(os.path.join(root,'*'+extension))
8            # apply function to all files
9            for f in files :
10
```

Figure 4.1: Loop initialisation of mel-spectrogram converter

Within the inner loop, the necessary Librosa functions were called to convert each audio file to a mel-spectrogram. The function librosa.feature.melspectrogram() is called to perform the spectrogram algorithm on the audio file. Along with the audio signal, the sampling rate, number of frequency bins, hop length and FFT window size are passed as arguments to the function. The returned spectrogram is then converted to a log scale, using the librosa.logamplitude() function. The mel-spectrogram is then trimmed slightly to make the array dimensions even, which is desirable for the CNN input because it allows faster computation. After the preprocessing, the mel-spectrogram is then appended to a list, to keep them in an ordered format. The snippet of code below shows these steps.

```
    for f in files :
            genre = f.split('/')[4].split('.')[0]

            if (genre == 'hiphop' or genre == 'rock' or genre == 'pop'
or genre == 'country'):
                print genre
                # Extract the mel-spectrogram
                y, sr = librosa.load(f)
                # Let's make and display a mel-scaled power (energy-
squared) spectrogram
                mel_spec = librosa.feature.melspectrogram(y, sr=sr,
n_mels=128,hop_length=1024,n_fft=2048)
                # Convert to log scale (dB). We'll use the peak power
as reference.
                log_mel_spec = librosa.logamplitude(mel_spec, ref_power
=np.max)
                #make dimensions of the array even 128x1292
                log_mel_spec = np.resize(log_mel_spec,(128,644))
                print log_mel_spec.shape
                #store into feature array
                features.append(log_mel_spec.flatten())
                # print len(np.array(log_mel_spec.T.flatten()))
                # Extract label
                label = genreDict.get(genre)
                labels.append(label)
            else:
                pass
    features = np.asarray(features).reshape(len(features),82432)
    print features.shape
    print len(labels)

    return (features, one_hot_encode(labels))

```

Figure 4.2: Mel-spectrogram converter

After each mel-spectrogram is created, each corresponding genre label is extracted from the filename of the audio track and is then encoded into a one-hot format. A one-hot encoding function was implemented to complete this step. The function can be seen in the code snippet below.

```
def one_hot_encode(labels,num_classes=4):

    assert len(labels) > 0

    if num_classes is None:
        num_classes = np.max(labels)+1
    else:
        assert num_classes > 0
        assert num_classes >= np.max(labels)

    result = np.zeros(shape=(len(labels), num_classes))
    result[np.arange(len(labels)), labels] = 1
    return np.array(result)

```

Figure 4.3: One-hot encoder function

The final step in this script is to serialise the data and save it to disk space using the pickle library. Two Python objects are saved: the list of the processed mel-spectrograms and a corresponding array of their respective genre labels in one-hot form. After specifying the filenames of the serialised objects, the pickle.dump() function is used to write the data. These steps can be seen in the code snippet below.

```
    # store preprocessed data in serialised format so we can save
computation time and power
    with open('../../4GenreTest.data', 'w') as f:
        pickle.dump(train_data, f)

    with open('../../4GenreTest.labels', 'w') as f:
        pickle.dump(train_labels, f)

```

Figure 4.4: Object Serialistion of mel-spectrograms

4.3 Convolutional Neural Network

The preprocessed mel-spectrograms from the file space had to be loaded into variables using the Sklearn function joblib.load(), this function is optimised for loading serialised objects from the disk space. This is especially useful because the arrays produced from the spectrogram conversion were approximately between 2-3GB. Once the data is loaded, it is common practice to apply a random permutation to the training data. This is to prevent the model from receiving large quantities of similar data e.g. a batch consisting solely of hip-hop songs which increases the likelihood of the neural network overfitting to the noise of a particular genre. Therefore, the data must be fed in a random order so that the neural network is able to analyse a variety of training data examples within a given cycle.

The permutation is achieved by creating a new order of array indexes using the np.random.permutation() function. The produced permutation is also applied to the genre labels to maintain their correspondence with the input data. The code snippet below demonstrates this.

```
trainData = []
batch1 = []
batch2 = []
batch1labels = []
batch2labels = []
trainLabels = []
with open('../../4genreBatch1.data', 'r') as f:
  batch1 = pickle.load(f)
with open('../../4genreBatch2.data', 'r') as f:
  batch2 = pickle.load(f)
with open('../../4genreBatch1.labels', 'r') as f:
  batch1labels = pickle.load(f)
with open('../../4genreBatch2.labels', 'r') as f:
  batch2labels = pickle.load(f)


trainData = np.concatenate((batch1, batch2), axis=0)
trainLabels = np.concatenate((batch1labels, batch2labels), axis=0)

# Shuffle data
permutation = np.random.permutation(trainData.shape[0])
trainData = trainData[permutation]
trainLabels = trainLabels[permutation]


```

Figure 4.5: The loading of training data and permutation operation

The next step was to initialise certain hyper parameters. The learning rate which dictates how much we update the unit weights was set to 0.001 which is the recommended value for a CNN task in tensorflow. The number of epochs is the number of times the whole training dataset is iterated through. Therefore, its value is subject to change depending on the data set size. This is also the case for the batch size parameter, which controls how many training examples are fed to the CNN per learning step.

An important hyper parameter is the dropout, which allows dropout regularisation to be applied to the network. It allows a regularisation method to be applied to a chosen layer, in which random output values of the previous layers are removed with probability P. Typically P is set to 0.5, however the author decided to set it to 0.75, reducing the dropout by 25% because of lack of training data. It is an effective method because it prevents co-adaptation between units because each unit is forced to learn meaningful features by itself. Essentially, it cannot depend on the presence of other units which can be removed at random.

The tensorflow placeholders were initialised next. These placeholders are different from normal variables because they do not require an initial value, they simply allocate block of memory for future use in the training phase. By default, they do not have a constrained shape but we can save computational power by setting shapes.

The input placeholder X will hold the spectrograms in a set of one-dimensional vectors, as this is the preferred format for batch data. The shape of X is (n, m) where n is the number of spectrograms and $m = width * height$ of the spectrogram. When each individual spectrogram is processed by the CNN, the placeholder X_reshaped will take the one-dimensional vector and restore it back to its original dimensions dynamically. The placeholder Y is used to store the output of the CNN, the softmax vector. Therefore it's shape is directly controlled by the number of genres being predicted. The image below shows the discussed placeholders.

```
# declare the training data placeholders
# input x - for 644 x 128 pixels = 82432 - this is the flattened image data that is drawn from
x = tf.placeholder(tf.float32, [None, 82432])
x_reshaped = tf.reshape(x, [-1, 644, 128, 1]) # dynamically reshape the input
y = tf.placeholder(tf.float32, [None, 4]) # placeholder for corresponding labels- 4 genres
keep_prob = tf.placeholder(tf.float32)  # dropout (keep probability)


```

Figure 4.6: TF Placeholder's initialisation

A function called conv_layer was implemented to create convolution layers. As there are multiple hidden layers this function minimised code duplication and improved readability. The function sets the filter shape which controls how large the convolution windows are. Then it initialises the weights and bias for the filter using two tf variables. These variables are passed to a convolutional layer operation using the tf.nn.conv2d() function. A ReLU nonlinear activation is then applied to the output and finally a max pooling layer is applied using the tf.nn.maxpool() function. The function can be seen in the image below.

