MASTER THESIS

Content-based Music Recommendation

Author:
Benjamin POHL

Supervisor:
Dr. George FAZEKAS

A thesis submitted in fulfillment of the requirements

for the degree of MSc. Big Data Science

in the

Centre for Digital Music
School of Electronic Engineering and Computer Science

QUEEN MARY UNIVERSITY OF LONDON

Abstract

School of Electronic Engineering and Computer Science

MSc. Big Data Science

Content-based Music Recommendation

by Benjamin POHL

Recent boom in the industry of digital music distribution has made recommendation systems more of a necessity than a convenience both for consumers as well as business. Recommendations hitherto are mostly provided using collaborative filtering methods. This, however, fails to provide appropriate recommendations if no or sparse feedback data is obtainable which is known as the cold start problem. This research project aims to evaluate whether different machine learning techniques like CNN, RNN, Triplet-Network and transfer learning can overcome the cold start problem through content-based recommendations by predicting latent factors obtained by using a collaborative-based model. Quantitative and qualitative evaluation showed that machine learning based models are able to provide relevant recommendations and outperform traditional approaches and that musical-embeddings play a crucial role in doing so. Findings from this work can be used to provide better recommendations, perhaps leading to more business turnover and higher customer satisfaction.

Acknowledgements

I would like to thank Dr. George Fazekas from Queen Mary University and Dr. Elio Quinton from Universal Music Group (UMG) for their valuable and constructive suggestions during the planning and development of this research work.

Contents

List of Figures

List of Tables

List of Abbreviations

BoT	Bag of Tones	20, 29
BoW	Bag of Words	20
CE	Compute Engine	26
CF	Collaborative Filtering	7, 13, 15, 24
CNN	Convolutional Neural Network	vii, 21, 22, 37, 38, 44, 46, 49, 50, 53
dB	Decibel	25
DBN	Deep Belief Networks	5
DCT	Discrete Cosine Transform	19
ELU	Exponential Linear Unit	32
GCP	Google Cloud Platform	26, 28
GMM	Gaussian Mixture Models	5
HAS	Human Auditory System	19
HMM	Hidden Markov Models	5
LSTM	Long Short-Term Memory	22, 36, 37, 46
MAU	Monthly Active User	11
MF	Matrix Factorization	12
MFCC	Mel Frequency Cepstral Coefficient	5, 19, 21, 29, 43
MIR	Music Information Retrieval	2, 4, 5, 14
ML	Machine Learning	5, 6, 20, 21
PCA	Principal Component Analysis	12

RD	Research and Development	8
ReLU	Rectified Linear Unit	32
RNN	Recurrent Neural Network	vii, 22, 36, 46, 47, 49
SGD	Stochastic Gradient Descent	12, 13
SLA	Service Level Agreement	28
STFT	Short Time Fourier Transform	18
SVD	Single Value Decomposition	12
SVM	Support Vector Machines	5, 6
UMG	Universal Music Group	3, 24–26, 28, 30, 58
VM	Virtual Machine	26, 28

Chapter 1

Introduction

1.1 Context and structure

Recommendation systems shape our everyday life. They are responsible for creating your personal YouTube-feed to watch while eating breakfast. They show you your top news-stories on your way to work, give you the best places to eat for lunch, help you find the perfect birthday present for a friend, prepare a relaxing evening with your top Netflix shows and, not to be forgotten, recommend the perfect music while performing all these activities [33], [49], [53], [64], [71].

Since the world wide web took off in the 1990s the amount of information accessible by the average person has exploded and recommendation systems have become more of a necessity than a convenience. A search for 'sweet cat's' yields about 71 million search results on YouTube. Without a recommendation system who knows which of these videos would appeal to me, I would need to spend more than my entire lifetime to find it. The same is true for music with top tier streaming services like Spotify or Apple Music reaching a music library of over 50 million songs [116].

Current state of the art recommendations are done using the user's historic preferences [79]. This is based on the assumption that users who agree in the past (i.e. listening to the same songs/artists) will, most likely, also agree in the future. This is known as the collaborative-based approach for recommendations. However, the drawback of this approach is, that music with no previous history (i.e. new music) or music in a niche genre is unlikely to be recommended because of scarce data. As most users listening preferences follow the power law, this is true for a large part of the musical archive [53]. With Spotify's founder Daniel Ek publicly stating the number of new tracks added to Spotify each year has grown from 20.000 songs per day [120] in 2018 to nearly 40.000 songs a day in 2019 [123], this *cold start problem* needs to be addressed.

For that reason, the Music Information Retrieval (MIR) research community has put a lot of effort into content-based recommendation systems. This involved mainly the development of features extraction methods and functions that define similarities. Approaches for similarity measurements between audio signals often rely on ad-hoc defined metrics that are based on prior information, hence, they might not be suitable for recommendation [55], [77]. To bridge the semantic gap between one's preferences towards a song and the corresponding signal, researches have taken latent factors models [49] and applied them to the musical domain with good initial results [85].

The main research objective of this thesis is to evaluate if the cold start problem can be overcome via content-based music recommendations incorporating latent factors models using machine learning techniques like artificial neural networks.

My contributions include:

- Identified the problem and stated relevant background as well as related papers
- Conducted several experiments with conventional artificial neural networks like CNN and RNN.
- Explain, develop and implement a novel machine learning technique for embedding creation e.g. Triplet-Network for content-based recommendations
- Evaluate the achieved recommendations using qualitative and quantitative analysis

- Present potential improvements as future work

This project was done in partnership with Universal Music Group (UMG) who provided all the data and computational resources as well as valuable and constructive suggestions during planning and development.

Chapter 2 will give an overview of the related work. Chapter 3 gives an overview of music recommendation and its challenges, the key audio features description, latent factor models and the used machine learning techniques. Chapter 4 will display the dataset, pre-processing steps and hardware environment. Chapter 5 will lay out the used methodologies. Chapter 6 will present the conducted experiments and result. Chapter 7 will close with the conclusion and future work.

1.2 Personal motivation

Music listening is an essential part of my life. For the most part, I am using music as a brain manipulation tool e.g. to focus or to relax and I am always actively searching for more music to enjoy. This, however, is not an easy task as current music recommendation systems still have a lot of room for improvement to satisfy my need for the discovery of new music. Therefore, I decided to pursue my master's project in the area of content-based music recommendation in cooperation with Universal Music Group with the hope to discover new techniques to improve current recommendation systems.

Chapter 2

Related Work

2.1 Introduction

The MIR research community is comprised of experts in the field of signal processing, musicology, information retrieval, physiology, machine learning, optical music recognition and more. Some of the core applications are music generation, playlist generation or continuation and music recommendation.

This chapter will give an overview of related work which will include different methods used to solve the same or similar problems like the one at hand. The first part will be an overview of research conducted into audio feature extraction and the second part over similarity metrics and recommendations.

2.2 Audio feature extraction

Early research into feature extraction yielded only very basic features like spectral centroids or more advanced ones like pitch or harmonicity. These features were then used in a basic classification task using distance metrics such as euclidean-distance or cosine-distance [18]. These basic features were replaced by more engineered ones like the

Spectrogram in its different variations (i.e. powered, mel-scaled or with cepstral coefficients). These were then used to build a Bag of Tones model (using the bag of words approach), and classified by histogram intersection [20]. However, these classifications had a limited amount of basic classes such as ′music′, ′no music, ′chatting′, ′noise′ etc [19]. [24] was one of the first researches to build a system that would provide similarity-based recommendations to a given song using the above-mentioned features.

The rise in popularity of Machine Learning (ML) techniques also influenced the field of MIR. In the beginning of the 2000s established statistical and ML techniques like Gaussian Mixture Models (GMM) in combination with K-means clustering [27], [29], Monte Carlo simulations, Hidden Markov Models (HMM) [39] and Support Vector Machines (SVM) [35], [43] in combination with the before mentioned Mel Frequency Cepstral Coefficient (MFCC) features were used. Research was also conducted if these new methods could be used with simpler features such as timbre or tonal ones [37] or if a combination of them would yield better predictions [47], [51]. The rise of neural networks or earlier architectures like Deep Belief Networks (DBN) has also made it possible to investigate if features can be learned instead of predefined [66], [73], [78].

2.3 Music recommendation

Another part of the research community focused on improvements to similarity metrics, which can serve as an important feature for recommendations, by learning them [55] or incorporating algorithm used in recommendation systems such as learning to rank [82]. Another option would be to incorporate latent vectors [85] which are based data retrieved from collaborative filtering. These content-based extensions are based on the assumption that collaborative recommendations outperform content-based ones [53], [79]. Pure content-based recommendations have not seen much attention as preliminary results did not show good results [38], [40], [43].

On the other hand, pure collaborative-based ones, have been a major topic for the MIR community, which might be due to the fact collaborative-based recommendations have been extensively researched in other domains such as movies [49] or E-commerce with good initial results [23] and can relatively easily be adapted towards music recommendation. This lead to improvements and interesting discoveries such as the elimination or dampening of the *grey sheep* problem which is, because of the sparse listening preferences of users, quite common in the musical domain [107].

There has also been research into hybrid recommendations i.e. the combination of both content-based and collaborative-based techniques. Early hybrid recommendations combined these two approaches through different ML techniques such as SVM [41], [45] or Bayesian network [48]. More advanced techniques were used by [82], [85]. Other researches incorporated more user-oriented contextual information such as self-created playlist [50] or social media information [56], [63]. These user-centric recommendations have been recognized as more promising as they cover a broader range of attributes such as *coverage* [65] or *novelty* [53] which seems to be more important than simple quantitative measurement such as accuracy. This has sparked many different research directions such as ones based on users' explicit or implicit feedback [32] or semantic description [84]. Another area of interest is the modeling of perceived similarity [46], [59]. Direct improvements towards producing more *novel* recommendations have been made by [68] and more recently by [101] which try to incorporate the popularity of songs.

Chapter 3

Background

3.1 Introduction

This chapter will give an overview of different recommendation methods and their use cases. It will also include an explanation of the variety of features that can be extracted from audio signals, how they are constructed and their use for the problem of content-based recommendation. The last part will outline machine learning techniques.

3.2 Recommendation methods

In the world of recommendation systems, there are 4 major groups of how a recommendation can be provided. These are Collaborative Filtering (CF), also referred to as collaborative-based filtering throughout this thesis, context-based filtering, content-based filtering and hybrid methods that combine a mix of the above-mentioned ones. As this thesis focuses on content-based recommendation using collaborative-based latent vectors, context-based filtering will not be explored in depth.

3.2.1 Collaborative filtering

This method uses data from a large user base (collaborating) which includes their preferences/tastes and is able to make intelligent recommendations (filtering/predictions) for a given user. Recommendations are made with the underlying assumption that, for given user X, finding a user Y with similar preferences, the probability of agreement between X and Y for a give statement S is higher than between X and a randomly chosen person [76] which is derived from the psychological analysis of humans that a recommendation from a person similar to us, will yield better ones than from foreigners [14].

The method was invented by the Research and Development (RD) team at PARC (formally know as XEROX PARC) [15] and has since become the most widely used recommendation systems for all aspects of life. It has also been demonstrated that collaborative-based consistently outperform alternatives [72]. The same is true for music recommendation as shown by [79]

The captured data, on which the filtering is performed, should include the user's taste towards the area of recommendation. Therefore, feedback data is mostly used. This can be in the form of explicit feedback which is mostly categorical (e.g. 0 to 5 stars) or binary (e.g. thumb up / down) or implicit feedback which is inferred from a user's usage-pattern (e.g. play-counts, watch-/listen-time, ...) [16]. Early CF adaptations into the music domain relied mostly on explicit feedback [53], [67]. This was due to the findings that explicit feedback is more accurate as it incorporates positive and negative treats and hence can provide better recommendations. Even though it might be more accurate, explicit feedback is very sparse, meaning, the amount of explicit feedback per user is very low. Research has shown that a large amount of implicit feedback can outperform small amounts of explicit feedback based CF recommendations [67]. Therefore, the use of implicit feedback through user tracking has become the norm.

As explained above, the main Idea of CF is to create a matrix M that captures users'

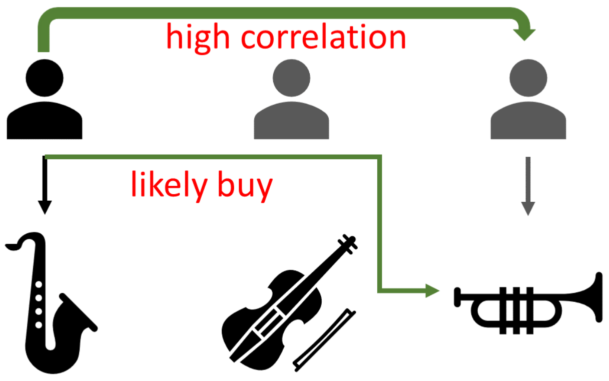


FIGURE 3.1: Nearest-neighbour user-based CF

preferences. The matrix M is build by a list of m users $U = u_1, u_2, ..., u_m$ and n items $I = i_1, i_2, ..., i_n$. This means each user u_i has its preferences captured in a list of items I_{ui}. For each active user $u_\alpha \in \mathcal{U}$, there are to main actions that can be performed: predictions and recommendations. Predictions are numerical probabilities, denoted as $P_{u,i}$, which represent the predicted overlap for a given item $i_j \notin I_{u_a}$. Recommendations are a list of N items I_r containing the highest N-made predictions from a list of items $I_r \cap I_{u_\alpha} = \Phi$ i.e. a list containing only items the active user u_α has not expressed preferences for [28]. There are a variety of algorithms that can perform these two actions which can be divided into two groups: memory-based and model-based approaches.

Memory-based Collaborative filtering

Memory-based approaches, also referred to as user-based or item-based, make use of the entire matrix M to generated predictions or make recommendations by finding similar users or items or user-item combinations. One of the simplest but also most used techniques, at least in the early 2000s, is to find similar users, that is, users whose preferences overlap, is the $nearest-neighbour$ technique. See figure 3.1 for illustration. We can make a prediction $p_{u,i}$ for our active user u_α with

$$p_{u,i} = \frac{\sum_k \textit{similarity-function}\left(s_i, s_j\right) R_{u,j}}{\text{number of ratings}}$$

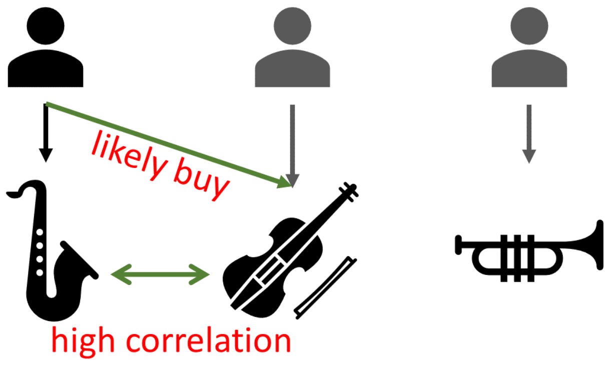


FIGURE 3.2: Item similarity, item-based CF

with $k \leq U - 1$. $p_{u,i}$ denotes the prediction for user u and item i and $R_{u,j}$ the provided rating for item j. Number of ratings can also be expressed as $\sum_k$ *similarity-function* $(|s_i, j|)$. To adjust for ones user individual ratings one can also use an adjusted average

$$p_{u,i} = \frac{\sum_k \textit{similarity-function}\,(s_i, s_j)\,(R_{u,j} - \overline{R}_j)}{\text{number of ratings}}$$

One can use a variety of functions to calculate similarities. One of the most widespread ones are Pearson correlation or cosine similarity [28]. The Pearson correlation is defined as:

$$sim(i,j) = \frac{\sum_{u \in U}\left(R_{u,i} - \overline{R}_i\right)\left(R_{u,j} - \overline{R}_j\right)}{\sqrt{\sum_{u \in U}\left(R_{u,i} - \overline{R}_i\right)^2}\sqrt{\sum_{u \in U}\left(R_{u,j} - \overline{R}_j\right)^2}}$$

[12]. The cosine similarity is defined as:

$$sim(i,j) = \cos(\vec{i}, \vec{j}) = \frac{\vec{i} \cdot \vec{j}}{\|\vec{i}\| * \|\vec{j}\|} = \frac{\sum_k R_{i,k} R_{j,k}}{\sqrt{\sum_k R_{i,k}^2}\sqrt{\sum_k R_{j,k}^2}}$$

[5].

These similarity-based measurements are also used in the item-based filtering where one does not want to find similar users, but similar items. See figure 3.2 for an illustration. This method can be used if only a small number of similar users can be found and hence the user-based recommendation might be quite poor. One can create an item-based similarity matrix M by the above-mentioned similarity metrics and take the (weighted) average number of ratings of the active user u_α on M to make item-based recommendations.

Both user-based and item-based filtering were successful in the past by providing decent recommendations, but they both face some challenges. One of these challenges is the sparsity of the matrix M [28]. In this case, this translates to a matrix with lost of zeros i.e. no ratings. Both Spotify and Apple Music claim a music library of more than 50 million songs. According to Spotify's financial statement 2019, the average Monthly Active User (MAU) spends around 25h listening to music. Even if one assumes each listening would be unique, which is a false assumption as stated in the financial report, but just for comparison, this results, at an average length of 4 minutes per song, in $4.500 = (25 * 60 * 12)/4$ songs per year, or an overlap of less than 0.0001% to the continuously growing library of songs held by Spotify or Apple Music [116]. The second problem is the scalability of the above-mentioned algorithms. As they need to both loop through all users and all items, they scale with a complexity of $O(Ux(UxI))$ or $O(Ix(UxI)$. Spotify claims a user base of more than 100 million users making the above-presented options unusable [116].

