On automated clique detection and personalized music generation

Audio Processing and Indexing 2021

Tobias Oberkofler
Leiden institute for advanced computer science - Leiden University
Leiden, Zuid Holland, The Netherlands

Tanya Dimitrov
Leiden institute for advanced computer science - Leiden University
Leiden, Zuid Holland, The Netherlands

Bart de Zoete
Leiden institute for advanced computer science - Leiden University
Leiden, Zuid Holland, The Netherlands

ABSTRACT

The music industry has seen an immense growth in new applications throughout recent years, aided by the advance of available and accessible computing power. Furthermore does the widespread availability of music mean that fans are no longer bound to radio or other forms of linear media but can create their completely personal soundtrack, anytime, anywhere. In this paper we propose and outline a first prototype of a pipeline for automated personalized music generation. The task is to create coherent song covers based on individual musical taste profiles. The pipeline is build upon three main parts: genre classification for automatically creating broad musical "taste profiles", state-of-the-art music generation techniques to transform a piece of music from one genre domain into another and finally cover song detection to create an objective measure of success for the quality of generated cover songs. In our work on automated music creation we are building upon existing open source software for music source separation, timbre transfer and audio mixing such as Demucs, DDSP and Audacity. We find that genre classification and cover song detection can be accomplished to a satisfying degree. We are furthermore able to provide a pipeline to successfully curated coherent cover songs in the style of a personalized genre with a lot of creative variation. Such a pipeline may be used by artists to discover new dimensions of their music in a explorative and playful way.

KEYWORDS

personalized music generation; automatic clique detection; automated audio processing; timbre transfer; music information retrieval

1 INTRODUCTION

The music industry has seen an immense growth in applications throughout recent years, following an unseen advance of computing power and technological abilities leading the way to new forms of music creation, production and marketing. The widespread availability of music means fans are no longer bound to radio or other forms of media but can create their completely personal soundtrack. Typically, users may find their unique "taste profiles" satisfied in several manners, one of which is through the enjoyment of a cover song: a popular song personalized to match an artists unique musical taste by adopting new vocals, instrumentation, tempo and genre.

The curation of a cover song may pose great creative difficulty for an artist, however these difficulties are multiplied when the task is posed to a computer. The automatic generation of a cover song has several layers of complexity as it is posed with several tasks: style translation, music generation and cover similarity detection.

In order to create a unique cover song the music must transcend its original space. One solution for this is the translation of music to a new genre, which can be achieved through a deep understanding of genre features. Genre classification should be considered a crucial part in cover song generation, as it can not only provide genre information but also an understanding of an individuals genre preferences. Many methods exist for it, but classical machine learning methods remain the most popular choice in regards to interpretable results. Such methods usually utilize music features of a song (such as the BPM and timbre) as input, rather than the raw audio. In recent years convolutional neural networks building genre classification on top of visual representations of the raw audio files have proven to be a highly successful alternative to classical methods in terms of the classification performance. Within this paper we will briefly outline and compare both approaches.

Genre translation, and generally music synthesis, are tasks in which there are few available existing models. Most models utilize deep and complex neural networks to understand musical syntax, requiring lengthy training times. Using a mixture of domain knowledge about existing prevalent patterns of a given genre and modern autoencoder networks to perform timbre transfer the complexity of the problem can be significantly reduced.

However, not only is music difficult to synthesize, but it is difficult to evaluate the quality of the generated music. In our case, it is the challenge of determining whether generated music contains enough of the same musical qualities as the original song. As humans this is not a difficult task, given that there is familiarity with the original piece. However, training a model to do so has proven to be difficult. The many creative freedoms that a musician has in their interpretation of a cover song provides a level of complexity in the correct classification of cover songs to the original music. There are several existing methods that evaluate cover song detection, none of which can accurately classify cover songs consistently.