```
  #function to create convolutional layers
def conv_layer(input_data, num_input_channels, num_filters,
    filter_shape, pool_shape, name):
    # setup the filter input shape for tf.nn.conv_2d
    conv_filt_shape = [filter_shape[0], filter_shape[1],
    num_input_channels, num_filters]

    # initialise weights and bias for the filter
    weights = tf.Variable(tf.truncated_normal(conv_filt_shape, stddev
    =0.03),name=name+'_W')

    bias = tf.Variable(tf.truncated_normal([num_filters]), name=name+'
    _b')

    # setup the convolutional layer operation
    out_layer = tf.nn.conv2d(input_data, weights, [1, 1, 1, 1], padding
    ='SAME')
    # add the bias
    out_layer += bias

    # apply a ReLU non-linear activation
    out_layer = tf.nn.relu(out_layer)

    # now perform max pooling
    ksize = [1, pool_shape[0], pool_shape[1], 1]
    strides = [1, pool_shape[0], pool_shape[1], 1]
    out_layer = tf.nn.max_pool(out_layer, ksize=ksize, strides=strides,
    padding='SAME')

    return out_layer

```

Figure 4.7: Hidden layer function

Next, the fully connected layer was implemented following the convolutional layers. This involved setting up the weights and biases which constitute each unit. 1024 units are used to capture the complex features which are fed into it. The output is then passed through a ReLU non-linear function called tf.nn.relu(). Lastly, the dropout regularisation is applied to 25% of the units to prevent co-adaptation.

The output of the fully connected layer is fed into the softmax classifier layer. The function tf.nn.softmax() will produce the categorical probability distribution vector of genres which is stored in a placeholder Y_. The code snippet below shows the fully connected and softmax layer.

```
# setup some weights and bias values for the fully connected layer, then activate with ReLU
wd1 = tf.Variable(tf.truncated_normal([41 * 8 * 64, 1024], stddev=0.03), name='wd1')
bd1 = tf.Variable(tf.truncated_normal([1024], stddev=0.01), name='bd1')
dense_layer1 = tf.matmul(flattened, wd1) + bd1
#Relu activation
dense_layer1 = tf.nn.relu(dense_layer1)
#Apply dropout here
dense_layer1 = tf.nn.dropout(dense_layer1, keep_prob)
# Softmax Classifier layer
wd2 = tf.Variable(tf.truncated_normal([1024, 4], stddev=0.03), name='wd2')
bd2 = tf.Variable(tf.truncated_normal([4], stddev=0.01), name='bd2')
logits = tf.matmul(dense_layer1, wd2) + bd2
y_ = tf.nn.softmax(logits)

```

Figure 4.8: FC and Softmax layer implementation

Then, the cross entropy variable is then initialised using the tf.nn.softmax_cross_entropy_with_logits() function. This variable is the loss value that the neural network will minimise using an optimiser. The optimiser used in this implementation is the Adam optimiser, this algorithm is an improvement of the stochastic gradient descent which has seen popular use in deep learning applications. It is useful for problems that have a large number of parameters and noisy gradients. It also has small memory requirements and is computationally efficient.

The accuracy metric is then defined, it is equal to the average difference between predicted genre and the ground truth label, the smaller the difference the higher the accuracy. The code snippet below shows hows the accuracy metric is calculated.

```
1 cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
      logits=logits, labels=y))
2
3 # add an optimiser
4 optimiser = tf.train.AdamOptimizer(learning_rate=learning_rate).
      minimize(cross_entropy)
5
6 # define an accuracy assessment operation
7 correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
8 accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
9
10
```

Figure 4.9: Cross-entropy and Accuracy setup

After the required variables and placeholders were initialised. The training loop was implemented using the tf.session() function. This function dynamically loads the convolutional neural network into memory so that training can begin. Within the session, a nested for loop is used to make a training loop.

Given the number of epochs, the loop will iterate through the dataset multiple times and produce batches of data at each step to feed into the network. Each epoch produces an accuracy and loss value for each batch, this allows the tracking of network performance throughout the training phase. The image below shows the training loop.

```
1
2 # setup the initialisation operator
3 init_op = tf.global_variables_initializer()
4
5 # allows us to save the model's weights and biases for further use
    after training
6 saver = tf.train.Saver()
7
8 with tf.Session() as sess:
9     # initialise the variables
10     sess.run(init_op)
11     for i in range(10):
12         print "epoch ", (i+1)
13         for epoch in range(num_of_epochs):
14             #average cross_entropy for each epoch
15             avg_cost = 0
16             # for i in range(batch_size):
17             batch_x, batch_y = getBatch(trainData, trainLabels,
    batch_size, epoch)
18             # train network with batch data
19             _, cost, batch_acc = sess.run([optimiser, cross_entropy,
    accuracy],feed_dict={x: batch_x, y: batch_y, keep_prob: dropout})
20             #output accuracy and loss for batch
21             print "Mini-Epoch:", (epoch + 1) , "accuracy: {:.3f}".
    format(batch_acc), "loss: {:.3f}".format(cost)
22
23     print "Training complete! Now time to save the model"
24
25
```

Figure 4.10: TF Training loop

After the training loop is exited, the trained CNN's parameters are saved to the local disk space. This is to ensure that the trained CNN can be used as the second component of the system following the mel-spectrogram converter. Lastly, the CNN is evaluated using unseen test data without optimising any feature kernels or weights, to truly grasp whether the model is able to generalise well on unseen data or if it is overfitting to the training data.

```
    saver.save(sess, "tmp/model.ckpt")

    for z in range(30):
        finalData,finalLabels = getBatch(testData, testLabels, 10, z)
        final_acc, softmaxOutput = sess.run([accuracy,y_], feed_dict={x
    : finalData, y: finalLabels,keep_prob: 1.0})
        print "final accuracy:  ",final_acc
        print softmaxOutput


```

Figure 4.11: Saving and evaluation of the model

When the CNN has been trained, it can be run in a prediction phase as part of the system as mentioned. So that when given a query song, it can produce a vector of genre probabilities for the playlist generation algorithm. This prediction phase consists of loading the previously trained CNN model and allowing it to produce its genre predictions without optimising its parameters. The code snippet below shows this process.

```
# allows us to save and restore the model's weights and biases for further use after training
saver = tf.train.Saver()

predictions = []

with tf.Session() as sess:
    # initialise the variables
    sess.run(init_op)

    saver.restore(sess, 'tmp/model.ckpt')

    for epoch in range(num_of_epochs):
        data = getBatch(targetData,batch_size,epoch)
        # predict and store output for these songs
        softmaxOutput = sess.run(y_, feed_dict={x: data,keep_prob: 1.0})
        print softmaxOutput
        if epoch == 0 :
            predictions = softmaxOutput
        else:
            predictions = np.vstack((predictions,softmaxOutput))

print predictions.shape
np.set_printoptions(suppress=True)
print predictions

# store predictions in serialised format for later processing
joblib.dump(predictions,'UserChosenSongs.prediction')


```

Figure 4.12: Prediction phase of the model

4.4 Playlist Generator

The playlist generator was also implemented in Python. The first step consists of reading the output vectors of the CNN from a pickled array, where each index represents the genre vector to a corresponding song. A for loop is used to match each vector to a list of song titles which is also stored in a text file. Each pair is then stored in a dictionary data structure called songLibrary, which sets the key as the song title and the value as the genre vector. This dictionary is also used as the basis for creating the high dimensional genre space as seen in the snippet below.

```
import numpy as np
from sklearn.externals import joblib
from math import *
import operator

songLibrary = {}
counter = 0
# 1. match predictions to corresponding songs
# predictions = joblib.load('UserTestSongs.prediction')
predictions = joblib.load('predictions.data')
rockPredictions = joblib.load('extraRock.prediction')
userPredictions = joblib.load('UserChosenSongs.prediction')
with open('songTitles.txt') as f:
    for line  in f:
        songLibrary[line.strip('\n')] = predictions[counter]
        counter += 1


```

Figure 4.13: The building of the genre space

Then, a variable called querySong is initialised which stores the name of the user's chosen song. The dictionary songLibrary which contains every song's coordinates in the genre space is then iterated through using the built in function iteritems(). In this loop, the coordinates of each song is used to calculate their Euclidean distance to the query song as seen in the image below.

