Informatics

Reproduction of Black-Box Music Analysis Algorithms through Machine Learning

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Data Science

eingereicht von

Andreas Schmidt, BSc
Matrikelnummer 01526918

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Associate Prof. Dipl.-Ing. Dr.techn. Peter Knees

Wien, 3. Juli 2023

Andreas Schmidt Peter Knees

Informatics

Reproduction of Black-Box Music Analysis Algorithms through Machine Learning

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Data Science

by

Andreas Schmidt, BSc
Registration Number 01526918

to the Faculty of Informatics

at the TU Wien

Advisor: Associate Prof. Dipl.-Ing. Dr.techn. Peter Knees

Vienna, 3rd July, 2023

Andreas Schmidt	Peter Knees

Erklärung zur Verfassung der Arbeit

Andreas Schmidt, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 3. Juli 2023

Andreas Schmidt

Danksagung

Ich möchte mich an dieser Stelle bei allen Menschen bedanken, die mich bei der Erstellung dieser Diplomarbeit unterstützt haben.

An erster Stelle möchte ich meinem Betreuer, Herrn Associate Prof. Dipl.-Ing. Dr.techn. Peter Knees, danken, der mein Interesse für dieses Thema geweckt hat und mich richtungsweisend während meiner Arbeit begleitet und mit seinen konskruktiven Ratschlägen unterstützt hat.

Mein herzlichster Dank gilt meinen Eltern, die mir mein Studium ermöglicht haben. Vielen Dank für eure Unterstützung, eure Geduld und eure Motivation!

Besonderer Dank gilt außerdem meiner Schwester Viktoria und meiner Freundin Lydia, sowie meinen Freudinnen und Freunden, die mir stets zur Seite standen.

Kurzfassung

Eine Sammlung von semantischen Musik-Features (d.h. Metriken, die bestimmte Merkmale von Musikstücken quantifizieren), die vom Musik-Streaming-Dienst *Spotify* angeboten wird, zeigt wissenschaftliche Relevanz im Forschungsfeld Music Information Retrieval aufgrund der Tatsache, dass sie in mehreren vorangegangenen Forschungsarbeiten eingesetzt wurde. Die Algorithmen, die zur Berechnung dieser semantischen Musik-Features verwendet werden sind nicht veröffentlicht und treten daher als „Black-Box“-Algorithmen auf.

Diese Arbeit untersucht die Eigenschaften dieser Musik-Features und Ansätze zur Nachbildung der Algorithmen, die zu ihrer Berechnung verwendet werden, bzw. Alternativen dazu anhand aktueller Methoden aus dem Bereich Music Information Retrieval. Konkret wurden Algorithmen zur Vorhersage der Features “Danceability”, “Acousticness”, “Instrumentalness”, “Speechiness”, “Liveness”, “Valence”, “Energy” and “Loudness” untersucht. Das Ergebnis dieser Arbeit ist eine Charakterisierung der Eigenschaften der Musik-Features, sowie eine Reihe von Machine-Learning-Modellen, die die Berechnungen der Features reproduzieren, und eine Evaluierung, wie gut die Black-Box-Algorithmen reproduziert werden können.

Abstract

In the research field of Music Information Retrieval a set of semantic music features (i.e. descriptors quantifying certain characteristics of pieces of music) that is offered by music streaming service *Spotify* has gained scientific relevance because of the fact that it has been used in various research works in the past. The algorithms that are originally used for the purpose of calculating these semantic music features are not public and therefore appear as "black-box" algorithms.

This work studies the characteristics of these music features and approaches to reproduce or approximate the algorithms that are originally used for calculating them using state of the art methods from the field of Music Information Retrieval. Specifically, the features "Danceability", "Acousticness", "Instrumentalness", "Speechiness", "Liveness", "Valence", "Energy" and "Loudness" are investigated. The outcome of this work is a characterization of the properties of the music features, as well as a set of machine learning models that are reproducing the algorithms for calculation of the features and an evaluation of how well the black-box algorithms can be reproduced.

Contents

CHAPTER 1

Introduction

Music is an important part of our culture and a loyal companion in many people's everyday lives all over the world. The multimodality of how music can be represented (e.g. via audio signals, symbolic music representations or musical score) and of artifacts that describe characteristics of pieces of music (e.g. song, album or artist information) offer many possibilities for extracting and retrieving useful information from music. However, this also issues the challenge of optimally utilizing these multiple modalities and possibly combining them in order to do so.

The research field of Music Information Retrieval (MIR) is concerned with the extraction of information from music. The field has been evolving since the 1990s, when computers as a medium for music consumption and later music streaming services like Spotify[1], Apple Music[2], Amazon Music[3] or Deezer[4] became popular. These technologies were facilitated by the invention of modern audio compression, which reduces the size of digital music files, allowing them to be broadcasted via the internet. Along with these technologies, research interest in retrieval of information from music data arouse [44]. Research in MIR finds application in tasks such as music recommendation, music annotation and classification or music similarity estimation, all of which are employed by commercial products by music streaming services and online selling platforms. These technologies have become ubiquitious technologies in our everyday's lifes, and considerably impacted music consumption as well as the music market itself.

[1]https://www.spotify.com/
[2]https://music.apple.com/
[3]https://music.amazon.com/
[4]https://www.deezer.com/

1.1 Problem Statement

The music streaming service Spotify[5] publishes a set of music features (i.e. descriptors quantifying certain characteristics of music) for every song available on Spotify. These music feature namely are “Danceability”, “Acousticness”, “Instrumentalness”, “Speechiness”, “Liveness”, “Valence”, “Energy” and “Loudness”. Originally, these features were offered for a database of about 30 million songs by *EchoNest*[6], a service that offered this data via a web API for commercial use. *EchoNest* was founded in 2005 as a spin-off from the Massachusetts Insitute of Technology and acquired in 2014 by Spotify, which has been offering this data under their name since then. This set of music features is relatively easily available via the internet for a large collection of pieces of music and appears to describe certain aspects of music well. For these reasons, these music features have been used in other research works on MIR and music recommender systems in the past, and have therefore gained scientific relevance [51, 33, 36, 18, 32, 3, 35].

However, Spotify’s music features lack a precise definition of what exactly they are intended to represent, how they are computed and which sources of information are used for their computation. As a consequence, the calculations that Spotify makes in order to extract these features appear as a black-box algorithm that might also change over time.

1.2 Aim of the Work

This motivates the aim of this work, which is to analyze the music features published by Spotify and to build machine learning models that are able to predict these music features for the purpose of investigating the reproducibility of these features using state-of-the-art music feature extraction methods, comparing how different feature extraction methods perform and obtaining a set of machine learning models that that can be used to extract these features in a stable version of the features (i.e. a version that does not change over time as Spotify might change their algorithms for producing the features). To this end, the following **research questions** are defined and elaborated in this work:

RQ1 Which prediction performances (measured by Mean Squared Error (MSE) with Spotify’s feature values serving as ground truth) can be obtained with state-of-the-art machine learning algorithms on the task of predicting Spotify’s music features using a test data set that covers a broad range of music styles and genres?

RQ2 Does the choice of different model parameters of state-of-the-art machine learning algorithms have an influence (measured by MSE with Spotify’s feature values serving as ground truth) on the task of predicting Spotify’s music features?

RQ3 Can the performances obtained from elaborating research question RQ1 be improved by using a Multi-Task Learning (MTL) approach?

[5]https://www.spotify.com/

[6]http://modelai.gettysburg.edu/2012/music/docs/EchoNestAnalyzeDocumentation.pdf

In the course of elaborating the research questions stated above an analysis of Spotify's music features will be given, and an investigation and comparison of different feature extraction methods will be conducted.

1.3 Methodology

The methodological approach that is followed in this work in order to answer the research questions stated above follows a series of steps. The structure of this work corresponds to this series of steps.

First, state-of-the-art methods of content-based MIR are studied in order to work out the approaches relevant for reproducing Spotify's black-box algorithms. This is done by conducting literature research. In order to be able to answer research question RQ3 the literature research also comprises a review of relevant approaches and prior research works on the application of MTL in the context of MIR. Chapter 2 introduces the research field of MIR and gives a brief overview of the relevant approaches. In Chapter 3 an introduction to Convolutional Neural Networks (CNNs) in the context of MIR is provided.

In order to understand the characteristics of the music features that are subject of this work, they are analyzed on an appropriate set of pieces of music. The set of data that is considered for this, as well as for the following parts of the work, are based on the *Million Playlists Dataset* [4], consisting of around 1.4 million observations. These data will subsequently also be used for experiments conducted for the purpose of answering research questions RQ1-RQ3. Considering the relatively large size of the data set, its creation necessitates planning and implementation of software for retrieving, preprocessing and storing these data. In Chapter 4 a characterization of the music features and of the data set is given.

With the knowledge obtained by these prior steps, a set of experiments is designed and conducted. To this end, a set of machine learning algorithms and hyperparameters is selected, and machine learning models are trained on the data set mentioned above. With an appropriate evaluation criterion, the quality of the models is evaluated and the results are finally interpreted and discussed. Chapter 5 contains the experiment results and a how these results can be interpreted.

CHAPTER 2

Music Information Retrieval

The research fields of Music Information Retrieval (MIR) is concerned with extracting information from music. Often, MIR tasks aim at the extraction of certain features (i.e. descriptors quantifying certain characteristics) from music. For example, such an MIR task would be the extraction of features quantifying emotions that a human listener would perceive when listening to a song (which is sometimes referred to as *Music Emotion Recognition*) [19].

The data that are used as the basis for Information Retrieval from music are discriminated between contextual data of music (which is referred to as *context-based MIR*) or the music content itself (which is referred to as *content-based MIR*). These two types of data are suited for the extraction of different types of music features and require different methods to be applied to process the data. There has also been research on hybrid MIR approaches that employ both content-based and context-based features. In the following, a brief overview of context-based and content-based MIR is given.

2.1 Context-Based Music Information Retrieval

Context-based MIR is concerned with retrieval of information from contextual data sources of music rather than from the audio of music. This approach to MIR is motivated by the idea that audio signal representations (i.e. music content) might encode only some of the aspects a human listener would associate with a piece of music. In this regard, music content is just one aspect of how music can be characterized and just one source to retrieve music information from. However, there might also be information about a piece of music that is contained in other (contextual) data sources rather than in audio. Depending on the application and the type of features that shall be extracted from music, contextual data sources might be better suited than content-based methods.

Contextual sources that might contain information important for MIR and that have been studied in prior research works are, for example, song metadata like artist information, song or album titles, release dates, genre information, tags or other annotations assigned to a song. Especially in music recommendation context-based sources have shown to be able to yield good performances as for example Schedl et al. (2021) [43] have shown.

Besides song metadata, textual sources can also be used in MIR. These are typically extracted from the World Wide Web, which makes it possible to retrieve them in an automated fashion, e.g. by web crawlers or from the semantic web. Textual sources are, for example, artist biographies, song lyrics, news articles, blog posts or status update messages in social media. To use these unstructured data, methods from classical Information Retrieval are employed [26].

2.2 Content-Based Music Information Retrieval

Since the late 1990s, research efforts have been made in content-based MIR to retrieve music information from a song's audio signal itself.

In general, features extracted from music content can be categorized as *low-level*, *mid-level* or *high-level* music features. The discrimination between these concepts is based on how "close" a feature is to a characterization of music in a way it is also naturally perceived by humans, and respectively how "close" a feature is to characteristics of the data (e.g. the audio signal) it is extracted from [27]. Low-level music features are features that describe a property of the data it is based on rather than a property that would also be perceived by humans. In the case of music features extracted from audio signals of pieces of music, for example, low-level features would measure some characteristics of the audio signal itself, e.g. the zero-crossing rate, which measures the rate at which an audio signal crosses the zero axis. On the other hand, high-level features (also termed *semantic* features) represent semantic characteristics of music, i.e. information that is also naturally perceived by a human listener. This information is generally contained in an audio signal in more complex structure than low-level features and therefore requires more sophisticated methods to model these features from the audio. The example of Music Emotion Recognition mentioned above would be considered a high-level music feature. Sometimes high-level music features are also calculated from low-level features. Typical semantic audio features that are subject of common MIR tasks are categorical features, e.g. classifying a piece of music into music genres or annotations of semantic tags [48]. Mid-level music features range between low- and high-level features and have properties of both low-level features and high-level features.

For example, Foote (1997) [14] presented the approach of calculating similarities between audio signals of pieces of music by transforming an audio signal to obtain a suitable representation and applying a statistical model. The techniques used in such early research were inspired by the scientific field of Speech Recognition, which is concerned with the extraction of text from audio signals containing spoken word. These early approaches mainly addressed retrieval of *low-level* music features. Common low-level music features that were also subject of early research works are characteristics like tempo, loudness, tone color or timbre [17, 44]. More recent research is often concerned with the extraction of *high-level* music features.

There are multiple approaches to processing audio signal in order to extract features representing particular characteristics. State-of-the-art content-based MIR approaches often transform an audio signal into a signal representation in the frequency domain and extract desired features using a complex statistical model, often a Machine Learning model. The experiments conducted in this work that aim at the reproduction of music features also use a similar approach. In Sections 2.2.1-2.2.3 a brief description of this approach is given.

2.2.1 Audio Signal Processing

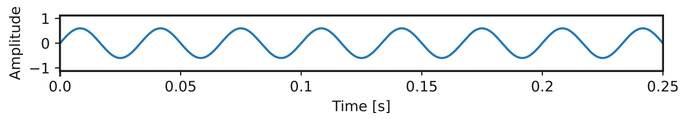
The music representation that is usually started out from in state-of-the-art content-based MIR is assumed as a time-discrete signal representation in form of a progression of samples $x[n] \in [-1, 1]$, with $n = i \cdot T_s$ and $i = 0, 1, 2, \ldots,$ sampled at an (equidistant) sampling rate of $\frac{1}{T_s}$ samples per second from a continuous analog audio signal with typical sampling rates of e.g. 44 100 Hz [44].

Music is usually composed of multiple different sounds of different frequencies and amplitudes, which may interfere with each other. A way of extracting information from music is to derive information about the frequencies and amplitudes of sounds that are present in a piece of music or a part of it.

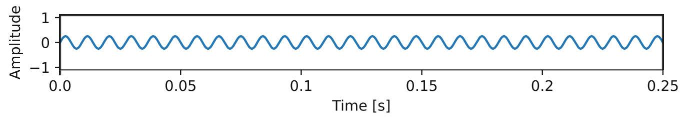
2.2.2 Short-Time Fourier Transform

To this end, a transform referred to as Fourier Transform, which converts an audio signal representation in time-domain into one in frequency-domain, is applied. The idea behind this is that an audio signal can be considered as the sum of multiple sine waves and that the signal can be represented by its frequency components, i.e. by a number of frequencies and their absolute amplitude [37, Chapter 2]. Figure 2.1 shows a signal along with its Fourier Transform.

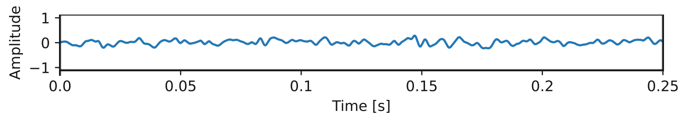
The result of a Fourier Transform represents the amplitudes of the frequencies that are present in the audio signal. Because the frequencies in the signal change over time, applying Fourier Transform on the audio signal of e.g. an entire song would average the frequency information over the entire song, and information of how frequencies change over time would get lost. In order to solve this problem, the audio signal is first split (also called *framed*) into small (possibly overlapping) chunks (called *frames*) of the signal and the amplitude of each occurring frequency for every frame is calculated using Fourier Transform. This is procedure is referred to as Short-Time Fourier-Transformation (STFT). The result of an STFT is then a series of vectors where each element represents a certain frequency and an element's value is the amplitude of the respective frequency in the respective time frame. Such a series of vectors is usually considered a matrix and called a *spectrogram*. Figure 2.2 shows a short piece of an audio signal and its corresponding



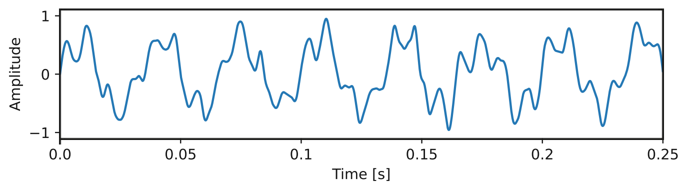
(a) Sine wave with frequency 30Hz and amplitude 0.6



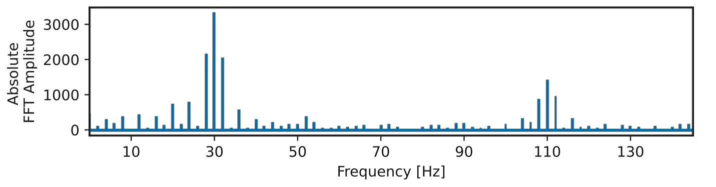
(b) Sine wave with frequency 110Hz and amplitude 0.35



(c) Random noise

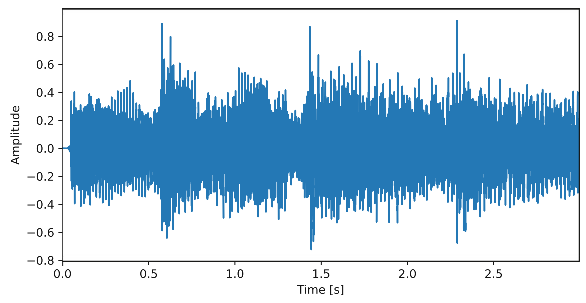


(d) Sum of the signals shown in Figures 2.1a-2.1c

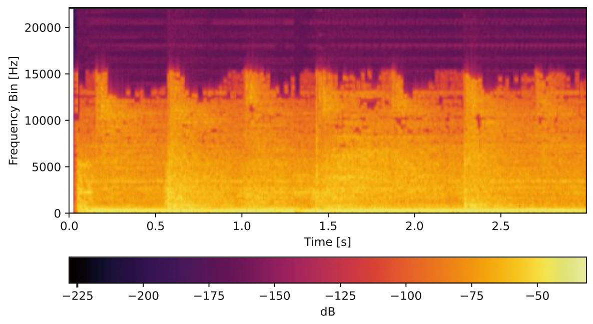


(e) Extract of the Fourier Transform of the signal shown in Figure 2.1d. Because the signal is composed of waves of 30Hz and 100Hz, the Fourier Transform shows maxima at 30Hz and 100Hz. Note that the amplitude around 30Hz is approximately double the amplitude around 110Hz, which corresponds to the amplitudes of the sine waves the signal is composed of. The random noise that is part of the signal is represented by random frequencies having an increased amplitude.

Figure 2.1: Example of a signal composed of multiple sine waves and its Fourier Transform.



(a) Audio Signal



(b) Log-Scaled Spectrogram

Figure 2.2: Audio signal and its corresponding Log-Scaled Spectrogram of a sample of 3 seconds from the song “Yellow Ledbetter” by Pearl Jam (1991).

frequency components over time that are calculated by an STFT. For its calculation, a frame size of 256 samples and a hop size of 128 were used and the resulting amplitudes were scaled with $20 \cdot \log_{10}$.

An element $X(m, k)$ representing the amplitude of frequency k present in the m^{th} time frame of a spectrogram X can be defined as

$$X(m,k) = \sum_{n=0}^{N-1} x(n+mH) \cdot w(n) \cdot e^{-\frac{2\pi ikn}{N}}$$

where N denotes the number of samples of the time frame, x denotes the audio signal (and thus $x(n+mH)$ denotes the n^{th} value of the signal in the time frame) and w denotes a window function [37, Chapter 2].

A spectrogram resulting from an audio signal is dependent on multiple parameters that affect the STFT. These are parameters that characterize the framing of the audio signal and the type of window function that is used.

The framing of the audio signal is characterized by the frame size $N \in \mathbb{N}$ (i.e. the number of samples that make up a frame), the number of samples $H \in \mathbb{N}$ a frame is *shifted* with respect to its previous frame (referred to as *hop size*) and the way samples at the beginning and end of the audio signal are treated (referred to as *padding*). Frame sizes used in audio signal framing typically range from 256 to 8192 samples (which results in frames of approx. 6 ms to 186 ms seconds assuming a sampling rate of 44100 Hz) [27, Chapter 2]. Hop sizes are oftentimes chosen as $H = \frac{N}{2}$. For padding, often "zero-padding" is used, which means that incomplete frames at the beginning and end of the signal are filled with zeros [37, Chapter 2].

The window function $w : [0 : N - 1] \rightarrow \mathbb{R}$ often is a bell-shaped function, like a Hann function or Gaussian function, which has the effect of increasing the effect samples towards the center of a frame have on a spectrogram's vector [37, Chapter 2].