Model-based Collaborative filtering

Model-based collaborative filtering approaches provide recommendations by first developing a model that captures the user's preferences. These models are developed through various techniques envisioned from the data mining or machine learning community, i.e. they take a probabilistic approach. A large variety of different methods such as Bayesian's networks, Markov decision process or clustering methods have been used [62]. To deal with the problems of sparsity and scalability of the memory-based methods, dimensionality reduction techniques are used. The idea behind them is to project the data into a lower-dimensional space while retaining most of the variance.

Some know algorithms are Single Value Decomposition (SVD) [4] or Principal Component Analysis (PCA) [11]. One adaption of SVD is Matrix Factorization (MF) [26]. This method caught a lot of attention after winning the Netflix prize competition in 2009 [49], [57]. It works by factoring our sparse matrix M into two smaller ones X and Y. These one hold the latent representation of the initial user and item data. Then a recommendation $\hat{r}_{u,i}$ can be made by taking the inner product between x_u and y_i i.e. $\hat{r}_{u,i} = x_u^T y_i$. To learn possible values for X and Y the objective function $\min_{x_\star, y_\star} \sum_{r_{u,i}} \left(r_{ui} - x_u^T y_i\right)^2$ can be minimised. One can also add a lasso or ridge regression $+\lambda \left(\|x_u\|^2 + \|y_i\|^2\right)$ as a regularisation method. λ can be tuned or learned to find the optimal amount of regularisation. The objective function can be optimized using Stochastic Gradient Descent (SGD) i.e. following the negative gradient of the objective (loss) function to find a local optimum [58].

This method works very well in practice and yields good results [58] but mostly relies on explicit feedback data. As explain in the previous section, most user preferences are recorded in an implicit way. To yield good results on these data sets one needs to make adjustments on the objective function and optimisation algorithm as proposed by [54]. The first proposed change is to binarize the captured preferences

$$p_{ui} = \begin{cases} 1 & r_{ui} > 0 \\ 0 & r_{ui} = 0 \end{cases}$$

and add a confidence measurement $c_{u,i}$ on the binarized observations $c_{ui} = 1 + \alpha r_{ui}$. This yields a new cost function incorporating the confidence measurements:

$$\min_{x_\star, y_\star} \sum_{u,i} c_{ui} \left(p_{ui} - x_u^T y_i\right)^2 + \lambda \left(\sum_u \|x_u\|^2 + \sum_i \|y_i\|^2\right)$$

To make use of the confidence measurement one needs to take all possible U and I pairs into account. As explained in the previous chapter this leads to a problem of scalability. This can be circumvented by using a alternating least square optimisation instead of the SGD. This reduces the complexity from $O(f^2 * I * U^2)$ to $K * \left(O\left(f^2\mathcal{N} + f^3m\right) + O\left(f^2\mathcal{N} + f^3n\right)\right)$ with $\mathcal{N}$ being the number of non-zero observed values, which as explained before, might be close to 0.0001% of the total number of items an K being the number of alternating least squares that are performed [54].

Even though memory-based and model-based approaches yield good results under the right circumstances, they both have, in addition to the sparsity and scalability, some major downsides. One of the most significant ones is the cold start problem. This refers to the situation in which appropriate recommendations can't be provided by their inherent lack of data. If either a new user or a new item enters the CF system, they have no ratings/preferences [22], [53], [90]. Another problem CF has to deal with is one individual's preferences which can be, because of the immense music corpus, unique. This can lead to users with a unique taste for whom it will be hard to find similar users. This is also known as the grey sheep or black sheep problem [22], [90]. The before-mentioned algorithm includes a confidence variable which increases if an item is observed multiple times. This can lead to a popularity bias in the recommendation as popular items have higher listening percentages and therefore might be recommended more often. This can end in a self-feeding, feedback loop [22], [90].

The next section will give an overview of content-based recommendation systems that can circumvent some of the CF problems outlined above.

3.2.2 Content-based filtering

Content-based recommendation systems are similar to item-based CF methods in the sense that, they try to find similarities between items to make a recommendation. But as the name suggests, content-based systems use the content of the items to determine similarities and not captured user preferences [76]. In the musical domain, the *content* refers to the audio signal. However, as pure audio is quite a dense data format (more in section 3.3), a variety of extracted features from the audio signal have been used.

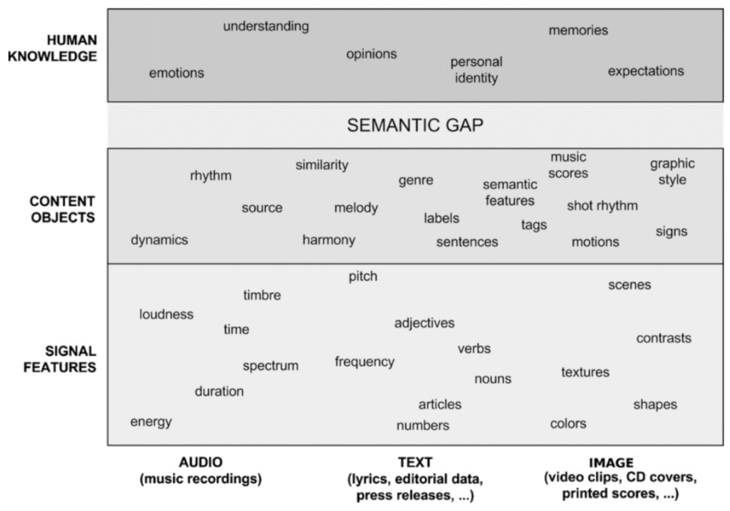


FIGURE 3.3: The semantic gap in the musical domain. Source: [53]

This can include but is not limited to features like timbral, tonal or temporal ones, or attached meta-data like genres, categorical or free form tags [53], [76], [83], [102].

The area of automatic feature extraction is one of the main topics for the MIR research community. Feature extraction techniques will be explained in section 3.3. To make accurate content-based recommendations researches have often used different extracted features in conjunction. Around the late 2000's the combinations involved mostly a technique for genre classification with the addition of simple features such as loudness or tempo analysis [40], [44], [60], [101]. One can also use more subjective features such as a song's mood, as emotional reactions can be a good indication for recommendations [13], [61]. The problem with these features is to accurately map them to a given user as emotional reactions also differ from person to person [13], [53]. This is also known as the 'semantic gap' problem in music. See figure 3.3 for an illustration. Whereas the MIR community has made astounding progress in the terms of low- mid-level feature extraction, bridging the semantic gap remains a challenging problem [53].

By focusing on the audio signal and omit user-captured preferences, content-based systems can solve some of the major challenges of the CF counterpart. The cold start problem is solved by measuring similarities of extracted features without relying on preference ratings. This also avoids the popularity bias as no ratings or listening counts are taken into account which or may not be desirable as one might want to recommend a *top ten hit list* for which popularity measurement will be needed. The grey sheep problem is not entirely solved as the quality of recommendation depends on the number of total music samples and their genre distribution. Some niche genres might have a low number of samples to draw from. However, content-based systems also introduce new challenges. The main one being the quality of the automatic feature extraction techniques. As explained above they also have to deal with the semantic gap problem. Other challenges might arise if one solely relies on similarity metrics for the recommendation as this might lead to a problem of novelty i.e. one will only get recommendations for similar songs and it lacks exploration into other genres. However, if the findings from [52] can be trusted, only 7% of users require a lot of exploration/novelty to be satisfied with their recommendations. However, depending on the purpose of the recommendations, they might not need to be novel and therefore the problem of novelty can be re-framed as a desirable property rather than a problem.

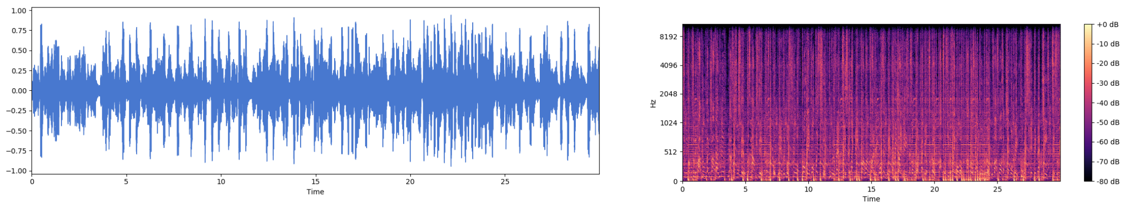
3.2.3 Hybrid approaches

Hybrid approaches are, as their names suggest, a combination of different filtering techniques with the goal to provide better recommendations. The idea behind a hybrid approach is that, with a combination of different techniques, the overall system can benefit from the strengths of the individual systems, while negating their drawbacks. For example can one combine the strengths of both a CF and content-based systems as proposed by [85] to circumvent the 'cold start' problem and also narrow the semantic gap by incorporating CF data in the model. There are numerous ways of combining said techniques. The example above would fall into the mixed category of combinations. Other main ways of combinations include weighting, switching and cascading. A weighted combination consists of two individual techniques whose output recommendations are combined using a fixed or learned weight. This allows for an easily understandable and flexible system as new techniques can be added quickly. Switching means that different techniques are used in parallel but only one provides a final recommendation. The confidence of the recommendations from each model could serve as a switching criterion. Cascading is a method to combine different techniques in a series. The output from the first technique will be used as the input for the following one. The in-out puts can differ depending on the goal. One can use recommendations as output i.e. a first technique would make coarse recommendations where the second one would refine them. Another way would be to uses the features from the first model as output and feed that as input for the second one [30], [53], [71].

3.3 Audio features

This section will focus on explaining audio feature representation techniques, namely the spectrogram and its variations e.g. mel-spectrogram and the mel frequency cepstral coefficient. Feature extraction methods are generally used if the raw data can't be used because of its high dimensionality or if single features do not carry enough information to contribute to the overall picture. Audio data is highly dimensional. For example, a typical image from the imagenet dataset is 256 by 256 pixel image with a color range of 0 to 255 can be represented in $16.711.680 = (256 * 256 * 2^8)$ bits. The typical audio file has a sample rate of 44.100Hz with a bit depth of 16 i.e. double that of an image and 2 channels (stereo). For a 4 minute songs this results in $338.688.000 = (44.100 * 16 * 2) * 240$ bits. Audio is also different from images in that it is serial and not static.

Whereas a single pixel in an image can carry useful information independent of its



(A) Raw audio signal / waveform

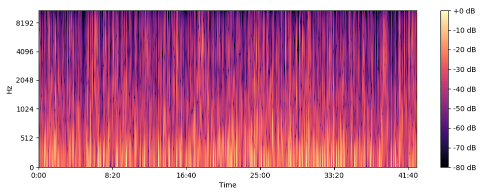
(B) Spectrogram linear scale

FIGURE 3.4: Audio transformation from waveform to spectrogram with 30sec sample taken from the song awol - drs everyman Hip-Hop genre

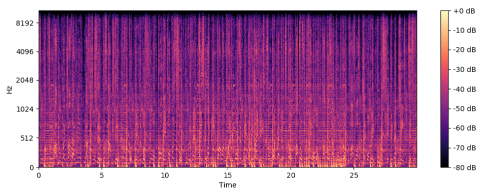
location for tasks like object detection, a single sample in audio can not be taken independently. This is due to the nature of sound being serial i.e. occurring in waves, which have to be analyzed in sequence. This is why most audio feature extraction techniques use a sliding window approach with significant overlap, to capture the meaning of a specific sequence [124].

3.3.1 Spectrograms

The spectrogram is a representation of how the audio signal's frequencies change over time [74]. It displays the magnitude of the Fourier transformation of the audio signal spitted into frames and weighted windows. These characteristics makes the spectrogram one of the most used representation for musical analysis, dating back into the mid to late 20th century [3], [8]. The spectrogram is created by splitting the audio signal into three dimensions and display it onto a, generally, two dimensional plot. One axis, normally the x-axis, displays time and the y-axis the frequency band. The energy or intensity of the audio signal is highlighted by color coding of the plot. The energy refers to the amount of pressure created by the source, which is in turn picked up by the microphone. In general, signals in the low frequency region (bass) carry a lot of energy whereas the high ones carry only small amounts of energy [74]. An example of the audio signal before transformation and be seen in figure 3.4a and after transformation in figure 3.4b.



(A) Short Window



(B) Long Window

FIGURE 3.5: Comparison of short and long window Fourier transformation

The first step for the spectrogram creation consist of the audio decomposition into frequencies. As one can see the waveform is a continuous wave of many overlapping frequencies with different energy along the time domain. To decompose the wave into multiple waves for each frequency the Fourier transformation can be used [10]. This transformation imagines each function as combination of waves but at different frequencies. However, performing the decomposition on the entire song, would yield very precise frequency identifications but nearly all information of frequency transition would be lost. That is why the audio signal is splitted into chunk's i.e. the transformation is performed using a sliding window where one can assume the signal is stationary. As one can see in figure 3.5a where a small window of 25 samples, which results in 1.1ms, is used, it is easy to identify at which times frequency changes occur but the frequency identification is blurry. In contrast figure 3.5b shows a very large window of 2048 samples (93ms) where frequency resolution increased but at the same time, the time resolution decreased. In practice, a variant know as Short Time Fourier Transform (STFT) is used which combines the window, overlap and Fourier transformation into one function. One can use different shapes of windows which are chosen based on the assumption of the signal. One common window function is the Hann-window [7], which has the shape of a raised cosine function with non-zero fall of at the edges. To compensate for information loss close to the edge, overlap is added.

This serves as the baseline representation for more further processing, namely log scale and mel scale and discrete cosine transformation. This will be explored in the following section 3.3.2.

3.3.2 Mel Frequency Cepstral Coefficient

The MFCC is a audio signal representation which is based on the spectrogram but with more added transformation. These reduce the dimensionality and encapsulates information which should resemble how the Human Auditory System (HAS) perceives the audio signal [6], [25]. The transformation from a spectrogram to MFCC can be broken into three steps:

- **Logarithmic scale**: A logarithmic transformation is a non-linear operation in which an interval is not increased linearly but by the a factor of its base logarithm. This is done as the HAS perceives loudness not on a linear but on a non-linear scale which is modeled by the Steven's power law with an exponent of 0.67 meaning to double the perceived loudness energy of the signal has to be about 7 times as strong [2]. Therefore, the log transformation will match the extracted energy levels from the spectrogram to perceives loudness by the HAS
- **Mel-scale**: The mel- or melody-scale, is another non-linear transformation which based on the perceived level of pitch by the HAS. [1] discovered that correct pitch identification/recognition varies at different frequencies. The HAS seem to be able to better identify the correct pitch at lower frequencies compared to high ones. The perceived pitch follows an somewhat exponential decay at higher frequencies. One can map an audio signal into m mel from frequency f with the following function: $m = 2595 \log_{10}\left(1 + \frac{f}{700}\right)$ [1]. After the conversion into mel's, one can construct triangular filter banks using $mel_i : mel_{i+1}$. To retrieve the mel-scaled output one has to multiply the log-scaled input with each filter bank and add the output together. The output of this transformation is also known as a mel-spectrogram which is the main data source for this project.
- **Discrete Cosine Transform (DCT)**: The last step of the MFCC creation is to take the DCT from the log- and mel-scaled spectrogram. This is similar to the first step of the spectrum creation where the DFT is used to assume a function as a wave.

> Here one assumes that output of the above explained step can be represented as (cosine) waves. Each n, m frame can be represented using $n * m$ cosine waves stacked together using different coefficients. By analyzing the coefficients one can identify the ones that contribute the most towards a signal and ones that carry less information. By taking the top k coefficients, the amout of features are reduced with a limited loss of information [86].

This operation is performed frame by frame, which means, that information in the time domain i.e. changes of frequencies / mels over time i.e frames is not taken into account, even though some timely information is used because of the windowing process. To incorporate more of the dynamics one can use a delta. This takes the difference between two consecutive frames as a new value with added padding at the end. The space between the frames is measure in *orders* i.e. taking two consecutive frames is a first order delta, taking the first and third frame is a second order delta etc. [124]

3.4 Machine Learning Methods

This section presents an overview of the selected ML methods and their functions. This should only be taken as a short introduction as it will only highlight their specialties and no broader context or explanation will be given.

3.4.1 Bag of Tones

This section will present an overview of the Bag of Tones (BoT) approach. The term was introduced by [87] but it is in its essentials an adaption of the well established Bag of Words (BoW) approach used mostly in Natural Language Processing tasks [70]. The main idea behind the BoW approach is that a word embedding space can be created which captures the meaning of a word in a sentence in a vector that can serve as an input for ML methods. The idea behind the BoT approach is the same only that the

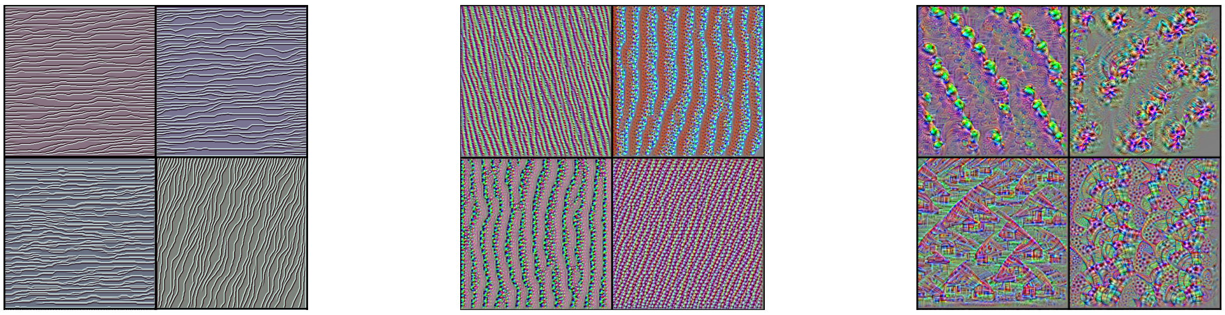

(A)
Low
(B)
Mid
(C)
High

FIGURE 3.6: Illustration of VGG16 low-, mid- and high-level feature extraction

inputs are not sentences but MFCC's. This is done by building a dictionary or a set of *tones* which serves as the embedding space. This space can be arbitrarily complex i.e. one could build an embedding space of 10 features or 10.000.000 features. This embedding space serves as a general representation of all songs. Therefore, each song can be mapped to the embedding by measuring the occurrences. In this case, the mappings for each song can then be used to train a regression model.