In this paper we propose and outline a first prototype of a pipeline for personalized music generation (outlined in Figure 1). The task is to create coherent song cover based on an individual's musical taste profile. The pipeline is built upon three main parts: genre classification for automatically creating broad musical "taste profiles", state-of-the-art (SOTA) music generation techniques to transform a piece of music from one genre domain into another and finally cover song detection to create an objective measure of success for

the quality of generated cover songs. The presented tool may be used by musicians as a quick form of experimentation with different genres and imagining their musical identities in different roles. As well as by fans as an interactive instrument to translate their favourite tracks into their favourite genres. Providing a novel notion of human computer interactions.

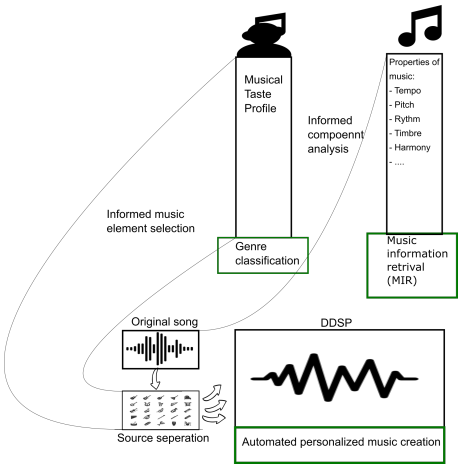

Figure 1: We outline an end-to-end pipeline that strives to consider and utilize both the intrinsic characteristics of a musical piece as well as a listeners personal musical taste profile, in order to adapt an original piece of work and create a new and unique cover version based on the discovered taste profile of a given individual listener.

2 THE DATA

Multiple datasets that were implemented to achieve our research. The datasets that were utilized were the GTZAN Dataset, and the Million Song Dataset, specifically the Tagtraum and SecondHandSongs subdatasets.

GTZAN Dataset . The dataset is composed of 10 different genres, each genre containing 100 audio clips of music. Each audio clip is a 30 second sample of the song. The audio clips are 22050 HZ monophonic files. The genres in the dataset include: blues, classical, country, disco, hiphop, jazz, metal, pop, reggae and rock. The audio clips were collected between the years 2000 and 2001 from personal CD's, radio and microphone recordings.

Million Song Dataset (MSD). The Million Song Dataset [1] is comprised of meta-data features of one million full length tracks. The data contains 53 features, for example: 'artist ID', 'year', 'energy', 'pitch','loudness' and 'timber'. The MSD is composed of several data subsets for different research focuses. In this analysis , we implemented the tagtraum genre annotations subdataset and the SecondHandSongs subdataset.

tagtraum Dataset. The tagtraum dataset [19] contains robust genre annotations to help aid in the genre classification task with the MSD. The MSD contains genre information for all of its tracks from the Last.fm database genre tags and the Top-MAGD Annotations (album level annotations from the All Music Guide). However, genre's are not consistent across both genre-sources and there is no genre hierarchy. The Tagtraum dataset merged several existing genre-labeled datasets to create a new dataset of genre annotations that is more robust than the ones implemented in the MSD. The tagtraum dataset ensures consistent labels and genre taxonomies. We used the CD2C tagtraum dataset that contains 191,401 track-genre labels of songs existing in the MSD. All track meta-features are from the MSD.

SecondHandSongs. The final dataset that was implemented was another MSD data subset,the SecondHandSongs dataset [1]. It contains 18,196 tracks from the Million Song Dataset, organized into 5854 cliqeus of tracks - all tracks of the same cover song. Although there are 5854 available cliques, only 29 cliques exist that contain a sufficient number of cover songs per title (>15). Therefore the data that was used for the cover song classification consisted of 441 tracks organized into 29 cliques. All track meta-features are from the MSD.

3 METHODOLOGY

3.1 Genre Classification

A first step in our pipeline creation is in obtaining a personalized musical taste profile on which later creation of new music can be based on. We reduce the complexity of the very nuanced task of generating musical taste profile by focusing on determining predominant genres in a given individual listening habits. From this predominant genres we hope to obtain a somewhat detailed insight on an individuals musical preferences. One can imagine executing this on his local music folder when then in turn each song could be classified towards a genre and a summary statistic for the available music collection computed. In a similar manner as it can be used to determine musical preference the approach of genre classification could also be used to objectively validate the music generation part of our pipeline by means of evaluating the classifiers confidence. The more confidently our classifier predicts the target genre for a cover, the better the cover corresponds to this genre and hence the closer the expected likeness to non-artificially generated music.