Oftentimes, before extracting features from a spectrogram representation of music, the vectors obtained from STFT are further transformed into *mel scale*, a representation of the frequencies at a logarithmic scale. Spectrograms transformed into mel space are referred to as *mel spectrograms*. This is done because of the fact that humans perceive pitch as the logarithm of the actual frequency. By transforming a spectrogram into mel scale the pitch of a sound is represented the way it is perceived by a human listener. This ultimately aims at representing music in a way that is close to the perception of humans in order to extract high-level music features. The transformation of a frequency spectrogram into a mel spectrogram is done by multiplication of each frame's Fourier Transform with a filter bank consisting of a triangular filter for each resulting mel bin. The transformation of a value f in frequency scale to a value m in mel scale is defined as $m = 2595 \cdot \log_{10}(1 + \frac{f}{700})$. Figure 2.3 shows a mel filter bank with 20 filters mapping frequencies from 0 to 22050Hz onto mel scale.

2.2.3 State of the Art in Content-Based Music Information Retrieval

From spectrogram representations of music, a number of low-level music features can be calculated, such as the *Band Energy Ratio*, which is used in speech/music classification tasks [29], the *Spectral Centroid*, which can be interpreted as the brightness of a sound or *Bandwidth*, which is a measure of the variance of the frequencies of an audio signal

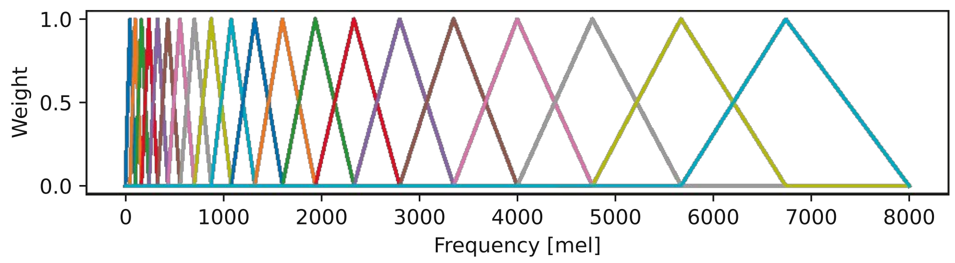


Figure 2.3: Mel filter bank

[27, Chapter 2].

Apart from low-level features that can be directly computed from frequency domain representations of music, the extraction of higher-level music features often requires more sophisticated techniques [24]. One such technique, which has been elaborated in a lot of research works, is to retrieve information from spectrograms of audio signals using machine learning techniques. There is literature showing that good results can be achieved using spectrograms as inputs to machine learning algorithms, such as Support Vector Machines or ensemble learning [9, 38, 34]. For example, Costa et al. (2011) [9] have analyzed the performance of Support Vector Machines to perform genre classifications from sprectrograms of pieces of music, or Nanni et al. (2016) [38] have extracted music genres from sprectrograms using an ensemble method. These approaches either extract low-level music features from the spectral music representation or so-called *Local Binary Patterns*, which represent local texture information of a visual image [11].

Another approach that has shown promising results is to use spectrograms as input to Convolutional Neural Networks (CNNs) [6, 39, 45]. CNNs are suited for processing 2-dimensional structures like spectrograms and exploiting 2-dimensional visual patterns from them. The motivation behind this approach is that the desired information in spectrograms is structured in such 2-dimensional patterns. With this approach, state-of-the-art results in extracting semantic music features could be achieved.

One general idea behind employing machine learning algorithms for MIR from audio spectrograms is that learning music features in this way limits the amount of preprocessing of audio signals that is required with other approaches where music features are hand-crafted [10]. Sometimes however a downside of using machine learning algorithms for MIR might be that usually larger amounts of training data are needed in order to learn music features.

CHAPTER 3

Music Feature Learning using Convolutional Neural Networks

Extracting high-level music features from audio representations of music can be quite a challenging task because of the fact that the desired information is often structured in a complex way. This necessitates appropriate methods for the extraction of such features. One such method that has become particularly popular in recent times is the use of machine learning algorithms to obtain models of such features from audio. Because of the complexity of the information structured in audio, appropriate machine learning algorithms have to have according expressive power.

In consideration of that, Convolutional Neural Networks (CNNs) have proven to be suitable for this task in ongoing research [6, 50]. The experiments conducted in this work that aim at the reproduction of music feature extraction algorithms also employ CNNs for the extraction of semantic music features. This Chapter gives a description of the theory of CNNs.

3.1 Neural Networks

A supervised machine learning model embodies a mathematical function $\hat{f}$ that depends on model parameters. Supervised machine learning describes the task of adapting these model parameters such that $\hat{f}$ approximates a target function $f : X \to Y$.

In general, supervised machine learning is done by an algorithm that takes labeled data (i.e. pairs of (x, y) with $x \in X$ and a label $y \in Y$), and adapts $\hat{f}$ in order to obtain a better approximation of f. The process of repeatedly adapting $\hat{f}$ based on training data is called *training*. The aim of training is to obtain a best-possible approximation $\hat{f}$ of f and to be able to use the resulting model to predict unlabeled data (i.e. to predict y from x).

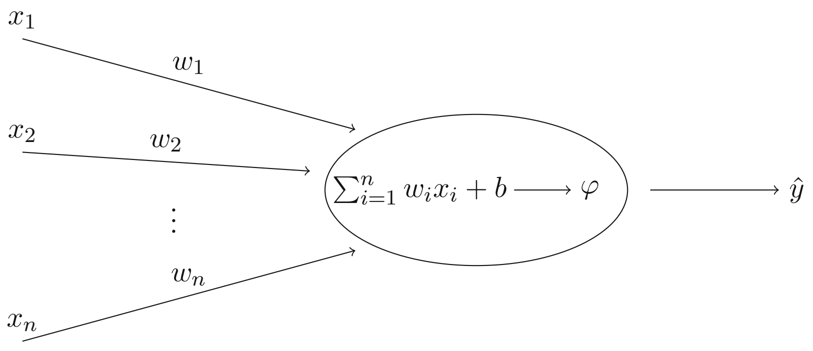


Figure 3.1: Schematic of an artificial neuron.

The approximation $\hat{f}$ can then be used in future to predict unlabeled data (i.e. data points (x, y) where x can be observed, but y is unknown).

The set of pairs (x, y) that are used for training the model are called the *training dataset*, x is called an *input data point*, often of high dimensionality, and y is called *ground truth* or *target* and often a discrete or real value.

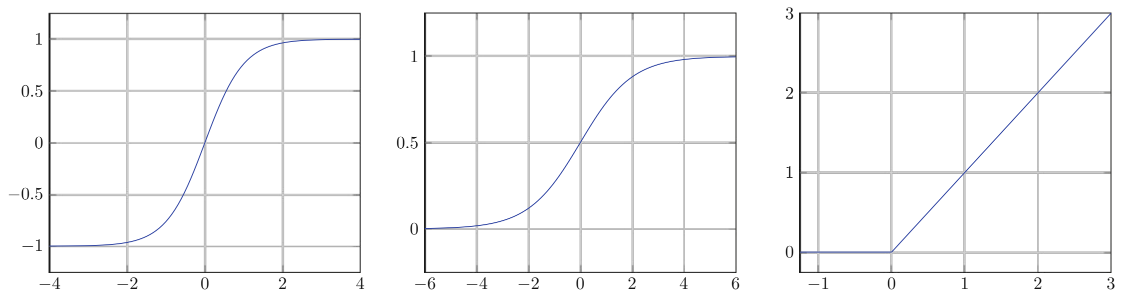
Training is usually done by computing the predicted label $\hat{y} = \hat{f}(x)$ for such a pair of (x, y) and modifying the model parameters in such a way that the error $\varepsilon = y - \hat{y}$ (where operator $-$ denotes an appropriate difference operator for measuring the difference between y and $\hat{y}$) is minimized.

Supervised machine learning is employed in many applications for solving problems in science and is employed in many consumer products in today's world. It plays to its strength in problems where the problem's solution is embodied in a function that cannot be specified explicitly (e.g. because an explicit function definition is not known or finding such a specification is too expensive), but function values are known (or can be observed) for a limited set of instances. Oftentimes, the object of machine learning is just to obtain a better representation of a particular problem instance, which can then be used, for example, in other artificial intelligence tasks. This is referred to as *Representation Learning*.

Neural networks - a type of machine learning models - describe a structure composed of artificial neurons, each of which applies a function to its input data. The result of such a function is either passed to another artificial neuron or is the entire neural network's output. The function parameters of each artificial neuron is adapted during training. In classical neural networks, the output of a neuron with n inputs is given by

$$y = \varphi\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

where $w_1, \ldots, w_n \in \mathbb{R}$ are the neuron's weights, $b \in \mathbb{R}$ is a bias, $x_1, \ldots, x_n$ are the inputs and φ is a function, called the activation function, which can either enhance or simplify the neuron's output.



(a) Sigmoid function (b) tanh function (c) ReLU function

Figure 3.2: Activation functions commonly used in artificial neurons.

The activation function φ is typically a differentiable, non-linear function that is usually the same for all neurons in hidden layers. The choice of φ determines different properties of the neural network and can help to avoid common problems that can occur during training. Commonly used activation functions, depicted in Figure 3.2, are the sigmoid function $\varphi(x) = \frac{1}{1+e^{-x}}$, the hyperbolic tangent function $\varphi(x) = \tanh(x)$ or a rectified linear unit (ReLU) function like $\varphi(x) = \max(0, x)$.

The weights $w_1, \dots, w_n$ and bias b of a neuron are the parameters that are adapted during training. Hence, one neuron on its own can already be interpreted as a machine learning model since it applies a trainable function to calculate the output for an input.

However, because the function embodied by a single neuron is just a relatively simple activation function applied to a linear combination of the input, one neuron alone lacks the power to express complex relationship between input and output data. By joining a number of neurons to form a neural network, relationships of arbitrary complexity can be expressed. There are a variety of topologies used for the arrangement of neurons in a neural network, with different advantages for different machine learning tasks. Many neural network topologies group the neurons in so-called *layers*. In *feedforward* neural networks, neurons are grouped in ordered layers, which group neurons such that the neurons of one layer are not connected with each other (i.e. there is no neuron that takes the output of another neuron of the same or a previous layer as its input) and all neurons of one layer are connected with all neurons of the consecutive layer. Recurrent neural networks additionally consist of special recurrent layers whose neurons are connected to each other (i.e. the output of a neuron is possibly connected to its input or the input of another neuron in the same layer).

If a neural network consists of multiple layers, it is called a *deep* neural network, with the first layer (i.e. the layer whose neurons take the model's input as their input) being referred to as the input layer, the last layer being referred to as the output layer and the other layers being called hidden layers. Figure 3.3 depicts a schematic representation of a deep neural network with two hidden layers.

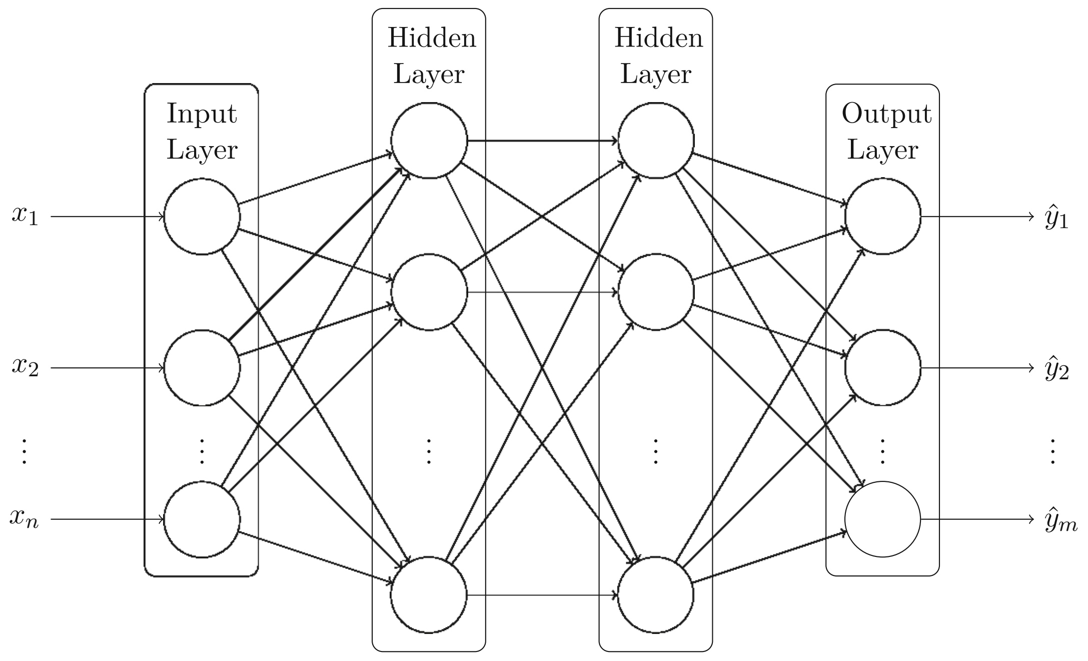


Figure 3.3: Illustration of a Deep Neural network with two hidden layers.

3.2 Optimization Algorithms

In order to calculate a neural network's output for a given input, the input data flows through the network's layers - one after the other - until the final output data exits the network. Information flowing in this way is referred to as *forward propagation* because information flows "forward" from the input layer of the network to its output layer.

As with some other supervised machine learning algorithms (and already described above), when training neural networks, training data points x are repeatedly fed into the network to calculate its output $\hat{y}$. *Loss* (also called *cost* or *error*) $C(y, \hat{y})$ is calculated using a defined loss function C, which measures the similarity of y and $\hat{y}$. The network's parameters Θ are then manipulated based on the loss $C(y, \hat{y})$ so that $C(y, \hat{y})$ would be decreased if the same x were fed into the network again. This way, with every training data point fed into the network and followed by an update of Θ, the model is repeatedly optimized towards minimizing C. This whole procedure is based on the idea that a small value of C would be equal to a predicted output $\hat{y}$ being similar to y, and in case $C(y, \hat{y})$ is small for any $y \in Y$, function $\hat{f}$ (which is represented by the model) would approximate target function f.

An algorithm that implements a method for minimizing the loss is called an *optimization algorithm* (or *optimizer*). Research on neural networks has established a variety of different optimization algorithms with different characteristics that also affect the training results in different ways. Hence, the optimization algorithm can be considered a hyperparameter when using neural networks for solving machine learning tasks.

The way optimizing a network's weights in order to minimize C works is that the gradient of C are calculated with respect to the network's weights for a pair (x, y), and the weights are manipulated in a way that C follows the gradient towards a local minimum of C. The calculation of the gradient of C can be efficiently done using the *Backpropagation Algorithm*, which feeds the output $\hat{y}$ calculated for a training data point x into the network backwards (i.e. from the output layer to the input layer). In the following, the principle of the Backpropagation Algorithm is explained.

Consider a deep neural network and training data consisting of n pairs (x_i, y_i) of input data points x_i and target values y_i with $i = 1, \ldots, n$. Let (l) be a layer of a neural network, with $K^{(l)(i)}$ being the weights of (l) at the time where the network has been trained with the ith training data point (i.e. before training the network, the weights of (l) are denoted by $K^{(l)(0)}$, after learning the first training data point x_1 the weights are updated to $K^{(l)(1)}$, etc.) and $B^{(l)(i)}$ being the bias of (l) respectively.

When training the network, the goal is to update the elements of $K^{(l)(i-1)}$ and $B^{(l)(i-1)}$ so that the loss for the ith training data point is minimized. The updated weights $K^{(l)(i)}$ can be given by

$$K_{q,r}^{(l)(i)} = K_{q,r}^{(l)(i-1)} - \epsilon \cdot \frac{\partial C(y, \hat{y})}{\partial K_{q,r}^{(l)(i-1)}}$$

for an element $K_{q,r}^{(l)(i)}$ of $K^{(l)(i)}$, and the updated bias $B^{(l)(i)}$ can be given by

$$B_q^{(l)(i)} = B_q^{(l)(i-1)} - \epsilon \cdot \frac{\partial C(y, \hat{y})}{\partial B_q^{(l)(i-1)}}$$

for an element $B_q^{(l)(i)}$ of $B^{(l)(i)}$. ϵ denotes the learning rate (see Section 3.3).

As shown above, calculating $K^{(l)(i)}$ and $B^{(l)(i)}$ involve calulating the gradients $\frac{\partial C(y,\hat{y})}{\partial K_{q,r}^{(l)(i-1)}}$ and $\frac{\partial C(y,\hat{y})}{\partial B_q^{(l)(i-1)}}$ of $C(y, \hat{y})$, which can be expressed by the Chain Rule as

$$\frac{\partial C(y, \hat{y})}{\partial K_{q,r}^{(l)(i-1)}} = \frac{\partial C(y, \hat{y})}{\partial z^{(l)}} \frac{\partial z^{(l)}}{\partial K_{q,r}^{(l)(i-1)}}$$

with $z^{(l)} = \sum_m^{r=1} K_{q,r}^{(l)(i-1)} Y_r^{(l-1)} + B_q^{(l)}$. For a neural network with L layers $(1), \ldots, (L)$ the Chain Rule can be applied iteratively to express the network's output y for an input x as an expression of x and the gradients [16, Chapter 6].

3.2.1 Stochastic Gradient Descent

For many machine learning tasks in practice, calculating the gradient of the loss function with respect to the entire training data set involves high computational efforts that are infeasible in many cases. In order to reduce computational effort, the gradient can be calculated with respect to only one training data point at a time and the weights can be updated after every training data point. This approach is known as Stochastic Gradient Descent (SGD), which was first described by Robbins (1951) [41].

The problem with updating the network's weights after training a single data point is that every training data point directly influences the network's weights, which adds noise to the loss function. In contrast, *Gradient Descent* (i.e. calculating the gradient with respect to the entire training data set) usually results in a smoother descent of the loss function.

In order to reduce the noise stemming from SGD and also avoid infeasible computational efforts for Gradient Descent, a preferred approach is to calculate the gradients with respect to small batches of training data and update the weights after each such batch. This is referred to as *Mini-batch* SGD (or often just SGD) [42].

Besides the fact that additional noise is added to the gradients when using SGD as compared to the "true" gradients (i.e. when the entire data set is used for computing the gradients), this noise can also be used to one's advantage. In cases where the model parameters are optimized towards a local minimum of the loss function that is not its global minimum, this noise can cause the parameters to be updated "away" from this local minimum and possibly towards the global minimum. The amount of noise that is added to the gradients is controlled by the learning rate (because the learning rates restricts how much the weights and biases are changed at each update), as well as the batch size (because with increasing batch size the impact and also the noise of a single training data point decreases) [52].

3.2.2 Adam

Adaptive Moment Estimation (Adam), introduced by Kingma and Ba (2014), is an optimization algorithm that is intended to perform a more efficient parameter optimization than SGD [25]. Indeed, empirical experiments have shown that Adam is generally computationally more efficient than SGD and models with Adam tend to learn a target function faster (i.e. with fewer training examples). However, there is research showing that SGD yields better overall prediction performance than Adam because of the fact that it is less prone to overfitting [42, 55].

3.3 Learning Rate

The learning rate is a hyperparameter of neural networks that represents the maximum amount that the weights and biases of the neural network are changed at a time in training in order to minimize the value of the loss function. The learning rate restricts the maximum change in the model parameters in order that the loss function is minimized gradually with additional training samples until it reaches a minimum. This is called *convergence*. On the one hand, the learning rate has to be large enough in order that the parameters are not adapted towards a local (and not a global) minimum of the loss function and in order that the goal of the training (which is to minimize the loss function) is reached within a reasonable amount of training samples. On the other hand, the learning rate has to be small enough so that the model parameters do not "step over" a minimum value of the loss function [40].

Oftentimes, when training neural networks, an approach of decreasing the learning during training is chosen. This way, the learning, and accordingly the changes of the model parameters, are big at the beginning of the training (resulting in a faster decrease of the loss function) and small at the end of the training (resulting in small and precise changes of the model parameters towards the convergence of the model) [40].

3.4 Convolutional Neural Networks

A Convolutional Neural Network (CNN) (sometimes abbreviated *ConvNet*) is a type of deep neural networks that is specialized in processing input data structured as multi-dimensional tensors, in particular visual images. In contrast to ordinary deep neural networks described above, CNN additionally consist of special layers making them especially capable of processing and extracting information that is structured in a similar way visual information is typically structured in images or characteristics of music are structured in a spectrogram representation of a song [16].

Recently, the application of CNNs has become more and more popular in the scientific field of Music Information Retrieval (MIR), especially in content-based MIR, due to the promising results that could be achieved with CNN approaches on extraction of information from music content, which made them the state of the art for many MIR tasks. For example, Korzeniowski and Widmer (2017) proposed CNN-based estimation of musical key from spectrogram representations of songs with state-of-the-art prediction performance [28], Schreiber and Müller (2018) presented a CNN approach for estimation of the tempo of pieces of music also with state-of-the-art performance [46], Holzapfel and Grill (2016) [23] and Durand et al. (2016) [12] both studied CNN-based systems for extraction of different rhythm-related music features, and Schindler et al. (2016) compared different CNN architectures for music tagging [45].