```
# 2. choose a query song then use similarity algorithm to return top
    ten similar songs
querySong = "Cocoa Butter Kisses (ft Vic Mensa & Twista) (Prod by Cam
    for JUSTICE League & Peter Cottont (DatPiff Exclusive)"

querySongData = songLibrary[querySong]

del songLibrary[querySong]

topSongs = {}

for key, value in songLibrary.iteritems():
    # calculate distance
    dist = np.linalg.norm(querySongData-songLibrary[key])
    # store in distance directory
    topSongs[key] = dist

# order top songs by distance
sortedSongs = sorted(topSongs.items(), key=operator.itemgetter(1))
# take top 10 closest songs
sortedSongs = sortedSongs[:10]


```

Figure 4.14: Using the chosen similarity metric, the most similar songs to the query are found

These distances are then stored in a new dictionary variable called dist. This dictionary of distances is then sorted using the built in function sorted(). With the distances sorted in ascending order the first 10 indexes will contain the closest songs to the query song. These 10 songs are used to create the recommended playlist. The output of the algorithm is then written to the disk space using the joblib.dump() function.

4.5 User Interface

The user interface was created using the Python micro-framework Flask. It is useful for building web applications that need to be connected to a python backend. The app consists of two pages: the homepage and the demo page. The frontend was coded using HTML 5 and CSS and JavaScript.

By incorporating the CSS framework Bootstrap, this allowed the web pages to be responsive across different device platforms. The homepage uses a minimalistic design which includes the system logo and a visible link to the demo page. The implemented homepage can be seen below.

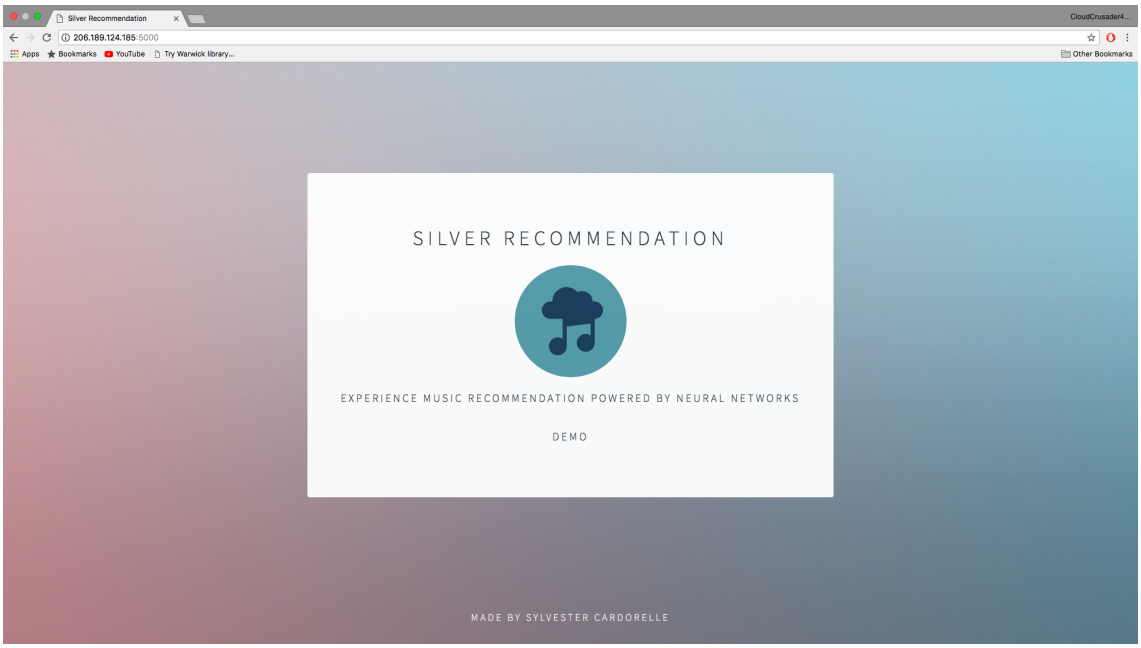

Figure 4.15: Home page

The demo page adheres to the same design as the homepage. It displays a three step procedure to use the system. The first step allows the user to choose a song from the available library which is displayed in a dropdown list.

The second step is automatically completed for the user upon selection of a query song. The page will display relevant information about the chosen song including it's genre and relevant statistics e.g. a scatter chart depicting the song within the genre space. The demo page can be seen in the image below.

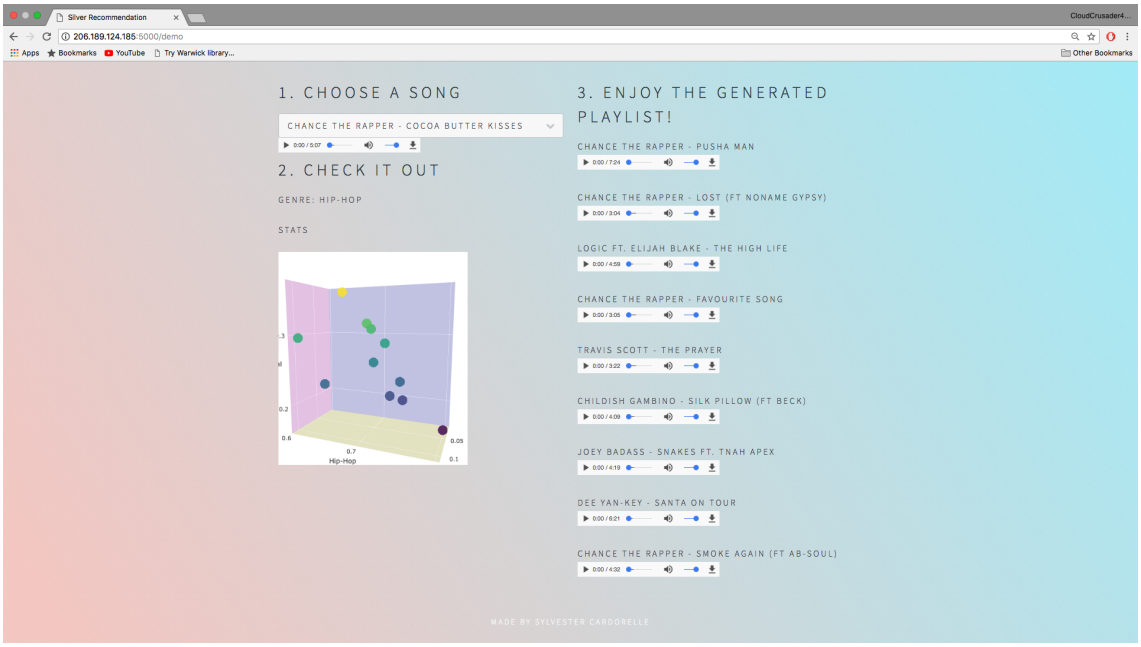

Figure 4.16: Demo page

The third and final step of the demo page is to display the recommended playlist. The user is able to listen to each song within the playlist, using the integrated audio player. This user interface satisfies the usability principles discussed in the design section because it is user friendly and easy to understand for the user.

Chapter 5

Testing

In this section, the author will discuss the training results of the CNN and the evaluation of its performance. The playlist generation algorithm is also tested and the overall system is evaluated in a recommendation setting using a human opinion study.

5.1 CNN training

The developed CNN initially performed a 10 genre classification task consisting of the following genres: Blues, Rock, Hip-Hop, Classical, Jazz, Heavy Metal, Reggae, Pop, Disco and Country. The GTZAN dataset conveniently provided audio tracks for these 10 genres (100 songs per genre). The preliminary results showed a low average accuracy of 12% which was due to the lack of training data to fit the model to such a complex task. This led to the decision of scaling down the number of genres to 4 (Pop, Hip-Hop, Rock and Folk) which more data was available on in the FMA dataset.

As expected, the model accuracy increased to an average of 30% with the scaling down of the number of genres. This score was compared to the benchmark score of 63% that Tao et al produced for the 4 genre classification task [12]. However, Tao did not use the same genres as the author instead he used Classical, Blues, Metal and Disco. This lead the author to the conclusion that the choice of genres greatly effects the model complexity because a given genre may be better defined than others. For example, there is not a clear decisive boundary between a Pop song and a Folk song, as many Folk songs could be considered Pop or vice-versa. Tao's choice of genres were better defined than the author's initial choice e.g. Classical and Blues have a clear distinction.

The CNN seemed to have trouble discerning between pop and folk music. As the folk genre has a very wide subset of styles. Thus, the author decided to reconstruct the dataset and choice of genres to more well defined ones, so that Tao's accuracy could be surpassed. This lead to doing a 3 genre classification task as this allowed the author to select 3 well defined genres (Hip-Hop, Rock and Classical) which were abundant in the FMA dataset. Now that the CNN could be trained on a 1000 songs per genre, the CNN would be able to learn the complex relationships in the data. When performing this task the CNN was able to achieve an average accuracy of 75%. The tables below summarises these results.

	4 Genre Classification	3 Genre Classification
Genres Used	Pop, Rock, Hip-Hop and Folk	Hip-Hop, Rock and Classical
No. of songs per genre	800	1000
No. of hidden layers	2	3
No. of Epochs	100	200
Batch-size	30	10
Dropout	0.75	0.75

Figure 5.1: Hyperparameter values used to train the CNNs for the genre prediction task. The 'Batch Size' is the number of training examples fed to the CNN per learning step (mini-epoch).