Genre classification is an inherently hard and context depended task. One crucial difficulty of the classification problem is that time changes perception. Hence what used to be considered pop in 1960 might well be considered something completely different when judged in our days. Hence a dataset which spans a long period of time like the million song dataset will always have some internal tension and harmonizing the labeling across decades is a non-trivial task [1, 19]. Furthermore are borders between genres often "fluid" and can not be capped at a clear threshold. Neither do most songs only contain elements of a single genre but are often a mix of various related genres. Lastly it is to mention that datasets found in the real world are rarely balanced. Music genre information is no exception to this as a look at the most popular songs of the last six decades illustrates. Within in this work we do not try to solve this inherent issues of genre classification but create algorithms and tools that tackle the issue with a proper understanding of their limitations.

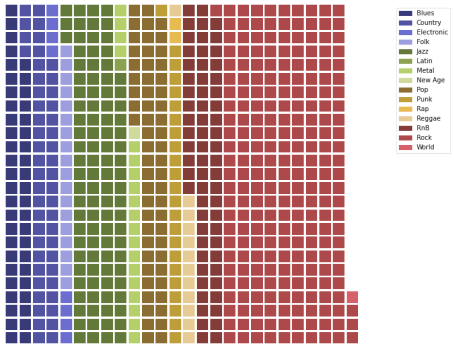

Figure 2: Looking at the most prominent genres in the MSD according to the Tagtraum-labeling we find that especially rock and jazz is predominant while other genres like world are heavily underrepresented.

We benchmark various ways of classification to achieve this goal. First we outline traditional machine learning methods based on pre-computed characteristics of a song such as support vector machines and random forest models.

Secondly we investigate the abilities of several SOTA convolutional neural networks to accuratly predict a given genre-label based on a songs visual timbre and pitch representation.

3.1.1 Random Forests and Support Vector machines. Random Forests are an effective combination of multiple tree models to reduce the overall generalization error of the classifier [4]. They have been successfully applied in various fields of study from ecology to bioinformatics and finance [2]. We also attempted classification with decision trees only, however the results turned out worse than the random forest model in every case which is why we focus only on RF here.

Support Vector machine(SVM) as Introduced by Vapnik et al. [3] is a class of algorithms which work by maximizing the margin of the separating hyperplane to the closest point of each existing class. This optimization procedure can be formulated as a convex problem and hence efficiently solved to optimality. The main statistical "trick" of SVMs is in allowing higher dimensional data embeddings via various specialized kernels which supports large separation margins and therefore oftentimes better classification.

In this approach we utilize the fact that this indicators can capture crucial properties of a song. The zero crossing rate for example can capture how "vocal (dependent)" a song is which we hypothesis to be a valuable in the distinction for example between jazz and pop or folk music. Similar arguments can be made for tempo, key, etc. We utilize 15 features for the MSD dataset and 60 features for GTZAN. A more detailed outline of those can be found in section 4.1.

3.1.2 Convolutional Neural Networks. Convolutional neural networks as first outlined by LeCun et al. [12][1] are a deep learning paradigm allowing state-of-the-art pattern recognition on a global and local scale. They have outperformed SOTA performance in various computer vision as well as audio processing applications [7, 13]. In essence a CNN is a multi-layer neural network with special constraints on the connections within the convolutional layers enforcing weight sharing between a given convlutional kernel as well as limited receptive fields. Because a given kernel is shared across a feature map, it becomes a pattern detector which can be activated on a certain pattern of the input. More shallow layers are usually respond strongly to simpler patterns such as stripes or checker patterns while deeper layers respond to more complex patterns such as the shape of a human, a dog or the qualities of a specific instrument detected in a MFCC. The wide success and applicability of the architecture has lead to an explosion of various specialised CNN architectures. For our propose we examined and fine tuned 3 universally recognised pretrained CNN models which all clearly outperformed the benchmark of a shallow naive CNN approach trained from scratch. We examined ResNet34 [10], ResNet50[10], Squeezenet1_0[11].