3.4.1 Layers of Convolutional Neural Networks

The architecture of CNNs is composed of a concatenation of *layers*, with each layer (l) performing an operation on the data by taking a tensor $Y^{(l-1)} \in \mathbb{R}^{w^{(l-1)} \times h^{(l-1)} \times c^{(l-1)}}$ as an input and producing a tensor $Y^{(l)} \in \mathbb{R}^{w^{(l)} \times h^{(l)} \times c^{(l)}}$ as an output, which is either the input of the subsequent layer $(l+1)$ or the CNN's final output. Matrix $Y_k^{(l)} \in \mathbb{R}^{w^{(l)} \times h^{(l)}}$ with ${Y_k^{(l)}}_{i,j} = Y_{i,j,k}^{(l)}$, $i \in \{1, \ldots, w^{(l)}\}$, $j \in \{1, \ldots, h^{(l)}\}$, $k \in \{1, \ldots, c^{(l)}\}$ is called the k^{th} *feature map* of $Y^{(l)}$.

Convolutional Layers

A Convolutional Layer consists of a filter kernel, which is a tensor containing the parameters that are tuned during training. It produces the output by performing a convolution on its input using the filter kernel. Because the filter kernel is usually much smaller in the first and second dimension (which are an image's width and height in case the CNN is processing images, or the time and frequency dimension of a spectrogram representation of music) a Convolutional Layer applies the filter kernel on each small area of the input tensor and thus is able to exploit local characteristics of the input [16].

Let l be a Convolutional Layer,
$Y^{(l-1)} \in \mathbb{R}^{w^{(l-1)} \times h^{(l-1)} \times c^{(l-1)}}$ be the input of (l),
$K^{(l)} \in \mathbb{R}^{f^{(l)} \times f^{(l)} \times c^{(l)}}$ be the filter kernel of (l),
$B^{(l)} \in \mathbb{R}^{c^{(l)}}$ called the layer's bias,
and $Y^{(l)} \in \mathbb{R}^{w^{(l)} \times h^{(l)} \times c^{(l)}}$ be the layer's output of (l)
with $w^{(l)} = \begin{cases} \left\lfloor \frac{w^{(l-1)}+2p^{(l)}-f^{(l)}}{s^{(l)}} + 1 \right\rfloor; & s^{(l)} > 0 \\ w^{(l-1)} + 2p^{(l)} - f^{(l)}; & s^{(l)} = 0 \end{cases}$ and $h^{(l)} = \begin{cases} \left\lfloor \frac{h^{(l-1)}+2p^{(l)}-f^{(l)}}{s^{(l)}} + 1 \right\rfloor; & s^{(l)} > 0 \\ h^{(l-1)} + 2p^{(l)} - f^{(l)}; & s^{(l)} = 0 \end{cases}$.

The i^{th} feature map of the Convolutional Layer's output $Y^{(l)}$ is given by

$$Y_i^{(l)} = B_i^{(l)} + \sum_{j=1}^{c^{(l-1)}} K_{i,j}^{(l)} * Y_j^{(l-1)}$$

The operator $*$ denotes a two-dimensional convolution operator, which is defined as

$$(K * Y)_{i,j} = \sum_{m=1}^{w_K} \sum_{n=1}^{h_K} K_{w_K,h_k} * Y_{i-m,j-n}$$

for $Y \in \mathbb{R}^{w_Y \times h_Y}$ and $K \in \mathbb{R}^{w_K \times h_K}$. However, in most CNN implementations $*$ is implemented as the cross-correlation operator

$$(K * Y)_{i,j} = \sum_{m=1}^{w_K} \sum_{n=1}^{h_K} K_{w_K,h_K} * Y_{i+m,j+n}$$

which is similar to the convolution operator, but computationally less expensive. By adapting the layer's kernel weights appropriately (which is done when training the CNN), a Convolutional Layer that performs cross-correlation will give results equal to a Convolutional Layer that actually performs the convolution operation. [16, 7]

Pooling Layers

A Pooling Layer performs a specified aggregation operation on the input tensor that reduces the size of the tensor. A very common aggregation function used in many CNN architectures' Pooling Layers is the maximum function. Pooling Layers depend on the two hyperparameters filter size and stride. The filter size determines the number of tensor elements on which the aggregation function is applied, and the stride determines the number of tensors the filter is shifted to the next part of the input tensor. Pooling Layers do not consist of any parameters that are tuned during training.

The output $Y^{(l)} \in \mathbb{R}^{w^{(l)} \times h^{(l)} \times c^{(l)}}$ of a max Pooling Layer (l) with a filter of size $f^{(l)} \times f^{(l)}$ and stride s applied on input $Y^{(l-1)} \in \mathbb{R}^{w^{(l-1)} \times h^{(l-1)} \times c^{(l-1)}}$ is given by

$$Y_{i,j,k}^{(l)} = \max(Y_{i*s+0,j*s+0,k}^{(l-1)}, \dots, Y_{i*s+f^{(l)},j*s+0,k}^{(l-1)}, \dots, Y_{i*s+0,j*s+f^{(l)},k}^{(l-1)}, \dots, Y_{i*s+f^{(l)},j*s+f^{(l)},k}^{(l-1)})$$

for $i = 1, \dots, w^{(l)}$, $j = 1, \dots, h^{(l)}$ and $k = 1, \dots, c^{(l)}$, with $w^{(l)} \approx \frac{w^{(l-1)}}{s}$, $h^{(l)} \approx \frac{h^{(l-1)}}{s}$ and $c^{(l)} = c^{(l-1)}$.

Pooling Layers are used for reducing the size of a tensor, but also have the effect of making features invariant to certain translations. For example, features of similar but slightly different shapes or positions that are present in the input feature maps would tend to result in the same output of a Pooling Layer. These invariances are often a desired property when extracting features whose exact shapes or positions are not of relevance, but their overall presence are [16].

Fully Connected Layers

A Fully Connected Layer (also called dense Layer or linear Layer) connects every element of the input tensor to every element of the output tensor so that every output tensor element is a linear transformation of the input tensor. Thus, a Fully Connected Layer acts like a layer in an ordinary multilayer neural network.

The output $Y^{(l)} \in \mathbb{R}^{c^{(l)}}$ of a Fully Connected Layer (l) is given by

$$Y_i^{(l)} = \sum_{j=1}^{c^{(l-1)}} K_{i,j}^{(l)} * Y_j^{(l-1)} + B_i^{(l)}$$

for an input $Y^{(l-1)} \in \mathbb{R}^{c^{(l-1)}}$, weight $K^{(l)} \in \mathbb{R}^{c^{(l)},c^{(l-1)}}$ and bias $B^{(l)} \in \mathbb{R}^{c^{(l)}}$ [16].

Recurrent Layers

Recurrent Layers describe a type of layers whose neurons are connected in a way that forms a directed cycle, i.e. (parts of) the output of a Recurrent Layer is used as its own input [49]. As a consequence, not only the output of the preceding layer is used as a Recurrent Layer's input, but also (part of) the output the Recurrent Layer produced from the previous data point. This means that a neural network consisting of Recurrent Layers is able to process sequences of input data and dependencies within such sequences. Usually, the sequences of data fed into such a recurrent neural network are time series data and the Recurrent Layers are utilized to exploit temporal information from them.

In the context of CNNs, networks that consist of Recurrent Layers (together with Convolutional, Pooling and Fully Connected Layers) are referred to as Convolutional Recurrent

Neural Networks (CRNNs). Many CRNN architectures that have been successfully employed in various applications in recent years consist of a concatenation of Convolutional and Pooling Layers followed by one or several Fully Connected and Recurrent Layers. When processing two- or three-dimensional images (like visual images or sprectrogram representations of audio signals), Convolutional and Pooling Layers are used for exploiting local dependencies of each element of a sequence of input data and Recurrent Layers following these Convolutional and Pooling Layers are used for exploiting temporal information that is structured across the elements of the sequence.

Literature describes a number of different types of Recurrent Layers used as part of CNNs, commonly used are Long Short-Term Memorys (LSTMs), introduced by Hochreiter and Schmidhuber (1997) [21], or the Gated Recurrent Unit (GRU), as introduced by Cho et al. (2014) [5]. Both LSTM and GRU contain a mechanism that allows them to store a feature of the input sequence across multiple elements of the input sequence.

In MIR research, in the recent past, CRNNs were able to achieve state-of-the-art results in tasks like music tagging, as shown by Choi et al. (2017) [8], from spectrogram representations. Here, CRNN models are used because of their ability to extract local information in a more flexible way than conventional CNN models. An explanation for this is that some parts of the information to be extracted might be influenced by the global structure of an input spectrogram and other parts might be influenced by short time segments of such a spectrogram [8].

3.4.2 Training Convolutional Neural Networks

Training CNNs is done using the same principle as in other feedforward neural networks, which is to iteratively minimize the loss function using gradient descent. For Convolutional Layers and Pooling Layers special adaptions to the optimization algorithm that is used have to be made.

Multi-Task Learning

Besides the approach described above, which is to train a neural network to predict one target variable to solve a machine learning task at hand (which is referred to as Single-Task Learning (STL)), another strategy is to train a single neural network to predict multiple target variables at once from the same input variables, i.e. to solve multiple machine learning tasks at once. This strategy is referred to as Multi-Task Learning (MTL) and has shown to be able to yield promising results in the context of deep neural networks in some cases as compared to using multiple networks where each one solves an individual task [53].

The benefit of prediction performance when using MTL is based on multiple mechanisms. One of the benefits of MTL is that when learning multiple target variables, the values of all of those variables will contain some sort of noise, which may be part of the parameters of hidden layers that learn internal feature representations. With MTL, however, a shared internal feature representation for mutliple target functions can help to average out these

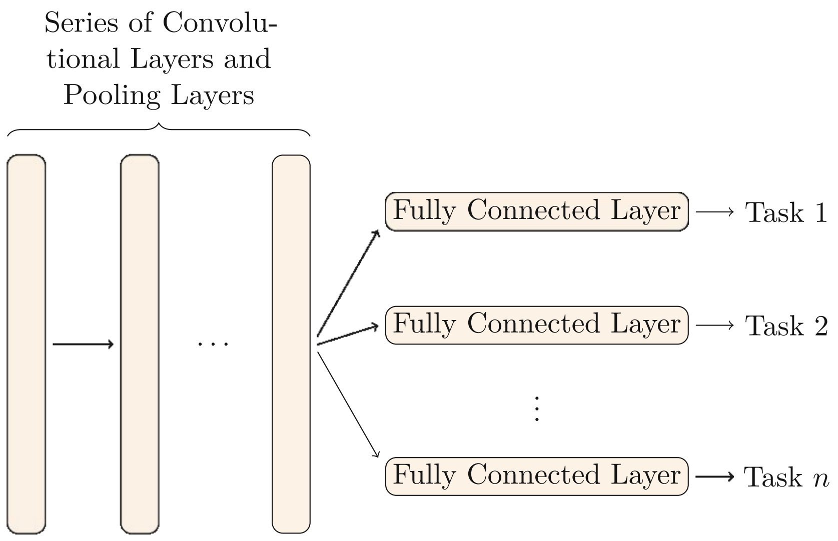


Figure 3.4: Schematic of a common MTL CNN architecture learning n tasks.

noises and thus reduce them. Furthermore, in cases where a network has difficulties learning a certain variable, learning other variables at the same time that are easier to learn, can improve the learning results for the variable, which is harder to learn. The reason for this is that, by learning other target variables, internal feature represenations can be created that are beneficial for learning the variable that was initially more difficult to learn [2].

There is a variety of different approaches described in literature of how the idea of MTL can be employed in deep neural networks, and especially in CNNs. Often, a neural network architecture is used that takes as its input a common set of input features for learning a set of tasks and calculates an output variable (or a set of output variables) for each of the tasks. In order to be able to optimize the weights of such a network, a loss function that combines the loss functions of all the tasks has to be found. A trivial loss function would be the sum of the losses of the individual tasks. There are also other strategies that attempt to regularize the losses during training [31, 15].

Besides different ways to handle the problem of loss weighting, there are also different approaches of how the architecture of MTL CNNs can be structured. One MTL approach suggests to use a neural network architecture where all or the majority of hidden layers is shared across different tasks. In the context of CNNs, architectures as depicted in Figure 3.4, which share a sequence of Convolutional and Pooling Layers across multiple tasks followed by a series of Fully Connected Layers for each individual task, have shown good prediction performances [54].

Also in the music domain, such MTL CNN models have shown promising results in various MIR tasks. For example, Böck et al. (2019) [1] have obtained state-of-the-art accuracy in tempo and beat estimation from spectrogram representations of songs using

MTL models. Singh and Biswas (2021) [47] were able to outperform a conventional CNN model with MTL CNNs in the tasks of artist and language recognition from spectrograms.

3.4.3 Convolutional Neural Network Architectures

The architecture of a CNN describes the way layers are connected in order to form the CNN. There is quite a lot of research works that has studied how different architectures affect the performance of CNNs for different machine learning tasks and data sets. Many CNN architectures that have been showing good prediction performance consist of a series of Convolutional Layers and Pooling Layers, followed by one or more Fully Connected Layers. Generally, CNNs with more layers tend to have higher expressive power. However, more complex architectures can also be more prone to generalization error. Furthermore, working with CNNs with a higher number of layers is computationally more expensive, which is the reason why early research works employed smaller CNN architectures than more recent research.

In the following, a brief overview of the architectures used for the attempt of reproduction of Spotify's black-box algorithms for calculating the music feature that are subject of this work is given.

LeNet-5

LeNet-5 is an early CNN architecture proposed by LeCun et al. (1989) [30]. A schematic of the architecture is given in Figure 3.5. LeNet-5 was originally employed and became famous for the task of handwritten digit recognition from monochrome images of 32×32 pixels. Compared to *ResNet-34*, described in Section 3.4.3, and many popular modern architectures that have yielded promising results in various applications, LeNet-5 is a relatively small network in that it has a small number of layers and thus has limited abilities with regard to exploiting complex structures from its inputs.

LeNet-5 is one of the neural network architectures that are considered for reproducing Spotify's music features as part of the experiments conducted in this work. The reason for this is that because of the characteristics of LeNet-5, it is presumably a good example for investigating how well a small architecture is suited for extracting semantic information from spectrogram music representations.

Residual Networks

Residual Network (*ResNet*) is a group of architectures that was first elaborated by He et al. (2016) [20], and is popular in various image recognition tasks because of its good prediction performance and its relatively low number of layers compared to other architectures. ResNets consist of a sequence of layers, just as e.g. LeNet-5, and additionally contain connections that pass the output of certain layers across subsequent layers so that these these layers are skipped. The output tensor of the sequence of skipped layers is then added to the input tensor of the sequence and forwarded to an activation function. Such a

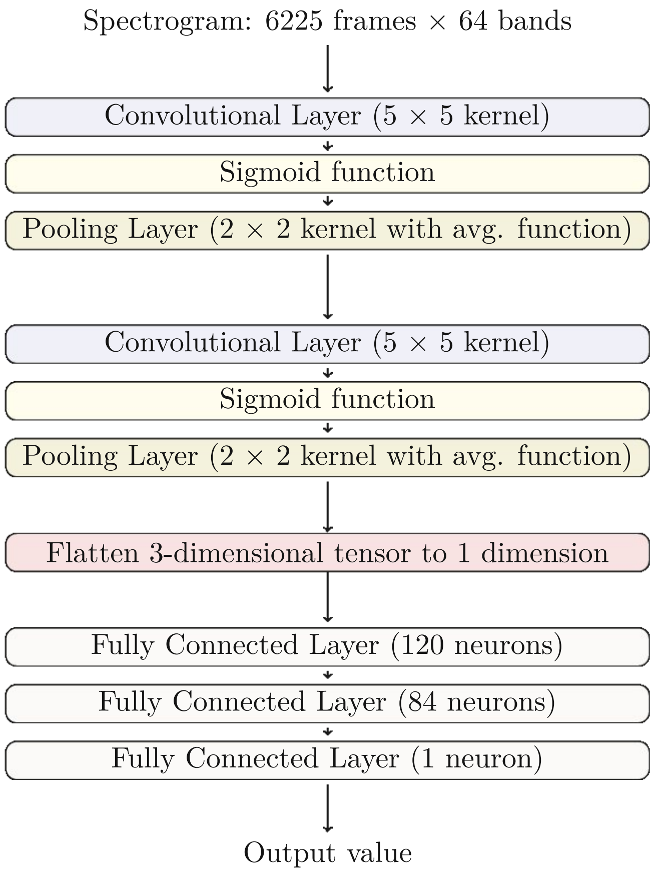

Spectrogram: 6225 frames × 64 bands
Convolutional Layer (5 × 5 kernel)
Sigmoid function
Pooling Layer (2 × 2 kernel with avg. function)
Convolutional Layer (5 × 5 kernel)
Sigmoid function
Pooling Layer (2 × 2 kernel with avg. function)
Flatten 3-dimensional tensor to 1 dimension
Fully Connected Layer (120 neurons)
Fully Connected Layer (84 neurons)
Fully Connected Layer (1 neuron)
Output value

Figure 3.5: Schematic of LeNet-5 architecture

sequence of layers is called a *residual block* and is depicted in Figure 3.6. The motivation for using residual blocks is to avoid the so-called *vanishing gradient problem*, which is the case when gradients become so small during backpropagation that the network's weights are effectively not updated or updated very slowly. Because residual blocks pass the gradient past themselves to the previous layer, it is less likely to vanish and, hence, residual networks may have better prediction performance [13].

Figure 3.7 shows the structure of *ResNet-34*, a residual network architecture, which is employed in the experiments of reproducing Spotify's black-box algorithms as part of this work. ResNet-34 contains 16 residual blocks of the same structure as the residual block shown in Figure 3.6.

Convolutional Recurrent Neural Networks

There is a variety of CRNN architectures described in literature in the field of MIR. However, for the experiments conducted as part of this work only one CRNN architecture is considered, whose structure is depicted in Figure 3.8. This architecture was already

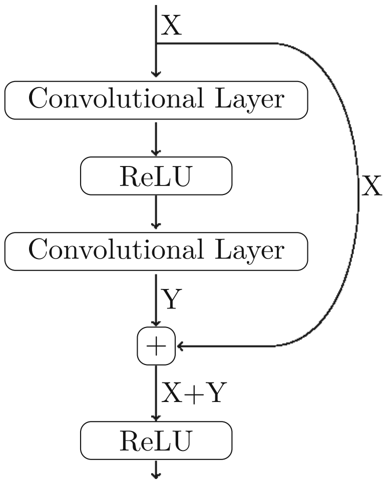


Figure 3.6: Schematic of a typical residual block.

used in other research works in the field of MIR. For example Choi et al. (2017) [8] employed this architecture for the task of music tagging and Nasrullah and Zhao [39] for the task of artist classification. In both works the CRNN showed promising results for the respective tasks.

Multi-Task Learning

The MTL architecture that is used in the attempt of reproducing Spotify's black-box algorithms is shown in Figure 3.9. A similar architecture was already used in a prior research work by Singh and Biswas (2021) [47] where a MTL model was trained to simultaneously perform music artist prediction and language recognition from spectrogram representations of songs.

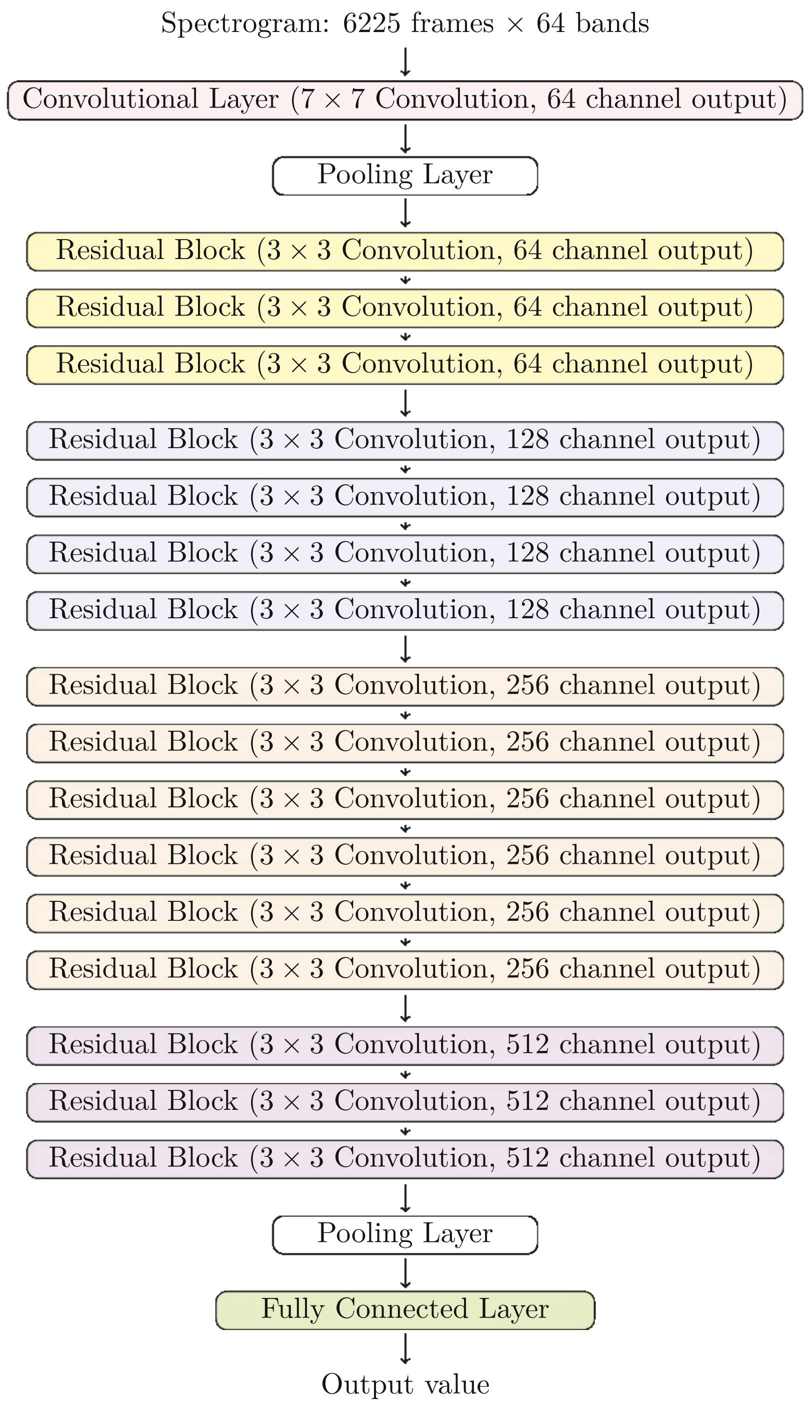

Spectrogram: 6225 frames × 64 bands
Convolutional Layer (7 × 7 Convolution, 64 channel output)
Pooling Layer
Residual Block (3 × 3 Convolution, 64 channel output)
Residual Block (3 × 3 Convolution, 64 channel output)
Residual Block (3 × 3 Convolution, 64 channel output)
Residual Block (3 × 3 Convolution, 128 channel output)
Residual Block (3 × 3 Convolution, 128 channel output)
Residual Block (3 × 3 Convolution, 128 channel output)
Residual Block (3 × 3 Convolution, 128 channel output)
Residual Block (3 × 3 Convolution, 256 channel output)
Residual Block (3 × 3 Convolution, 256 channel output)
Residual Block (3 × 3 Convolution, 256 channel output)
Residual Block (3 × 3 Convolution, 256 channel output)
Residual Block (3 × 3 Convolution, 256 channel output)
Residual Block (3 × 3 Convolution, 256 channel output)
Residual Block (3 × 3 Convolution, 512 channel output)
Residual Block (3 × 3 Convolution, 512 channel output)
Residual Block (3 × 3 Convolution, 512 channel output)
Pooling Layer
Fully Connected Layer
Output value

Figure 3.7: Schematic of ResNet-34 architecture.