No. of genres	Average CNN Accuracy	Tao Feng NN Benchmark Accuracy
4	30%	63%
3	75%	70%

Figure 5.2: CNN training accuracy results

5.2 Playlist generation

Upon completion of the playlist generation algorithm, the optimal similarity metric was decided and the output playlists were analysed to determine whether the recommendations produced by the algorithm were sensible. Two similarity metrics were compared: Euclidean distance and cosine distance.

Euclidean distance similarity was chosen over cosine similarity after generating multiple playlists using both metrics. The results produced using the Euclidean metric appeared varied and sensible, whereas the results using the cosine metric did not. The playlists produced for songs of different genres were almost identical when using cosine distance. This led to the conclusion that this set of re-occurring songs were situated on a particular plane of the genre space, that had an optimal cosine distance to the majority of the songs, explaining the duplicate playlists.

After the similarity metric was decided, the generated playlists were analysed in a qualitative manner using data visualisation. An example playlist was generated using the query song 'Cocoa butter kisses' by Chance the rapper. This is a popular Hip-Hop song with gospel influences. The pie chart below visualises the song's genre composition.

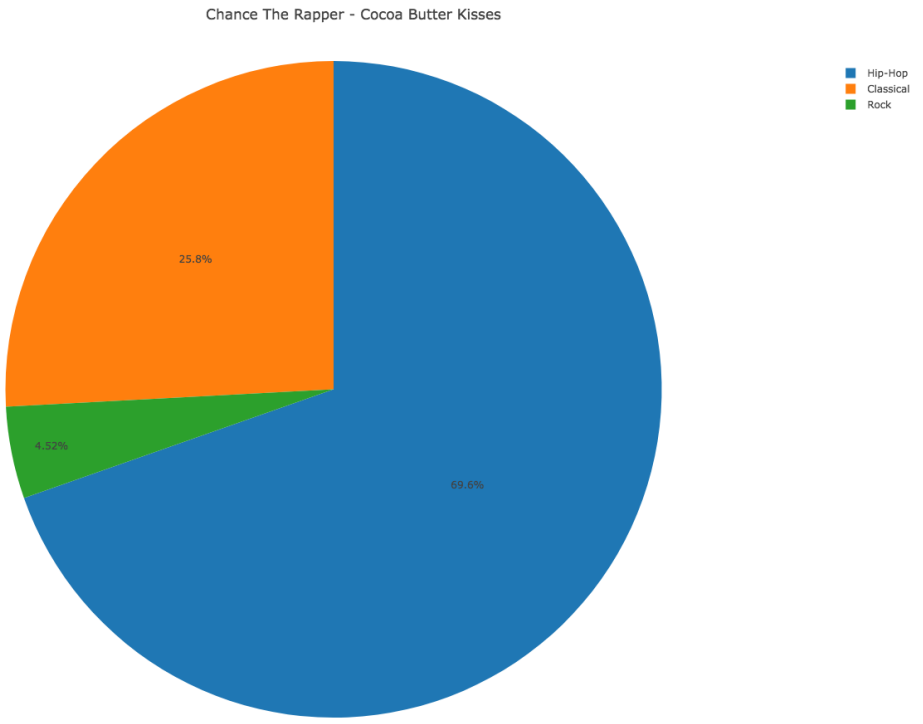

Figure 5.3: The genre composition of 'Cocoa butter kisses' (69.6% Hip-Hop, 25.8% Classical and 4.52% Rock)

The CNN was able to correctly predict the song's genre (Hip-Hop receiving the highest probability score). It was evidently able to identify other genre elements such as Classical. Also, the CNN did not think much rock was contained within the song, explaining the low probability score of 4%.

The following playlist was generated using the query song 'Cocoa butter kisses'. The songs are in descending order of similarity (the more similar to the query song, the higher the song's rank in the playlist).

1. Chance the rapper - Pusha man **(Genre: Hip-Hop)**
2. Chance the rapper - Lost **(Genre: Hip-Hop)**
3. Travis Scott - The prayer **(Genre: Hip-Hop)**
4. Chance the rapper ft Childish Gambino - Favourite song **(Genre: Hip-Hop)**
5. Logic ft Elijah Blake - The high life **(Genre: Hip-Hop)**
6. Childish Gambino ft Beck - Silk pillow **(Genre: Hip-Hop)**
7. Dee Yan-Key - Santa on tour **(Genre: Jazz)**
8. Childish Gambino - Shoulda known **(Genre: Hip-Hop)**
9. Chance the rapper - Juice **(Genre: Hip-Hop)**
10. Chance the rapper ft Ab-Soul - Smoke again **(Genre: Hip-Hop)**

The 3D scatter graph below visualises the playlist and the query song in the genre space. The centre green point represents the query song and the surrounding points represents the playlist cluster. The resulting playlist is interesting because 3 out of the 4 closest songs were also made by the same artist, Chance the rapper. This indicates that the CNN could be identifying higher-level features such as vocal style and range from the data, which is why it was able to identify the artist's songs with similar vocal energies and predict them similar genre scores.

Also, one of the songs on the playlist called Santa on tour was a jazz instrumental track, surprisingly the track contained a similar ambience and slow-paced drums to the query song. This highlights the system's ability to recommend similar songs regardless of genre.

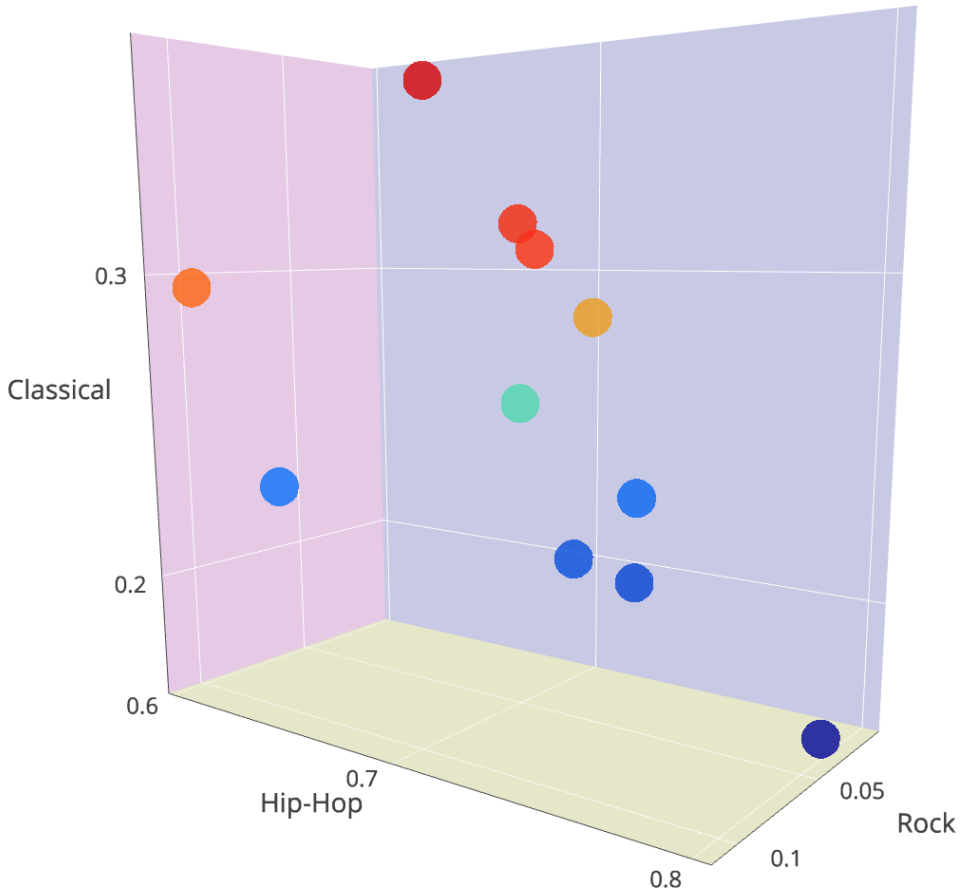

Figure 5.4: 3-D representation of the playlist in the genre space

5.3 Human evaluation study

Although many automatic playlist generation algorithms have been proposed over the years, there is currently no standard evaluation procedure [43]. As a result, it is difficult to quantitatively compare different algorithms and objectively determine if any progress is being made. At present, the predominant approach to playlist algorithm evaluation is to conduct human opinion surveys, which can give a direct and realistic measure of performance [44].

Alternatively, current quantitative evaluation schemes either reduce the problem to a (discriminative) information retrieval setting, or rely on simplifying assumptions that may not hold in practice. An example of a considered automated evaluation scheme, is the playlist semantic cohesion test. Semantic Cohesion can be defined by the frequency of meta-data co-occurrence (e.g. songs by the same artist or of the same genre) or the entropy of the distribution of genres within a given playlist.

This assumption is truly unrealistic, as songs generally map to multiple meta-data tags. Assigning each song to precisely one semantic description may therefore discard a great deal of information, and conceal the semantic content of the playlist. Issues of semantic ambiguity aside, a more fundamental flaw lies in the assumption that cohesion accurately characterises playlist quality. In reality, this assumption is rarely justified, and evidence suggests that users often prefer playlists that are diverse in style and genre [45].

Human evaluation studies are a form of evaluation that helps us to determine the perceived quality of playlists with good confidence. One of the major drawbacks of such studies however is that they can be time consuming and expensive [46]. One particular problem in this domain is that in many experimental designs, the participants are asked to listen to all the tracks of the playlist. This can take a long time, even if only excerpts are played.