3.4.2 Convolutional Neural Networks

Convolutional Neural Network (CNN) is a ML technique that is mostly used in the computer vision domain i.e. for the analysis of images. They can be broken down into two sub parts: The first one are the convolutional operations which serve as a feature extraction technique. The second one are fully connected layers that perform the classification or regression task using the features from the convolutional layers as input. CNN's first appeared around the 1980s [9] but gained broad popularity after it has been shown that they can achieve excellent performances in computer vision challenges [81].

The main advantage of CNN's over traditional feed-forward neural networks is given through the use of filters with shared weights which slide over the input (i.e. image) instead of fully connected neurons. It provides much more manageable and trainable amounts of parameters and it resembles the processing of visual information in humans/animals [36]. The *sliding* operation is done via a matrix multiplication with a chosen filter size. The stride controls the shift for each operation i.e. a stride of 1 means the filter is shifted by 1 pixel after each operation. To capture information at the edges of the input, padding can be used. These convolutional operations are stacked after each other in layers with pooling operations in between. Pooling operations reduce the size of the image or feature map using the same matrix multiplication technique as the convolutional operations. This is done to make the network more robust i.e. prevent overfitting but also to extract features in a higher level of abstraction. Compressing the input will force the network to focus more on the broader context of an image. This means the deeper the network the more fine-grained level of abstractions can be captured by the filters [81]. An illustration can be seen in figure 3.6.

3.4.3 Recurrent Neural Networks

The Recurrent Neural Network (RNN) architecture is quite different from CNN's and more close to a normal neural network structure but with one major difference as they allow loops and therefore allow an analysis of time-dependent data. This is done by keeping information of the previous input while processing new inputs. This previous information is kept in the *state* of an RNN cell. This mimics the way humans analyze time-dependent information such as text [98]. A special version using the RNN architecture is the Long Short-Term Memory (LSTM) [21]. As the name suggests this network is able to analyze short and long term dependencies. This has proven to work very well not only for text analysis but also for speech recognition and music analysis [104], [112].

An illustration of an LSTM cell can be found in figure 3.7. Every input is passed through the upper part of the cell called the *cell gate*. The first interaction seen in the top left is called the *forget gate*. This gate decides if the information passed through the *cell*

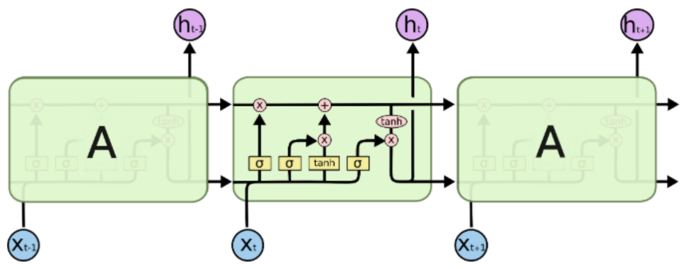


FIGURE 3.7: Illustration of an LSTM cell. Source: [126]

gate at that point in time is kept or not. The output is a continuous number between 0 (don't keep) and 1 (keep). This information is then passed to the *input gate* which updates the old state of the cell with the information from the *forget gate*. The last step is the *output gate*. This transforms the input from the *cell gate* to the output which will be the new *cell state* which can be seen in the top right of the image. But the most important information is kept in the *hidden state* which is carried over to the new cycle of the network. This can be seen at the bottom right of the image. This continuous forward pass of the *cell state* and *hidden state* allows it to keep track of short and long term dependencies [21].

Chapter 4

Dataset and Hardware Setup

4.1 Dataset

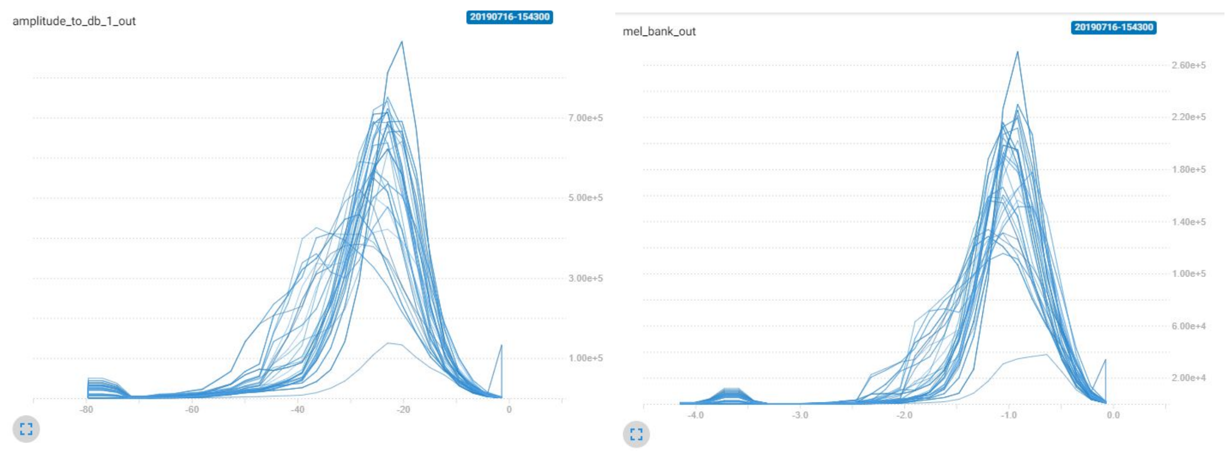
The dataset for the project was provided by UMG. It consists of 53.202 unique songs, taken from different genres. The audio was not provided in raw format (i.e. audio signal) but as magnitude spectrograms. The spectrograms were computed with the following settings: a sampling rate of 22.050Hz with summed stereo channels, a window length of 1024 samples (46ms) a hop size of 512 samples (23ms) a frame length of 256. The outputs for each song were split into 6sec patches with 513bins and 256 frames each with zero-padding added for the last patch. This resulted in 22,35,781 patches and about 1.1 terabytes of data. The dataset was divided into 10 folds each one holding 10% of the data. Each fold contained complete songs i.e. all patches for a given song. Fold 0 to 6 were used as a training set. Fold 7 served as a validation set. Fold 8 and 9 were used as a test set and served as the input for the predictions seen later on. The labels for this project are latent factors. They were learned using an implicit CF dataset from UMG which contained both in house implicit feedback as well as feedback for their music listed on Spotify. However, in contrast to a traditional CF dataset, this one contained playlist and item (songs) relations and not user and item relations. The model outlined in section 3.2.1 was used to learn the latent factors. This resulted in (200, 1) dimensions for the latent factors for each song. UMG also provided some metadata of the songs which contained artist/band names as well as song names.

4.2 Preprocessing

The music data was provided as magnitude spectrograms, but as explained in section 3.3.1 to enhance the individual audio features, it is beneficial to convert the amplitude to Decibel (dB) and apply the mel-scale. This was done using the *Kapre* [109] library which is based on the *Keras* [95] layer architecture and therefore allows On-GPU prepossessing. The *AmplitudeToDB* layer was used to convert the input to dB and apply a log scale and the layer *Filterbank* to apply the mel-scale. The variable *NO_MELS* refers to the number of filter-banks i.e. the number of output features. The default setting was 128 filter-banks. This setup was used as the first two layers in all following models. Looking at the output distribution of these two layers one can confirm they work as intended. See figure 4.1a and 4.1b.

```
model.add(kapre.utils.AmplitudeToDB(input_shape=(513, 256, 1), amin=1e-10,
 ↪  top_db=80.0))
model.add(kapre.filterbank.Filterbank(n_fbs=NO_MELS, trainable_fb=False,
 ↪  sr=22050, init='mel',
                                      fmin=0, fmax=22050 // 2,
                                       ↪  bins_per_octave=12,
                                       ↪  name='mel_bank'))
```

The provided latent factors had very small numerical (absolute) values, in the range of roughly -0.015 to 0.031. Following intuition and suggestions found in [119] and [17] the factors were normalised into a range of -1 to 1 using $x' = \frac{x - \min(x)}{\max(x) - \min(x)}$ with x' being the normalised value. This was a bad idea for two reasons. First, the output of the matrix factorization models tends to be normally distributed as can be seen in figure 4.2. It would have been better to use mean normalisation in that case i.e. $x' =$



(A) Amplitude to Decibel (B) Mel filter-banks

FIGURE 4.1: Output distribution

$\frac{x - average(x)}{\max(x) - \min(x)}$. The second reason is that min-max normalization is very sensitive to outliers. Given a large enough sample size and a normal distribution, outliers tend to occur. This meant most of the information captured by the latent vectors, were lost. This meant that all following models would learn a mean-representation over all songs and therefore produce non-usable output. Looking at figure 4.3a one can see nearly all songs, except a few at the beginning, yield similar latent vectors after the min-max normalization. However, it might not be a good idea to normalize the latent factors at all as the relative magnitude of the different factors is correlated with their importance for recommendation purposes and this information should be kept. Hence, the latent vectors were only scaled around the mean with unit variance. This can be seen in figure 4.3b.

4.3 Hardware requirements and its difficulties

Music data has a lot of licensing implications which are very complicated [121], [122]. That is why UMG required that no data leaves their environment. This meant all experiments were conducted using a Virtual Machine (VM) on the Google Cloud Platform (GCP), Compute Engine (CE). The provided VM had 26 Gigabyte (GB) of memory, a

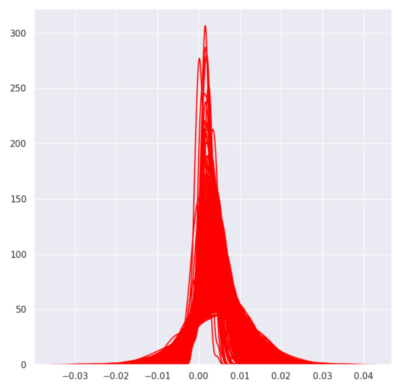
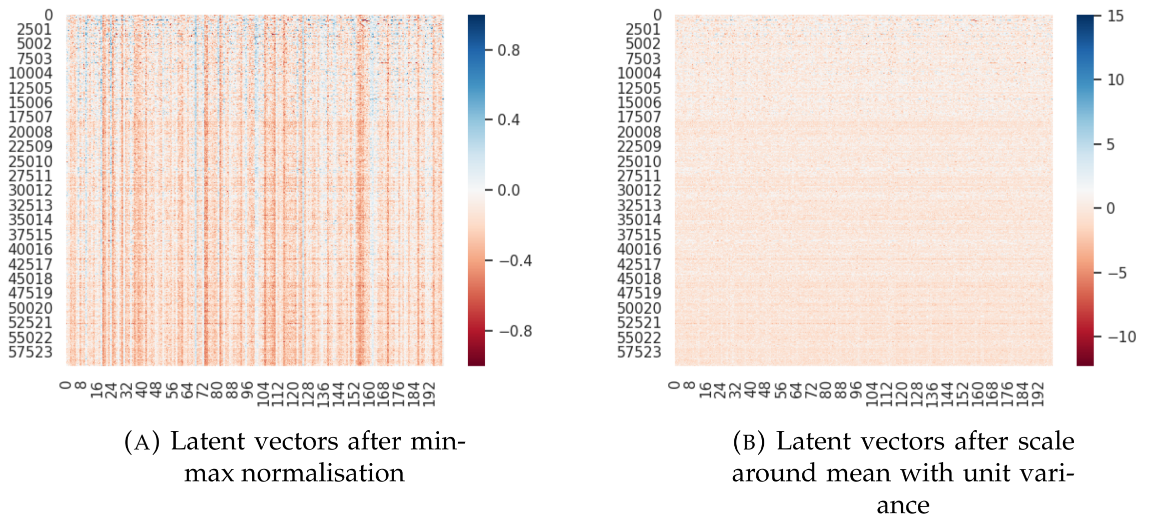


FIGURE 4.2: Distribution of latent factor


(A) Latent vectors after min-max normalisation

(B) Latent vectors after scale around mean with unit variance

FIGURE 4.3: Latent factors preprocessing

4 core CPU based on the Intel Haswell architecture (exact model unknown) and one Nvidia Tesla K80 GPU.

A dataset that spans about 1.130 Gigabyte brings some difficulties with it. The first one being that it does not fit into Memory. GCP provides VM's with 1.4 to 3.8 TB of memory but the smallest variant starts at about £8.000 per month [117]. To make use of the CPU for batch preparation while the GPU is performing training of the model, a custom data generator for train, validation and test data was written [115]. It is based on the Keras sequence object. This resulted in a training time of about 24hours per epoch on 10% of the data. It took about 4 days and a lot of tinkering with the HDF5 library to find that the slow learning was due to I/O bound of the chosen storage option: *Zonal standard persistent disks*. GCP offers faster storage options namely *Local SSD*. However, this option had the unfortunate drawback that the VM could not be shut down. This also meant that adding a second GPU, later on in the project, was not an option anymore. On very simple models with < 1.000.000 parameters and only a few layers the train time was still I/O bound at about 40 minutes per epoch on 10% of training data but this was acceptable compared to the 24h from before. Being I/O bound, at least for simple networks, also means that loading had to be as fast as possible. This meant optimizing every line of code for fast operations. Therefore, a lot of helper functions and files (dictionaries and numpy arrays) were created to allow fast indexing and access to information.

Another problem arises if the 99,99% up-time that GCP guarantees in its Service Level Agreement (SLA) is not enough and the VM dies. This occurred on the 06. August 2019. Because of the chosen storage options recovery was unfortunately not possible which resulted in the loss of all saved models, predictions and metrics/figures. For this reason, the qualitative evaluation had to be changed from experts interviews with musicology experts from UMG to a small recommendation comparison performed by me.

Chapter 5

Methodology

5.1 Bag of Tones model

The BoT model, described in section 3.4.1, should serve as a base-line model to compare the performance and recommendations for the latter discussed models. The results of this model can be found in section 6.1. The Bag of Tones representation can be split into 4 parts:

- **Feature extraction**: As with all models that try to build embeddings i.e. an abstract representation of the input, one needs to extract features. This was done using MFCC's, in detail explained in section 3.3.2. In contrast to the following neural network models this method does not learn a feature extraction but requires a pre-computed one. For this reason, the MFCC with first and second order delta was chosen as this tries to encode more information into lower dimensions but also including time-dependencies. The lower dimensionality with MFCC's compared to mel-bands reduces the computing time for the dictionary creation. This was done using 13 coefficients. With added first and second order delta this resulted in a total of 39 dimensions. Only the training data was used for the dictionary creation.
- **Dictionary**: The dictionary serves as the embedding space and was learned using

the K-Mean algorithms. The dictionary size was set to $k = 4000$ somewhat arbitrarily but [85] has shown good performance using this setting. In a future version, one could use the 'Elbow' method to find a suitable number of clusters. Due to the large training size of 770 Gigabyte and the properties behind the classical K-means algorithm which requires all samples (in memory) to compute distances and update centroids, a mini-batch K-means variant proposed by [69] has been used. The training would stop after the average movement per centroid update was lower than 1×10^{-6} for 10 following batches.

- **Compute song embedding**: To compute the embedding per song, one can simply create a histogram representation of each song by counting how often each centroid has been selected.

- **Regression**: In the last step, one needs to build a mapping between the embeddings per song and the desired label. There are multiple regressions options like linear, lasso or rigid regression, one could also use a decision tree for this problem. Another option would be a Neural Network constructed with dense layers only, which was chosen for this problem. The network consisted of two dense layers with 1024 neurons each and an output layer with 200 dimensions. The exponential linear unit function was used as the activation function for the dense layers and the mean squared error function was used as the objective function. A detailed description of these two functions can be found in the following section 5.2.1.

5.2 Convolutional Neural Networks

This section will outline the methodology behind the use of a convolutional neural network outlined in section 3.4.2. The input for each following model will be the provided data by UMG i.e. powered spectrograms and the first two layers for each model will

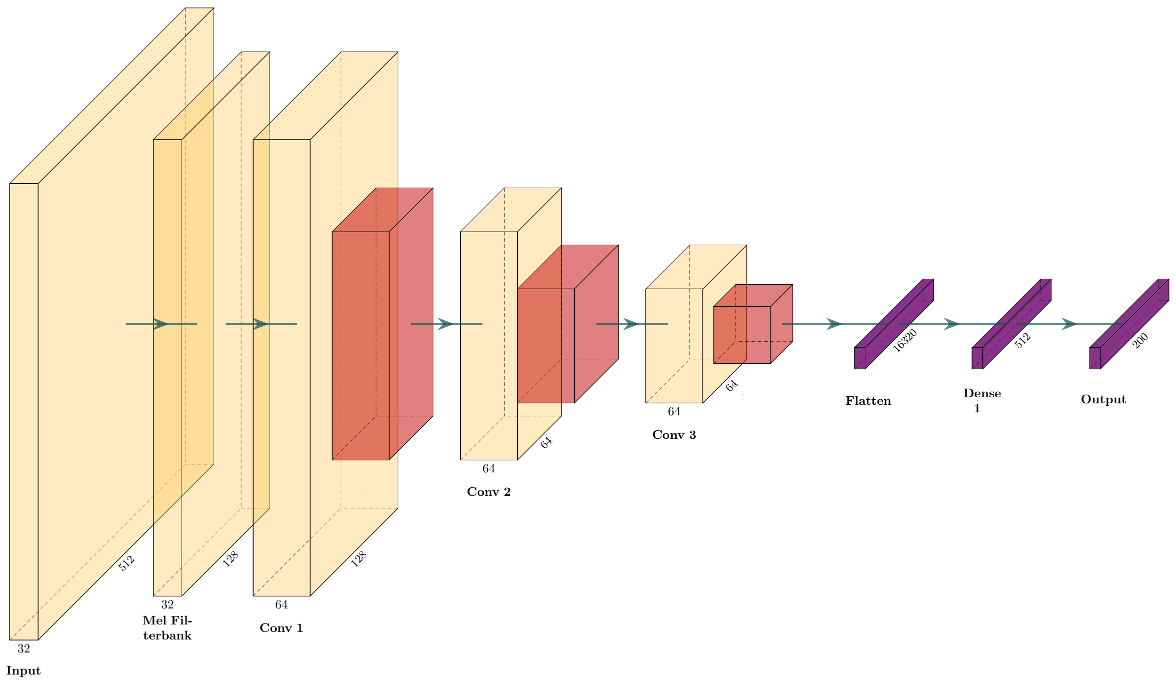


FIGURE 5.1: Basic Convnet

be the *AmplitudeToDB* and *Filterbank* as described in section 4.2. Each following model has been trained using 10% of the training data, 100% of the validation data and 100% of the test data and 5 epochs. Using 10% of the training data was chosen because of the long training due to the difficulties outlined in section 4.3.