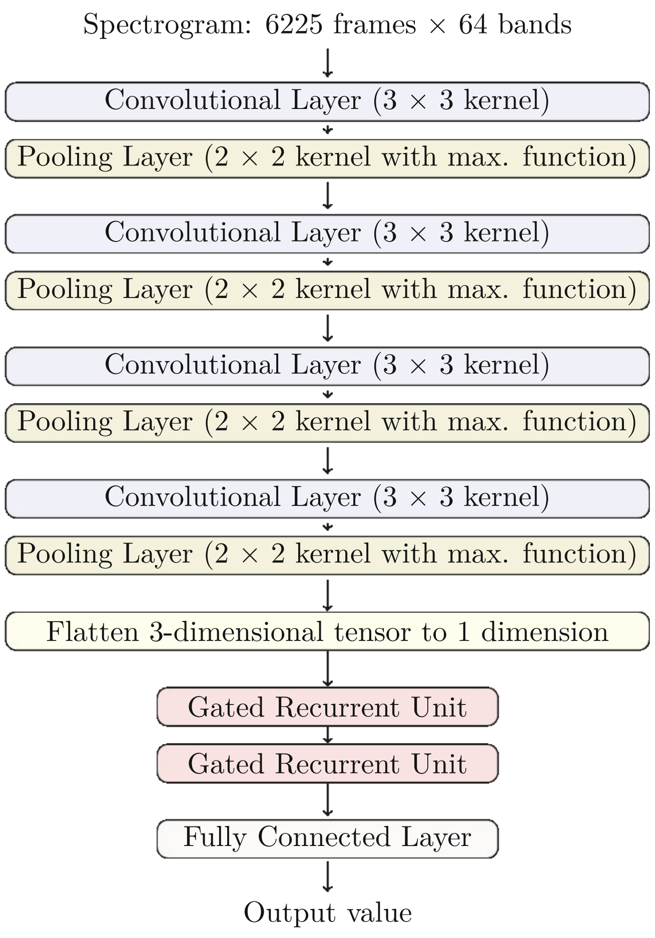

Spectrogram: 6225 frames × 64 bands
Convolutional Layer (3 × 3 kernel)
Pooling Layer (2 × 2 kernel with max. function)
Convolutional Layer (3 × 3 kernel)
Pooling Layer (2 × 2 kernel with max. function)
Convolutional Layer (3 × 3 kernel)
Pooling Layer (2 × 2 kernel with max. function)
Convolutional Layer (3 × 3 kernel)
Pooling Layer (2 × 2 kernel with max. function)
Flatten 3-dimensional tensor to 1 dimension
Gated Recurrent Unit
Gated Recurrent Unit
Fully Connected Layer
Output value

Figure 3.8: Schematic of a CRNN.

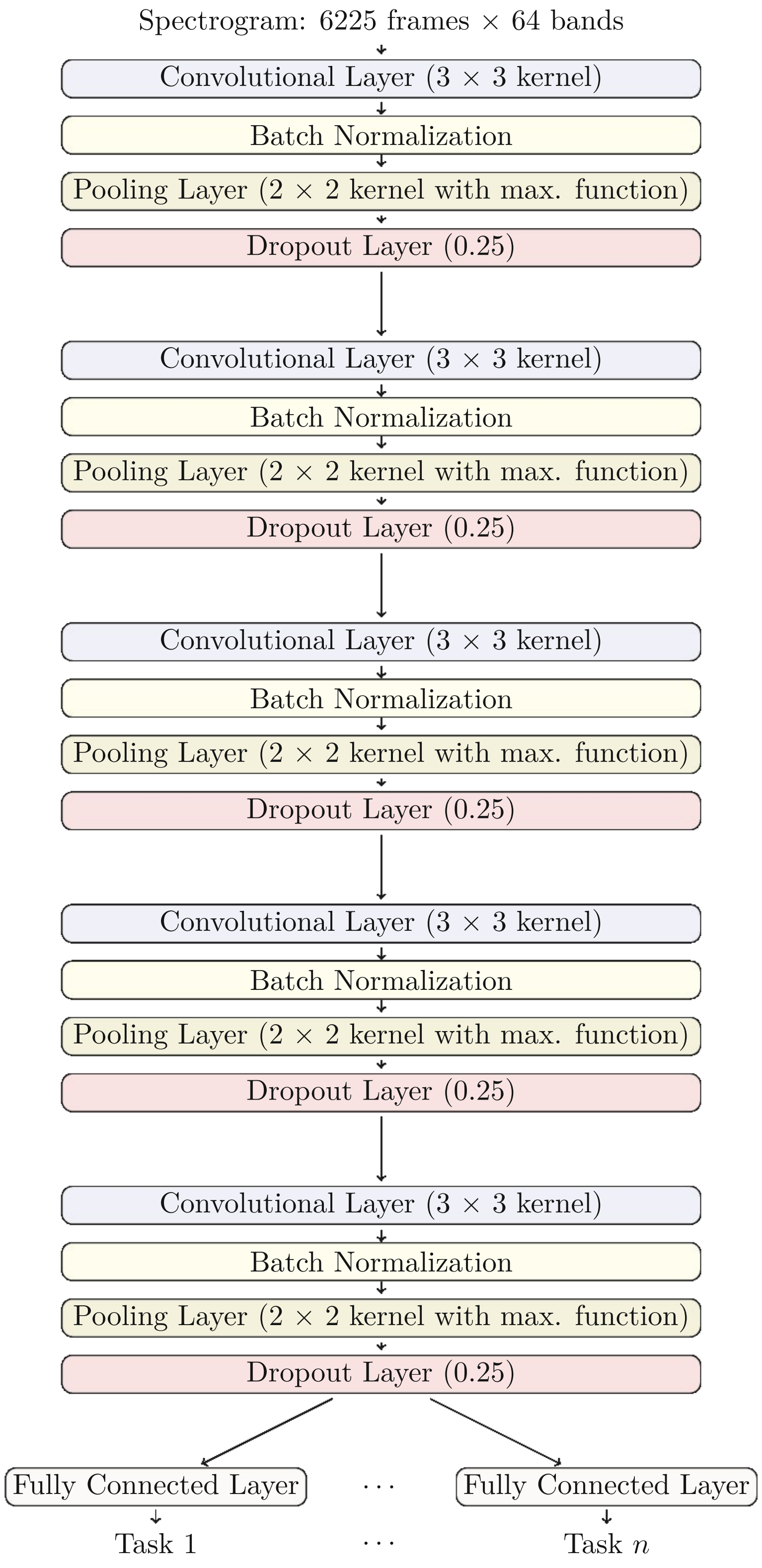

Spectrogram: 6225 frames × 64 bands
Convolutional Layer (3 × 3 kernel)
Batch Normalization
Pooling Layer (2 × 2 kernel with max. function)
Dropout Layer (0.25)
Convolutional Layer (3 × 3 kernel)
Batch Normalization
Pooling Layer (2 × 2 kernel with max. function)
Dropout Layer (0.25)
Convolutional Layer (3 × 3 kernel)
Batch Normalization
Pooling Layer (2 × 2 kernel with max. function)
Dropout Layer (0.25)
Convolutional Layer (3 × 3 kernel)
Batch Normalization
Pooling Layer (2 × 2 kernel with max. function)
Dropout Layer (0.25)
Convolutional Layer (3 × 3 kernel)
Batch Normalization
Pooling Layer (2 × 2 kernel with max. function)
Dropout Layer (0.25)
Fully Connected Layer
· · ·
Fully Connected Layer
Task 1
· · ·
Task n

Figure 3.9: Schematic of a MTL CNN.

CHAPTER 4

Experiments

4.1 Data Set

The data set that is considered for the attempt of reproducing the black-box algorithms that are the subject of this work is a set of audio sequences of songs along with the feature values of the music features that are calculated by these algorithms. The machine learning models that are trained to estimate these music features will use the audio sequences as their input and the feature values as ground truth.

The selection of songs that is considered in the course of this work is supposed to be composed of songs that differ across various genres and origin times and regions with the intention to map a broad variety of different music in order to obtain a preferably "general" algorithm for approximating Spotify's features in the sense that this algorithm yields good results for different types of music, and to be able to evaluate the results in terms of different properties of the evaluated music. For the sake of simplicity there is no such selection of songs created in the course of this work, but a data set of songs readily available - the *Million Playlists Dataset* [4] - that has been created with sufficiently similar intentions is used instead.

The *Million Playlists Dataset* is a set of 1 000 000 playlists that was created for the *RecSys Challenge 2018* for the task of finding a list of songs that continues a playlist in the best possible way [4]. For the experiments conducted in this work all songs contained in the playlists of the *Million Playlists Dataset* are taken to form a data set of 2 261 469 songs.

For each of these 2 261 469 songs, a variety of metadata can be retrieved through Spotify's web API[1], including the audio features that are attempted to be approximated in this work, as well as an audio sample that is used for approximating the features. However, only for 1 405 808 songs such a sample is available. Therefore, the dataset that is considered throughout this work is restricted to those 1 405 808 songs.

[1]https://developer.spotify.com/documentation/web-api/reference/, accessed 2022-11-26

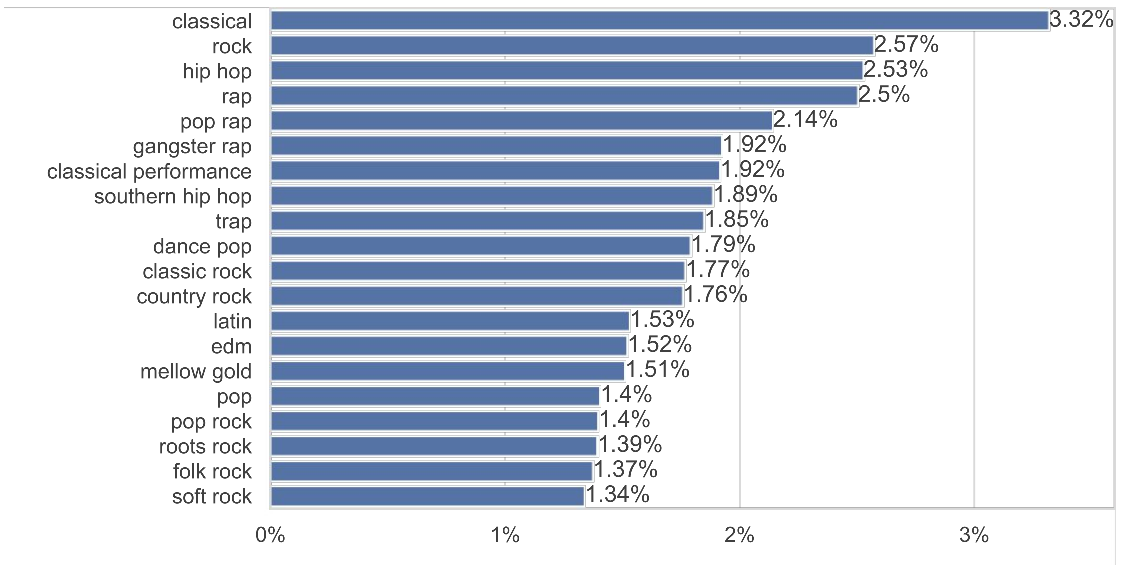


Figure 4.1: The 20 most frequent genres in the data set along with the relative frequency they are assigned to a song (relative to the total number of songs). The genres are based on annotations of a song's artists with genres and were retrieved from Spotify's web API[3]. The reason artists' genres were used here rather than a song's genre annotations is that Spotify does not provide this information. Because of the fact that a song can potentially have multiple artists and an artist can potentially be annotated with multiple genres, a song can potentially be assigned multiple artists.

For each of these songs the data set that will be used for training the machine learning models consists of the audio signal of a sample of 30 seconds, as well as the ground truth labels of the audio features “Danceability”, “Acousticness”, “Instrumentalness”, “Speechiness”, “Liveness”, “Valence”, “Energy” and “Loudness”.

As described above, the audio samples that are used for approximating the audio features are downloaded from Spotify's web API[2]. There they are available in MP3 format with a sample rate of 44 100Hz. Each of these audio samples contains a continuous sequence of 30 seconds. Generally, these samples do not contain the very beginning or end of a song, but some part in between.

The songs considered in the *Million Playlists Dataset* are intended to cover a broad selection of different styles and genres of music. Because of the fact that the selection of songs considered in this work only consists of songs available on Spotify, the selection of songs is biased towards mainstream music originated from “Western cultures” [22]. Figure 4.1 shows the 20 most frequent genres of the songs in the data set.

As described in Section 4.1, the data set that is used for the reproduction of the algorithms for calculating the semantic audio features contains 1 405 808 data points.

[2]https://developer.spotify.com/documentation/web-api/reference/, accessed 2022-11-26

For the purpose of training and evaluating the performances of the machine learning models that are the subject of the experiments conducted as part of this work, the data set is split into three subsets: a training set, a validation set and a test set. This approach is standard practice in designing machine learning experiments and has the intention of preventing biased evaluation, i.e. evaluating a model's performance during parameter optimization based on data that has already been used for training the model or evaluating a final model's performance on data that has already been used for selecting the final model. Research literature describes different ways of how a data set is best split into these three subsets depending on characteristics of the respective data set. Depending on the size of a data set and the distribution of the individual subsets with respect to the population distribution different split ratios (i.e. the ratio of the sizes between training, validation and test set) are suggested.

In case of the data set at hand, $\sim 5\%$ of the total number of data points (i.e. 70 290 data points) are assigned to each the validation and the test set. The remaining 90% of the data points (i.e. 1 265 228 data points) are assigned to the training set. This split ratio is chosen because of the relatively large amount of available data, considering that a validation (respectively test) set of 70 290 data points is expected to give sufficient confidence when evaluating the models' performances. The assignment of data to one of the three subsets is done by random sampling.

4.2 Music Features

In the following, a brief characterization of each of the music features is given.

4.2.1 Danceability

The feature "Danceability" is intended to provide information about "how suitable a track is for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity"[4]. Values of "Danceability" are real values ranging from 0 to 1[4], where 0 describes a "least danceable" song, and 1 a "most danceable"[4] one. Figure 4.3 shows the distribution of the values of "Danceability" in the data set with a mean of ~ 0.550, $q_{0.25} \approx 0.425$, $q_{0.50} \approx 0.565$, $q_{0.75} \approx 0.69$ and a variance of ~ 0.034. As shown in Figure 4.2, the values are positively correlated with feature "Valence" (correlation coefficient $\rho \approx 0.530$). Figure 4.2 does not indicate correlations between "Danceability" and any other features.

4.2.2 Acousticness

According to Spotify's web API documentation "Acousticness" is "a confidence measure from 0.0 to 1.0 of whether the track is acoustic"[4]. 1 represents high confidence that the track is acoustic, whereas 0 represents low confidence the track is acoustic. Figure 4.4 shows the distribution of the values of "Acousticness" present in the data set. The

[4]https://developer.spotify.com/documentation/web-api/reference/, accessed 2022-11-26

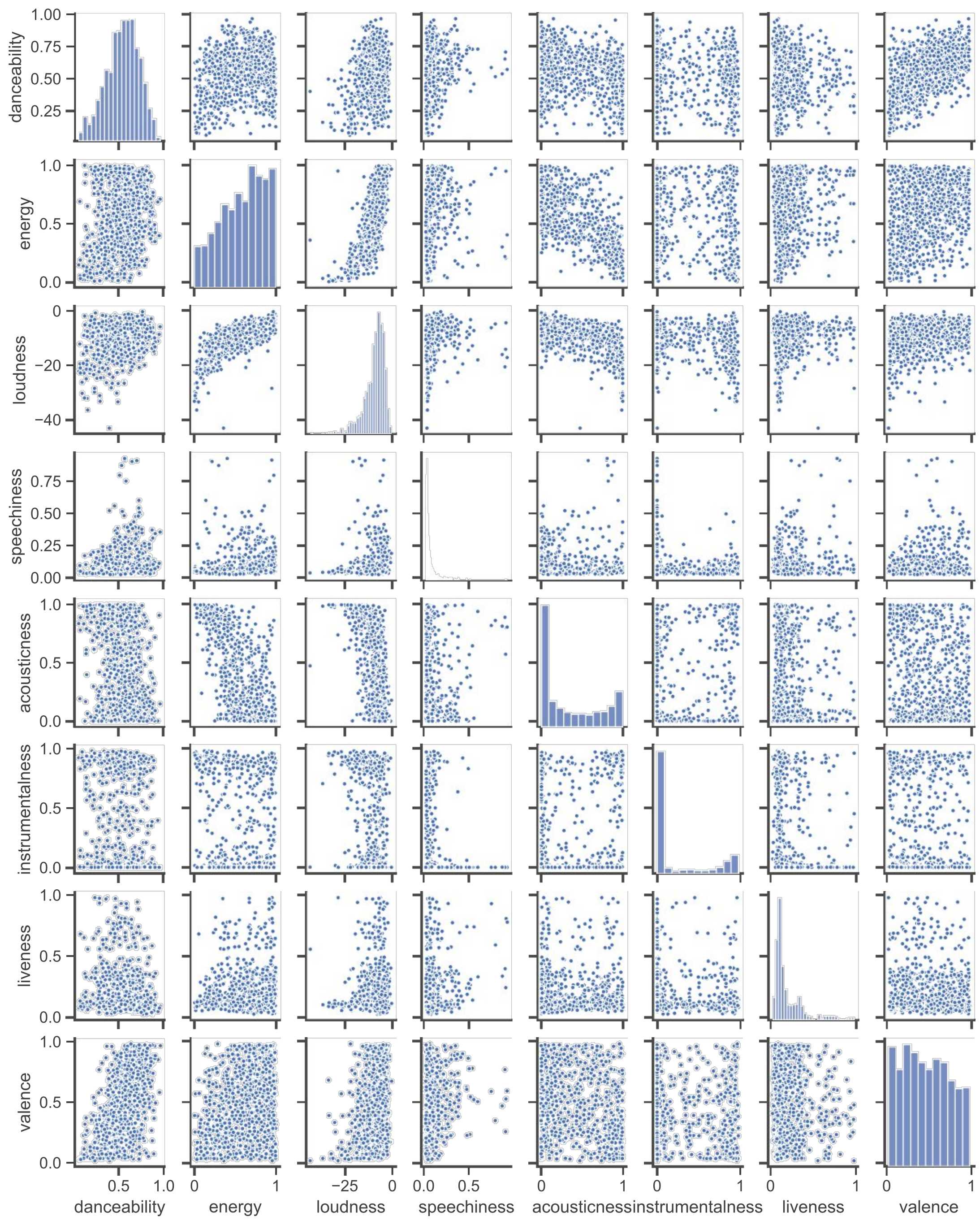


Figure 4.2: Scatter plot matrix showing correlations between the music features. This visualization is based on feature values of a random sample of 1000 songs of the data set.