The study the author carried out consisted of gathering 15 randomly sampled students from the University of Warwick. They were randomly sampled through an anonymous Facebook post and the subjects remained anonymous throughout the study to remove any bias. Each subject was asked to choose a query song constraint by the genres the CNN was able to predict (Hip-Hop, Classical and rock). A playlist was then generated for each subject based on their chosen song for them to listen to. They were then each given a survey to fill in, so that they could evaluate the playlist.

The survey was made using the Google forms platform. It consisted of three important questions which are commonly used in similar studies. These questions would allow for meaningful conclusions and statistics to be made. The 3 questions can be seen below:

1. How would you describe yourself as a music listener? (on a scale from 1 to 5, 1 = casual listener and 5 = musicophile)
2. How would you rate the playlist based on the given song? (on a scale from 1 to 5, 1 = unrepresentative of the query song and 5 = a well-curated playlist)
3. What factors were important when deciding playlist quality? (tick the relevant choices from the following)
   (a) Similar sounds
   (b) Similar genres
   (c) Similar energy
   (d) Similar instrumentation
   (e) Similar artists
   (f) Diversity in the playlist

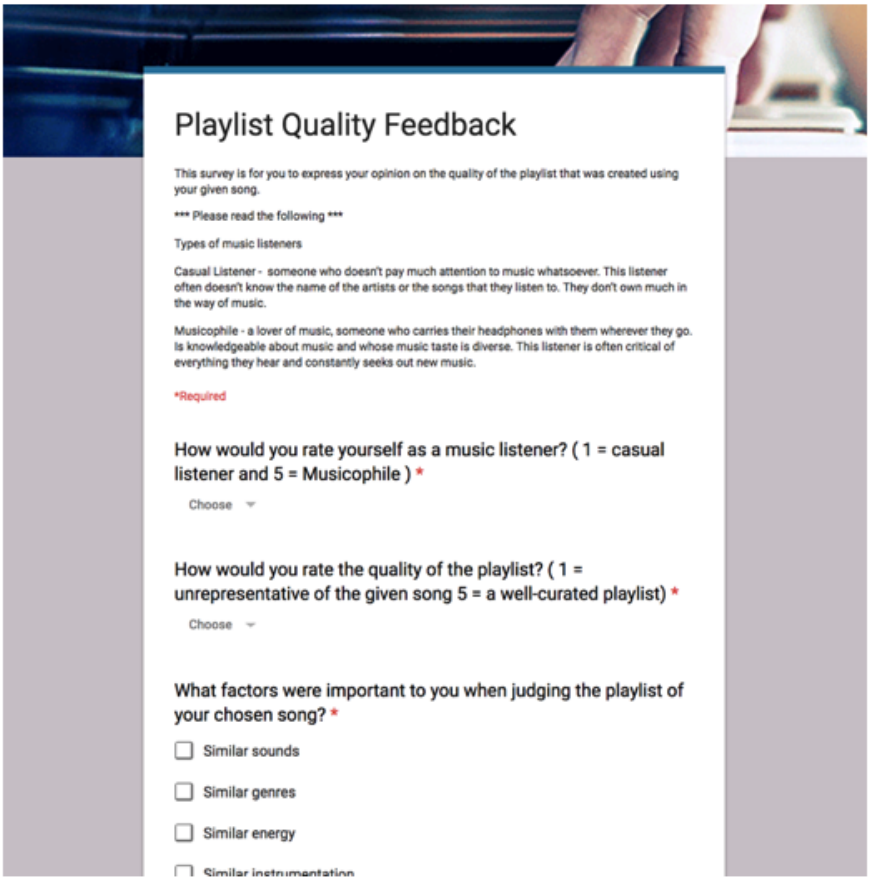

Figure 5.5: The Google form survey used in the study

The survey also provided a space at the end for any additional comments about the playlist. The first question allowed the author to analyse the distribution of listeners in the study. In music theory, there is a distinction between two types of music listeners. There is the casual listener who is described as someone who doesn't pay much attention to music and often doesn't know the name of the artists or songs that they listen to. At the other end of the spectrum is the musicophile. The musicophile is a lover of music and is described as someone who is critical of everything they hear and constantly seeks out new music.

This is important because if the study consisted solely of musicophiles for example, this would not make the study representative of a typical recommendation setting where there is a variety of listeners.

The results below show a great variation of listeners in the study, making the consequent results more reliable and representative. With approximately 30% of the subjects being at either spectrum(casual listener or audiophile) and the remaining subjects being at the equilibrium.

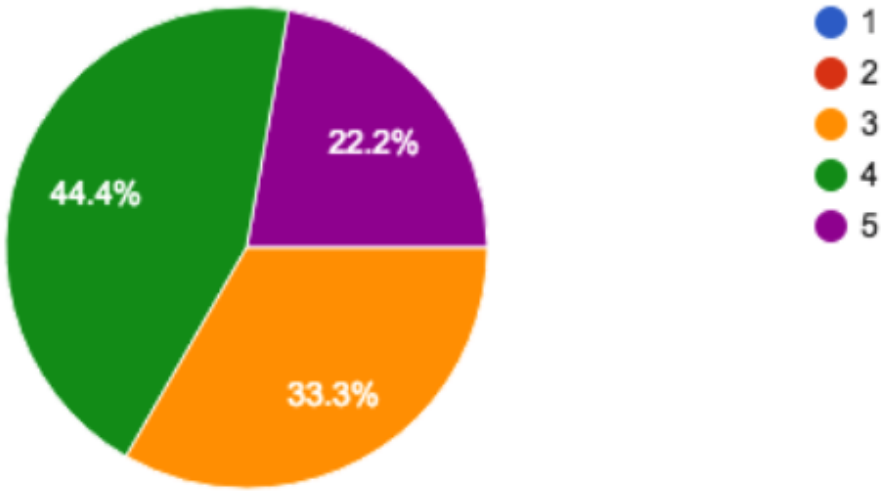

Figure 5.6: Distribution of listeners in study

When the subjects were asked to rate the playlist on a scale of 1 to 5. The results showed a positive response from the subjects, with 55.6% giving a score of 4 and 22.2% giving a score of 5. This shows that the system is able to produce sensible recommendations. The pie chart below shows the results.

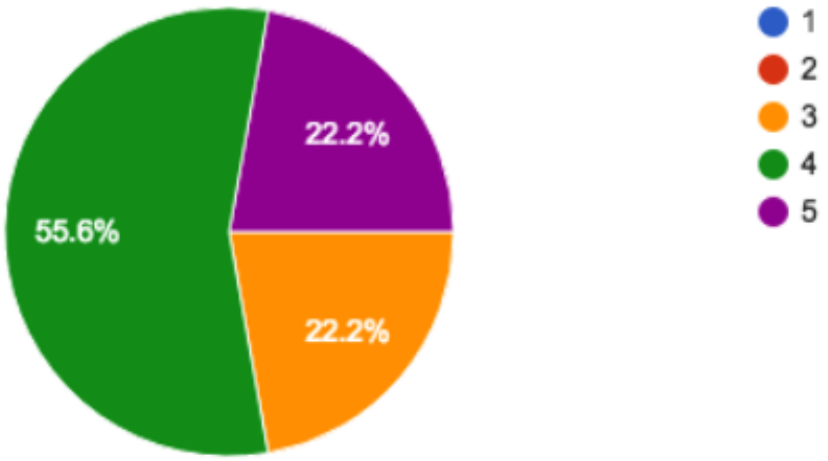

Figure 5.7: Playlist quality results

When the subjects were asked to choose the factors that were important to them when deciding the playlist quality. 76.9% of the subjects stated that similar energy to the query song is a key quality of a good playlist and this characteristic was the most popular of all the given factors. Other factors that stood out was similar sounds and similar instrumentation which over 50% of subjects saw these as important factors also. The least popular factor among the subjects was similar artists with only 15.4% of the subjects deeming it and important factor. This supports the research showing that playlist diversity is more important than semantic cohesion.

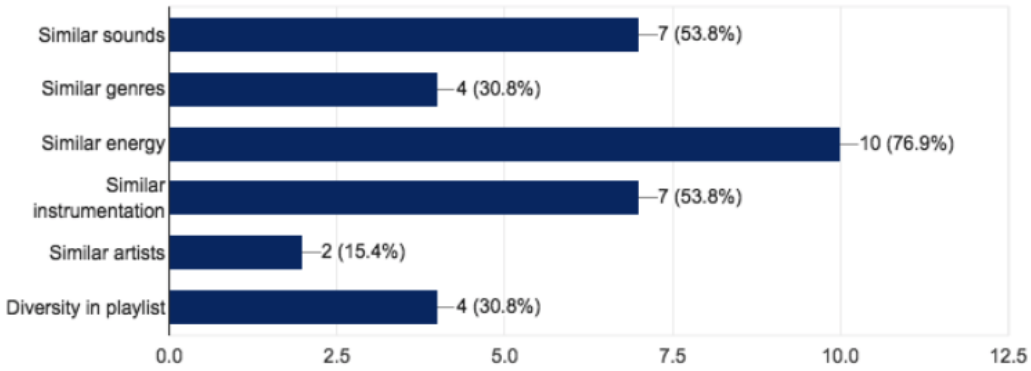

Figure 5.8: Playlist factors results

Chapter 6

Author's Assessment of the Project

This section contains the author's thoughts and reflections of the project, the technical challenges faced and how they were overcome. Additionally, reflecting on the achievements gained from the project and it's contributions to the field of music information retrieval

6.1 Technical challenges

A significant challenge the author faced was constructing a suitable dataset. As CNN require a large amount of data to train its parameters, a large collection of audio tracks needed to be used for the training phase. However, it is very difficult to find large-scale datasets of music audio due to the legal requirements of free music. The situation is improving with the recent release of the FMA dataset which contains 100,000 songs but this still pales in comparison to industry level datasets used by Spotify and Apple Music. The majority of the available datasets are created by volunteers and researchers in the MIR field, therefore there is little support when problems are faced with the dataset(s). For example, human error can lead to mislabelling of song genres which can be hard to detect.

The Million Song Dataset (MSD) was considered by the author, but it no longer supplies the audio of the songs but only contains the precomputed features which were not relevant to the task. The GTZAN dataset was then discovered by the author, which provided audio for songs spanning 10 genres. However, a key issue arose when the author discovered that the dataset contained only 1000 songs and that the free music archive contained corrupt MP3 files which were not detectable until the attempted spectrogram conversion. This proved troublesome because the corrupt files would interrupt the spectrogram conversion process and lead to the restarting of the script, lengthening the conversion from an average of 1-2 hours to 4 hours.