5.2.1 Basic Convnet

The first experiment was conducted using a very simple network architecture. An illustration of the network architecture can be seen in figure 5.1. The graphic was created using an adapted version of [118]. As described in section 4.1 the input had the dimensions of 513 by 256. Using a batch size of 32, this resulted in the input dimensions of 32 by 513 by 256. The *Filterbank* layer reduces the number of features from 513 to *NO_MELS* which has been set to 128. This is passed into the first convolutional layer with 64 kernels, each having a size of (3, 3) and a stride of (1, 1) with padding so input and output dimensions are identical. The padding setting was used more because of convenience to keep track of the dimensions throughout the network than of necessity. After each convolutional layer follows a max-pooling operation. This was done to reduce the size of the feature maps which is a computational benefit and might dampen overfitting but also to return an abstract of the features maps (by applying the max operation) and let the following layer focus on these, more abstract, representations. One can also use a convolutional layer with a stride and no padding to reduce the size, this might lead to better results than max-pooling in some cases [92]. The max-pooling filter sizes for the first layer are (2, 2) followed by (2, 4) and lastly again (2,2) resulting in a 16 by 16 output. A three-layer design was chosen with the assumption that the first layer would capture low-level features (like edges), the second one mid-level features and the last layer high-level features. After the feature extraction a flatten operation was performed which turns all (n, m) feature maps into a $(n * m, 1)$ vector, which is followed by a fully connected (dense) layer with 512 neurons. The size of 512 was chosen somewhat arbitrary but with the goal in mind to keep the total number of parameters of the model < 10 million and most of them are located in the dense layers i.e. $16.384 * 512 = 8.3 million$. The output layer has 200 neurons which are identical to the target vectors of (200, 1). All layers, expect from the output, were initialized using the 'HE normal' distribution as it shows superior performance to a random initialization [97]. The output layer was initialized with zeros. Rectified Linear Unit (ReLU) was one of the major reasons deep convolutional networks work so well in the first place as its fast calculation allows converges at reasonable training times while keeping non-linearity [81]. But ReLU has the drawback that dead neurons i.e. gradients of 0 can occur. To avoid dead neurons and improve converges an adaption of ReLU the Exponential Linear Unit (ELU) with a chosen α of 1.0 was used [96].

$$R(x) = \left\{ \begin{array}{cc} x & x > 0 \\ \alpha \cdot (e^x - 1) & x <= 0 \end{array} \right\}$$

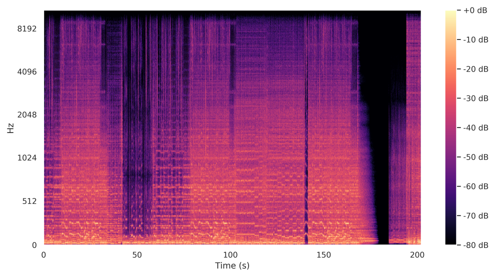
Optimisations were performed using the adam optimiser with an initial learning rate of 1×10^{-4}, beta1 of 0.9, beta2 of 0.999, epsilon of 1×10^{-6} and a decay of 1×10^{-8}. Adam was chosen over stochastic gradient decent as it shows better, faster and more stable converges which is beneficial given the low number of epochs [89], [114]. Mean squared error $\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \hat{Y}_i\right)^2$ was chosen as the objective function as it has been proven well for regression task and aligns with the goals of predicting latent factors.

The results from this model can be seen in section 6 in table 6.2 and the training history in figure 6.1a.

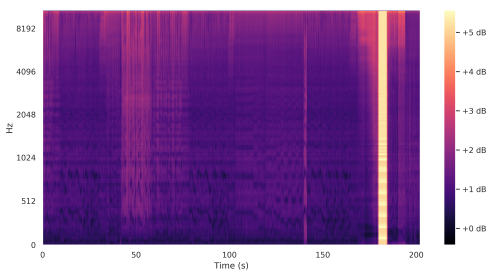
5.2.2 Convnet architecture adjustments

From the previous section, we have seen that the basic covnet model seems to overfit to our training data. One common technique to prevent overfitting is to use more training data. This could be done as currently, only 10% of the training data is in use, but would increase the training time significantly. Another way to reduce overfitting is to introduce dropout which will randomly *deactivate* a percentage of neurons and hence create a more robust network [93]. However, dropout will require significantly more computational resources. A computational less intensive approach is to introduce batch normalization. By reducing the covariance shift between our learned mapping function and the target, the model should be able to generalize better with shifted, but similar in terms of distribution, input data. For this model batch normalization was added after each convolutional operation. The results can be seen in table 6.2 and figure 6.1b

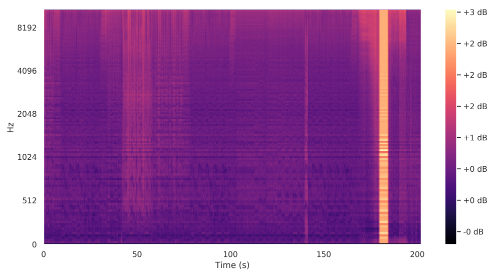
As the experiments have shown, the problem of overfitting is removed by batch normalization. Now one can change the network architecture with the hope of a better optimization i.e. lower loss. One common way would be to add more layers i.e. making the network *deeper*. There are two options, increasing the number of convolutional



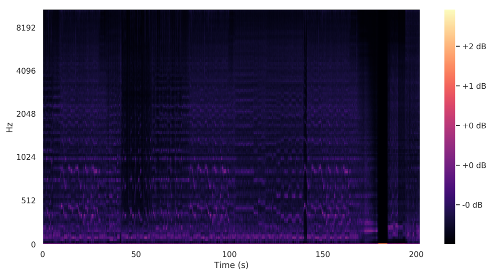
(A) Original Input



(B) Output filter 0



(C) Output filter 1



(D) Output filter 4

FIGURE 5.2: Compare original Input and output activation's after first convolutional operation

operations and therefore making more fine-grained feature representations or increasing the number of dense layers. The latter option will be investigated first. Adding two more dense layers increases the number of parameters by about 500k. The results can be seen in table 6.2. Having more layers the number of neurons in each layer can be adjusted. For the second experiment, the number of neurons in the hidden layers were increased to 2048. This large increase in neuron was chosen to investigate if significant changes occur as a 4-fold increase in the number of neurons will result in a 4+ fold increase in the number of parameters.

This might lead to the assumption that the feature extraction phase i.e. convolutional operations are not able to capture the necessary information that more neurons in the dense layers would have any effect. One can also vary the filter sizes to smaller or larger ones and therefore focus on more fine-grade or coarse representations. However, this experiment was also unsuccessful as it only yields a slight improvement in validation and test loss. All of the experiments so far have shown only small improvements over the base-line model. This seems to indicate that there is some architectural problem in the network. One can try to investigate the problem by visualizing the filters, and maybe more importantly, the filter outputs to get a sense of what the network is learning. Figure A.1 in appendix A shows the visualisation of filter weights. Following a similar analysis as in [105] one can see that the assumption of edge detection holds true in figure A.1a with some filters resembling low or high pass filters. This can be confirmed by looking at the output actions of these filters. This can be seen in figure 5.2. The sample input, seen in figure 5.2a is the instrumental version from the song *Still Falls The Rain* from the J-Pop genre. Analyzing the outputs one sees that filter 0 i.e. figure 5.2b seems to act as a high pass filter while filter 4 i.e. figure 5.2d seem to be the opposite acts like a low pass filter. Notably, filter 1 i.e. figure 5.2c seems to focus on silent parts as it has its highest activation at the time the original song is silent.

Using a filter size of (3, 3) has shown great success in music classification and tagging tasks [80], [100] and the inspection seems to indicate the filters learn some use full feature extractions. This makes sense as the idea of small filters i.e. make use of local correlations, overlaps with the spectrograms which also display local correlations [105]. But [106] has shown that different filter shapes, ones that only slide in the frequency or time domain, can result in better performance. It might be the case that the task of latent factor prediction relays on longer time dependencies or it needs to capture a wider range of frequencies. For this experiment, the 2D convolutional operations were switched out for 1D convolutional operations. The results can be seen in table 6.2.

Given that all of the above experiments have failed or yielded only slight improvements over the baseline, instead of trying more complex adjustments to the model, one can also change the input or target selection for the model. As described in detail in section 4.1 the input spectrograms are only 6 seconds long. Most successfully networks in the field of audio classification seem to use longer input sequences of about 30 seconds or longer [48], [85], [88]. Unfortunately, using a longer input sequence of about 30 seconds is not possible with the provided hardware, given a batch size of 32, as the allocation of memory for the larger inputs per batch yields an out of memory error. A reduction in batch size from 32 to 8 however worked. As one might recall from personal experience, most songs are comprised of an introduction, the main part or body of the song and a conclusion [34] were the body of the song should hold the most amount of information. To extract the main part, one can simply ignore the first *t* second at the beginning and end of a song. According to the analysis of Billboard Top 100 Songs in 2015-2016 by [111] about 87% of songs have an intro and outro length of 30 seconds or shorter. Therefore, two experiments have be conducted: one with a input length of 3 segments (18 seconds) and a skipping of the first 5 and last 5 segments (30 seconds each) with a batch size of 32 and a second with the same skipping but an input length of 10 segments (60 seconds) an a batch size of 8. The results can be seen in table 6.2 as well as the quantitative evaluation in section 6.6.1.

5.3 Recurrent Neural Network

The main idea of RNN is that data is analyzed in the temporal domain while keeping an internal state (i.e. memory) of the previous time steps which yielded impressive performances in audio related tasks such as speech recognition or music generation, for more detail see section 3.4.3.

The reasoning behind using a variation of the RNN i.e. the LSTM architecture for this task is the same as the idea behind a convolutional operation in the time domain. However, the convolutional operations have seen limited success, but this does not mean analyzing the time domain is irrelevant. Convolutional operations in the time domain learn a static representation of filters which will be applied to the input. Their output is not influenced by the output of the previous operation. This is not the case with LSTM's as they keep a time dynamic representation.

The first layers of the network i.e. *AmplitudeToDB* and *Filterbank* haven been kept the same. The previous section has shown that using longer sequences from the main body of the music seems to increases performance. Using longer sequences should, in theory, align with the goals of the LSTM. For faster execution, the CUDA deep neural network (CuDNN) implementation of the LSTM (CuDNNLSTM) has been chosen.

The exact implementation and results from this model can be seen in section 6.3.

5.4 Pre-trained Neural Networks

Given the methodology of CNN's outlined in section 5.2.1 and 5.2.2 and the achieved results outline in section 6.2 one can assume that CNN's are able to learn representative features and a regression function for this task. The train trajectory seen in figure 6.1b shows a non-converged model which is heading to a better optima. This leads to the assumption that longer training time i.e. more epochs, could lead to a better model. However, this was not possible given the constraints outlined in section 4.3. Here one can use pre-trained neural networks.

Using pre-trained networks is one of the main research areas in the domain of transfer-learning. This domain set out to address the idea of transfer knowledge learned in one domain and adapt it to a new domain [91]. As discussed in section 3.4.2 and 5.2.2 different layers of a CNN learn different sets of filters for low-, mid- and high level feature extraction. These filters which serve as feature extraction methods can be adapted into new domains by leveraging their learned weights. It has been shown that this not only works in the domain of image classification [91] but also in the domain of music tagging [94], [108].

Two different pre-trained networks have been selected. The first one being the Xception network trained on the ImageNet dataset which is the current best performing model in top 1 and top 5 classification accuracy for the ImageNet classification challenge. This model can be broken down into three blocks: Entry flow, Middle Flow, Exit Flow. As this model is trained on an image classification task we are not interested in high-level feature maps. Therefore, only the Entry Flow and Middle Flow up to block *block11_sepconv3_bn* or layer 104 have been used [110]. The second model is a much simpler 5 layer CNN model which was trained for music tagging [108]. From this model, all 5 layers were used, as the input data on which the model was trained on, is similar to the one used in this project and therefore low-, mid- and high-level should align.

The domain adaption/network alignment can be done in multiple ways ranging from very simple to very complex. For this project, only two simple approaches have been chosen as they do not require large amounts of training data or computational resources. The first approach utilizes the pre-trained network with fixed weights and only the newly added layers are trained. The second approach uses a very low learning rate to adapt the learned weights for the target domain without losing the information gained from the source domain. The exact implementation of these models and approaches, as well as the results, can be found in section 6.4.

5.5 Triplet Neural Network

Analyzing the results given by the quantitative evaluation outlined in section 6.6.1 so far, one can see that the base-line model outperforms all other models in terms of overlap so far. There might be multiple reasons for this behavior. One might be that the low complexity of the BoT model fits better given the amount of training data. Even though extensive research and experiments into the chosen hyperparameters of the above-discussed models has been conducted, it might be that they are not suitable for the task at hand. It will most probably be a combination of multiple reasons. Given that the model using embeddings (i.e. BoT) presents the best performance so far, the last conducted experiments also involves a model that builds embeddings of the songs before a regression model is trained.

The embeddings were build using a special objective function called **triple loss**. This was first introduced by [103] and the idea is similar to [42]. As the name suggest, the objective function contains three samples which are called anchor, positive (sample) and negative (sample). To create an embedding the distance between anchor and positive should be minimize while the distance between anchor and the negative should be maximized. This can be formulated as:

$$\sum_{i}^{N} \left[\|f\left(x_i^a\right) - f\left(x_i^p\right)\|_2^2 - \|f\left(x_i^a\right) - f\left(x_i^n\right)\|_2^2 + \alpha \right]$$

which can be simplified into:

$$\mathcal{L} = \max(d(a,p) - d(a,n) + \text{margin}, 0)$$

were $a = anchor$, $p = positive$, $n = negative$ and $\alpha = margin$. Given the objective function the sampling of the triplets (a, p, n) is of importance. Choosing triples which fulfill $d(a,p) + margin < d(a,n)$ is called *easy triplets*. However, this might lead to an embedding collapse i.e. a loss of 0 which is not desirable as the network is not incentivized to learn meaningfully embeddings. The second option is to create *hard triplets* which is the complete opposite i.e. $d(a,n) < d(a,p)$ but this is hard to train i.e. requires more computational resources. The last option is called *semi-hard negatives* were the loss should not collapse to zero as the added margin pushes $d(a,p)$ to be larger than $d(a,n)$ i.e. $d(a,p) < d(a,n) < d(a,p) + margin$ [103], [125].

The batches of triplets can be created in two ways, one creates them beforehand and the other during training. Generally, the second option is favorable as it creates *better* triples for each training cycle. However, given the provided hardware outlined in section 4.3 this was not possible for two reasons: the I/O bound slows the triplet selection severely and this method requires fairly large batch sizes (larger than 1.000) which yields out of memory errors. Therefore, the inferior offline selection method has been chosen.

The exact implementation and results from this method can be seen in section 6.5.

5.6 Evaluations

Determining the quality of the provided recommendations in an offline scenario is not easy. As discussed in section 3.2.1, a recommendation system needs to provide novelty, diversity and serendipity. But these factors are difficult to measure in an offline scenario. This section will discuss some common quantitative evaluation methods and which type of qualitative evaluation was performed.

5.6.1 Quantitative evaluation

A relatively simple measurement for the quality of a model, in general, is its objective function. This means one can plot the loss and make assumptions about its quality but it is difficult to make assumptions about recommendations directly from the loss of the model. One way this could be done would be to measure the overlap between recommendations generated from the model and recommendations using the labels as ground truth.

Recommendations can be made based on the similarity of predicted and true latent factors and with the dot product between the two latent vectors i.e. playlist and songs. For similarity one can measure the cosine or euclidean distance, see section 3.2.1, between predictions and labels and find the top k ones with the smallest distance and present them as recommendations. This can be done for each song using the predictions and the labels as input and calculate the overlap between the two generated recommendations. One has to find a reasonable setting for k as a value to large would increase the overlap based on probability i.e. for k = 60.000 the overlap would always be 100%. For this project, a k of 10, 50, 100 and 500 was chosen. A small k might yield an overview on the accuracy of the model predictions and a large k might give some information about the novelty. One has to take into account that predictions can be correct by pure chance. This can be done using $\sum_{i=1}^{k} \frac{\frac{k}{(N-1)^k}}{k}$ with k equal to 10, 50, 100 or 500 and N equal to our number of songs. The number of unique songs in the test set is 11.333. This will results in probabilities of <0.01% for all chosen number of k's and hence can be ignored it this case.

The results for the quantitative evaluation can be found in section 6.6.1 and table 6.6 for similarity based recommendations and in table 6.7 for the dot product.