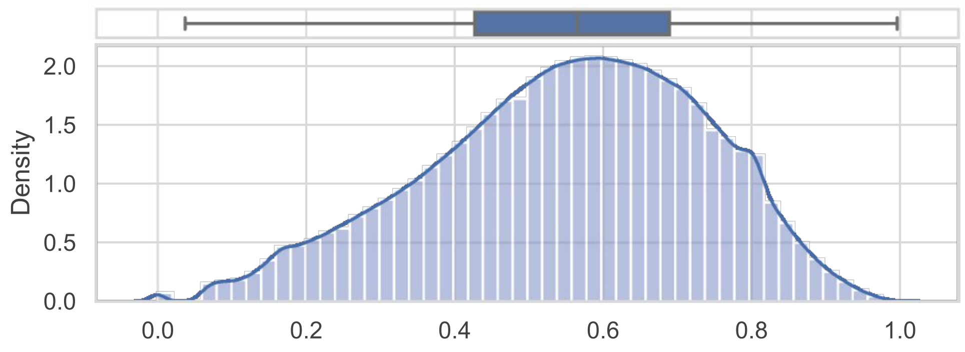


Figure 4.3: Distribution of the values of “Danceability”

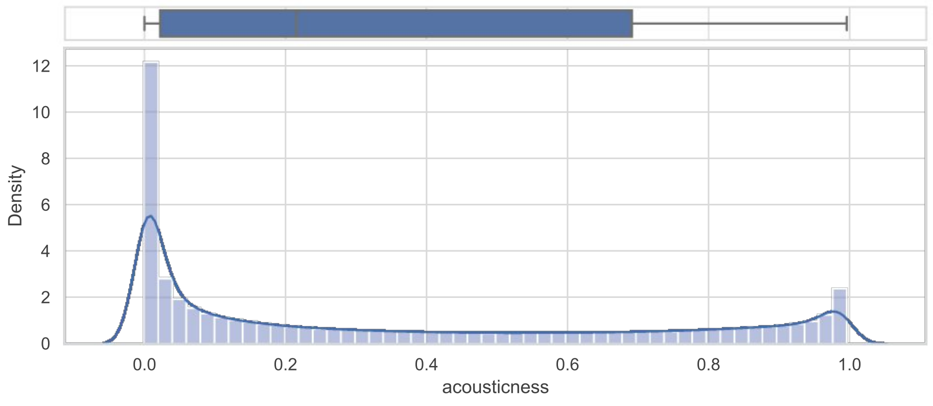


Figure 4.4: Distribution of the values of “Acousticness”

values have a mean of ~ 0.358, $q_{0.25} \approx 0.021$, $q_{0.5} \approx 0.216$, $q_{0.75} \approx 0.704$ and a variance of ~ 0.128. Figure 4.2 shows that “Acousticness” is negatively correlated with “Energy” ($\rho \approx -0.746$) and “Loudness” ($\rho \approx -0.6222$).

4.2.3 Instrumentalness

Music feature “Instrumentalness” expresses the probability of a track containing no vocals[5]. The values range from 0 to 1 with higher values representing higher probabilities that a song contains no vocals. with a mean of ~ 0.235, $q_{0.25} = 0$, $q_{0.5} \approx 0.001$, $q_{0.75} \approx 0.52$ and a variance of ~ 0.128. The distribution of the “Instrumentalness” values in the data set is visualized in Figure 4.5. From Figure 4.2 no correlations between feature “Instrumentalness” and other features are detectable. Spotify provides a description stating that vocal “ooh” and “aah” sounds are counted as instrumentals,

[5]https://developer.spotify.com/documentation/web-api/reference/, accessed 2022-11-26

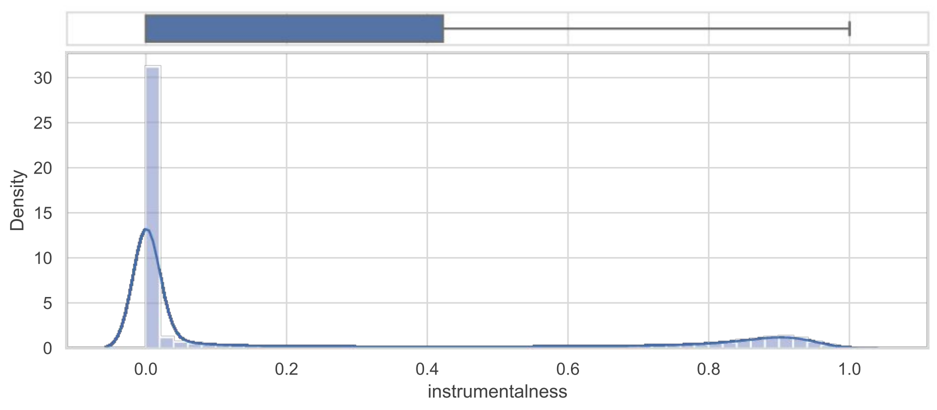


Figure 4.5: Distribution of the values of “Instrumentalness”

“Instrumentalness” values greater than 0.5 are inteded to be assigned to instrumental songs and rap or track containing spoken word are vocal[6].

4.2.4 Speechiness

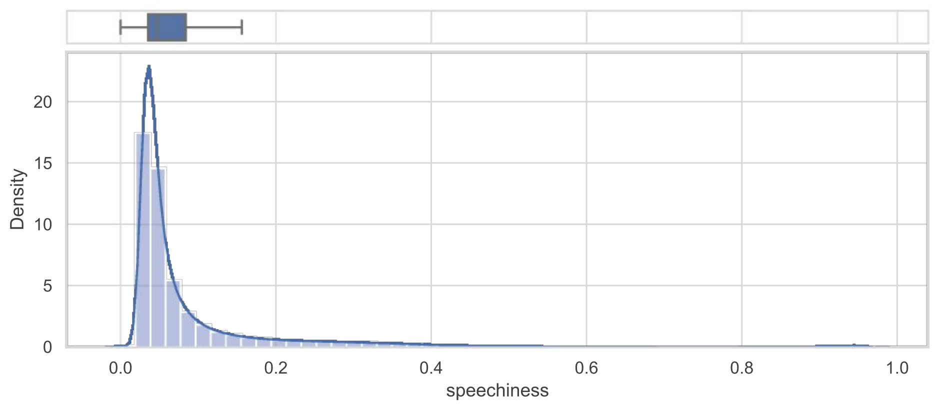


Figure 4.6: Distribution of the values of “Speechiness”

Feature “Speechiness” is intended to express to what extent spoken word is present in a track, where tracks with more speaking tend to have a higher feature value[6]. Values of “Speechiness” are real an between 0 and 1. Spotify furthermore describes in its API documentation that music (and other “non-speech-like” tracks) have “Speechiness” values of less than 0.33. Songs that contain music as well as spoken word (e.g. rap or other tracks that contain sections of speech and sections of music) have values between 0.33 and 0.66, and tracks that do not contain any music have values greater than 0.66. Figure 4.6

[6]https://developer.spotify.com/documentation/web-api/reference/, accessed 2022-11-26

gives an impression of the distribution of the values of feature “Speechiness” in the data set and that the majority of values are distributed below a value of 0.33. The mean value for “Speechiness” is ~ 0.089, $q_{0.25} \approx 0.035$, $q_{0.5} \approx 0.047$ and $q_{0.5} \approx 0.082$. No correlations between “Speechiness” and other music features are noticable in Figure 4.2.

Figure 4.6 also indicates a relatively small number of songs clustered between 0.8 and 1 and further analysis shows that among those songs with values above 0.66, the median is ~ 0.921, which seems to be approximately the center of this cluster. The genre annotations that the artists of these songs are annotated with the most are “comedy”, “new comedy” and “comic”. Upon further inveisitgation, it appears that some of these tracks indeed seem to be recordings of comedy programs rather than music. However, the large majority of observations in the data set does not seem to be such non-music tracks.

4.2.5 Liveness

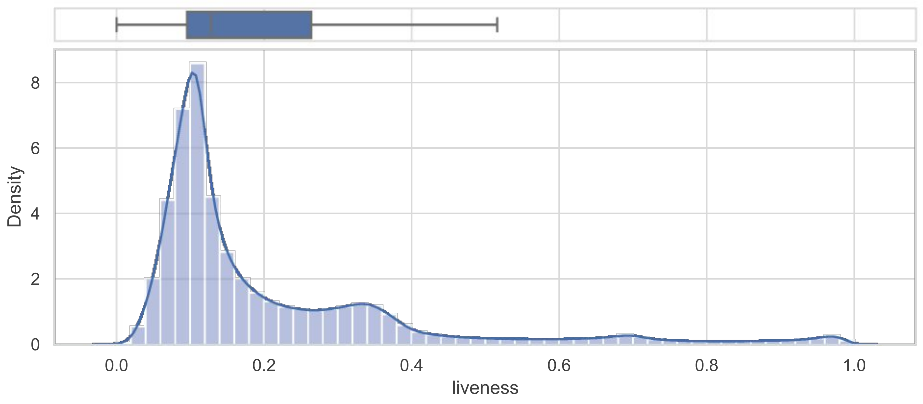


Figure 4.7: Distribution of the values of “Liveness”

“Liveness” is a music feature that estimates the probability of the recording containing noise from an audience. The feature values are real numbers ranging from 0 to 1 with higher values representing a higher probability that the recording contains noise from an audience. Figure 4.7 shows the probability density function and a histogram along with a boxplot of the values of “Liveness” in the data set. The values have a mean of ~ 0.206, $q_{0.25} \approx 0.095$, $q_{0.5} \approx 0.126$, $q_{0.75} \approx 0.285$ and variance ~ 0.035. Figure 4.2 does not indicate any obvious correlations between “Liveness” and other features.

4.2.6 Valence

Music feature “Valence” represents how positive a song is perceived emotionally. Its values are real and range from 0 to 1 with low values describing songs that sound negative and high ‘Valence” values that sound positive. Figure 4.8 shows that the values are relatively broadly distributed within their range with noticable local maxima of the

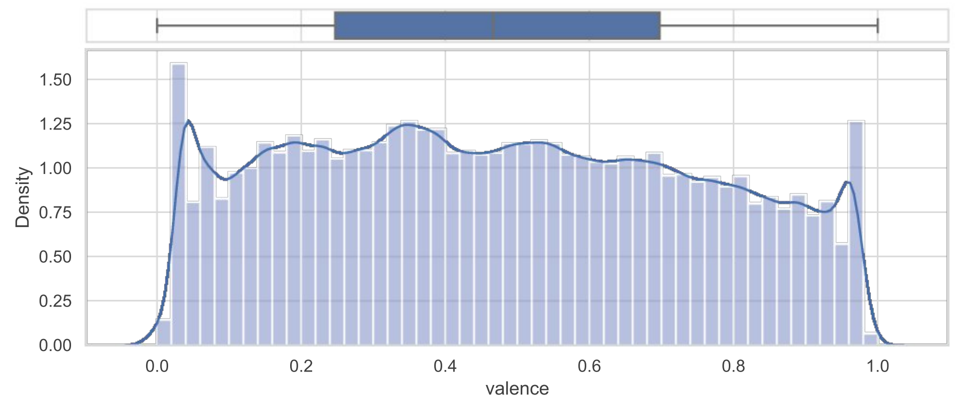


Figure 4.8: Distribution of the values of “Valence”

probability distribution function at each end of the range of values. In the data set “Valence” has mean ~ 0.470, $q_{0.25} \approx 0.239$, $q_{0.5} \approx 0.457$, $q_{0.75} \approx 0.690$ and variance ~ 0.073. As mentioned above, the songs in the data set show correlations between “Valence” and “Danceability” ($\rho \approx 0.530$).

Further analysis shows that among songs with particularly low “Valence” values, genre “classical” (followed by “classical performance”, “soundtrack” and “orchestra”) is the most prevalent. Among songs with particularly high values of “Valence” the three most assigned genres are “regional mexican”, “corrido” (a music genre originated from Mexico) and “nuevo regional mexicano”.

4.2.7 Energy

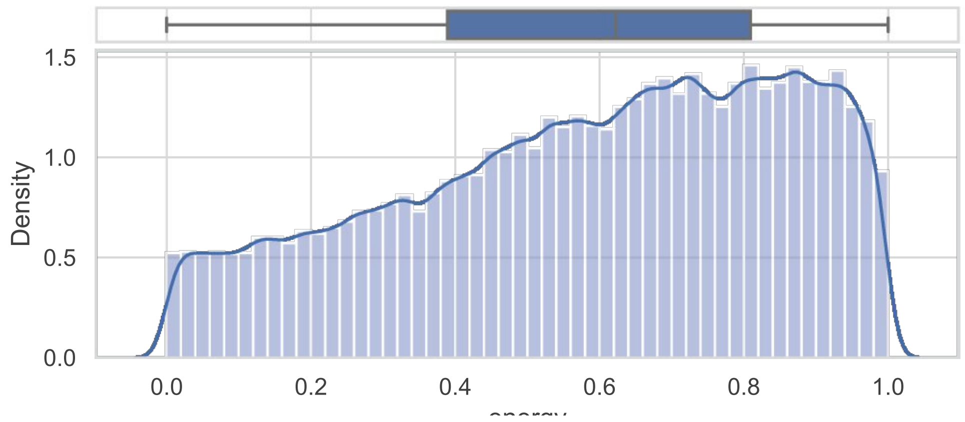


Figure 4.9: Distribution of the values of “Energy”

Feature “Energy” intends to describe how energetic, intensive and active a song is perceived. The values of “Energy” are real numbers ranging from 0 to 1. The distribution of the values of feature “Energy” in the data set is visualized in Figure 4.9. The values have a mean of ~ 0.581, quantiles $q_{0.25} \approx 0.382$, $q_{0.5} \approx 0.619$ and $q_{0.75} \approx 0.808$ and variance ~ 0.072. Figure 4.2 suggests that there is relatively strong positive correlation between “Energy” and “Loudness” and negative correlation between “Energy” and “Acousticness” ($\rho \approx -0.746$). According to Spotify’s web API documentation “Energy” correlates with “dynamic range, perceived loudness, timbre, onset rate, and general entropy”[7].

4.2.8 Loudness

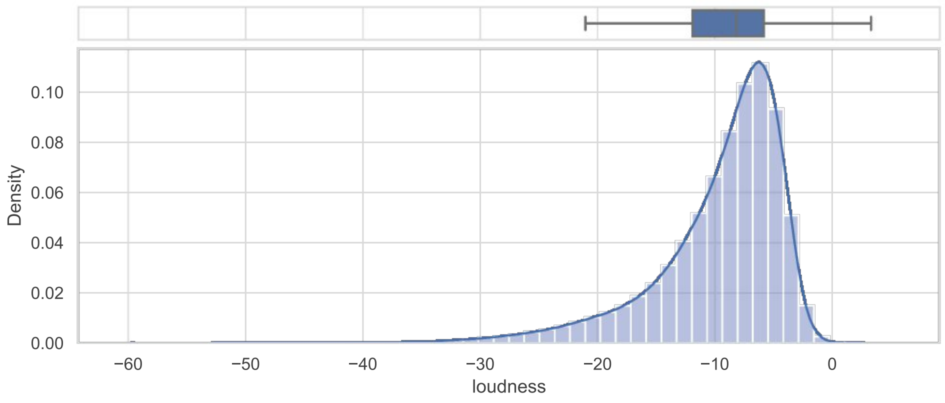


Figure 4.10: Distribution of the values of “Loudness”

Feature “Loudness” measures the average loudness of a song on decibel scale ranging from −60dB to 0dB. The values in the data set are real numbers. Figure 4.9 shows the distribution of the values of feature “Loudness” in the data set. The values have a mean of ~ -9.786, quantiles $q_{0.25} \approx 12.009$, $q_{0.5} \approx -8.253$ and $q_{0.75} \approx -5.888$ and variance ~ 32.924. As already mentioned, “Loudness” is positively correlated with “Energy” ($\rho \approx 0.790$) and negatively correlated with “Acousticness” ($\rho \approx -0.622$).

4.3 Experiment Design

As part of this work, a number of experiments are being conducted. The aim of these experiments is to answer the research questions formulated in Section 1.2, which address the question to what extent it is possible to reproduce the algorithms that are employed by Spotify for calculating the music features detailed in Section 4.2 using Convolutional Neural Networks (CNNs) for extracting them from spectrogram music representations. The evaluation of the experiments is based on the performance of the CNN models.

[7]https://developer.spotify.com/documentation/web-api/reference/, accessed 2022-11-26

The attempt of reproducing the black-box algorithms is restricted to the approaches described in Chapter 3 and moreover, not all CNN architectures described in 3.4.3 and combinations of their parameters will be fully exploited in order to determine the best approximation for each of the semantic audio features. These restrictions are made in order to keep the computational efforts for training the CNN models in feasible bounds. A more detailed description of which CNN architectures and parameters are considered in the experiments is given in the following.

In order to be able to answer research questions RQ1 and RQ2 a set of models with different parameters is trained for each of the music features described in Section 4.2. For each music feature, the CNN architectures *LeNet-5* and *ResNet-34*, as well as the Convolutional Recurrent Neural Network (CRNN) architecture presented in Section 3.4.3 are investigated. For each music feature and architecture two different learning rates, in particular 0.001 and 0.0001, and two different optimization algorithms, namely Adam and Stochastic Gradient Descent (SGD) (see Sections 3.2.1 and 3.2.2), are furthermore considered. These parameter combinations amount to twelve models per music feature or 96 models in total. Each model is trained on the training data set and subsequently evaluated on the validation set to be able to compare the quality of the models against each other. Based on the MSE obtained from evaluation the best performing model of each music feature is then evaluated again on the test data set. The evaluation results on the test set are considered as the final prediction results and are used for discussing research question RQ1.

For the purpose of answering research question RQ3, which addresses the comparison of the results obtained in the experiments described above with a Multi-Task Learning (MTL) approach, a set of MTL models is trained. To that end, multiple music features are learned by the same model at the same time. For the sake of keeping the computational efforts within feasible bounds, not all possible combinations of music features are learned by an MTL model. Instead, exemplarily, four pairs (of two music features each) are selected and one MTL model is trained for each pair. The pairs that are considered are the features “Danceability” and “Valence”, the features “Acousticness” and “Energy”, the features “Instrumentalness” and “Speechiness” and the features “Liveness” and “Loudness”. The reason for considering these pairs of features is that they cover all individual features and contain pairs of features that show correlations with each other (which are the pair of “Danceability” and “Valence” and the pair of “Acousticness” and “Energy”), as well as pairs of features that do not indicate correlations with each other (which are the pair of “Instrumentalness” and “Speechiness” and the pair of “Liveness” and “Loudness”). Thus, it might be possible to derive whether the prediction performance benefits from using correlated features in such an MTL setting. As with the Single-Task Learning (STL) experiments described above, every model is first trained on the training data set. The models are then evaluated using the validation data set. Based on the results, research question RQ3 can be assessed.

4.4 Experiment Implementation

The software that executes data preprocessing, training and evaluating of the models that are subject of the experiments conducted as part of this work are implemented in *Python* (version 3.8.5)[8]. The neural networks and the code for preprocessing of audio signals are implemented using the machine learning framework *PyTorch* (version 1.8.1)[9] and its audio processing library *Torchaudio*[10]. Additional notable libraries that were used for the implementations are *NumPy* (version 1.19.2)[11] and *pandas* (version 1.1.3)[12], as well as *Matplotlib* (version 3.3.2)[13] and *seaborn*[14] for creating visualizations depicted in this work.

All experiments conducted in the course of this work were executed on a machine equipped with an *Intel Core i5-9600K* CPU, 32 gigabytes of RAM and an *Nvidia GeForce GTX 1080* GPU. For model training and evaluation, the capabilities of *PyTorch*, which allow execution of vectorized operations on GPUs, were utilized. This allowed for accelerated execution speeds as compared to execution on CPUs.

4.5 Performance Evaluation

The performance of the CNN models that are trained as part of this work is evaluated by measuring the quality of the predictions on the test set. This is done by comparing the predictions to the ground truth.

Since all the target variables considered in the experiments have continuous values, measuring the quality of a model can be done by using a metric that expresses the similarity between predicted values $\hat{y}$ and ground truth values y.

One possible metric with this property would be the Mean Absolute Error (MAE).

$$\text{MAE} = \frac{\sum_{i=1}^{n} |y_i - \hat{y}_i|}{n}$$

It has the advantage of representing the error between predicted and ground truth values on the same scale, on which the values themselves are given. Thus, the interpretation of MAE values is particularly intuitive.

Beside MAE, it is also worth considering the Mean Squared Error (MSE).

$$\text{MSE} = \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{n}$$

[8]https://www.python.org/
[9]https://pytorch.org/
[10]https://pytorch.org/audio/stable/index.html
[11]https://numpy.org/
[12]https://pandas.pydata.org/
[13]https://matplotlib.org/
[14]https://seaborn.pydata.org/

It expresses the average squared differences between predicted and ground truth values, with MSE values ranging from 0 to ∞ where lower values describe better prediction. Because of the fact that MSE uses the squared differences between $\hat{y}$ and y rather than the absolute difference, greater deviations of $\hat{y}$ from y have a greater effect on the measured performance. In this way, predicted values with greater errors are penalized more than predictions that are closer to ground truth.