ResNet34 and ResNet50 have been chosen for their well-known ability of transfer learning between domains. We chose a deeper and a more constrained version in order to pin point differences of the network depth on the classification performance. The Squeezenet was selected as a contrast to investigate whether classification would also be possible with a heavily reduced parameter space which would be attractive for applications on devices with limited memory such as mobile devices or a raspberry pi.

The Resnet is a very deep CNN architecture which was one of the first ones to circumvent the vanishing gradient problem by allowing skip connections between the various layers enabling backpropagation through tens (to this day even hundreds) of layers. It uses an initale $7x7$ convolutional kernel with stride 2 and 32 output channels followed by a max pooling layers and various $3x3$ kernels with stride 2 and 64 output channels which are grouped together in blocks of common operations. The squeeznet follows a similar structure, however employs smaller kernels ($1x1$), reduces the number of input channels per layer and downsamples only in the end of the network so that each convolution layer has larger activation maps. Together with some smart adaptions to the standard convolutional block operations this architecture leads to design with up 50 times less parameters than comparable CNNs with competitive accuracy [11]. We obtain the original pretrained models from the open source fast-ai library [2].

It is important to note that to execute our experiments with the Resnet50 at least 10GB of gRAM needs to be available, hence we do recommend to utilize modern GPUs whenever possible. Amazon AWS can offers an alternative for people without the required hardware.

3.2 Music generation

In order to generate a cover song in a style that is more like the music which a user enjoys, we have made use of music translation. The goal of this task is to take an input audio file and create an output audio file which contains the same music, but sounds as if from another domain (e.g. another musical style or genre). SOTA works on music translation make use of neural networks to perform the translations. We have experimented with two such networks.

[1] who in turn was inspired by neurological research in perception and cognition [16]

[2] https://www.fast.ai/

3.2.1 End-to-end music translation. The first network which we tried to use was proposed by Mor et al. from the Facebook AI Research group in [15]. It is an end-to-end network that can take songs from any arbitrary input domain and translate them into domains on which it was trained.

For this, the network makes use of WaveNets [22] to function as the encoders and decoders of an autoencoder-like network. The network is trained to denoise audio segments, and each decoder is trained on only the audio segments from a certain domain. This essentially makes the decoders learn to recreate the noises that are associated with their domain.

Just a single, universal encoder is used. This is achieved using an additional network that is trained to identify the domain of the input song based on the latent space from the encoder. The encoder is then trained to optimize the performance of the decoders and to confuse this domain classification network at the same time. As both networks keep getting better, the domain specific information becomes more and more obfuscated by the encoder, whilst also improving the overall quality of the outputs. And thus, the encoder becomes universal. Now with this, we can use the network to process songs from domains unseen during training (though they can only be translated into the domains seen during training).

We have attempted to train a new decoder on a pretrained model of this network, so that we could translate songs to the domain of this decoder. However, after considerable effort we were unable to reproduce the results of the original paper. We found that the new decoder could not be trained well enough to learn the sounds of its domain, and instead simply learned the identify function. One possible reason for this failure is that we did not have access to enough data for our desired output domain.

3.2.2 Multi-step pipeline. Because our first network did not provide us with satisfactory results, we came up with a multi-step procedure for music generation. Here we combine two audio processing techniques to together create covers of a song.

Source separation. The first step is to apply source separation to the input song. As the name implies, with this technique the different audio sources are separated into their own separate tracks. This allows us to apply different transformations on the instruments to make our song sound more like a different genre. It is also needed as the timbre transfer network which we discuss later can only work with single-source inputs.

For source separation, we make use of Demucs [5], which is a waveform-to-waveform model for source separation. This network has an autoencoder structure that directly creates the output waveforms, where each output is corresponds to a different instrument. It was shown to beat SOTA models and create natural sounding outputs.