5.6.2 Qualitative evaluation

Quantitative evaluation is useful to get a quick overview of the performance of the models/recommendations. However, these measurements only capture the ability of the model to predict the latent factors and therefore to mimic the information encapsulated in them. Having a high overlap can assure correct recommendations but a low overlap, in contrast, does not have to necessarily mean recommendations are bad. Using qualitative evaluation could reveal if recommendations outside of the overlap are still valid i.e. if they make *sense* in the context of the recommended songs. This means to evaluate if the recommendations are similar in genre, mood or another aspect. Qualitative evaluation was performed using a human trial. Spotify playlist were created out of the recommendations by querying the recommended ISRC's using the Spotify API through the *Spotipy* [127] library for python. This was only done for the top 10 recommendations as total playtime for each playlist had to be manageable. The songs who are selected as the source of the recommendations for whom the recommendations are made for should be carefully selected to capture some meaning while analyzing the predictions. However, I am not a musicology expert, therefore, my music select might not capture the optimal amount of information. The song *Swan Lake* the 20th composition by *Pyotr Ilyich Tchaikovsky* has been chosen to see whether suitable recommendations for classical music can be provided as they tend to include the challenge of varying the style of music within a song. The second song *Klavier* by the band *Rammstein* has been chosen to check whether the model recommendation might focus on a certain language e.g. German and if the recommendations fall into the chosen genre i.e. rock.

The results can be found in section 6.6.2.

Chapter 6

Experiments and Results

This chapter will give an overview of experiments and results for the methods discussed in detail in chapter 5.

6.1 Bag of Tones

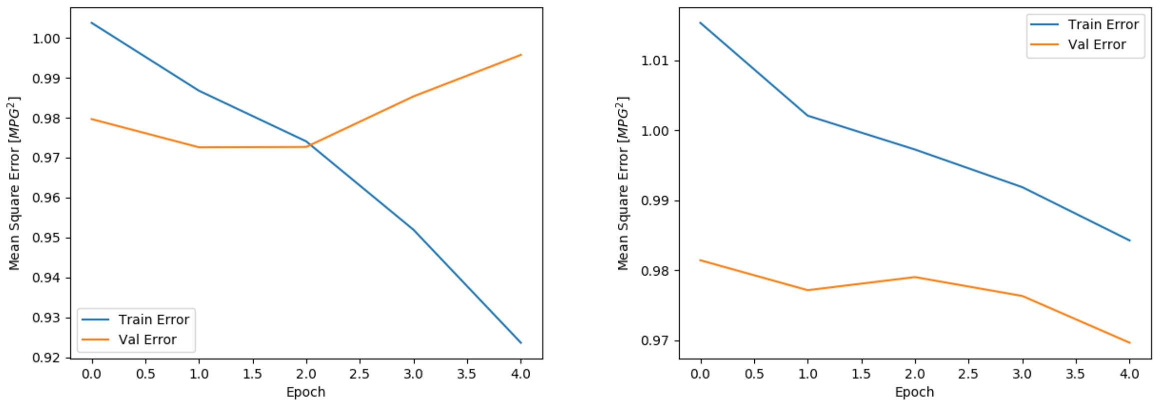
This section will outline the experiments and results using the Bag of Tones representation in detail explained in section 3.4.1 and 5.1 respectively. The results can be seen in table 6.1.

The MFCC features, as well as the first and second order deltas, were computed using the Librosa library [99]. The settings for the intermediate mel-scaled spectrograms are identical to the ones outlined in section 4.2. For the mini-batch k-means, the version implemented in scikit-learn was used [75]. The dictionary was trained using 10% of the training data which took about 72 hours. The regression was performed using a neural network implemented in Keras [95]. The main bottleneck for this model was CPU performance as both the mini-batch k-means well as the MFCC transformation were CPU bound.

The results of the baseline model look promising with decent quantitative evaluation seen in table 6.6 and 6.7.

Name	Parameters (trainable)	Training time	Validation Loss	Test Loss
BoT normal	$6,401,224$	840 min	0.9603	0.9598
BoT three dense, 2048 neurons	$16,996,552$	840 min	0.9599	0.9572

TABLE 6.1: Results table for Bag of Tones model



(A) Basic convnet

(B) with batch normalisation

FIGURE 6.1: Train history

6.2 Convolutional Neural Network

This section will outline the experiments and results using CNN's in detail with the methods discussed in section 5.2.

The results from the basic model outlined in section 5.2.1 can be seen in in table 6.2 and the training history in figure 6.1a. The loss seems to decrease stably over the 5 epochs which is a good sign. However, after the first two epochs, the validation loss is increasing which might be an indication of overfitting. This means the model is *remembering* the training data instead of learning a general representation.

- **CNN with batch normalization**: The train history for the model with batch normalization can be seen in figure 6.1b. The model shows better performance in validation and test loss with no signs of overfitting so far.
- **CNN with batch, 10 epochs**: To check whether the performance can be improved

by longer training time the number of epochs was increased to 10. This, however, only showed small improvements, therefore, other architectural changes were explored.

- **More dense**: Adding more dense layers results in a slight improvement in validation and test loss.
- **More dense and more neurons**: Even though adding more dense layers yielded an improvement, adding more neurons in these dense layers yielded only negligible improvements in validation and test loss.
- **CNN with more conv layers**: Unfortunately, this experiment only yielded a slight improvement in validation and test loss. This might lead to the assumption that the amount of feature extraction achieved by three layers is sufficient.
- **Slide in frequency domain / Slide in time domain**: It does not seem that a slide in the frequency domain or time domain, which should capture more of the serial nature of audio, seems to work in this case as it achieves lower validation and test loss than the basic model.
- **18sec sequence, skip 30sec, batch 32**: This was the first model were the I/O bound has been eliminated and the complexity of the model made it GPU bound. As explained in section 4.3, adding a second GPU was not an option. The combination of input sequences and the focus on the body of the song seem to help as it achieved a lower validation and test loss.
- **60sec sequence, skip 30sec, batch 8**: The increased input size which leads to more parameters in combination with a smaller batch lets the model overfit for or train and validation data. This could be prevented by adding Dropout. However, due to time constraints, this was not explored.

Name	Parameters (trainable)	Training time	Validation Loss	Test Loss
CNN normal	8, 566, 216	290 min	1.0015	0.9998
CNN with batch normalisation	8, 566, 600	294 min	0.9701	0.9823
CNN with batch, 10 epochs	8, 566, 600	2750 min	0.9642	0.9687
More dense	9, 091, 912	293 min	0.9685	0.9792
More dense and more neurons	42, 433, 864	290 min	0.9639	0.9827
More conv layers	36, 349, 640	294 min	0.9701	0.9799
Slide in frequency domain	2, 751, 464	290 min	1.1637	1.1774
Slide in time domain	2, 243, 560	293 min	1.0347	0.9889
18sec input sequence, skip 30sec, batch 32	4, 549, 320	480 min	0.9553	0.9457
60sec input sequence, skip 30sec, batch 8	16, 609, 992	920 min	0.8984	0.9545

TABLE 6.2: Results table for all CNN models

6.3 Recurrent Neural Network

This section will outline the experiments and results using RNN's in detail with the methods discussed in section 3.4.3 and 5.3. The results can be found in table 6.3.

The RNN model consists of two stacked LSTM layer with 64 cells in the first and 128 cells in the second layer. The output of the LSTM layer is flattened and feed into two dense layers with 1024 neurons each. This was done to increases the number of features that are feed into dense layers.

- **RNN normal**: The basic RNN models seem to perform on par with a more complex CNN model.
- **18sec input sequence, skip 30sec, batch 32**: The assumption outlined in section 5.3 that longer input sequences should favor the RNN seems to hold as it improves performance.
- **60sec input sequence, skip 30sec, batch 8**: Using even longer input sequences seem to improve performance even more. Also, the model does not overfit for the train and validation data as it was the case using the CNN with the smaller batch

size. A further interesting experiment would be to feed an entire song into the RNN but, with the provided hardware, this could only be done using a batch size of 1, therefore this experiment was not conducted.

Name	Parameters (trainable)	Training time	Validation Loss	Test Loss
RNN normal	$18,571,850$	290 min	0.9853	0.9823
18sec input sequence, skip 30sec, batch 32	$52,126,282$	520 min	0.9764	0.9747
60sec input sequence, skip 30sec, batch 8	$169,566,794$	1120 min	0.9423	0.9487

TABLE 6.3: Results table for all RNN models

6.4 Pre-trained Neural Networks

This section gives an overview of the exact implementation of the models outlined in section 5.4 as well as the achieved results which can be seen in table 6.4.

- **Xception Network**: This model is provided through the keras applications catalog therefore the can be loaded directly as a model object into keras. To get the output only up to the desired block *block11_sepconv3_bn* can be done using the code snippet below:

```
model_xception_load =
    keras.applications.xception.Xception(include_top=False,
    weights='imagenet', input_shape=(NO_MELS, 256, 3), pooling=None)
model_xception = keras.models.Model(inputs=model_xception_load.input,
    outputs=model_xception_load.get_layer('block11_sepconv3_bn').output)
```

The *Kapre* library, which is used for pre-processing the input for these models, only supports sequential models in contrast to functional models. Therefore, the final network architectures consist of three stacked sequential networks. The first being a model solely for the pre-processing which includes the *AmplitudeToDB* and *Filterbank* layers as well as a convolutional operation with a (1, 1) filter and 3 kernels which were set to be non-trainable. This was done as the Xception model requires 3 input channels (normally R, G, B). This was found to be faster than concatenations. The second one being the Xception network up to the chosen layer. The last one consists of two convolutional layers with the same design as the one outlined in section 5.2.1 which are hoped to learn a high-level representation. They are followed by two dense layers with 1024 neurons each and an output layer using 200 neurons. The models can then be stacked together:

```
keras.models.Model(inputs=model_mel.input,
↪   outputs=[model_reg(model_xception(model_mel.output))])
```

The layers from the imported Xception model can be set to *trainable* = *False* for the first approach. The adaption (second approach) was done by setting the learning rate of the adam optimizer to 1×10^{-8}.

- **Choi music tagging model**: The model and weights are provided in a TensorFlow compatible file format (hdf5). However, the model was built using an older version of Keras and an older version of Kapre which included an error in its normalization algorithm. Therefore, the model was not imported but rebuild using the current Keras version and only the parameters were imported and loaded into the constructed model. The parameters include both the weights of the filters as well as the associated bias. The model included weights for batch normalization which were not used but replaced by a trainable batch normalization layer. The max-pooling shapes were changed to compensate for the different input shape used in this project. The adaption was performed using the same settings as the one used in the Xception model.

The Xcpetion model without adaptation does not show good performance. This would indicate that learned low- and mid-level features using images from the ImageNet

Name	Parameters (trainable)	Training time	Validation Loss	Test Loss
Xception normal	6, 082, 088	290 min	1.0996	1.0987
Xception adaptation	18, 486, 680	560 min	0.9954	0.9930
Choi normal	3, 775, 434	290 min	0.9774	0.9762
Choi adaptation	3, 812, 746	290 min	0.9695	0.9701
Choi long sequence	22, 687, 114	290 min	0.9498	0.9427

TABLE 6.4: Results table for all pre-trained models

dataset differ at lot from the target domain (Spectrograms) and are not easily transferable. Also, it should be noted that the learned weights for the included batch normalization of the model should not make sense in the context of spectrograms and scale the input in an insufficient way. However, there is not an easy way for removing them and the learned weights of the filters depend on the scaled output. Using adaptation i.e. allowing the model to be able to learn with a slow learning rate, slightly improves performance. But this might be more due to a change of all trained weights as using only the network architecture i.e. not included its pre-trained weights, yielded better performance at a test loss of 0.9747. Because of the poor performance of this model the experiment using the longer input sequences was not conducted.

The Choi model without adaption seems to perform on par with one of the more advanced CNN's. However, if we allow the model to learn from our target domain, the performance does not seem to improve. This might suggest that even though the model was trained using mel-spectrogram as input i.e. identical to the input for this project, a domain transfer seems not possible. However, using a longer input sequence which mimics the input of the source domain even further, improves performance. Nonetheless, the main assumption that using pre-trained networks would outperform the CNN or RNN models significantly without requiring longer training time, can be falsified. This might indicate that musical feature extraction is very sensitive and therefore requires detailed engineering of the input for both source and target models.

6.5 Triplet Network

This section will present the implementation and results from the method in detail discussed in section 5.5.

As explained in the above-mentioned section, the offline sampling method for triplet creation has been used. The triplets were created by computing the distance matrix for each latent factor and the positive (sample) was randomly chosen out of 20 samples with the closest distance and the negative (sample) out of the 500 samples with the farthest distance. This was done to incorporate some randomness and variety into the triplet creation. The average distance between the 20 closest and 500 farthest samples is 0.3285, therefore, an α of 0.4 was chosen to create *semi-hard negative* triplets.

For feature extraction the basic CNN model explained in section 5.2.1 was used. The dimension of the embedding space were set to 128. This might seem much lower than the 4.000 of the BoT model but these ones are an embedding space for the entire dataset whereas the 128 are per individual sample. However, the dimensions were chosen somewhat arbitrary but [103] seem to achieve good results with it. The training of this model was performed using 100% of the train-, test- and validation-data and two epochs which took about 96 hours. This resulted in a validation loss of 0.6489 and a test loss of 0.6854. This might indicate that the triple selection worked as intended as the distance did not collapse to 0. The predicted embeddings were then used to train a regression model to predict the latent factors. The settings of the regression model are identical to the ones explained in section 5.1. The reduction of dimension in the input data allowed in-memory processing with short training time per epoch. Therefore, the regression model was trained for 50 epochs.

Name	Parameters (trainable)	Training time	Validation Loss	Test Loss
Triple loss embeddings	$8,408,320$	2315 min	0.6489	0.6854
Triple loss regression	$2,071,240$	120 min	0.7328	0.7459

TABLE 6.5: Results table for triplet model

6.6 Evaluations

6.6.1 Quantitative evaluation

This section will give an overview of the results for quantitative evaluation. A detailed description of the method uses can be found in section 5.6.1. The overlap for different models can be found in table 6.6 for similarity-based recommendations and in table 6.7 for the dot product between the latent vectors.

Analyzing the results one can see that the embeddings models e.g. Bag of Tones and triple network seem to outperform the CNN and RNN models. One of the reasons for this might be due to the two-stage learning process that the chosen embeddings models incorporate. By splitting the feature extraction and regression phases into two separate models, different hyperparameters can be selected for optimal training. In this particular case, the dimensionality reduction through the embedding models allowed much faster training for the regression model. The total percentages of overlap still seem very low with only 7% for a top ten recommendation i.e. on average, less than one song was predicted correctly.

6.6.2 Qualitative evaluation

This section will give an overview of results for qualitative evaluation and should reveal if the low scores from the quantitative evaluation translate to *bad* recommendations or not.

Name	top 10	top 50	top 100	top 500
Bag of Tones	0.3784%	1.5497%	2.6008%	9.4746%
CNN normal	0.0192%	0.1529%	0.3241%	3.5417%
CNN more dense and more neurons	0.0201%	0.1527%	0.3305%	3.5922%
CNN 18sec sequence, skip 30sec	0.9695%	0.3461%	0.6382%	2.8306%
RNN 18sec sequence, skip 30sec	0.2063%	0.4029%	1.0002%	4.0864%
Choi long sequence	0.9702%	0.3496%	0.6356%	2.9107%
Triplet	7.0035%	13.3401%	17.1463%	33.9973%

TABLE 6.6: Results table for quantitative evaluations based on similarity

Name	top 10	top 50	top 100	top 500
Bag of Tones	1.2918%	4.8163%	7.4602%	16.3851%
CNN normal	0.0265%	0.3771%	0.8616%	4.3907%
CNN more dense and more neurons	0.0268%	0.3274%	0.7691%	4.2088%
CNN 18sec sequence, skip 30sec	0.0285%	0.3159%	0.7646%	4.2257%
RNN 18sec sequence, skip 30sec	0.1023%	0.4192%	1.2307%	6.2639%
Choi long sequence	0.0279%	0.3208%	0.7594%	4.2853%
Triplet	48.5346%	53.8995%	55.2048%	52.8310%

TABLE 6.7: Results table for quantitative evaluations based on dot product

The Bag of Tones, CNN normal (with batch normalization) and Triplet models have been chosen. The recommendations playlist can be found in appendix B. Figure B.1 displays an overview of the recommendations for the song *Swan Lake* by *Tchaikovsky* and figure B.2 displays an overview of the recommendations for the song *Klavier* by *Rammstein*.

The reader can also listen to the created playlist through the following links (please note that availability of songs is subject to change and based on licensing):

- ***Swan Lake* by *Tchaikovsky***
  - **True**: Playlist
  - **CNN**: Playlist
  - **BoT**: Playlist

  - **Triplet**: Playlist

- ***Klavier* by *Rammstein***
  - **True**: Playlist
  - **CNN**: Playlist
  - **BoT**: Playlist
  - **Triplet**: Playlist

It is quite apparent that the recommendations provided by the CNN model for both songs are not usable at all. However, it is notable that, even though they are far off from the indented genre, they are relatively coherent i.e. all recommendations for the first song seem to be German schlager whereas for the second it all seems to be electronic dance music. For the first song, the bag of tone models seem to be better than the CNN but far from perfect. Only two out of the ten recommendations are somewhat close to the original song with most of them drifting more into jazz or samba genre. The triplet network performs best with all recommendations being close to the original. I even prefer the recommendations of the triplet network over the true ones. The true ones are deep into the classic genre with most of them slow, silent and not very rhythmic. However, I would categorize the original window of the swan lake composition as *energetic* with pronounced rhythm and bass. This is more reflected in the triplet recommendations. All true recommendations for the second song are from the same band i.e. *Rammstein*. This might be one of the reasons why the result from the quantitative analysis are so low. Here the recommendations from both the BoT and Triplet model are acceptable. Both do not seem to be language-dependent as only one recommended song is in German. I would say that the BoT recommendations are more on the *soft* side of Rock and may even be influenced by Hip-Hop whereas the Triplet recommendations are more *hard* Rock with some leaning towards Heavy Metal.