Because of the different dispersions of the feature value distributions of the different features that are the subject of the experiments, the scale of the MSEs of the resulting machine learning models might differ across the features. In order to be able to interpret how "good" an MSE value obtained from evaluation is in contrast to the scale of a feature's value, the R^2 Score, defined as

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2},$$

can be considered. R^2 Score puts the MSE in relation to a feature's variance, with values ranging from $-\infty$ to 1 where values below 0 describe an MSE greater than the feature values' variance, values above 0 describe an MSE lower than the feature values' variance and 1 describes a perfect fit (i.e. an error of 0). Ultimately, MSE metric will be used to choose which model performs best.

CHAPTER 5

Reproduction of Spotify's Audio Features

5.1 Audio Signal Processing

The audio signals of the songs in the dataset that are considered for the experiments conducted as part of this work are downloaded from Spotify's web API [1] as MP3 files containing audio samples of 30 seconds length with a sample rate of 44 100Hz, which makes $30 \cdot 44\,100 = 1\,323\,000$ samples per song.

The audio signal of each song is transformed into frequency domain by applying Short-Time Fourier-Transformation (STFT). To this end, the audio signal is segmented with a frame size of 400, a hop size of 200 and "zero-padding", which results in 6 625 frames.

Each frame of N samples is multiplied with a Hann function

$$w(n) = \sin^2\left(\frac{\pi n}{N-1}\right).$$

The STFT is then performed on each of the 6 625 frames, producing 6 225 vectors of 201 values. The number of values per vector is limited to 201 because this is the 201Hz is the lowest frequency that can be present in a frame of 400 samples.

The resulting spectrograms are then transformed into Mel scale using 128 filter banks. This transformation results in a Mel spectrogram of 6 225 frames with 128 bins in each frame.

These $6\,225 \times 128$ matrices are used as the inputs to Convolutional Neural Network (CNN) models of different architectures and hyperparameters. The architectures and hyperparameters are described below together with the experiment results.

[1] https://developer.spotify.com/documentation/web-api/reference/, accessed 2022-11-26

5.2 Experiment Results

5.2.1 Single-Task Learning Results

Danceability

Experiment Results					
CNN architecture	Learning rate	Optimization algorithm	MAE	MSE	R^2 Score
LeNet-5	SGD	0.0001	~ 0.088	~ 0.013	~ 0.626
LeNet-5	SGD	0.0010	~ 0.084	~ 0.012	~ 0.655
LeNet-5	Adam	0.0001	~ 0.086	~ 0.012	~ 0.640
LeNet-5	Adam	0.0010	~ 0.088	~ 0.013	~ 0.634
ResNet-34	SGD	0.0001	~ 0.088	~ 0.013	~ 0.615
ResNet-34	SGD	0.0010	~ 0.078	~ 0.010	~ 0.701
ResNet-34	Adam	0.0001	~ 0.071	~ 0.008	~ 0.767
ResNet-34	Adam	0.0010	~ 0.071	~ 0.009	~ 0.748
CRNN	SGD	0.0001	~ 0.099	~ 0.016	~ 0.526
CRNN	SGD	0.0010	~ 0.096	~ 0.016	~ 0.546
CRNN	Adam	0.0001	~ 0.096	~ 0.015	~ 0.563
CRNN	Adam	0.0010	~ 0.094	~ 0.015	~ 0.569

Table 5.1: Experiment results for feature "Danceability"

Table 5.1 contains the results for feature "Danceability" of the experiments that are described in Section 4.3. Each row in Table 5.1 represents one model that was trained on the training data set to extract feature "Danceability". The models are based on the hyperparameters specified in the respective row and yielded a Mean Absolute Error (MAE) and Mean Squared Error (MSE) as given in the respective columns when predicting the validation data set. The training time using the hardware as described in Section 4.4 was approximately 19 hours for each of the models (with the models with ResNet-34 architecture having slightly longer training times).

As can be seen from the table, based on the MSE when predicting the validation data set, the best-performing model is the ResNet-34 model with learning rate 0.0001 and Adam optimization with an MSE of ~ 0.008. Also, it is noticable that the ResNet-34 models with Adam optimization yield approximately the same MSE, regardless of the chosen learning rate, and they are performing better than any other models.

Figure 5.1 shows the MSE for the different models when predicting the validation data set in relation to the number of data points already trained. It shows that the two best-performing ResNet-34 models have lower errors throughout the entire training process (except for the very beginning). The other curves representing the other models show roughly the same location. All curves show a relatively steep descent at the beginning of training (until approximately 250 000 trained samples) and a relatively shallow descent afterwards. The MSE on the validation set of the model with ResNet-34 architecture, learning rate 0.001 and Stochastic Gradient Descent (SGD) optimization shows an especially high variability during training. Also, the validation curves of the LeNet-5 and Convolutional Recurrent Neural Network (CRNN) models have a higher

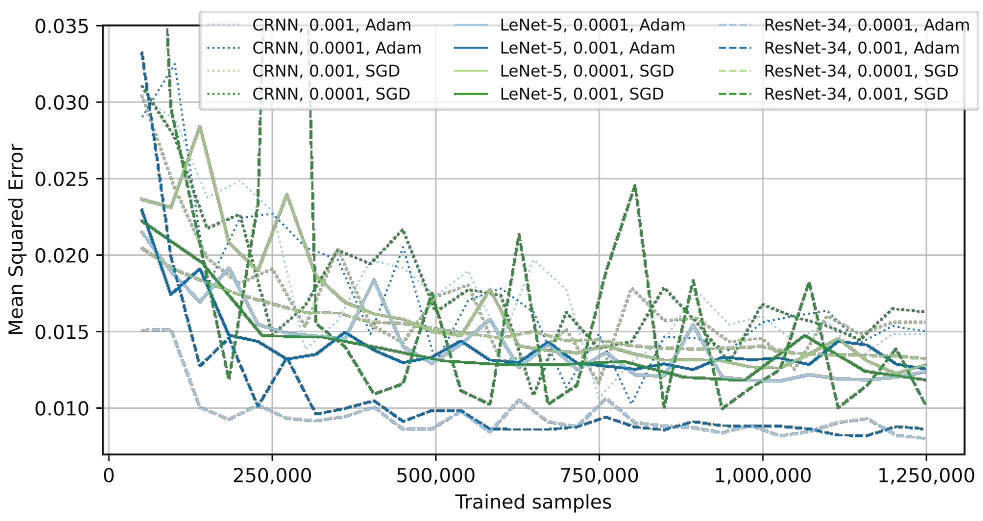


Figure 5.1: MSEs for the different models when predicting feature "Danceability" on the validation data set in relation to the number of data points already trained.

variablity than the other ResNet-34 models.

The training errors (i.e. the MSE when predicting the training data set) and the validation errors are shown in Figure 5.2 in relation to the number of data points already trained. Here, the values are exemplarily shown for the ResNet-34 model with learning rate 0.0001 and Adam optimization only as this is the best performing model. However, the respective graphs of the other models have similar characteristics. Note that for the training error shown in Figure 5.2 each point of the graph is calculated as the MSE of the previous 640 training samples. The relatively great noise of the curve showing the training error in Figure 5.2 can be explained by the fact that the training error was calculated on a small subsets of the training data. Contrarily, the validation errors are calculated on the entire validation set, which is a larger set of data, and therefore are less prone to noise.

It is apparent that both curves shown in Figure 5.2 have a steeper slope in the section that represents the beginning of the training process than later in the training process. The reason for this is that in the beginning of training the model weights are adapted more in order to minimize loss as compared to later in the training process when the model is already fitted and represents the target function better. Usually, with more training the error would reduce until the model reaches a point where it cannot improve anymore considering the limitations of its expressive power (i.e. its ability to express the relationship between input data and target). In the given case, the error seems to decrease even at the very right part of the curve, which means that the model would probably be still improving with more training data (or when learning the training data set again in another epoch). However, because of the chosen training data set and because

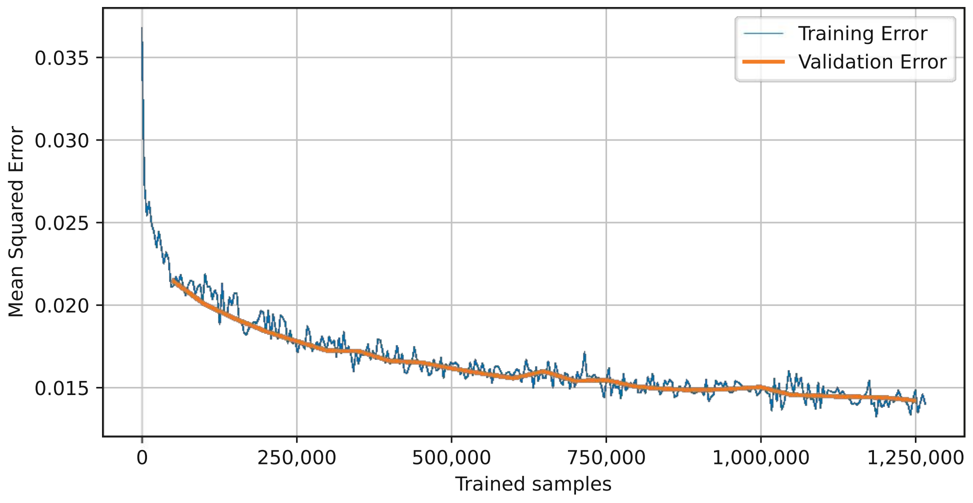


Figure 5.2: Comparison of MSEs when predicting feature "Danceability" on the training data set vs. on the validation data set in relation to the number of data points already trained. The predictions are made with the ResNet-34 model with learning rate 0.0001 and Adam optimization.

of the fact that the experiments conducted as part of this work are designed to include only one training epoch, the training process is stopped after learning the training data once, even though the model probably might have been improved with more training. As can be seen, both curves have a similar shape and the training and validation errors are approximately the same throughout the entire process of training the model. This usually means that the model performance when predicting unseen data is not worse than when predicting data that was used for model training and the model is expected to generalize well on unseen data.

The ResNet-34 model with Adam optimization and learning rate 0.0001 yields an MSE of ~ 0.009 on the test data set.

Acousticness

The results of the experiments that study to what extent it is possible to reproduce Spotify's algorithms that calculate music feature "Acousticness" are given in Table 5.2. As described in Section 4.4 the experiment setup is the same for the experiments for feature "Danceability". Also, the resulting prediction performances of the trained models show similarities with the models that predict feature "Danceability" in that the ResNet-34 models perform best regarding MSE. The overall best performing model is the one with ResNet-34 architecture, learning rate 0.001 and Adam optimization algorithm yielding an MSE of ~ 0.046 on the validation data set. All LeNet-5 and CRNN models perform worse than the ResNet-34 models and the models that use Adam optimization seem to

Experiment Results					
CNN architecture	Learning rate	Optimization algorithm	MAE	MSE	R^2 Score
LeNet-5	SGD	0.0001	∼ 0.172	∼ 0.052	∼ 0.599
LeNet-5	SGD	0.0010	∼ 0.159	∼ 0.050	∼ 0.610
LeNet-5	Adam	0.0001	∼ 0.176	∼ 0.062	∼ 0.524
LeNet-5	Adam	0.0010	∼ 0.166	∼ 0.051	∼ 0.602
ResNet-34	SGD	0.0001	∼ 0.156	∼ 0.052	∼ 0.600
ResNet-34	SGD	0.0010	∼ 0.097	∼ 0.021	∼ 0.835
ResNet-34	Adam	0.0001	∼ 0.093	∼ 0.019	∼ 0.853
ResNet-34	Adam	0.0010	∼ 0.093	∼ 0.018	∼ 0.860
CRNN	SGD	0.0001	∼ 0.171	∼ 0.056	∼ 0.568
CRNN	SGD	0.0010	∼ 0.168	∼ 0.055	∼ 0.579
CRNN	Adam	0.0001	∼ 0.179	∼ 0.057	∼ 0.560
CRNN	Adam	0.0010	∼ 0.182	∼ 0.062	∼ 0.523

Table 5.2: Experiment results for feature "Acousticness"

be consistently better than those using SGD across all other hyperparameter settings.

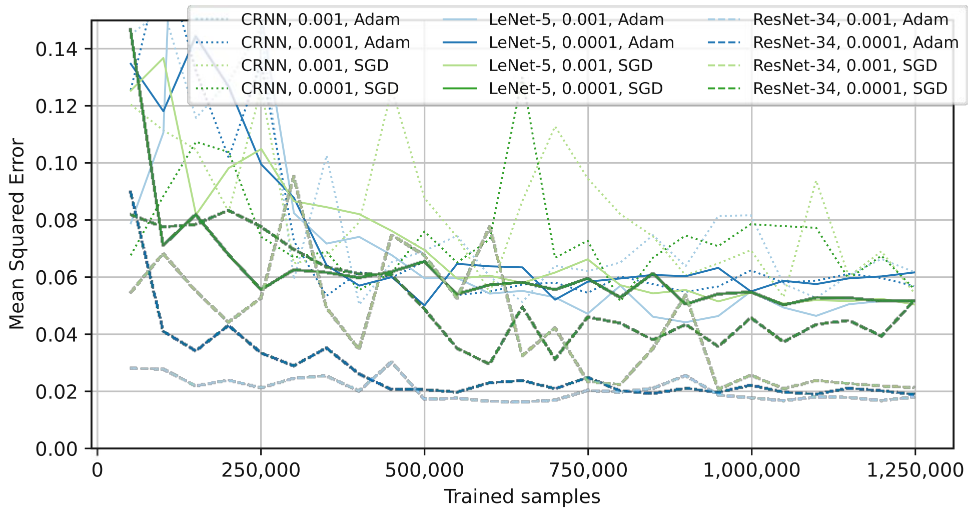


Figure 5.3: MSEs for the different models when predicting feature "Acousticness" on the validation data set in relation to the number of data points already trained.

Figure 5.1 shows the MSE for the different models when predicting the validation data set in relation to the number of data points already trained. As can be seen at the beginning of training all models have relatively high errors compared to the errors after training. Moreover, the two ResNet-34 models with Adam optimization show a relatively low MSE already early in the training process.

The ResNet-34 model with Adam optimization and learning rate 0.001 yields an MSE of ∼ 0.015 on the test data set.

Instrumentalness

Experiment Results					
CNN architecture	Learning rate	Optimization algorithm	MAE	MSE	R^2 Score
LeNet-5	SGD	0.0001	~ 0.261	~ 0.101	~ 0.224
LeNet-5	SGD	0.0010	~ 0.338	~ 0.174	~ -0.343
LeNet-5	Adam	0.0001	~ 0.263	~ 0.109	~ 0.163
LeNet-5	Adam	0.0010	~ 0.257	~ 0.103	~ 0.208
ResNet-34	SGD	0.0001	~ 0.170	~ 0.055	~ 0.580
ResNet-34	SGD	0.0010	~ 0.178	~ 0.058	~ 0.556
ResNet-34	Adam	0.0001	~ 0.150	~ 0.045	~ 0.655
ResNet-34	Adam	0.0010	~ 0.148	~ 0.040	~ 0.690
CRNN	SGD	0.0001	~ 0.267	~ 0.104	~ 0.201
CRNN	SGD	0.0010	~ 0.256	~ 0.098	~ 0.244
CRNN	Adam	0.0001	~ 0.278	~ 0.109	~ 0.157
CRNN	Adam	0.0010	~ 0.257	~ 0.103	~ 0.204

Table 5.3: Experiment results for feature “Instrumentalness”

Table 5.3 contains the results of the Single-Task Learning (STL) experiments for feature “Instrumentalness”. The prediction performances shown in the table show that the best performing models are the ones with ResNet-34 architecture. The overall best model is the ResNet-34 model with learning rate 0.001 and Adam optimization, which yields an MSE of ~ 0.04 when predicting the validation set. All models with LeNet-5 or CRNN architectures perfrom worse than any of the ResNet-34 models. The R^2 Scores of the LeNet-5 and CRNN models show that the MSEs of these models are only slightly lower than the variance $\mathrm{Var}(X) \approx 0.122$ of feature “Instrumentalness”. This means that a trivial model, which would always predict the mean value of feature “Instrumentalness” (and therefore yield an MSE $= \mathrm{Var}(X)$), would only be slightly worse than the LeNet-5 and CRNN models. From Table 5.3 no relationships between the other hyperparameters and prediction performance can be observed.

Figure 5.4 shows the MSE for the different models when predicting the validation data set in relation to the number of data points already trained. It shows that the ResNet-34 models have better validation errors throughout the training process compared to the LeNet-5 or CRNN models. All models have similar validation errors at the beginning of the training process, which improve relatively fast during the first $\sim 250\,000$ training samples. Later in the training process, the LeNet-5 and CRNN models all seem to have an MSE of ~ 0.11 with a higher variablity as compared to the ResNet-34 models. An MSE of ~ 0.11 is only slightly lower than the variance of $\mathrm{Var}(X) \approx 0.122$ of feature “Instrumentalness”, which is also shown by the R^2 Scores. The curves of the ResNet-34 models with learning rate 0.0001 seem to descent during the entire training process, even at the end of the training (which suggests that these models would likely still be improving with more training).

The ResNet-34 model with Adam optimization and learning rate 0.001 yields an MSE of ~ 0.041 on the test data set.

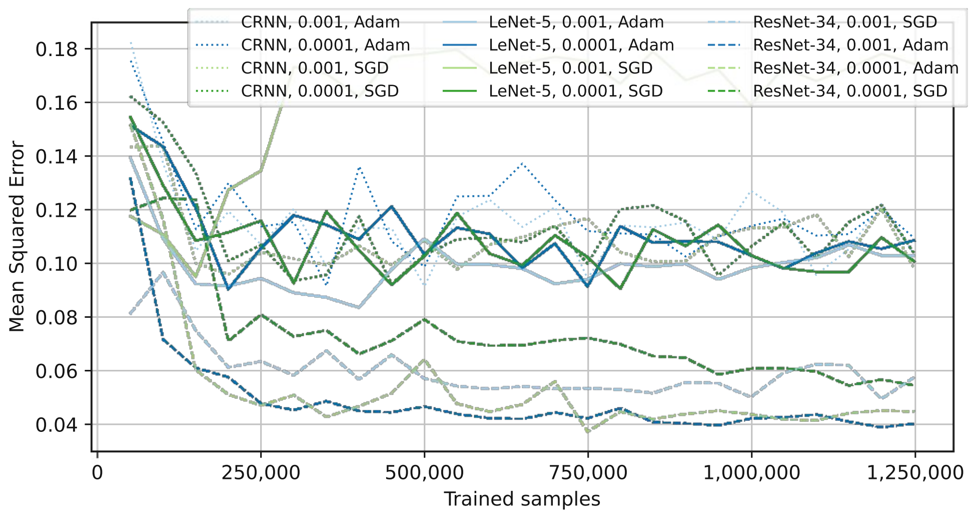


Figure 5.4: MSEs for the different models when predicting feature “Instrumentalness” on the validation data set in relation to the number of data points already trained.

Speechiness

Experiment Results					
CNN architecture	Learning rate	Optimization algorithm	MAE	MSE	R^2 Score
LeNet-5	SGD	0.0001	~ 0.076	~ 0.013	~ 0.085
LeNet-5	SGD	0.0010	~ 0.063	~ 0.009	~ 0.328
LeNet-5	Adam	0.0001	~ 0.074	~ 0.011	~ 0.196
LeNet-5	Adam	0.0010	~ 0.071	~ 0.011	~ 0.209
ResNet-34	SGD	0.0001	~ 0.041	~ 0.005	~ 0.627
ResNet-34	SGD	0.0010	~ 0.044	~ 0.005	~ 0.600
ResNet-34	Adam	0.0001	~ 0.040	~ 0.005	~ 0.640
ResNet-34	Adam	0.0010	~ 0.037	~ 0.004	~ 0.687
CRNN	SGD	0.0001	~ 0.083	~ 0.015	~ -0.097
CRNN	SGD	0.0010	~ 0.079	~ 0.014	~ -0.029
CRNN	Adam	0.0001	~ 0.088	~ 0.017	~ -0.231
CRNN	Adam	0.0010	~ 0.079	~ 0.015	~ -0.069

Table 5.4: Experiment results for feature “Speechiness”

The result of the experiments that are described in Section 4.3 are given in Table 5.4 for feature “Speechiness”. Note that compared to the MSE values obtained for the other features described above, the MSEs of “Speechiness” are much lower in general. The main reason for this is that the feature value distribution of “Speechiness” is also narrower than the distribution of other features. The R^2 Scores adjust for these differences in the feature value distributions and show that the CRNN models perfrom worse than the feature’s variance $\mathrm{Var}(X) \approx 0.013$ (which can be interpreted as a trivial model that would always predict the mean value of “Speechiness”) and the LeNet-5 models are only slightly better than the variance. The "Speechiness" model that yields the best prediction performance is the model with ResNet-34 architecture, 0.001 learning rate and Adam optimizer with an MSE of ~ 0.004.