As is also mentioned in the original paper, the network does suffer from bleeding between the "vocal" and "other" tracks. The extend to which this occurs differs greatly between songs. After listening to some examples, we deemed the quality of Demucs sufficient for our purpose. Another limitation of Demucs is that it only separates four different tracks: vocals, drums, bass, and everything else. However, most songs have at least some of these elements, and therefore we believe that it can be used well in the majority of cases.

Timbre transfer. After obtaining the different instruments as different tracks, we can make the tune played by these instruments sound as if played by another instrument. For that we apply timbre transfer, making use of Differentiable Digital Signal Processing [6]. This method uses a neural network to extract complex digital signal processing features from the input. These features can be tuned and used to perform various interesting tasks, but in our case we use it to perform timbre transfer. We namely alter the characteristics of the sound to be more similar to the characteristics from desired instruments. Meanwhile, other features such as the loudness and frequency are left unchanged. As a result we get exactly what we wanted: the same tune as played in the input, but now sounding as if played by another instrument.

In our pipeline, we pick and choose the tracks from specific instruments and transfer them into specific others. Which tracks and which translations we choose depends on the genre of the input song and the desired genre of the cover. For example, in rock music, guitars are found in most songs, so when we want to create a rock cover, it would make sense to translate one on the original instruments to sound like a guitar. We have currently tuned the pipeline for translations from pop to jazz, but different translations can be easily added.

As with Demucs, here we are again bound to limitations of the network. In this case, we can only make translations towards instruments on which the network has been trained. It is partly for this reason that we chose translations to jazz, as most of the available instruments are jazz instruments. However, it is possible to train the network on new instruments and this has the nice benefit that it does not require a lot of data; the available instruments were trained on less than 13 minutes of audio. Because of this, training for new instruments could greatly help make our pipeline more varied and flexible.

3.2.3 Automatic mixing. After we have separated the different tracks and turned (some of) them into other instruments, we combine the different tracks into the final cover song. This is done based on heuristics that depend on the genre of the input song and the desired genre of the cover, as does the choice of which tracks and instrument translations to use. For example in our pop-to-jazz translation, the original drum track is kept but the volume is set to be much lower. This makes sense because in pop music the drum beat is often very prominent whereas it is more in the background with jazz music. Another example is that we decrease the tempo of the song by 15 BPM, as that is the difference between the average BPM of pop and jazz music.

These are the main effects that we apply, volume adjustments and tempo changes. The latter can be automated when the average tempo of the genres are known. However, finding a good set of instructions for volume adjustments has to be done manually. As with our example, some logic can be used to determine what instruments to adjust in which way, but all songs are different so finding settings that work well for most songs from the input genre does require some effort. Another thing to keep in mind with volume adjustments is that the timbre transfer makes certain instruments sound more quiet than others.

These adjustments are all done using Audacity [3], which is free, open-source audio software. The benefit of this software is that every action can be automated using scripts. This makes it easy to do the mixing with the heuristic-based settings.

3.3 Cover Song Detection

We implement cover song detection to create an objective measure of song similarity with cover songs from the music generation task. Validation via a downstream task approach (such as the one we have outlinesd) is often used in modern deep learning applications in particular natural language processing [9] and computer vision [23] but also has application in music generation [8, 14, 18].

Cover Detection is a difficult task that has been attempted many times over in literature. In the paper [20], there are several experiments in cover song detection that address the question of feature selection, analysis in both the time and frequency domain, and different model implementation. For simplicity, we will be implementing a method from [20] that was distinguished as a high scoring model. In addition, we use the SecondHandSong dataset (18.196 tracks, 4.128 covers) consisting of metadata on songs[1]. Our analysis uses the tracks' 'chromas', 'beat tracking' and 'segment analysis' metafeatures. The chroma vectors are processed to achieve beat-alignment by using the beat and segment feature variables. The beat-aligned chroma are then windowed into chroma patches (each patch consisting of 75 consecutive chroma). The chroma patches then undergo a 2D Fourier Transform. The median of the fft-transformed matrix is taken and used as input to a K-nn clustering model. A detailed outline of the model is given in section 4.2.