There was also a lack of genre labelling in the meta-data supplied within the FMA dataset. Resulting in more than 3000 songs receiving no genre label, rendering them useless for the task. Consequently, leading to another problem which was an unbalanced dataset. In the FMA dataset, due to the missing labels; the number of rock songs were much higher than the number of hip-hop or classical songs. When training a neural network with an unbalanced dataset it can have a negative impact on its performance, especially when a particular class is over represented in the data. It is a known principle that using a balanced dataset with a CNN will yield the best results because it will learn to generalise well rather than overfitting.

To overcome the problem of corrupt files and an unbalanced dataset, a portion of the FMA dataset was ignored to avoid these problems. Instead, the author implemented a Python web Crawler that could directly access the free music archive servers and automatically download songs of the desired genre to balance the dataset.

Furthermore, prior to this project the author had no experience in the field of machine learning or music information retrieval. As a result, a considerable amount of time was spent researching and self teaching the required concepts of these fields using online courses and resources such as Coursera. With the support of his supervisor and peers, the author was able to gain a sufficient amount of knowledge needed to progress through the development of the project.

Further to the forementioned, there was the challenge of using a convolutional neural network in the system. Although a powerful model, it consists of a plethora of hyper parameters. Thus, the author constantly faced questions concerning these parameters such as:

- How many hidden layers should be used?
- What value should the learning rate be?
- What layer should dropout regularisation be applied to?

However, there are no set formulas or answers to these questions as they depend on the nature of the task. Therefore, many of these hyper parameters were decided through 'trial and error' and experimentation.

Time management is also crucial when using a CNN, on average it would take 6 to 7 hours to train with the FMA dataset. Therefore, the optimisation and tweaking of the network's hyper parameters became very time-consuming. The Author planned strategic counter measures to combat this, ensuring that meaningful tasks could be completed whilst the CNN was training, leading to a much more efficient project.

Also, as the CNN performs computationally heavy tasks, the author was unable to train it on the planned OSX machine which used an intel i5 processor and 8GB RAM. Instead, the author migrated the project to a more powerful Ubuntu machine (16GB RAM and 4vCPUs) which was provided by the DigitalOcean cloud service. This also allowed a larger dataset to be used with the increased storage space.

6.2 Project management

The project development used an agile methodology. To manage tasks efficiently, the objectives were broken down into smaller tasks that could be tackled individually. These smaller tasks were then placed onto a physical agile progress board, where tasks are either to be started, in progress or completed.

Each task was tackled within a weekly sprint cycle and at the end of each sprint cycle a review meeting would take place. In this meeting, the author would carry out a review of the progress made within that week and place countermeasures to account for unforeseen challenges.

These meetings also provided time for the author to prepare for future tasks. The project progress was also tracked using an online diary application known as Day One. This diary allowed the author to reflect on progress and identify potential solutions.

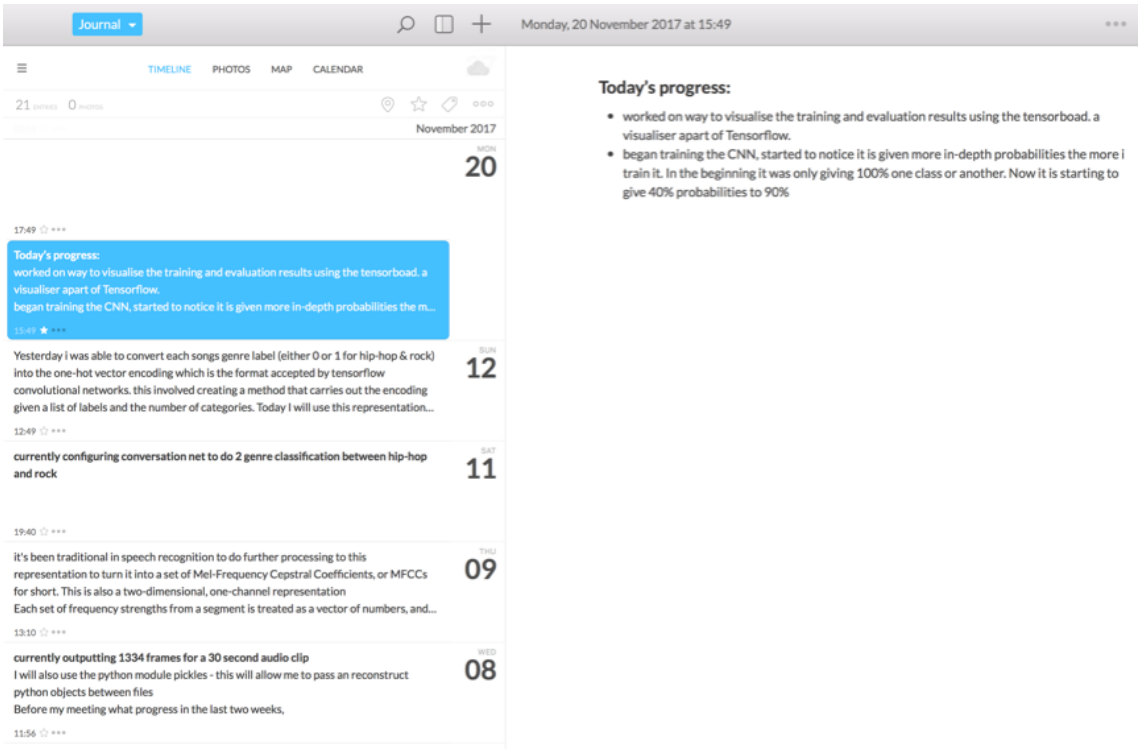

Figure 6.1: Diary application Day One used to track project

To keep track of the project's development, the version control platform Github was used. It allowed the backing up of different iterations of the system to prevent any data loss. Various development branches were created throughout the project for the different functionalities of the system to allow a clear separation between tasks. Overall, the project management methodology worked well and provided sufficient countermeasures against time constraints and unforeseen circumstances.

6.3 Project contributions

This project was able to make contributions to the field of music information retrieval by incorporating deep learning techniques to a music recommendation task.

This project was able to present a new recommendation model and solution to the cold start problem. By not relying on a collaborative filtering model, the produced system can never experience a cold start because it uses audio signals which are always available, unlike historical usage data.

The project was able to utilise a CNN in a creative manner that has never been done before. The adopted approach shows that the genre classification task can be quite powerful when used as acoustically meaningful measure of music.