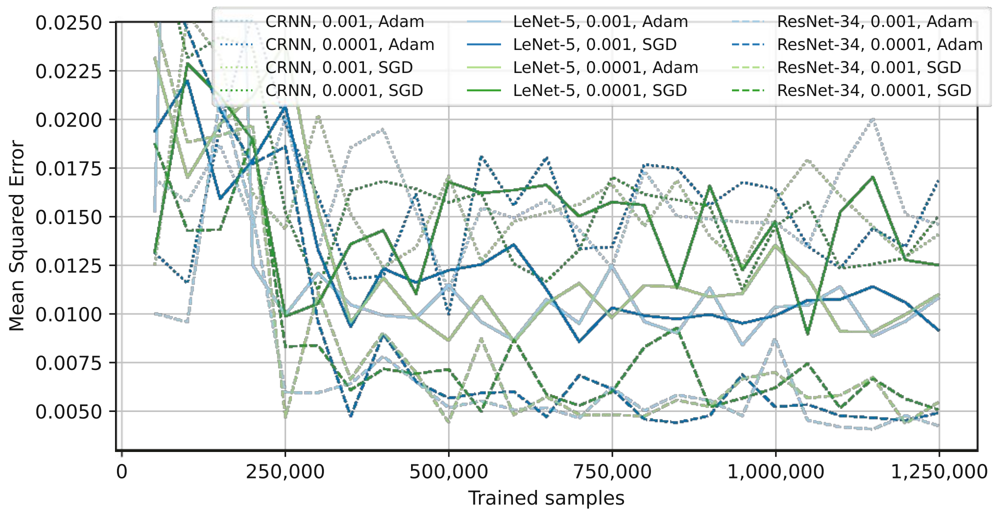


Figure 5.5: MSEs when predicting feature "Speechiness" on the validation data set in relation to the number of data points already trained for the different models.

Figure 5.5 shows curves that represent the validation errors in the course of the training process for the different models that were considered in the experiments. Here, a relationship between the models' CNN architecture and the error during training can be observed. The CRNN models perform the worst, followed by the LeNet-5 models and followed by the Resnet-34 models.

The ResNet-34 model with Adam optimization and learning rate 0.001 yields an MSE of ~ 0.005 on the test data set.

Liveness

Table 5.5 contains the results of the experiments conducted for feature 'Liveness". The model with the lowest MSE of ~ 0.032 is the ResNet-34 model with Adam optimization algorithm and a learning rate of 0.001. However, the other MSE values of the other models on the validation set are not much worse. All models range at an MSE of ~ 0.032 to ~ 0.038, which is only approximately the variance of feature "Liveness" of $\mathrm{Var}(X) \approx 0.036$, yielding R^2 Scores of around 0.

The curves in Figure 5.6 show that the MSEs of the different models obtained on the validation data set during training all reach a value of approximately 0.033 to 0.04 after around 250 000 to 500 000 trained samples and stay in that range throughout the remainder of the training process (with some "outlying" validation MSEs). There is

Experiment Results					
CNN architecture	Learning rate	Optimization algorithm	MAE	MSE	R^2 Score
LeNet-5	SGD	0.0001	~ 0.139	~ 0.038	~ -0.062
LeNet-5	SGD	0.0010	~ 0.136	~ 0.035	~ 0.017
LeNet-5	Adam	0.0001	~ 0.135	~ 0.036	~ -0.003
LeNet-5	Adam	0.0010	~ 0.135	~ 0.035	~ 0.021
ResNet-34	SGD	0.0001	~ 0.123	~ 0.033	~ 0.063
ResNet-34	SGD	0.0010	~ 0.131	~ 0.035	~ 0.006
ResNet-34	Adam	0.0001	~ 0.121	~ 0.033	~ 0.069
ResNet-34	Adam	0.0010	~ 0.123	~ 0.032	~ 0.105
CRNN	SGD	0.0001	~ 0.141	~ 0.038	~ -0.061
CRNN	SGD	0.0010	~ 0.135	~ 0.036	~ -0.010
CRNN	Adam	0.0001	~ 0.136	~ 0.037	~ -0.034
CRNN	Adam	0.0010	~ 0.137	~ 0.038	~ -0.065

Table 5.5: Experiment results for feature “Liveness”

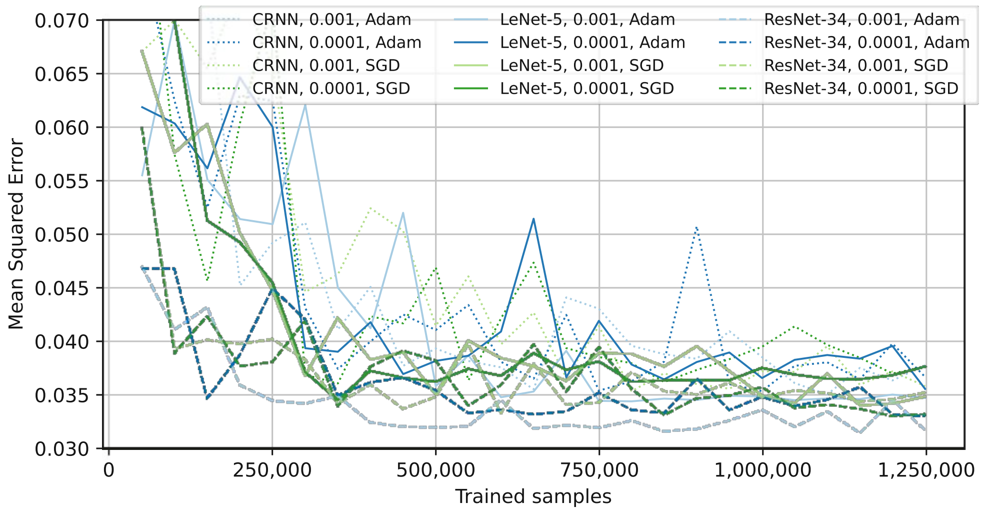


Figure 5.6: MSEs when predicting feature “Liveness” on the validation data set in relation to the number of data points already trained for the different models.

no relationship observable between any of the hyperparameters and the progression of validation MSE or the final MSE.

The ResNet-34 model with Adam optimization and learning rate 0.001 yields an MSE of ~ 0.031 on the test data set.

Valence

Experiment Results					
CNN architecture	Learning rate	Optimization algorithm	MAE	MSE	R^2 Score
LeNet-5	SGD	0.0001	~ 0.226	~ 0.075	~ -0.012
LeNet-5	SGD	0.0010	~ 0.199	~ 0.056	~ 0.237
LeNet-5	Adam	0.0001	~ 0.201	~ 0.059	~ 0.209
LeNet-5	Adam	0.0010	~ 0.193	~ 0.054	~ 0.277
ResNet-34	SGD	0.0001	~ 0.144	~ 0.035	~ 0.530
ResNet-34	SGD	0.0010	~ 0.150	~ 0.036	~ 0.515
ResNet-34	Adam	0.0001	~ 0.131	~ 0.030	~ 0.601
ResNet-34	Adam	0.0010	~ 0.129	~ 0.027	~ 0.636
CRNN	SGD	0.0001	~ 0.235	~ 0.078	~ -0.058
CRNN	SGD	0.0010	~ 0.212	~ 0.066	~ 0.107
CRNN	Adam	0.0001	~ 0.217	~ 0.069	~ 0.071
CRNN	Adam	0.0010	~ 0.231	~ 0.080	~ -0.075

Table 5.6: Experiment results for feature “Valence”

The STL experiment results for feature “Valence” are given in Table 5.6. Here, the model with ResNet-34 architecture, Adam optimizer and learning rate 0.001 performs best, yielding an MSE of ~ 0.027 on the validation set. In general, the ResNet-34 models give MSE values approximately two times lower than the LeNet-5 or CRNN models.

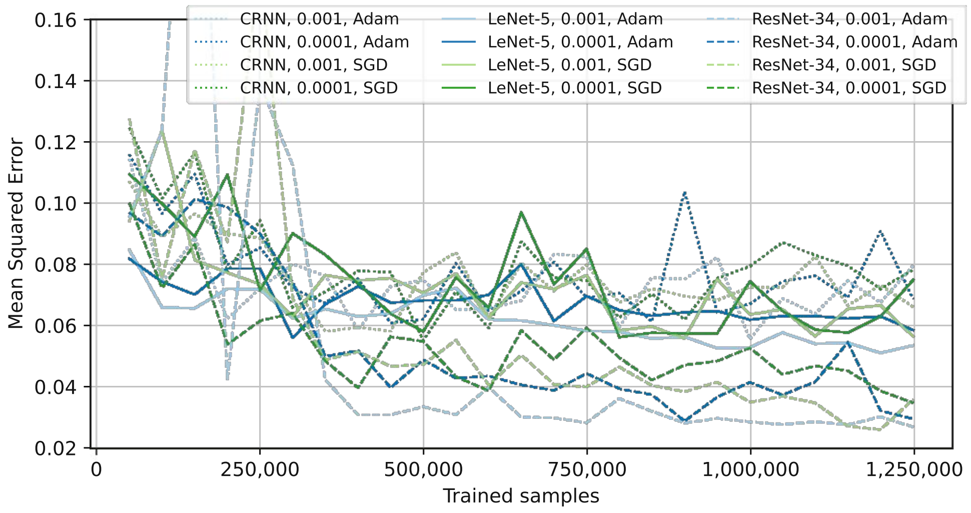


Figure 5.7: MSEs when predicting feature “Valence” on the validation data set in relation to the number of data points already trained for the different models.

Figure 5.7 shows the MSEs of the different models on the validation data set during the training process. The curves representing the LeNet-5 and CRNN models have similar locations and show that the MSEs range from ~ 0.06 to ~ 0.08 during the process of training after the first $\sim 250\,000$ training samples. The MSEs do not seem to improve anymore after the first ~ 250 000 samples. The curves representing the ResNet-34 models show that the MSE values of these models seem to improve throughout the entire training process.

The ResNet-34 model with Adam optimization and learning rate 0.001 yields an MSE of ~ 0.027 on the test data set.

Energy

Experiment Results					
CNN architecture	Learning rate	Optimization algorithm	MAE	MSE	R^2 Score
LeNet-5	SGD	0.0001	~ 0.124	~ 0.024	~ 0.679
LeNet-5	SGD	0.0010	~ 0.127	~ 0.024	~ 0.672
LeNet-5	Adam	0.0001	~ 0.135	~ 0.028	~ 0.615
LeNet-5	Adam	0.0010	~ 0.119	~ 0.022	~ 0.694
ResNet-34	SGD	0.0001	~ 0.090	~ 0.013	~ 0.823
ResNet-34	SGD	0.0010	~ 0.092	~ 0.014	~ 0.809
ResNet-34	Adam	0.0001	~ 0.104	~ 0.018	~ 0.754
ResNet-34	Adam	0.0010	~ 0.099	~ 0.016	~ 0.775
CRNN	SGD	0.0001	~ 0.211	~ 0.067	~ 0.092
CRNN	SGD	0.0010	~ 0.196	~ 0.058	~ 0.211
CRNN	Adam	0.0001	~ 0.190	~ 0.057	~ 0.227
CRNN	Adam	0.0010	~ 0.212	~ 0.067	~ 0.086

Table 5.7: Experiment results for feature "Energy"

Table 5.7 shows that the prediction performances of the considered models are greatly dependent on the choice of the network architecture with the ResNet-34 models yielding the best results, followed by the LeNet-5 models and followed by the CRNNs. The overall best performance is achieved by the ResNet-34 model with SGD optimization and a learning rate of 0.0001 with an MSE of ~ 0.013.

Figure 5.8 shows the models' validation MSEs during training. Here, a clear difference between the performances of the CRNN models and the other models can be seen in that the CRNN models have much worse prediction performances during training and also higher variabilities of the validation curves than most other models. The CRNN models do not seem to improve at all during training.

The ResNet-34 model with Adam optimization and learning rate 0.001 yields an MSE of ~ 0.010 on the test data set.

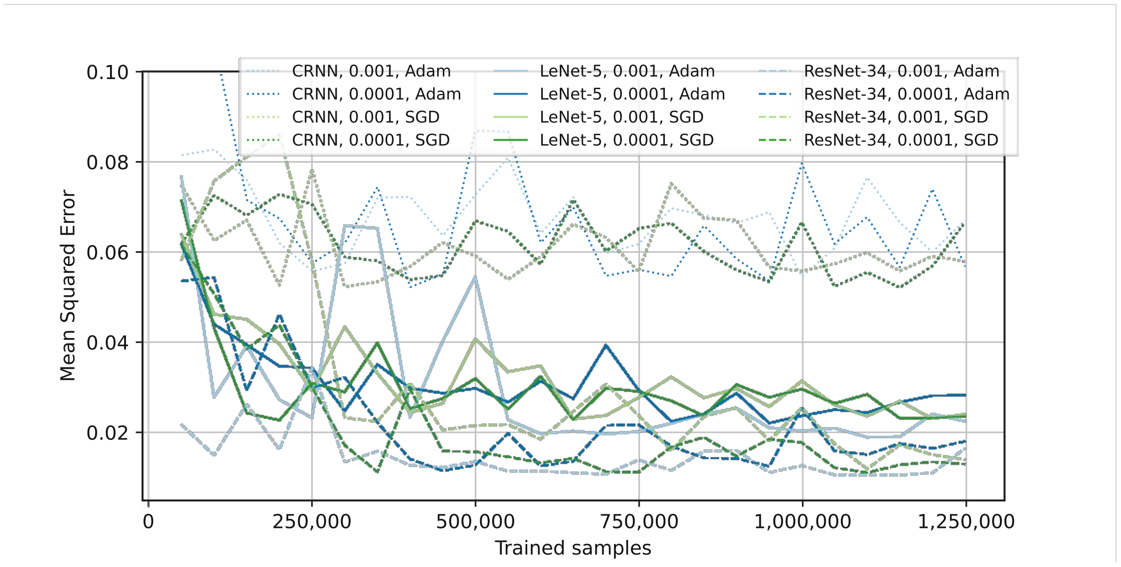


Figure 5.8: MSEs when predicting feature “Energy” on the validation data set in relation to the number of data points already trained for the different models.

Loudness

Experiment Results					
CNN architecture	Learning rate	Optimization algorithm	MAE	MSE	R^2 Score
LeNet-5	SGD	0.0001	∼ 1.911	∼ 7.163	∼ 0.779
LeNet-5	SGD	0.0010	∼ 1.738	∼ 5.981	∼ 0.815
LeNet-5	Adam	0.0001	∼ 1.818	∼ 6.045	∼ 0.813
LeNet-5	Adam	0.0010	∼ 1.901	∼ 6.319	∼ 0.805
ResNet-34	SGD	0.0001	∼ 1.429	∼ 5.314	∼ 0.836
ResNet-34	SGD	0.0010	∼ 1.936	∼ 9.557	∼ 0.705
ResNet-34	Adam	0.0001	∼ 1.535	∼ 5.812	∼ 0.820
ResNet-34	Adam	0.0010	∼ 1.443	∼ 5.500	∼ 0.830
CRNN	SGD	0.0001	∼ 1.907	∼ 6.504	∼ 0.799
CRNN	SGD	0.0010	∼ 1.695	∼ 6.026	∼ 0.814
CRNN	Adam	0.0001	∼ 1.887	∼ 6.689	∼ 0.793
CRNN	Adam	0.0010	∼ 1.868	∼ 6.649	∼ 0.795

Table 5.8: Experiment results for feature “Loudness”

The experiment results for feature “Loudness” are given in Table 5.8. The ResNet-34 model using SGD optimization with learning rate 0.0001 and the two ResNet-34 models with Adam optimizer show relatively good prediction performance on the validation set compared to the other models. Especially, the ResNet-34 with SGD optimization and a learning rate of 0.001 has a noticeably worse MSE than any of the other considered models.

Figure 5.9 shows curves that represent the validation errors in the course of the training

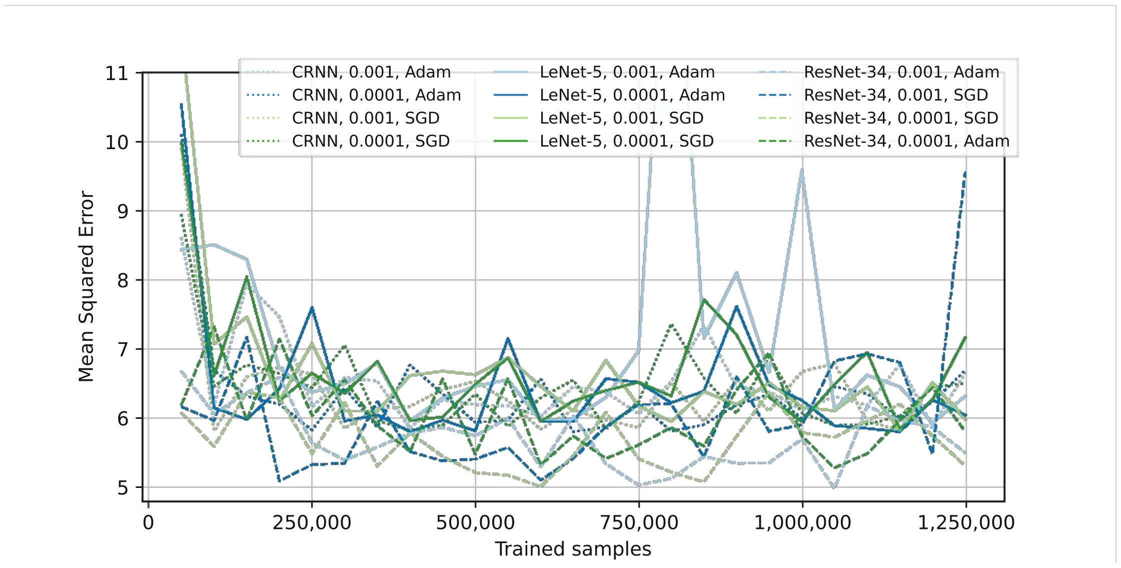


Figure 5.9: MSEs when predicting feature "Loudness" on the validation data set in relation to the number of data points already trained for the different models.

process for the different models that were considered in the experiments. Interestingly, the graph representing the ResNet-34 model with SGD optimizer and learning rate 0.001, which yielded the worse prediction performance on the validation set as stated in Table 5.8, performed relatively well when validated during the training process, but had a significantly worse MSE after training was finished. The reason for this is not evident, but is likely to be caused by random fluctuations of the MSE during adaptation of the network's weights. It is also noticable that the graph representing MSEs of the LeNet-5 model with Adam optimizer and learning rate 0.001 shows two relatively high peaks during the process of training.

The ResNet-34 model with Adam optimization and learning rate 0.001 yields an MSE of ~ 5.608 on the test data set.

5.2.2 Multi-Task Learning Results

In the following, the results of the Multi-Task Learning (MTL) experiments conducted as part of this work are given. These experiments are carried out in order to be able to answer research question RQ3, which addresses the comparison of the results described in Section 5.2.1 above with an MTL approach.

The data preprocessing for the MTL experiments is done the same way as described in Section 5.1. The CNN architecture used here is the one described in Section 3.4.3. All models were trained using Adam optimization with a learning rate of 0.001 and without loss weighting.

As described in Section 4.3 the combinations of features that are considered in the MTL experiments are features “Danceability” and “Valence”, features “Acousticness” and “Energy”, features “Instrumentalness” and “Speechiness” and features “Liveness” and “Loudness”. For each of these combinations, a CNN model with two heads is trained. The resulting models are then evaluated on the test data set in order to estimate a final model’s prediction performance.

Experiment Results				
Tasks	Feature	MAE	MSE	R^2 Score
“Danceability”, “Valence”	“Danceability”	~ 0.094	~ 0.015	~ 0.565
	“Valence”	~ 0.173	~ 0.047	~ 0.366
“Acousticness”, “Energy”	“Acousticness”	~ 0.144	~ 0.038	~ 0.703
	“Energy”	~ 0.126	~ 0.024	~ 0.675
“Instrumentalness”, “Speechiness”	“Instrumentalness”	~ 0.212	~ 0.082	~ 0.358
	“Speechiness”	~ 0.044	~ 0.009	~ 0.321
“Liveness”, “Loudness”	“Liveness”	~ 0.128	~ 0.034	~ 0.020
	“Loudness”	~ 3.977	~ 24.359	~ 0.264

Table 5.9: MTL experiment results. The left column contains the music features that have been learned simultaneously, the three right columns contain the prediction performances for the feature given in the second column from the left.

“Danceability” - “Valence”

As shown in Table 5.9, the MTL model trained to extract features “Danceability” and “Valence” yields an MSE of ~ 0.015 when predicting feature “Danceability” on the test data set and an MSE of ~ 0.047 when predicting feature “Valence”.

Compared to the result obtained from the best STL models described in Section 5.2.1, which yield an MSE of 0.009 when predicting feature “Danceability” on the test data set, the MTL approach achieved worse prediction performance. Also compared to the best STL model for predicting feature “Valence” yielding an MSE of ~ 0.027, the prediction performance of the MTL approach is worse.