4 EXPERIMENTS

4.1 Genre classification

We use the the Random Forest and Support Vector machine algorithms as well as the ResNet34, ResNet50 and Squeeznet models to classify between the 10 distinct extracted genres for the GTZAN and MSD dataset.

For the feature based algorithms we use the following setup: For the GTZAN audio data we follow the outline of Olteanu et al. [17] and first translate the raw audio into meaningful audio features including: 'length', 'chroma_stft_mean', 'chroma_stft_mean', 'zero_crossing_rate_mean', 'harmony_mean', 'tempo', ' mean of mel_frequency cepstrum for various song segments',.. (a full list may be found in Appendix 4). In total 60 features are presented.

For the MSD we use the features provided by Bertin-Mahieux et al. [1]. Those include: 'danceability', 'familiarity', 'hotness', 'duration', 'loudness', 'year', 'tempo', 'analysis_rate', 'key', 'key_confidence', 'mode', 'mode_confidence', 'time_signature' and 'time_signature_conf'(a full list in Appendix 5). In total 15 numerical features are presented. As, due to copyright considerations, there is no access of the underlying audio files of the MSD no new features other than the ones provided can be computed.[4] We apply a 10-fold cross-validated grid search in sklearn[5] to determine optimal hyperparamers for the respective algorithm.

For the CNN based approaches we transform the pitch and timbre information into images of dimension $12x256$, cropping overflowing length of tracks and we use the MFCC scaled to a $256x256$ image as provided by [17]. We use the fast-ai learning rate finder to determine optimal learning rates for the models respectively for each dataset and otherwise leave the hyperparameters as set by the original implementations. The pretrained models were first trained 5 epochs with all forzen layers on each dataset to determine respective learning rate and then fine tuned with unfrozen layers for another 5 epochs in the case of ResNet and 50 epochs for the squeeznet (as the model has many fewer parameters this still ends in less computation time, however more tuning turned out to be not beneficial to generalization).

A 80-20 train test split is used for validation. We investigate varying performance of the algorithms providing confusion matrices and F1-score.

4.2 Cover Song detection and classification

We carry out two main experiments on cover song detection and binary cover song classification.

Model 1 : Cover Song Prediction. For the first case the model is a K-nn clustering model that uses a euclidean distance metric as outlined in [20] and 25 neighbors with uniform distance, as determined by a random hyperparameter search. All other model parameters are the default implementation as found in the scikit-learn toolbox [21]. Due to the fact that there are many songs in the SecondHandSong Dataset with few covers, only songs with more than 15 covers are used in this analysis. This resulted in 441 tracks and 29 covers. This method aims to cluster cover songs together and to predict cover clusters of song candidates based on existing cover songs in the K-nn model.

Model 2: Binary Cover Song Classification. In the second case, we implement a binary interpretation of the K-nn clustering model outlined in Model 1. The processing of tracks and the model is identical, differing only in the model input. The model contains all covers of the generated song candidate as well as all covers of a randomly selected song in the database. Therefore the model training input will consist of approximately 30 samples. The model then predicts whether the generated cover song is a cover song of the correct song.

4.3 Automated personalized music generation

For the automated personalized music generation we randomly selected several pop song examples from the MSD and let them run through our pipeline. We applied our multi-step pipeline to first separate the track into 4 source components ("vocal", "bass", "drums", "other") and then carried out a timber transfer using a selection of pretrained DDSP models that are fitting to the genre. Finally we used our Audacity automation script to mix and fine tune the tracks. Results and source files can be listened to on our project webpage [6].

[3] https://www.audacityteam.org/

[4] The original features were compute according to the methodology of the echonest API (https://github.com/echonest) which is a discontinued projected however was recently acquired by spotify and integrated in the spotify API which provides similar functionality and feature extraction for the entire music catalog of spotify.com.