Chapter 7

Conclusions and future work

This research project aimed to identify possible machine learning techniques that are able to overcome the *cold start* problem from collaborative-based music recommendation. This was done by using the audio content as the source of the recommendation and the learned latent factors from a collaborative-based model as its target to achieve more coherent and accurate recommendations.

Both quantitative and qualitative evaluation have shown that this might indeed be possible. It has long been a challenge for content-based recommendations to capture enough information to make recommendations that reflect one user's preferences and affects towards music. Using the latent factors from a collaborative-model as the target for training seems to help mitigate this problem. Among all the models I have tested, experiments suggests that the Triplet-network provides the most fitting recommendations. Qualitative evaluation has shown that the recommendations are able to capture more nuance than recommendations just based on genre or another category would. However, this is not to say that all user preferences were captured. Therefore, this approach might be the first step to bridge the semantic gap outlined by [53].

The project has also shown that analyzing the quality of recommendations purely based on quantitative analysis is not suitable. An overlap of around 1% or 7% might not sound promising but qualitative evaluation has shown that even 7% overlap recommendations are promising. This, however, should undergo further investigation as this statement relays solely on my own assumption and analysis of the recommendations. Conducting interviews with experts in the field of musicology could provide more insight into the recommendations and identify possible issues. A/B testing the resulting using a broader audience would also help to highlight potential disadvantages. Further research into this area could yield a way of comparable qualitative evaluations which would make comparisons between recommendation methods and achieved *improvements* much more usable.

Followup work could also investigate the potential of a hybrid recommendation approach between the resulting content-based recommendations of this project and pure collaborative-based ones. Similar to the approach by [31] the results from this work could aid a CF model to make final recommendations.

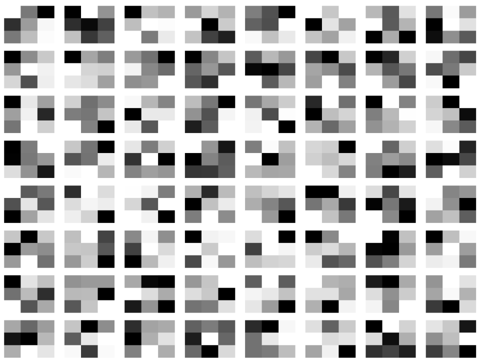
Using a large variety of possible machine learning techniques has also shown that to build a reasonable performing model a two-stage approach seems to work much better than the more favored end-to-end approach. Historically it has been shown that two-stage approaches like bag of words for natural language processing or SIFT feature extraction and classification using SVM or another classifier are outperformed significantly by end-to-end approaches like a CNN or RNN [81], [98]. This might still be true for the task at hand but from a computer engineering standpoint, it has been shown that this is not feasible. Learning musical-embeddings and feeding them into a second regression model which will require only a small portion of the computational resources outperforms end-to-end approaches in training time and result given the constrained of the provided hardware. However, using an end-to-end approach to create the embeddings as proposed by [113] could yield more suitable embeddings than a Triplet-network.

Further to the above, thorough source selection i.e. which patches of a song are taken into account, has to be chosen carefully as it impacts performance. Here more advanced selection methods which are magnitude- or entropy-based could yield further improvements. It has also been shown that domain transfer learning for music recommendation will require further research as out of the box approaches using pre-trained networks do not work.

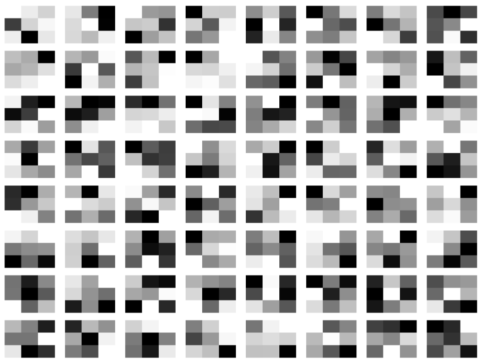
Summing up, this research project has shown that the cold start problem can be circumvented using novel machine learning techniques like Triplet-networks and using latent factors as prediction targets narrows the semantic gap which will yield recommendations that are more suitable to a user's preferences.

Appendix A

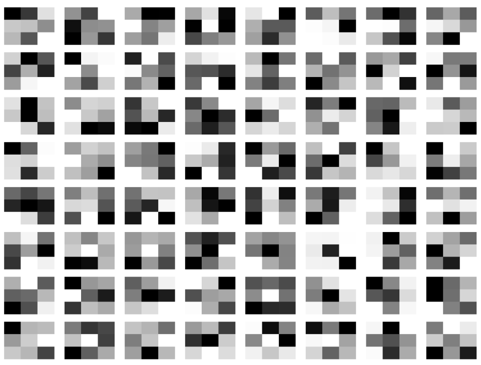
CNN visualisation of filter weights



(A) Layer 1



(B) Layer 2



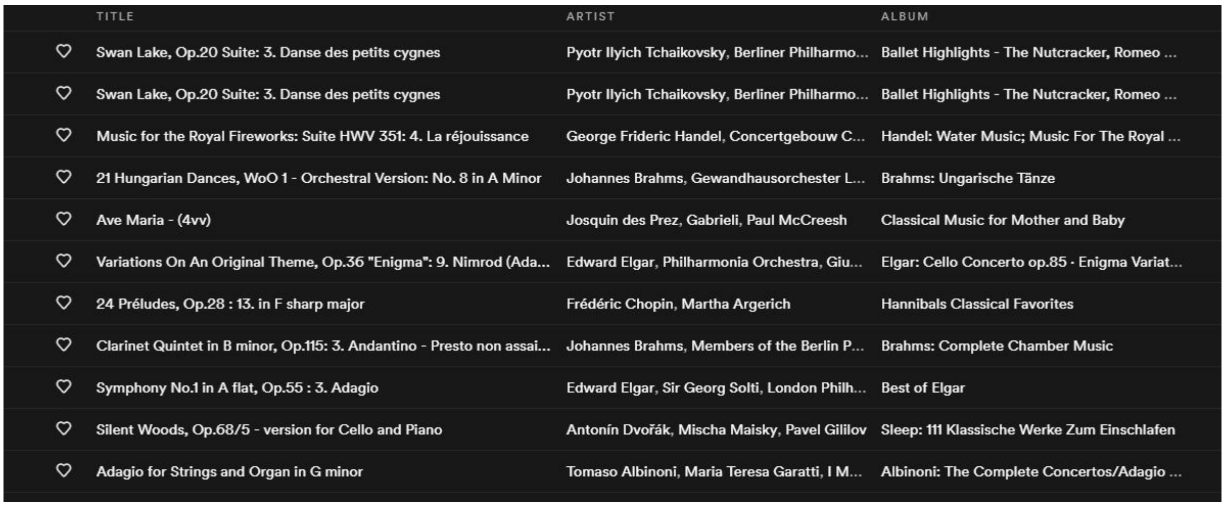
(C) Layer 3

FIGURE A.1: Visualisation of filter weights

Appendix B

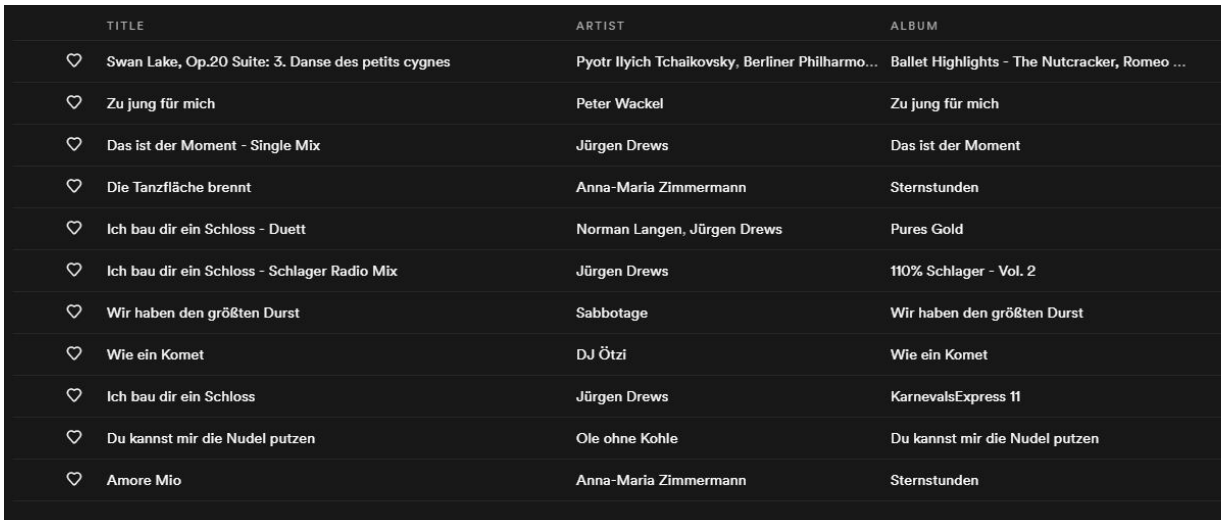
Spotify recommendations

This is an overview of recommendations from the three different models and for the two different selected songs. The playlist *True* indicates the recommendations using the provided latent factors by UMG.



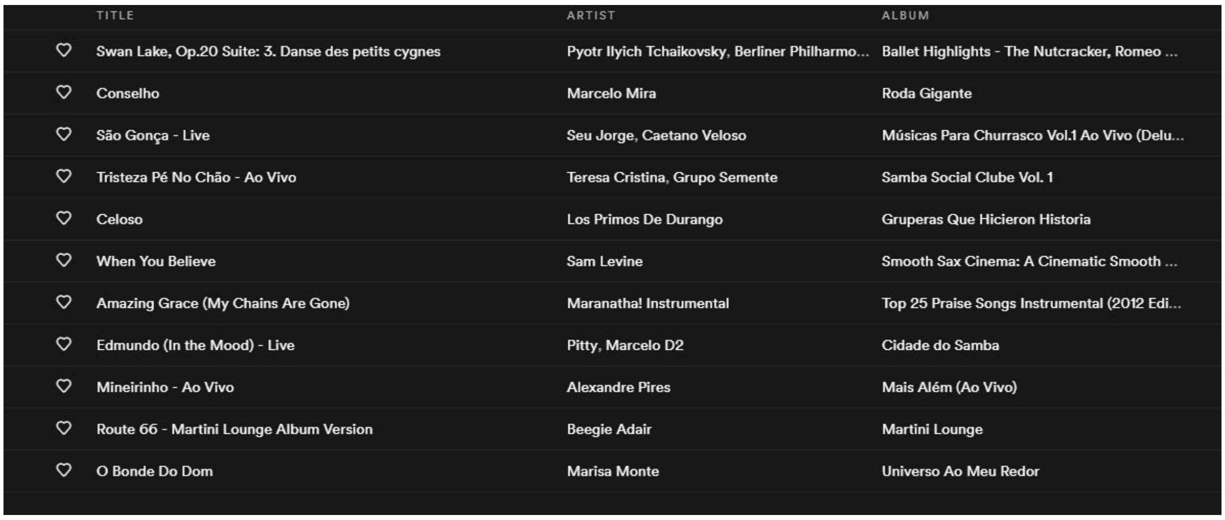
TITLE	ARTIST	ALBUM
Swan Lake, Op.20 Suite: 3. Danse des petits cygnes	Pyotr Ilyich Tchaikovsky, Berliner Philharmo...	Ballet Highlights - The Nutcracker, Romeo ...
Swan Lake, Op.20 Suite: 3. Danse des petits cygnes	Pyotr Ilyich Tchaikovsky, Berliner Philharmo...	Ballet Highlights - The Nutcracker, Romeo ...
Music for the Royal Fireworks: Suite HWV 351: 4. La réjouissance	George Frideric Handel, Concertgebouw C...	Handel: Water Music; Music For The Royal ...
21 Hungarian Dances, WoO 1 - Orchestral Version: No. 8 in A Minor	Johannes Brahms, Gewandhausorchester L...	Brahms: Ungarische Tänze
Ave Maria - (4vv)	Josquin des Prez, Gabrieli, Paul McCreesh	Classical Music for Mother and Baby
Variations On An Original Theme, Op.36 "Enigma": 9. Nimrod (Ada...	Edward Elgar, Philharmonia Orchestra, Giu...	Elgar: Cello Concerto op.85 - Enigma Variat...
24 Préludes, Op.28 : 13. in F sharp major	Frédéric Chopin, Martha Argerich	Hannibals Classical Favorites
Clarinet Quintet in B minor, Op.115: 3. Andantino - Presto non assai...	Johannes Brahms, Members of the Berlin P...	Brahms: Complete Chamber Music
Symphony No.1 in A flat, Op.55 : 3. Adagio	Edward Elgar, Sir Georg Solti, London Philh...	Best of Elgar
Silent Woods, Op.68/5 - version for Cello and Piano	Antonín Dvořák, Mischa Maisky, Pavel Gililov	Sleep: 111 Klassische Werke Zum Einschlafen
Adagio for Strings and Organ in G minor	Tomaso Albinoni, Maria Teresa Garatti, I M...	Albinoni: The Complete Concertos/Adagio ...

(A) True



TITLE	ARTIST	ALBUM
Swan Lake, Op.20 Suite: 3. Danse des petits cygnes	Pyotr Ilyich Tchaikovsky, Berliner Philharmo...	Ballet Highlights - The Nutcracker, Romeo ...
Zu jung für mich	Peter Wackel	Zu jung für mich
Das ist der Moment - Single Mix	Jürgen Drews	Das ist der Moment
Die Tanzfläche brennt	Anna-Maria Zimmermann	Sternstunden
Ich bau dir ein Schloss - Duett	Norman Langen, Jürgen Drews	Pures Gold
Ich bau dir ein Schloss - Schlager Radio Mix	Jürgen Drews	110% Schlager - Vol. 2
Wir haben den größten Durst	Sabbotage	Wir haben den größten Durst
Wie ein Komet	DJ Ötzi	Wie ein Komet
Ich bau dir ein Schloss	Jürgen Drews	KarnevalsExpress 11
Du kannst mir die Nudel putzen	Ole ohne Kohle	Du kannst mir die Nudel putzen
Amore Mio	Anna-Maria Zimmermann	Sternstunden

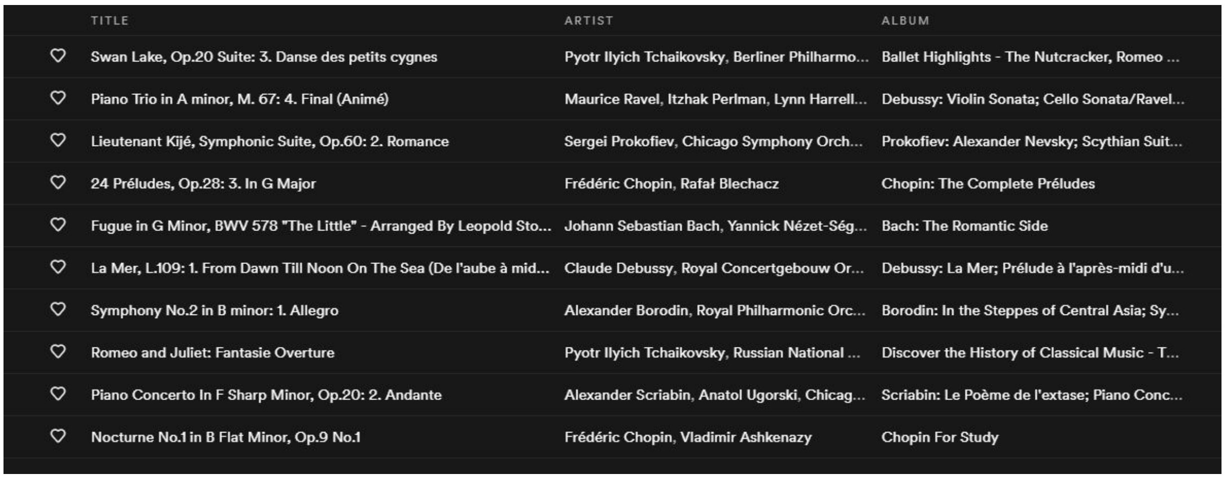
(B) CNN



TITLE	ARTIST	ALBUM
Swan Lake, Op.20 Suite: 3. Danse des petits cygnes	Pyotr Ilyich Tchaikovsky, Berliner Philharmo...	Ballet Highlights - The Nutcracker, Romeo ...
Conselho	Marcelo Mira	Roda Gigante
São Gonça - Live	Seu Jorge, Caetano Veloso	Músicas Para Churrasco Vol.1 Ao Vivo (Delu...
Tristeza Pé No Chão - Ao Vivo	Teresa Cristina, Grupo Semente	Samba Social Clube Vol. 1
Celoso	Los Primos De Durango	Gruperas Que Hicieron Historia
When You Believe	Sam Levine	Smooth Sax Cinema: A Cinematic Smooth ...
Amazing Grace (My Chains Are Gone)	Maranatha! Instrumental	Top 25 Praise Songs Instrumental (2012 Edi...
Edmundo (In the Mood) - Live	Pitty, Marcelo D2	Cidade do Samba
Mineirinho - Ao Vivo	Alexandre Pires	Mais Além (Ao Vivo)
Route 66 - Martini Lounge Album Version	Beegie Adair	Martini Lounge
O Bonde Do Dom	Marisa Monte	Universo Ao Meu Redor