“Acousticness” - “Energy”

The MTL model that attempts to reproduce the feature extraction algorithms for features “Acousticness” and “Energy” achieves an MSE of ~ 0.038 for feature “Acousticness” and an MSE of ~ 0.018 for feature “Energy”. Also both “Acousticness” and “Energy” have worse errors than respective best STL models.

“Instrumentalness” - “Speechiness”

As can be seen from Table 5.9, the MTL model that was trained to predict features “Instrumentalness” and “Speechiness” yields an MSE of ~ 0.082 for “Instrumentalness” and ~ 0.009 for “Speechiness”. Also here, the results obtained from the MTL approach are worse than the respective best STL results, but better than the worst STL models.

“Liveness” - “Loudness”

As shown in Table 5.9, the MTL approach for feature extraction of “Liveness” and “Loudness” yields an MSE of ~ 0.034 for “Liveness” and ~ 24.359 for “Loudness”.

In comparison with the results obtained from the STL approaches for learning features “Liveness” as described above, the MTL has a higher error than the best STL model, but is better than the LeNet-5 and CRNN models (compared when evaluating on the validation data set). Compared with the STL result for “Loudness”, the MTL approach is worse than any of the STL models.

5.3 Discussion

The results described in Section 5.2 show that the prediction performances of the machine learning models that were considered in the experiments differ substantially across the music features that are subject of this work. Such a comparison of the quality of the feature extraction across the different music features is only reasonable when comparing R^2 Scores, which adjusts the MSEs by the features’ variances, which of course differ across the features.

The final model for prediction of features “Acousticness”, “Energy” and “Loudness” achieve an R^2 Score greater than 0.8 and the final model for features “Danceability”, “Instrumentalness”, “Speechiness” and “Valence” an R^2 Score between 0.5 and 0.8, whereas the final model that predicts feature “Liveness” shows an R^2 Score of only 0.122. This shows that Spotify’s “black-box” algorithms that calculate features “Danceability”, “Acousticness”, “Instrumentalness”, “Speechiness”, “Valence”, “Energy” and “Loudness” can be reproduced quite well using spectrogram representations of audio samples compared to the algorithm calculating feature “Liveness”. The latter one can only be reproduced marginally better than a trivial model that would always just predict a feature values’ mean (which would yield an R^2 Score of 0).

A comparison of the behavior of the individual prediction performances when varying the different hyperparameters that are considered in the STL experiments described above shows a pattern, which holds for almost all the music features subject of this work. It shows that any of the ResNet-34 models is able to outperform all the LeNet-5 or CRNN models (with the exception of feature “Loudness”). Both, the LeNet-5 and the CRNN architecture, have in common that they are generally smaller architectures (i.e. architectures with less Convolutional Layers) than the ResNet-34 architecture. This suggests that the size of a neural network might determine its capability of extracting the music feature considered in the experiments. This would also be suggested in comparison of the MTL results with those of the ResNet-34 models, since the neural network employed in the MTL experiments also has fewer layers than ResNet-34 and the ResNet-34 models outperformed the MTL results with respect to any of the investigated music features. However, note that there might also be different reasons appearing in correlation with the choice of the neural network architecture rather than the size of a neural network.

Also note that these observations only hold true with respect to the other parameter settings that were considered as part of the experiments. This means in particular that the settings of other experiment variables might have caused the LeNet-5 or CRNN models to perform worse than the ResNet-34 models, and by using different settings of e.g. the learning rate or the optimization algorithm (or any other control variable used in data preprocessing or training) the LeNet-5 or CRNN models might have performed differently in comparison with the results achieved by the ResNet-34 models. In order to be able to draw a conclusion about the question of whether the number of layers or other characteristics of the considered models caused the ResNet-34 models to perform better in general, further investigation of the impact of the number of layers would have been necessary.

CHAPTER 6

Conclusion

This work is concerned with a set of semantic music features that is published for every song available on Spotify. As described in Section 1.1 this set of music features was used in multiple research works in the field of Music Information Retrieval (MIR) in the past and therefore gained scientific relevance. The aim of this work is to explore the characteristics of these music features and especially to what extent it is possible to reproduce the algorithms Spotify uses for the calculation of these features using state-of-the-art methods of MIR. The research questions defined in Section 1.2 are supposed to frame the scope of this work and to specify its aim. In order to be able to give scientifically sound answers to the research questions, a series of experiments is designed and conducted. A detailed description of the experimental setup is described in Chapter 4.

The results of the Single-Task Learning (STL) experiments show which prediction performances can be obtained when predicting the music features published by Spotify that are subject of this work. As the experiment results stated in Section 5.2 show, feature “Danceability” can be predicted with an Mean Squared Error (MSE) of ~ 0.009, “Acousticness” with ~ 0.015, “Instrumentalness” with ~ 0.041, “Speechiness” with ~ 0.005, “Liveness” with ~ 0.031, “Valence” with ~ 0.027, “Energy” with ~ 0.010, and feature “Loudness” with ~ 5.608. These measurements give an answer to research question RQ1 as defined in Section 1.2. Because of the fact that the test data set that was used in the experiments to measure the final models’ results for each of the features represents a broad range of music (to the extent it is offered by Spotify) and because the machine learning algorithms that were applied in the experiments can be considered as *state of the art*, the prerequisites of “using a test data set that covers a broad range of music styles and genres” and “state-of-the-art machine learning algorithms” that are defined in research question RQ1 are arguably met by the experiments. A description of the contents of the data set that is used in this work and basis of the experimental results is given in Section 4.1 and is intended to argue that the test data set actually “covers a broad range of music styles and genres”. The explanations about the state of the art in MIR and especially the state of the art in deep learning for music feature learning is attempting to act as a justification of the claim that the machine learning algorithms used in the experiments are indeed *state of the art*.

A comparison of the different model hyperparameters that are considered in the machine learning experiments suggests that there is a correlation between some of the chosen hyperparameters and the performance obtained from the models on the task of predicting Spotify's music features that are subject of this work. The hyperparameters that are tuned in the STL experiments are the Convolutional Neural Network (CNN) architecture, the optimization algorithm and the learning rate. For all considered features (with the exception of feature "Loudness") the choice of the network architecture has a noticable impact on the resulting model performance. The experiments show that the models that apply ResNet-34 architecture are able to outperform the other CNN architectures under consideration with regards to MSE. The exact reason for this cannot be derived from the conducted experiments. One potential explanation for the fact that the considered models with ResNet-34 architecture performed better than the other models could be that the depth of the employed CNN has an impact on the resulting model performance. It is common that CNNs with more hidden layers are able to achieve better prediction results because of the fact that with more hidden layers more complex internal feature representations can be expressed by a network and accordingly more complex relationships between input and output can be mapped.

The other model hyperparameters that are tuned as part of the conducted experiments do not show any relationship with the resulting models' ability to reproduce the algorithms for calculating the music features at hand. These observations give an answer to research question RQ2, which addresses the comparison of the influence of different model parameters on the task of predicting these music features with regards to the models' MSE.

Besides the experiments conducted in this work, which are concerned with the assessment of the abilities to reproduce Spotify's algorithms for calculating the music features using STL CNN models, a series of experiments is conducted, which assesses the capabilities of Multi-Task Learning (MTL) for this purpose. In comparison with the prediction performances obtained using STL, the MTL models generally yield worse results for all considered music features. The exact reason for this cannot be derived from the experiment results. One potential explanation would be that the smaller size of the used MTL networks has a disadvantage in comparison with the larger ResNet-34 architecture used in the STL experiments. This would also be indicated by the fact that for some of the music features the MTL models yield equal performance as the other network architectures considered in the STL experiments, which have a size similar to the MTL architecture. Another explanation would be that with MTL it is simply not possible to outperform STL on the machine learning tasks at hand regardless of the size of the used networks.

It should be noted that the approaches considered in this work for predicting the music features are only a selection of possible approaches and that there would be a wide variety of other approaches that is left out of consideration. The selection of approaches considered in this work is made in order to keep the extent of the experiments in a feasible scope. However, it remains unclear whether other approaches could have yielded better results in the task of predicting the music features. It is very likely that with other approaches or, for example, by further tuning the considered hyperparameters, better prediction performance results could have been obtained. The results actually obtained in this work yet give an approximate measurement of well each of the music feature can be extracted.

Also, it should be noted that the aim of this work, which is to study to what extent it is possible to reproduce the algorithms Spotify uses for calculation of the music features, is limited with respect to the fact that a number of premises are made. One such premise is that the machine learning models for reproducing the feature extraction algorithms only rely on spectrogram representations of audio signals as their input. This work does not address other types of music descriptors that could be suitable for prediction of the music features subject of this work. Furthermore, the audio signals only cover a sample of each song of 30 seconds. It is evident that an audio signal covering an entire song would contain more information that could be relevant for extraction of the music features.

Moreover, it should be noted that the music feature values used in this work, which are offered by Spotify and were downloaded via a web API from Spotify, may change as Spotify changes their algorithms that calculate these music features. Because of the fact that the feature values were only downloaded once for the purpose to be used as part of this work, this work considers only a "snapshot" of the feature values that might change over time. This of course limits the validity of the findings of this work, since the gained insights might not be true anymore as Spotify changes the algorithms for calculating the music features. On the other hand, the reproduced feature extraction algorithms provide a stable version of the set of features, which does not change even if Spotify would change their feature extraction algorithms. For this reason, the algorithms that are produced as part of this work might be particularly valuable for other research works that use these music features because they ensure reproducibility, which is not given by Spotify's verions of the feature extraction algorithms.

In conclusion, this work gives a characterization of a set of semantic music features, which appear to be scientifically relevant in the research field of MIR, and it attempts to study to what extent these music features can be extracted from audio signals and to reproduce the algorithms originally used to calculate these features. The results of this attempt shows that it is indeed possible to perform the extraction of these features to a certain degree using the methods applied in this work.

As described above, the findings of this work underlie certain limitations. Apart from the limited validity of the findings that is caused by these limitations, they also show that this work leaves room for future works, which could extend the research efforts already made by this work. For example, a possible future work could study other music descriptors, whether contextual or other audio representations, in order to further improve the extraction of the features. Also, an analysis of how well the features can be learned when using the audio signals of entire songs rather than only samples - as it is done in this work - would help to gain further insights.

List of Figures

[1]https://developer.spotify.com/documentation/web-api/reference/, accessed 2022-11-26

List of Tables

Bibliography

[1] Sebastian Böck, Matthew E. P. Davies, and Peter Knees. Multi-task learning of tempo and beat: Learning one to improve the other. In *Proceedings of the 20th International Society for Music Information Retrieval Conference*, 2019.

[2] Rich Caruana. Multitask learning. *Machine Learning*, 28, 07 1997.

[3] Chih-Ming Chen, Ming-Feng Tsai, Jen-Yu Liu, and Yi-Hsuan Yang. Using emotional context from article for contextual music recommendation. In *Proceedings of the 21st ACM International Conference on Multimedia*, MM '13, page 649–652, New York, NY, USA, 2013. Association for Computing Machinery.

[4] Ching-Wei Chen, Paul Lamere, Markus Schedl, and Hamed Zamani. Recsys challenge 2018: Automatic music playlist continuation. In *Proceedings of the 12th ACM Conference on Recommender Systems*, RecSys '18, page 527–528, New York, NY, USA, 2018. Association for Computing Machinery.

[5] Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder–decoder approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111, Doha, Qatar, October 2014. Association for Computational Linguistics.

[6] Keunwoo Choi. *Deep neural networks for music tagging*. PhD thesis, Queen Mary University of London, UK, 2018.

[7] Keunwoo Choi, György Fazekas, Kyunghyun Cho, and Mark B. Sandler. A tutorial on deep learning for music information retrieval. *CoRR*, abs/1709.04396, 2017.

[8] Keunwoo Choi, György Fazekas, Mark Sandler, and Kyunghyun Cho. Convolutional recurrent neural networks for music classification. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2392–2396, 2017.

[9] Yandre M. G. Costa, Luiz S. Oliveira, Alessandro L. Koericb, and Fabien Gouyon. Music genre recognition using spectrograms. In *2011 18th International Conference on Systems, Signals and Image Processing*, pages 1–4, 2011.

[10] Yandre M.G. Costa, Luiz S. Oliveira, and Carlos N. Silla. An evaluation of convolutional neural networks for music classification using spectrograms. *Applied Soft Computing*, 52:28–38, 2017.

[11] Y.M.G. Costa, L.S. Oliveira, A.L. Koerich, F. Gouyon, and J.G. Martins. Music genre classification using lbp textural features. *Signal Processing*, 92(11):2723–2737, 2012.

[12] Simon Durand, Juan Pablo Bello, Bertrand David, and Gaël Richard. Robust downbeat tracking using an ensemble of convolutional networks. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 25(1):76–89, 2017.

[13] Mohammad Sadegh Ebrahimi and Hossein Karkeh Abadi. Study of residual networks for image recognition. In Kohei Arai, editor, *Intelligent Computing*, pages 754–763, Cham, 2021. Springer International Publishing.

[14] Jonathan T Foote. Content-based retrieval of music and audio. In *Multimedia Storage and Archiving Systems II*, volume 3229, pages 138–147. International Society for Optics and Photonics, 1997.

[15] Ting Gong, Tyler Lee, Cory Stephenson, Venkata Renduchintala, Suchismita Padhy, Anthony Ndirango, Gokce Keskin, and Oguz H. Elibol. A comparison of loss weighting strategies for multi task learning in deep neural networks. *IEEE Access*, 7:141627–141632, 2019.

[16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[17] M. Goto and Y. Muraoka. A beat tracking system for acoustic signals of music. In *Proceedings of the Second ACM International Conference on Multimedia*, MULTIMEDIA '94, page 365–372, New York, NY, USA, 1994. Association for Computing Machinery.

[18] Sophia Hadash, Yu Liang, and Martijn C. Willemsen. How playlist evaluation compares to track evaluations in music recommender systems. In *IntRS 2019 Interfaces and Human Decision Making for Recommender Systems 2019*, CEUR Workshop Proceedings, pages 1–9, January 2019.

[19] Donghong Han, Yanru Kong, Jiayi Han, and Guoren Wang. A survey of music emotion recognition. *Frontiers of Computer Science*, 16, 12 2022.

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[21] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 11 1997.

[22] Thomas Hodgson. Spotify and the democratisation of music. *Popular Music*, 40(1):1–17, 2021.

[23] Andre Holzapfel and Thomas Grill. Bayesian Meter Tracking on Learned Signal Representations. In *Proceedings of the 17th International Society for Music Information Retrieval Conference*, pages 262–268, New York City, United States, August 2016. ISMIR.

[24] Eric J. Humphrey, Juan Pablo Bello, and Yann LeCun. Moving Beyond Feature Design: Deep Architectures and Automatic Feature Learning in Music Informatics. In *Proceedings of the 13th International Society for Music Information Retrieval Conference*, pages 403–408, Porto, Portugal, October 2012. ISMIR.

[25] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[26] Peter Knees and Markus Schedl. A survey of music similarity and recommendation from music context data. *ACM Trans. Multimedia Comput. Commun. Appl.*, 10(1), dec 2013.

[27] Peter Knees and Markus Schedl. *Music Similarity and Retrieval - An Introduction to Audio- and Web-based Strategies*, volume 36 of *The Information Retrieval Series*. Springer, 2016.

[28] Filip Korzeniowski and Gerhard Widmer. End-to-end musical key estimation using a convolutional neural network. In *First published in the Proceedings of the 25th European Signal Processing Conference*, pages 966–970. EURASIP, 08 2017.

[29] Yizhar Lavner and Dima Ruinskiy. A decision-tree-based algorithm for speech/music classification and segmentation. *EURASIP Journal on Audio, Speech, and Music Processing*, 2009:1–14, 2009.

[30] Y. LeCun. Generalization and network design strategies. In R. Pfeifer, Z. Schreter, F. Fogelman, and L. Steels, editors, *Connectionism in Perspective*, Zurich, Switzerland, 1989. Elsevier.

[31] S. Liang, C. Deng, and Y. Zhang. A simple approach to balance task loss in multi-task learning. In *2021 IEEE International Conference on Big Data (Big Data)*, pages 812–823, Los Alamitos, CA, USA, dec 2021. IEEE Computer Society.

[32] Yu Liang and Martijn C. Willemsen. Personalized recommendations for music genre exploration. In *Proceedings of the 27th ACM Conference on User Modeling, Adaptation and Personalization*, UMAP '19, page 276–284, New York, NY, USA, 2019. Association for Computing Machinery.

[33] Kehan Luo. Machine learning approach for genre prediction on spotify top ranking songs. Master's thesis, University of North Carolina at Chapel Hill, 2008.

[34] Michael I Mandel, Graham E Poliner, and Daniel PW Ellis. Support vector machine active learning for music retrieval. *Multimedia systems*, 12(1):3–13, 2006.

[35] Alessandro B. Melchiorre and Markus Schedl. Personality correlates of music audio preferences for modelling music listeners. In *Proceedings of the 28th ACM Conference on User Modeling, Adaptation and Personalization*, UMAP '20, page 313–317, New York, NY, USA, 2020. Association for Computing Machinery.

[36] Martijn Millecamp, Nyi Nyi Htun, Yucheng Jin, and Katrien Verbert. Controlling spotify recommendations: Effects of personal characteristics on music recommender user interfaces. In *Proceedings of the 26th Conference on User Modeling, Adaptation and Personalization*, UMAP '18, page 101–109, New York, NY, USA, 2018. Association for Computing Machinery.

[37] Meinard Müller. *Fundamentals of Music Processing - Audio, Analysis, Algorithms, Applications*. Springer, 2015.

[38] Loris Nanni, Yandre M.G. Costa, Alessandra Lumini, Moo Young Kim, and Seung Ryul Baek. Combining visual and acoustic features for music genre classification. *Expert Systems with Applications*, 45:108–117, 2016.

[39] Zain Nasrullah and Yue Zhao. Music artist classification with convolutional recurrent neural networks. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2019.

[40] Josh Patterson and Adam Gibson. *Deep Learning: A Practitioner's Approach*. O'Reilly, Beijing, 2017.

[41] Herbert E. Robbins. A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407, 2007.

[42] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.

[43] Markus Schedl, Christine Bauer, Wolfgang Reisinger, Dominik Kowald, and Elisabeth Lex. Listener modeling and context-aware music recommendation based on country archetypes. *Frontiers in Artificial Intelligence*, 3, 2021.

[44] Markus Schedl, Emilia Gómez, and Julián Urbano. Music information retrieval: Recent developments and applications. *Foundations and Trends in Information Retrieval*, 8:127–261, 01 2014.

[45] Alexander Schindler, Thomas Lidy, and Andreas Rauber. Comparing shallow versus deep neural network architectures for automatic music genre classification. In

Proceedings of the 9th Forum Media Technology 2016 and 2nd All Around Audio Symposium 2016, St. Pölten, Austria, November 23-24, 2016, volume 1734 of *CEUR Workshop Proceedings*, pages 17–21, 2016.

[46] Hendrik Schreiber and Meinard Müller. A single-step approach to musical tempo estimation using a convolutional neural network. In *Proceedings of the 19th International Society for Music Information Retrieval Conference, ISMIR 2018, Paris, France, September 23-27, 2018*, pages 98–105, 2018.

[47] Yeshwant Singh and Anupam Biswas. Multitask learning based deep learning model for music artist and language recognition. In *Proceedings of the Workshop on Speech and Music Processing 2021*, pages 20–23, NIT Silchar, India, December 2021. NLP Association of India (NLPAI).

[48] Mohamed Sordo. *Semantic annotation of music collections: A computational approach*. PhD thesis, Pompeu Fabra University, Spain, 2012.

[49] Ralf C. Staudemeyer and Eric Rothstein Morris. Understanding LSTM - a tutorial into long short-term memory recurrent neural networks. *CoRR*, abs/1909.09586, 2019.

[50] Karen Ullrich, Jan Schlüter, and Thomas Grill. Boundary detection in music structure analysis using convolutional neural networks. In *Proceedings of the 15th International Society for Music Information Retrieval Conference, ISMIR 2014, Taipei, Taiwan, October 27-31, 2014*, pages 417–422, 2014.

[51] Maksims Volkovs, Himanshu Rai, Zhaoyue Cheng, Ga Wu, Yichao Lu, and Scott Sanner. Two-stage model for automatic playlist continuation at scale. In *Proceedings of the ACM Recommender Systems Challenge 2018*, RecSys Challenge '18, New York, NY, USA, 2018. Association for Computing Machinery.

[52] Jingfeng Wu, Wenqing Hu, Haoyi Xiong, Jun Huan, Vladimir Braverman, and Zhanxing Zhu. On the noisy gradient descent that generalizes as SGD. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 10367–10376. PMLR, 2020.

[53] Yu Zhang and Qiang Yang. An overview of multi-task learning. *National Science Review*, 5(1):30–43, 09 2017.

[54] Yu Zhang and Qiang Yang. A survey on multi-task learning. *IEEE Transactions on Knowledge and Data Engineering*, 34(12):5586–5609, 2022.

[55] Pan Zhou, Jiashi Feng, Chao Ma, Caiming Xiong, Steven Hoi, and E. Weinan. Towards theoretically understanding why SGD generalizes better than ADAM in deep learning. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS'20, Red Hook, NY, USA, 2020. Curran Associates Inc.