[5] https://scikit-learn.org/stable/index.html

[6] https://liacs.leidenuniv.nl/~s2030098/index.html

5 RESULTS

5.1 Genre classification for GTZAN

For the GTZAN we find that the traditional feature based algorithms as well as the image based deep learning methods achieve remarkable results. The ResNet dominates the field but the random forest model follows up close. Figure 3 illustrates the high prediction quality on the case by outlining the confusion tables for the three different benchmarked CNN architectures.

5.2 Genre classification for MSD

For the MSD we observe that all algorithms perform significantly worse. We hypothesis this to be a consequence of the heavily imbalanced data property distribution of this dataset. The ResNet50 achieves the best performance however looking at the confusion tables (which can be found in the appendix figure 7) we find that it is still heavily driven by the majority classes and greatly under predicting the appearances of marginal classes. None of the algorithms is really able to properly label all minority classes correctly to a high degree.

F1-score	RF	SVM	ResNet34	ResNet50	Squeeznet
GTZAN	0.77	0.32	0.73	0.79	0.74
MSD	0.47	0.37	0.41	0.49	0.47

Table 1: F-1 score for the outlined algorithms on GTZAN and MSD.

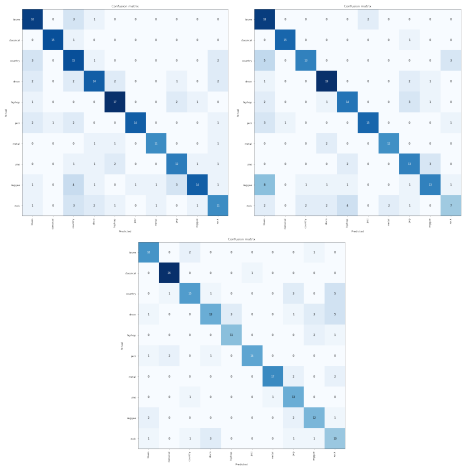

Figure 3: Comparison of the three CNN models fine tuned on the mfcc image representation of the GTZAN audio files. Top left: Resnet34, Top right: Resnet50, Bottom: Squeeznet1_0. We observe that the larger Resnet architectures marginally outperforms the parameter reduced squeeznet. We do not observe a significant difference between the larger and the smaller resnet model here.

5.3 Cover Song Recognition for MSD

Model 1. The kNN clustering model of all tracks in the SecondHandDataset with sufficient covers(>15 tracks per cover) was able to achieve a clustering accuracy of 0.289 with a train-test split of 90%-10%. This is lower than the metric reported in [20],0.602.

Model 2. The binary clustering classification method is designed as a tool for a quick objective measure in the music translation pipeline for our own use. The accuracy of the binary clustering method cannot be reported on robustly for a variety of reasons. First, we are limited in the sample size of our tests. There are only 29 cover songs with a sufficient number of covers in the SecondHandSongs, of which due to very long translation processing times we were only able to translate 7 covers so far. Second, our music generation pipeline is designed for the translation of genres pop to jazz, where most of the 29 covers in the subset are not pop tracks. As a result we do not believe our results to be robust in all instances, however we have chosen to report the accuracy here as an interesting indication of construction quality. We were able to achieve an accuracy of 1.00 when using a binary cover classification on 7 generated covers.

5.4 Generated Covers and Their Quality

In this work we have outlined a pipeline for automatic cover song generation, which we have implemented for covers for pop songs to jazz songs. We have applied our methodology to a number of pop songs and the resulting covers can be listened to on our project webpage [7].

In addition to an objective measure of cover-song similarity, we must also report upon other measures of audio quality, namely audio coherence. This, unfortunately, cannot be objectively measured yet and as a result we must rely on our observations and experiences when listening to the generated cover songs. Overall we find that the quality of the covers has a lot of creative range. Some are rather convincing whereas others sound unnatural and awkward. Most of them, however, are clearly recognizable as covers of the original while still entertaining the qualities of the target genre. Thus, we consider our goal to be achieved on this domain.