(C) BoT

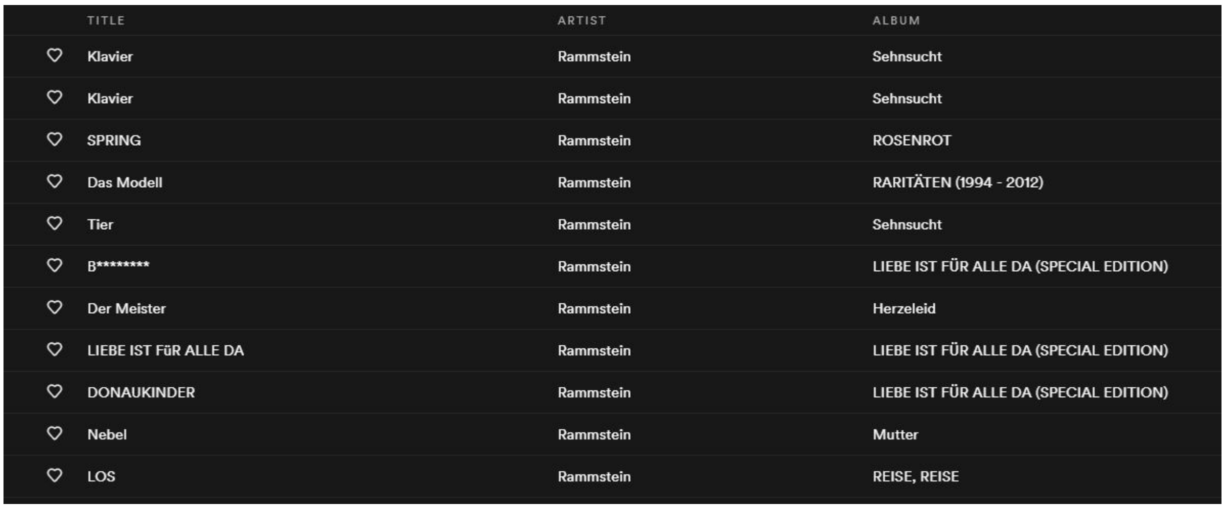
FIGURE B.1: Playlist recommendation for *Swan Lake* by *Tchaikovsky*



TITLE	ARTIST	ALBUM
Swan Lake, Op.20 Suite: 3. Danse des petits cygnes	Pyotr Ilyich Tchaikovsky, Berliner Philharmo...	Ballet Highlights - The Nutcracker, Romeo ...
Piano Trio in A minor, M. 67: 4. Final (Animé)	Maurice Ravel, Itzhak Perlman, Lynn Harrell...	Debussy: Violin Sonata; Cello Sonata/Ravel...
Lieutenant Kijé, Symphonic Suite, Op.60: 2. Romance	Sergei Prokofiev, Chicago Symphony Orch...	Prokofiev: Alexander Nevsky; Scythian Suit...
24 Préludes, Op.28: 3. In G Major	Frédéric Chopin, Rafał Blechacz	Chopin: The Complete Préludes
Fugue in G Minor, BWV 578 "The Little" - Arranged By Leopold Sto...	Johann Sebastian Bach, Yannick Nézet-Ség...	Bach: The Romantic Side
La Mer, L.109: 1. From Dawn Till Noon On The Sea (De l'aube à mid...	Claude Debussy, Royal Concertgebouw Or...	Debussy: La Mer; Prélude à l'après-midi d'u...
Symphony No.2 in B minor: 1. Allegro	Alexander Borodin, Royal Philharmonic Orc...	Borodin: In the Steppes of Central Asia; Sy...
Romeo and Juliet: Fantasie Overture	Pyotr Ilyich Tchaikovsky, Russian National ...	Discover the History of Classical Music - T...
Piano Concerto In F Sharp Minor, Op.20: 2. Andante	Alexander Scriabin, Anatol Ugorski, Chicag...	Scriabin: Le Poème de l'extase; Piano Conc...
Nocturne No.1 in B Flat Minor, Op.9 No.1	Frédéric Chopin, Vladimir Ashkenazy	Chopin For Study

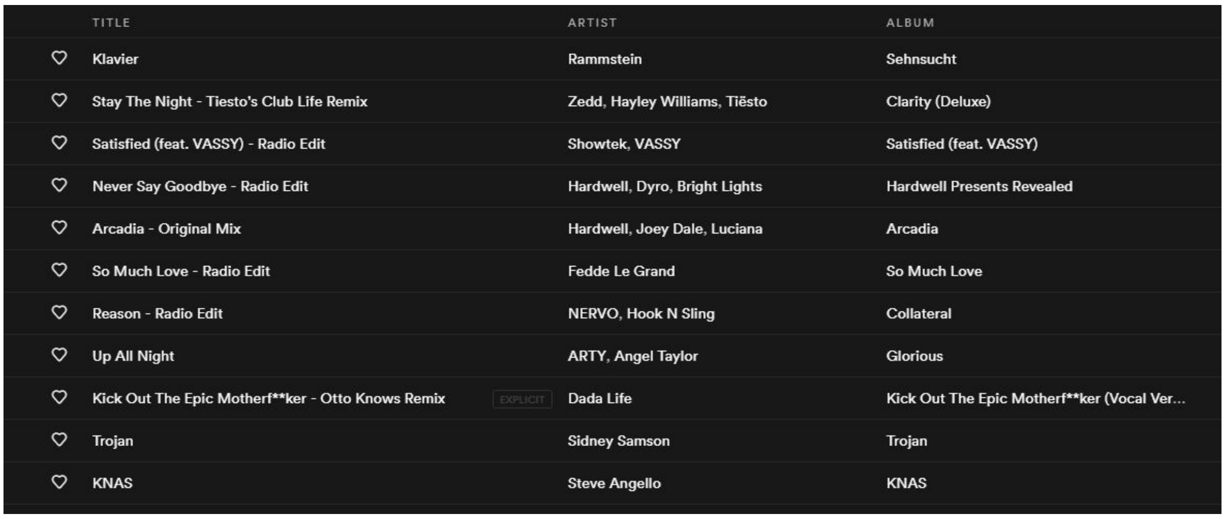
(D) Triplet

FIGURE B.1: Playlist recommendation for *Swan Lake* by *Tchaikovsky*



TITLE	ARTIST	ALBUM
Klavier	Rammstein	Sehnsucht
Klavier	Rammstein	Sehnsucht
SPRING	Rammstein	ROSENROT
Das Modell	Rammstein	RARITÄTEN (1994 - 2012)
Tier	Rammstein	Sehnsucht
B********	Rammstein	LIEBE IST FÜR ALLE DA (SPECIAL EDITION)
Der Meister	Rammstein	Herzeleid
LIEBE IST FüR ALLE DA	Rammstein	LIEBE IST FÜR ALLE DA (SPECIAL EDITION)
DONAUKINDER	Rammstein	LIEBE IST FÜR ALLE DA (SPECIAL EDITION)
Nebel	Rammstein	Mutter
LOS	Rammstein	REISE, REISE

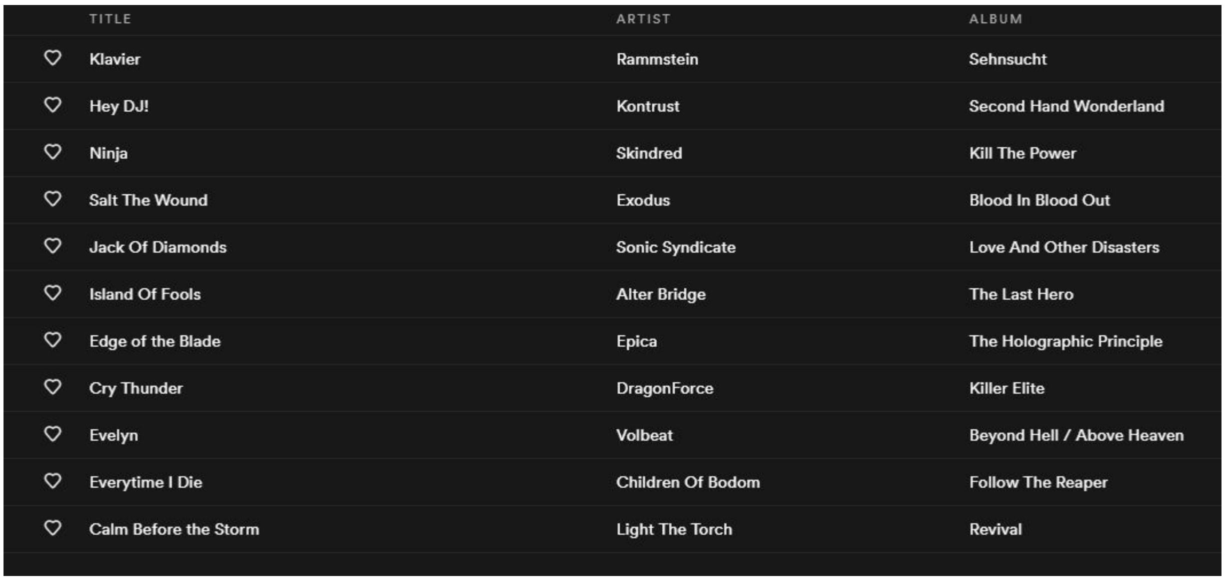
(A) True



TITLE	ARTIST	ALBUM
Klavier	Rammstein	Sehnsucht
Stay The Night - Tiesto's Club Life Remix	Zedd, Hayley Williams, Tiësto	Clarity (Deluxe)
Satisfied (feat. VASSY) - Radio Edit	Showtek, VASSY	Satisfied (feat. VASSY)
Never Say Goodbye - Radio Edit	Hardwell, Dyro, Bright Lights	Hardwell Presents Revealed
Arcadia - Original Mix	Hardwell, Joey Dale, Luciana	Arcadia
So Much Love - Radio Edit	Fedde Le Grand	So Much Love
Reason - Radio Edit	NERVO, Hook N Sling	Collateral
Up All Night	ARTY, Angel Taylor	Glorious
Kick Out The Epic Motherf**ker - Otto Knows Remix EXPLICIT	Dada Life	Kick Out The Epic Motherf**ker (Vocal Ver...
Trojan	Sidney Samson	Trojan
KNAS	Steve Angello	KNAS

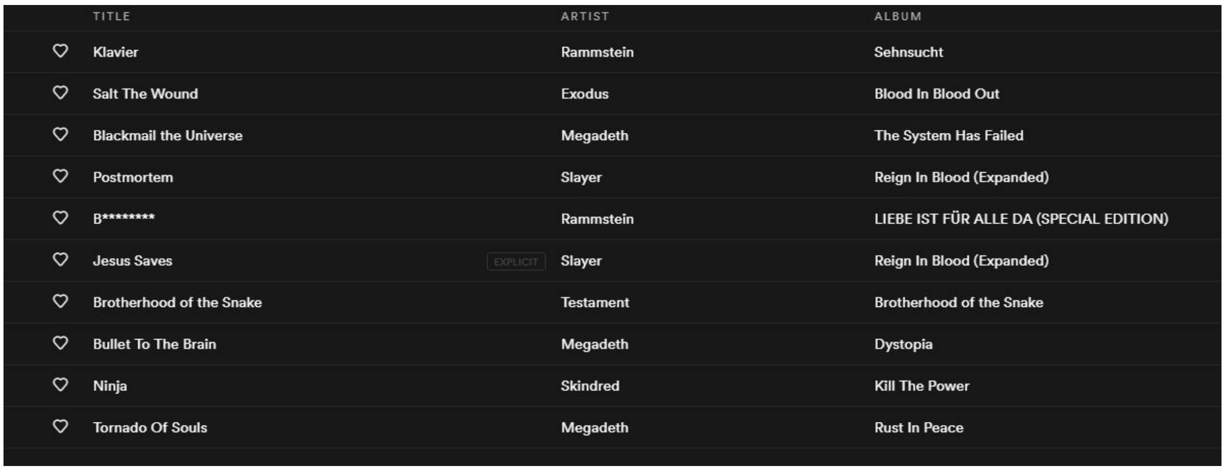
(B) CNN

FIGURE B.2: Playlist recommendation for *Klavier* by *Rammstein*



TITLE	ARTIST	ALBUM
Klavier	Rammstein	Sehnsucht
Hey DJ!	Kontrust	Second Hand Wonderland
Ninja	Skindred	Kill The Power
Salt The Wound	Exodus	Blood In Blood Out
Jack Of Diamonds	Sonic Syndicate	Love And Other Disasters
Island Of Fools	Alter Bridge	The Last Hero
Edge of the Blade	Epica	The Holographic Principle
Cry Thunder	DragonForce	Killer Elite
Evelyn	Volbeat	Beyond Hell / Above Heaven
Everytime I Die	Children Of Bodom	Follow The Reaper
Calm Before the Storm	Light The Torch	Revival

(C) BoT



TITLE		ARTIST	ALBUM
Klavier		Rammstein	Sehnsucht
Salt The Wound		Exodus	Blood In Blood Out
Blackmail the Universe		Megadeth	The System Has Failed
Postmortem		Slayer	Reign In Blood (Expanded)
B********		Rammstein	LIEBE IST FÜR ALLE DA (SPECIAL EDITION)
Jesus Saves	EXPLICIT	Slayer	Reign In Blood (Expanded)
Brotherhood of the Snake		Testament	Brotherhood of the Snake
Bullet To The Brain		Megadeth	Dystopia
Ninja		Skindred	Kill The Power
Tornado Of Souls		Megadeth	Rust In Peace

(D) Triplet

FIGURE B.2: Playlist recommendation for *Klavier* by *Rammstein*

Bibliography

[1] SS Stevens, J Volkmann, and EB Newman, “A scale for the measurement of the psychological magnitude pitch”, *The Journal of the Acoustical Society of America*, vol. 8, no. 3, pp. 185–190, 1937.

[2] SS Stevens, “The measurement of loudness”, *The Journal of the Acoustical Society of America*, vol. 27, no. 5, pp. 815–829, 1955.

[3] AV Oppenheim, “Speech spectrograms using the fast fourier transform”, *IEEE spectrum*, vol. 7, no. 8, pp. 57–62, 1970.

[4] GH Golub and C Reinsch, “Singular value decomposition and least squares solutions”, in *Linear Algebra*, Springer, 1971, pp. 134–151.

[5] N Ahmed, T Natarajan, and KR Rao, “Discrete cosine transform”, *IEEE transactions on Computers*, vol. 100, no. 1, pp. 90–93, 1974.

[6] P Mermelstein, “Distance measures for speech recognition, psychological and instrumental”, *Pattern recognition and artificial intelligence*, vol. 116, pp. 374–388, 1976.

[7] FJ Harris, “On the use of windows for harmonic analysis with the discrete fourier transform”, *Proceedings of the IEEE*, vol. 66, no. 1, pp. 51–83, 1978.

[8] RA Altes, “Detection, estimation, and classification with spectrograms”, *The Journal of the Acoustical Society of America*, vol. 67, no. 4, pp. 1232–1246, 1980.

[9] K Fukushima, “Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position”, *Biological cybernetics*, vol. 36, no. 4, pp. 193–202, 1980.

[10] RN Bracewell and RN Bracewell, *The fourier transform and its applications*. McGraw-Hill New York, 1986, vol. 31999.

[11] S Wold, K Esbensen, and P Geladi, "Principal component analysis", *Chemometrics and intelligent laboratory systems*, vol. 2, no. 1-3, pp. 37–52, 1987.

[12] I Lawrence and K Lin, "A concordance correlation coefficient to evaluate reproducibility", *Biometrics*, pp. 255–268, 1989.

[13] RE Thayer, *The biopsychology of mood and arousal*. Oxford University Press, 1990.

[14] PM Herr, FR Kardes, and J Kim, "Effects of word-of-mouth and product-attribute information on persuasion: An accessibility-diagnosticity perspective", *Journal of consumer research*, vol. 17, no. 4, pp. 454–462, 1991.

[15] D Goldberg, D Nichols, BM Oki, *et al.*, "Using collaborative filtering to weave an information tapestry", *Communications of the ACM*, vol. 35, no. 12, pp. 61–71, 1992.

[16] S Carroll and M Swain, "Explicit and implicit negative feedback: An empirical study of the learning of linguistic generalizations", *Studies in second language acquisition*, vol. 15, no. 3, pp. 357–386, 1993.

[17] CM Bishop *et al.*, *Neural networks for pattern recognition*. Oxford university press, 1995.

[18] E Wold, T Blum, D Keislar, *et al.*, "Content-based classification, search, and retrieval of audio", *IEEE multimedia*, vol. 3, no. 3, pp. 27–36, 1996.

[19] J Foote, "A similarity measure for automatic audio classification", in *Proc. AAAI 1997 Spring Symposium on Intelligent Integration and Use of Text, Image, Video, and Audio Corpora*, 1997.

[20] JT Foote, "Content-based retrieval of music and audio", in *Multimedia Storage and Archiving Systems II*, International Society for Optics and Photonics, vol. 3229, 1997, pp. 138–147.

[21] S Hochreiter and J Schmidhuber, "Long short-term memory", *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[22] M Claypool, A Gokhale, T Miranda, *et al.*, "Combing content-based and collaborative filters in an online newspaper", 1999.

[23] JB Schafer, J Konstan, and J Riedl, "Recommender systems in e-commerce", in *Proceedings of the 1st ACM conference on Electronic commerce*, ACM, 1999, pp. 158–166.

[24] M Welsh, N Borishov, J Hill, *et al.*, "Querying large collections of music for similarity", Technical report, University of California, Berkeley, CA, Tech. Rep., 1999.

[25] B Logan *et al.*, "Mel frequency cepstral coefficients for music modeling.", in *ISMIR*, vol. 270, 2000, pp. 1–11.

[26] DD Lee and HS Seung, "Algorithms for non-negative matrix factorization", in *Advances in neural information processing systems*, 2001, pp. 556–562.

[27] B Logan and A Salomon, "A music similarity function based on signal analysis.", in *ICME*, 2001, pp. 22–25.