Finally, the system was able to achieve the underlying objective of avoiding the popularity bias that is commonly seen in collaborative filtering, by solely using music audio as data the system is able to recommend new and unpopular songs. Thus in a recommendation setting, listeners can be exposed to larger proportions of a music library and find new artists/songs that match their taste.

Hopefully, this research will be able to further aid the field of music information retrieval and music recommendation.

6.4 Project achievements

The recommendation system that was created was able to achieve the objectives that were originally set out in the introduction. Firstly, it was able to apply feature engineering techniques to the data to provide a better representation using the mel-spectrogram. Secondly, the system was then able to train a convolutional neural network which was able to identify a song's genre and output categorical genre probabilities using a tensorflow implementation.

Additionally, the system was able to process a library of songs and plot them in a high dimensional genre space using the predicted probabilities. Further enabling the system to use a similarity metric to find similar songs using Euclidean distance in this space and produce recommendations to the user in the form of a playlist.

Moreover, the system was programmed in a modular fashion which will allow future modifications to be made easily without developers having to make huge changes to the code.

Chapter 7

Conclusion

The project was able to use a deep convolutional neural network to recommend songs using predicted genre probability scores. The system was evaluated by producing recommendation playlists in a human evaluation study. Despite the audio signal being the most difficult feature to extract meaningful information from. The results show that the produced recommendations are of a sensible nature. Hence, the project shows that using deep learning translates very well to a music recommendation setting and has the potential to outperform existing systems such as collaborative filtering models.

An improvement for the project would be to increase the number of genres used within the CNN and genre space. In Music theory, the idea that every song contains elements of several genres is frequently discussed. Therefore, this could potentially allow the system to produce more detailed and complex genre compositions of each song. To carry out this task the system would require the usage of a larger dataset. Traditionally, CNNs are known to scale very well with dataset size.

Another important improvement, would be the use of temporal stitching. In musical terms, temporal stitching is a technique that involves stitching multiple excerpts from different temporal locations to produce a single sample. The system currently uses 30 second audio samples to train the network which are taken from the middle of each song. However, these excerpts may not be representative of the given songs because the audio profile of a song is subject to change over time. Therefore, if temporal stitching is employed, multiple excerpts can be taken from each song over different points in time e.g. the beginning, middle and end. Each audio sample would become more representative of the whole song. Thus, resulting in more accurate recommendations.

A final thought for future work, would be to use the deep learning component in a completely different manner. Instead of the system choosing songs to recommend, it could generate the music instead. Recent studies in MIR have attempted to generate music using deep learning [47]. As progress is made in this area, it could lead to a revolution of music recommendation, such that when given a query song a similar sounding song could be generated from a deep network.

Hence, this project demonstrates how deep learning can be applied to a music recommendation system to solve the cold start problem. This project created a new recommendation model using solely audio signals without relying on historical usage data like collaborative filtering. Furthermore, this system not only has the potential for stand-alone use but could enhance a hybrid music recommendation system.

In conclusion, this project has provided a viable method to recommend new and unpopular music to enrich the listener experience which is the underlying objective of music recommendation itself.

- *Sylvester Cardorelle*

Bibliography

[1] J. Bennett, S. Lanning, *et al.*, "The netflix prize," in *Proceedings of KDD cup and workshop*, vol. 2007, p. 35, New York, NY, USA, 2007.

[2] P. Knees, T. Pohle, M. Schedl, and G. Widmer, "Combining audio-based similarity with web-based data to accelerate automatic music playlist generation," in *Proceedings of the 8th ACM international workshop on Multimedia information retrieval*, pp. 147–154, ACM, 2006.

[3] S. K. Lee, Y. H. Cho, and S. H. Kim, "Collaborative filtering with ordinal scale-based implicit ratings for mobile music recommendations," *Information Sciences*, vol. 180, no. 11, pp. 2142–2155, 2010.

[4] X. N. Lam, T. Vu, T. D. Le, and A. D. Duong, "Addressing cold-start problem in recommendation systems," in *Proceedings of the 2nd international conference on Ubiquitous information management and communication*, pp. 208–211, ACM, 2008.

[5] M. A. Casey, R. Veltkamp, M. Goto, M. Leman, C. Rhodes, and M. Slaney, "Content-based music information retrieval: Current directions and future challenges," *Proceedings of the IEEE*, vol. 96, no. 4, pp. 668–696, 2008.

[6] D. Bogdanov, M. Haro, F. Fuhrmann, A. Xambó, E. Gómez, and P. Herrera, "Semantic audio content-based music recommendation and visualization based on user preference examples," *Information Processing & Management*, vol. 49, no. 1, pp. 13–33, 2013.