[28] BM Sarwar, G Karypis, JA Konstan, *et al.*, "Item-based collaborative filtering recommendation algorithms.", *Www*, vol. 1, pp. 285–295, 2001.

[29] J-J Aucouturier, F Pachet, *et al.*, "Music similarity measures: What's the use?", in *ISMIR*, 2002, pp. 13–17.

[30] R Burke, "Hybrid recommender systems: Survey and experiments", *User modeling and user-adapted interaction*, vol. 12, no. 4, pp. 331–370, 2002.

[31] P Melville, RJ Mooney, and R Nagarajan, "Content-boosted collaborative filtering for improved recommendations", *Aaai/iaai*, vol. 23, pp. 187–192, 2002.

[32] K Hoashi, K Matsumoto, and N Inoue, "Personalization of user profiles for content-based music retrieval based on relevance feedback", in *Proceedings of the eleventh ACM international conference on Multimedia*, ACM, 2003, pp. 110–119.

[33] G Linden, B Smith, and J York, "Amazon. com recommendations: Item-to-item collaborative filtering", *IEEE Internet computing*, no. 1, pp. 76–80, 2003.

[34] F Pease, T Pease, and R Mattingly, *Jazz composition: Theory and practice*. Berklee Pr Pubns, 2003.

[35] C Xu, NC Maddage, X Shao, *et al.*, “Musical genre classification using support vector machines”, in *2003 IEEE International Conference on Acoustics, Speech, and Signal Processing, 2003. Proceedings.(ICASSP’03).*, IEEE, vol. 5, 2003, pp. V–429.

[36] DH Hubel and TN Wiesel, *Brain and visual perception: The story of a 25-year collaboration*. Oxford University Press, 2004.

[37] T Li and M Ogihara, “Content-based music similarity search and emotion detection”, in *2004 IEEE International Conference on Acoustics, Speech, and Signal Processing*, IEEE, vol. 5, 2004, pp. V–705.

[38] B Logan, “Music recommendation from song sets.”, in *ISMIR*, 2004, pp. 425–428.

[39] F Pachet and J-J Aucouturier, “Improving timbre similarity: How high is the sky”, *Journal of negative results in speech and audio sciences*, vol. 1, no. 1, pp. 1–13, 2004.

[40] P Cano, M Koppenberger, and N Wack, “Content based music audio recommendation”, in *Proceedings of the 13th annual ACM international conference on Multimedia*, ACM, 2005, pp. 211–212.

[41] H-C Chen and AL Chen, “A music recommendation system based on music and user grouping”, *Journal of Intelligent Information Systems*, vol. 24, no. 2-3, pp. 113–132, 2005.

[42] S Chopra, R Hadsell, Y LeCun, *et al.*, “Learning a similarity metric discriminatively, with application to face verification”, in *CVPR (1)*, 2005, pp. 539–546.

[43] MI Mandel and DP Ellis, “Song-level features and support vector machines for music classification”, 2005.

[44] E Pampalk, A Flexer, G Widmer, *et al.*, “Improvements of audio-based music similarity and genre classificaton.”, in *ISMIR*, London, UK, vol. 5, 2005, pp. 634–637.

[45] R Stenzel and T Kamps, “Improving content-based similarity measures by training a collaborative model.”, in *ISMIR*, Citeseer, 2005, pp. 264–271.

[46] F Vignoli and S Pauws, "A music retrieval system based on user driven similarity and its evaluation.", in *ISMIR*, Citeseer, 2005, pp. 272–279.

[47] E Gómez, "Tonal description of polyphonic audio for music content processing", *INFORMS Journal on Computing*, vol. 18, no. 3, pp. 294–304, 2006.

[48] K Yoshii, M Goto, K Komatani, *et al.*, "Hybrid collaborative and content-based music recommendation using probabilistic model with latent user preferences.", in *ISMIR*, vol. 6, 2006, 7th.

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

[50] J Donaldson, "A hybrid social-acoustic recommendation system for popular music", in *Proceedings of the 2007 ACM conference on Recommender systems*, ACM, 2007, pp. 187–190.

[51] DP Ellis, "Classifying music audio with timbral and chroma features", 2007.

[52] D Jennings, *Net, blogs and rock'n'roll: How digital discovery works and what it means for consumers, creators and culture*. Nicholas Brealey Publishing, 2007.

[53] Ò Celma, "Music recommendation and discovery in the long tail", PhD thesis, Universitat Pompeu Fabra, Barcelona, 2008.

[54] Y Hu, Y Koren, and C Volinsky, "Collaborative filtering for implicit feedback datasets", in *2008 Eighth IEEE International Conference on Data Mining*, Ieee, 2008, pp. 263–272.

[55] M Slaney, K Weinberger, and W White, "Learning a metric for music similarity", in *International Symposium on Music Information Retrieval (ISMIR)*, 2008.

[56] Z Chedrawy and SSR Abidi, "A web recommender system for recommending, predicting and personalizing music playlists", in *International Conference on Web Information Systems Engineering*, Springer, 2009, pp. 335–342.

[57] Y Koren, "The bellkor solution to the netflix grand prize", *Netflix prize documentation*, vol. 81, no. 2009, pp. 1–10, 2009.

[58] Y Koren, R Bell, and C Volinsky, "Matrix factorization techniques for recommender systems", *Computer*, no. 8, pp. 30–37, 2009.

[59] C-C Lu and VS Tseng, "A novel method for personalized music recommendation", *Expert Systems with Applications*, vol. 36, no. 6, pp. 10 035–10 044, 2009.

[60] F Maillet, D Eck, G Desjardins, *et al.*, "Steerable playlist generation by learning song similarity from radio station playlists.", in *ISMIR*, 2009, pp. 345–350.

[61] B McFee and GR Lanckriet, "Heterogeneous embedding for subjective artist similarity.", in *ISMIR*, 2009, pp. 513–518.

[62] X Su and TM Khoshgoftaar, "A survey of collaborative filtering techniques", *Advances in artificial intelligence*, vol. 2009, 2009.

[63] J Bu, S Tan, C Chen, *et al.*, "Music recommendation by unified hypergraph: Combining social media information and music content", in *Proceedings of the 18th ACM international conference on Multimedia*, ACM, 2010, pp. 391–400.

[64] J Davidson, B Liebald, J Liu, *et al.*, "The youtube video recommendation system", in *Proceedings of the fourth ACM conference on Recommender systems*, ACM, 2010, pp. 293–296.

[65] M Ge, C Delgado-Battenfeld, and D Jannach, "Beyond accuracy: Evaluating recommender systems by coverage and serendipity", in *Proceedings of the fourth ACM conference on Recommender systems*, ACM, 2010, pp. 257–260.

[66] P Hamel and D Eck, "Learning features from music audio with deep belief networks.", in *ISMIR*, Utrecht, The Netherlands, vol. 10, 2010, pp. 339–344.

[67] G Jawaheer, M Szomszor, and P Kostkova, "Comparison of implicit and explicit feedback from an online music recommendation service", in *Proceedings of the 1st international workshop on information heterogeneity and fusion in recommender systems*, ACM, 2010, pp. 47–51.

[68] M Nakatsuji, Y Fujiwara, A Tanaka, *et al.*, "Classical music for rock fans?: Novel recommendations for expanding user interests", in *Proceedings of the 19th ACM*

international conference on Information and knowledge management, ACM, 2010, pp. 949–958.

[69] D Sculley, "Web-scale k-means clustering", in *Proceedings of the 19th international conference on World wide web*, ACM, 2010, pp. 1177–1178.

[70] Y Zhang, R Jin, and Z-H Zhou, "Understanding bag-of-words model: A statistical framework", *International Journal of Machine Learning and Cybernetics*, vol. 1, no. 1-4, pp. 43–52, 2010.

[71] R Burke, A Felfernig, and MH Göker, "Recommender systems: An overview", *Ai Magazine*, vol. 32, no. 3, pp. 13–18, 2011.

[72] F Cacheda, V Carneiro, D Fernández, *et al.*, "Comparison of collaborative filtering algorithms: Limitations of current techniques and proposals for scalable, high-performance recommender systems", *ACM Transactions on the Web (TWEB)*, vol. 5, no. 1, p. 2, 2011.

[73] M Henaff, K Jarrett, K Kavukcuoglu, *et al.*, "Unsupervised learning of sparse features for scalable audio classification.", in *ISMIR*, vol. 11, 2011, p. 2011.

[74] M Muller, DP Ellis, A Klapuri, *et al.*, "Signal processing for music analysis", *IEEE Journal of Selected Topics in Signal Processing*, vol. 5, no. 6, pp. 1088–1110, 2011.

[75] F Pedregosa, G Varoquaux, A Gramfort, *et al.*, "Scikit-learn: Machine learning in Python", *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[76] F Ricci, L Rokach, and B Shapira, "Introduction to recommender systems handbook", in *Recommender systems handbook*, Springer, 2011, pp. 1–35.

[77] J Schluter and C Osendorfer, "Music similarity estimation with the mean-covariance restricted boltzmann machine", in *2011 10th International Conference on Machine Learning and Applications and Workshops*, IEEE, vol. 2, 2011, pp. 118–123.

[78] EM Schmidt and YE Kim, "Learning emotion-based acoustic features with deep belief networks", in *2011 IEEE workshop on applications of signal processing to audio and acoustics (Waspaa)*, IEEE, 2011, pp. 65–68.

[79] M Slaney, “Web-scale multimedia analysis: Does content matter?”, *IEEE MultiMedia*, vol. 18, no. 2, pp. 12–15, 2011.

[80] EJ Humphrey and JP Bello, “Rethinking automatic chord recognition with convolutional neural networks”, in *2012 11th International Conference on Machine Learning and Applications*, IEEE, vol. 2, 2012, pp. 357–362.

[81] A Krizhevsky, I Sutskever, and GE Hinton, “Imagenet classification with deep convolutional neural networks”, in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[82] B McFee, L Barrington, and G Lanckriet, “Learning content similarity for music recommendation”, *IEEE transactions on audio, speech, and language processing*, vol. 20, no. 8, pp. 2207–2218, 2012.

[83] Y Song, S Dixon, and M Pearce, “A survey of music recommendation systems and future perspectives”, in *9th International Symposium on Computer Music Modeling and Retrieval*, vol. 4, 2012.

[84] D Bogdanov, M Haro, F Fuhrmann, *et al.*, “Semantic audio content-based music recommendation and visualization based on user preference examples”, *Information Processing & Management*, vol. 49, no. 1, pp. 13–33, 2013.

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

[86] M Patil, A Gupta, A Varma, *et al.*, “Audio and speech compression using dct and dwt techniques”, *International Journal of Innovative Research in Science, Engineering and Technology*, vol. 2, no. 5, pp. 1712–1719, 2013.

[87] Z Qin, W Liu, and T Wan, “A bag-of-tones model with mfcc features for musical genre classification”, in *International Conference on Advanced Data Mining and Applications*, Springer, 2013, pp. 564–575.

[88] S Dieleman and B Schrauwen, “End-to-end learning for music audio”, in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2014, pp. 6964–6968.

[89] DP Kingma and J Ba, “Adam: A method for stochastic optimization”, *ArXiv preprint arXiv:1412.6980*, 2014.

[90] B Lika, K Kolomvatsos, and S Hadjiefthymiades, “Facing the cold start problem in recommender systems”, *Expert Systems with Applications*, vol. 41, no. 4, pp. 2065–2073, 2014.

[91] M Oquab, L Bottou, I Laptev, *et al.*, “Learning and transferring mid-level image representations using convolutional neural networks”, in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 1717–1724.

[92] JT Springenberg, A Dosovitskiy, T Brox, *et al.*, “Striving for simplicity: The all convolutional net”, *ArXiv preprint arXiv:1412.6806*, 2014.

[93] N Srivastava, G Hinton, A Krizhevsky, *et al.*, “Dropout: A simple way to prevent neural networks from overfitting”, *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[94] A Van Den Oord, S Dieleman, and B Schrauwen, “Transfer learning by supervised pre-training for audio-based music classification”, in *Conference of the International Society for Music Information Retrieval (ISMIR 2014)*, 2014.

[95] F Chollet *et al.*, *Keras*, `https://keras.io`, 2015.

[96] D-A Clevert, T Unterthiner, and S Hochreiter, “Fast and accurate deep network learning by exponential linear units (elus)”, *ArXiv preprint arXiv:1511.07289*, 2015.

[97] K He, X Zhang, S Ren, *et al.*, “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification”, in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[98] ZC Lipton, J Berkowitz, and C Elkan, “A critical review of recurrent neural networks for sequence learning”, *ArXiv preprint arXiv:1506.00019*, 2015.

[99] B McFee, C Raffel, D Liang, *et al.*, “Librosa: Audio and music signal analysis in python”, in *Proceedings of the 14th python in science conference*, vol. 8, 2015.

[100] T Park and T Lee, “Music-noise segmentation in spectrotemporal domain using convolutional neural networks”, in *16th International Society for Music Information Retrieval Conference (ISMIR)*, 2015.

[101] M Schedl and D Hauger, “Tailoring music recommendations to users by considering diversity, mainstreaminess, and novelty”, in *Proceedings of the 38th international acm sigir conference on research and development in information retrieval*, ACM, 2015, pp. 947–950.

[102] M Schedl, P Knees, B McFee, *et al.*, “Music recommender systems”, in *Recommender systems handbook*, Springer, 2015, pp. 453–492.

[103] F Schroff, D Kalenichenko, and J Philbin, “Facenet: A unified embedding for face recognition and clustering”, in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 815–823.

[104] TO ’Brien, “A recurrent neural network for musical structure processing and expectation”, 2016.

[105] K Choi, G Fazekas, and M Sandler, “Explaining deep convolutional neural networks on music classification”, *ArXiv preprint arXiv:1607.02444*, 2016.

[106] J Pons Puig, T Lidy, and X Serra, “Experimenting with musically motivated convolutional neural networks”, in *14th International Workshop on Content-Based Multimedia Indexing (CBMI); 2016 June 15-17; Bucharest, Romania.[Unknown place]: IEEE, 2016. p. 1-6.*, Institute of Electrical and Electronics Engineers (IEEE), 2016.

[107] D Sánchez-Moreno, ABG González, MDM Vicente, *et al.*, “A collaborative filtering method for music recommendation using playing coefficients for artists and users”, *Expert Systems with Applications*, vol. 66, pp. 234–244, 2016.

[108] K Choi, G Fazekas, M Sandler, *et al.*, “Transfer learning for music classification and regression tasks”, *ArXiv preprint arXiv:1703.09179*, 2017.

[109] K Choi, D Joo, and J Kim, “Kapre: On-gpu audio preprocessing layers for a quick implementation of deep neural network models with keras”, in *Machine Learning for Music Discovery Workshop at 34th International Conference on Machine Learning*, ICML, 2017.

[110] F Chollet, “Xception: Deep learning with depthwise separable convolutions”, in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1251–1258.

[111] DT Tough, “An analysis of common songwriting and production practices in 2014-2015 billboard hot 100 songs”, *MEIEA Journal*, vol. 17, no. 1, p. 79, 2017.

[112] A Ycart, E Benetos, *et al.*, “A study on lstm networks for polyphonic music sequence modelling”, ISMIR, 2017.

[113] J Lee, K Lee, J Park, *et al.*, “Deep content-user embedding model for music recommendation”, *ArXiv preprint arXiv:1807.06786*, 2018.

[114] SJ Reddi, S Kale, and S Kumar, “On the convergence of adam and beyond”, *ArXiv preprint arXiv:1904.09237*, 2019.

[115] *A detailed example of data generators with keras*, `https://stanford.edu/~shervine/blog/keras-how-to-generate-data-on-the-fly`, (Accessed on 07/29/2019).

[116] *Form 6-k spotify technology s.a.* `https://s22.q4cdn.com/540910603/files/doc_financials/quarterly/2019/Financial-Statements-Q1-2019.pdf`, (Accessed on 07/22/2019).

[117] *Google cloud platform pricing calculator*, `https://cloud.google.com/products/calculator`, (Accessed on 07/29/2019).

[118] *Harisiqbal88/plotneuralnet: Latex code for making neural networks diagrams*, `https://github.com/HarisIqbal88/PlotNeuralNet`, (Accessed on 07/30/2019).

[119] *How to use data scaling improve deep learning model stability and performance*, `https://machinelearningmastery.com/how-to-improve-neural-network-stability-and-modeling-performance-with-data-scaling/`, (Accessed on 07/28/2019).

[120] *In ar, gut vs. data isn't a binary choice*, https://www.musicbusinessworldwide.com/in-ar-gut-vs-data-isnt-actually-a-binary-choice/, (Accessed 07/12/2019).

[121] *Music copyright: A quick guide | practical law*, https://uk.practicallaw.thomsonreuters.com/3-532-4069?transitionType=Default&contextData=(sc.Default)&firstPage=true, (Accessed on 07/29/2019).

[122] *Music copyright uk | how to copyright a song & uk copyright law*, https://www.openmicuk.co.uk/advice/music-copyright-uk/, (Accessed on 07/29/2019).

[123] *Nearly 40,000 tracks are now being added to spotify every single day*, https://www.musicbusinessworldwide.com/nearly-40000-tracks-are-now-being-added-to-spotify-every-single-day/, (Accessed 07/12/2019).

[124] *Practical cryptography*, http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/, (Accessed on 08/10/2019).

[125] *Triplet loss and online triplet mining in tensorflow | olivier moindrot blog*, https://omoindrot.github.io/triplet-loss, (Accessed on 08/14/2019).

[126] *Understanding lstm networks – colah's blog*, https://colah.github.io/posts/2015-08-Understanding-LSTMs/, (Accessed on 08/13/2019).

[127] *Welcome to spotipy! — spotipy 2.0 documentation*, https://spotipy.readthedocs.io/en/latest/features, (Accessed on 08/19/2019).