[7] ISMIR, "About the international society of music information retrieval," 2008.

[8] A. Jeffries and J. Lagomarsino, "What happened to pandora's music genome project?," Jan 2017.

[9] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms," in *Proceedings of the 10th international conference on World Wide Web*, pp. 285–295, ACM, 2001.

[10] K. Jacobson, V. Murali, E. Newett, B. Whitman, and R. Yon, "Music personalization at spotify," in *Proceedings of the 10th ACM Conference on Recommender Systems*, pp. 373–373, ACM, 2016.

[11] A. Van den Oord, S. Dieleman, and B. Schrauwen, "Deep content-based music recommendation," in *Advances in neural information processing systems*, pp. 2643–2651, 2013.

[12] T. Feng, "Deep learning for music genre classification," *private document*, 2014.

[13] P. Kozakowski and B. Michalak, "Music genre recognition," Oct 2016.

[14] E. J. Humphrey, J. P. Bello, and Y. LeCun, "Moving beyond feature design: Deep architectures and automatic feature learning in music informatics.," in *ISMIR*, pp. 403–408, 2012.

[15] S. Dieleman, P. Brakel, and B. Schrauwen, "Audio-based music classification with a pretrained convolutional network," in *12th International Society for Music Information Retrieval Conference (ISMIR-2011)*, pp. 669–674, University of Miami, 2011.

[16] "Recommending music on spotify with deep learning," Aug 2014.

[17] A. T. Mick, "Generating spectrograms the hard way with numpy," May 2017.

[18] A. V. Oppenheim, "Speech spectrograms using the fast fourier transform," *IEEE spectrum*, vol. 7, no. 8, pp. 57–62, 1970.

[19] D. Ellis, "Chroma feature analysis and synthesis," Apr 2007.

[20] E. Alpaydin, *Introduction to machine learning*. MIT press, 2014.

[21] S. M. Weiss and C. A. Kulikowski, *Computer systems that learn: classification and prediction methods from statistics, neural nets, machine learning, and expert systems*. Morgan Kaufmann Publishers Inc., 1991.

[22] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, "Scikit-learn: Machine learning in python," *Journal of machine learning research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[23] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*, vol. 1. MIT press Cambridge, 2016.

[24] G. Bebis and M. Georgiopoulos, "Feed-forward neural networks," *IEEE Potentials*, vol. 13, no. 4, pp. 27–31, 1994.

[25] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256, 2010.

[26] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[27] A. Giusti, D. C. Ciresan, J. Masci, L. M. Gambardella, and J. Schmidhuber, "Fast image scanning with deep max-pooling convolutional neural networks," in *Image Processing (ICIP), 2013 20th IEEE International Conference on*, pp. 4034–4038, IEEE, 2013.

[28] N. Kalchbrenner, E. Grefenstette, and P. Blunsom, "A convolutional neural network for modelling sentences," *arXiv preprint arXiv:1404.2188*, 2014.

[29] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[30] T. Bertin-Mahieux, D. P. Ellis, B. Whitman, and P. Lamere, "The million song dataset.," in *Ismir*, vol. 2, p. 10, 2011.

[31] B. 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*, pp. 7–12, ACM, 2012.

[32] K. Benzi, M. Defferrard, P. Vandergheynst, and X. Bresson, "Fma: A dataset for music analysis," *arXiv preprint arXiv:1612.01840*, 2016.

[33] A. Coates and A. Y. Ng, "The importance of encoding versus training with sparse coding and vector quantization," in *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pp. 921–928, 2011.

[34] J. Salamon and J. P. Bello, "Deep convolutional neural networks and data augmentation for environmental sound classification," *IEEE Signal Processing Letters*, vol. 24, no. 3, pp. 279–283, 2017.

[35] S.-H. Cha, "Comprehensive survey on distance/similarity measures between probability density functions," *City*, vol. 1, no. 2, p. 1, 2007.

[36] S. Polamuri, "Five most popular similarity measures implementation in python," Mar 2017.

[37] J. Nielsen, *Designing web usability: The practice of simplicity*. New Riders Publishing, 1999.

[38] D. Budko, "Aws vs. digitalocean: Which cloud server is better – hacker noon," Oct 2017.

[39] B. McFee, C. Raffel, D. Liang, D. P. Ellis, M. McVicar, E. Battenberg, and O. Nieto, "librosa: Audio and music signal analysis in python," in *Proceedings of the 14th python in science conference*, pp. 18–25, 2015.

[40] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[41] G. van Rossum and F. L. Drake, "pickle–python object serialization," *Available from World Wide Web: http://docs. python. org/lib/module-pickle. html*.

[42] C. Adams, *Learning Python Data Visualization*. Packt Publishing Ltd, 2014.

[43] B. McFee and G. R. Lanckriet, "The natural language of playlists.," in *ISMIR*, vol. 11, pp. 537–542, 2011.

[44] G. Bonnin and D. Jannach, "Automated generation of music playlists: Survey and experiments," *ACM Computing Surveys (CSUR)*, vol. 47, no. 2, p. 26, 2015.

[45] A. Berenzweig, B. Logan, D. P. Ellis, and B. Whitman, "A large-scale evaluation of acoustic and subjective music-similarity measures," *Computer Music Journal*, vol. 28, no. 2, pp. 63–76, 2004.

[46] L. Barrington, R. Oda, and G. R. Lanckriet, "Smarter than genius? human evaluation of music recommender systems.," in *ISMIR*, vol. 9, pp. 357–362, Citeseer, 2009.

[47] A. Huang and R. Wu, "Deep learning for music," *arXiv preprint arXiv:1606.04930*, 2016.