As mentioned earlier, in the music generation pipeline we were bound to the pretrained models from two networks. This limited the creativity and realism of the covers substantially. However we created a script to extract different instruments from any given song in a determined musical collection and then create a dataset out of the separate tracks for each instrument, which then in turn can be used to train new DDSP models. We also had to make the pipeline universal to pop songs, which is difficult as even within a single genre there can be ample variety. All things considered, we find our results satisfactory and inspiring for future endeavours in the direction of automated personalized music creation. Still, the pipeline has to be taken as a prototype and many optimizations could be made were it to be used in a production ready environment.

6 DISCUSSION

We find that genre classification and cover song detection can be accomplished to a satisfying degree. However, although respectable progress has recently been made in the field of music generation,

[7] https://liacs.leidenuniv.nl/~s2030098/index.html

creating coherent sound samples for a wider range of applications still poses a huge challenge given the currently available tools. Even SOTA networks often fail to deliver consistent quality and performance.

For the Genre classification we observed that classification on balanced dataset such as the GTZAN can be carried out to a remarkable accuracy considering the inherent complexity and ambiguity of the task. However there is some criticism to be made on the GTZAN dataset which with its 100 songs per genre is clearly too small to provide a representative sample of music in its entirety. Observations have been made for example that roughly a quarter (24%) of all pop songs within the dataset stem from a single artist (Britney Spears). A similar observation can be made for the reggae genre and Bob Marley (35% of the genre) [19]. Hence the dataset is simply too small to be able to represent the richness of the modern music industry. The MSD on the other hand, although much closer to the real world, makes it incredibly hard to build reliable models for all genres due to the a great data imbalance, favouring accuracy on popular genres. Creation of more balanced data, for example by artificially reducing or synthesizing, might greatly boost performance of all models.

There were several instances in which we faced difficulty in reproducing reference paper results. Overall, we found there to be vague wording in papers when describing methodology and a lack of code transparency.

In the case of the Cover Song Classification found in [20], the authors reported a clustering accuracy of 0.602 when using a kNN clustering method on Fast Fourier Transformed patches using a euclidean distance. We followed the methodology outlined but were not able to reproduce such a high clustering accuracy. The highest clustering accuracy we were able to achieve is 0.289. This is a large margin of difference between the reported metric and the reproduction metric. We believe there to be several reasons as to why this may have been the case. As mentioned there is vague wording and a lack of code-transparency. The authors mentioned that they used beat-alignment but did not outline the methods they used to achieve it. In addition, other than the distance metric, the authors did not provide any hyperparameter information on the kNN model. However, since the cover song classification was meant to be a supplementary tool as an objective measure for our music generation analysis, we do not believe this to have been a huge roadblock. In addition, although it lacked robustness the binary cover-song classification proved to be a useful "check" for our music generation task nonetheless.

We also found that the output of [15] was not reproducible in our experiments. We have already mentioned some of the issues that we had with training a new decoder for the universal music translation network found in [15]. For this work, source code was available, but it had to be adapted heavily to be able to train new decoders. We were unable to create any audio that has any of the domain specific characteristics that would be expected. In our attempts to get the network working, we tweaked several parameter settings, most importantly the learning rate. As the loss of the network always got stuck around the same point, we suspect that our only chance of getting it to work would be with more training data. This was, however, not possible to us due to the difficulty in getting music datasets with actual audio.

We believe that this might illustrate a larger issue of the field where full access to samples and/or code is rarely given. With this paper we furthermore hope to encourage colleges to be as open as possible about data and source code, and outline interesting successes alongside failures of proposed models, so the field as a whole can learn and adapt faster and steady.

In the future, we hope to extend the model to different musical genres. In our experiments, we focused on the pop to jazz pipeline and as it presented a stark contrast in two musical tastes and it made for an enjoyable experiment for us as listeners to our generated cover songs. In addition, this pipeline was convenient as there were several jazz instruments available in the music translation network. In the future, we hope to train more instruments (mainly guitar and piano) in order to be able to translate to a wider range of genres. With information about genre features, this will be a straightforward task to replicate in a new musical genre.

7 CONCLUSION

In the past few decades, research in audio processing has seen many great advancements in various different tasks. With the rise of new platforms and analysis of user data, music fans have also enjoyed an increasingly personalized listening experience. Given that machine-made music is also a popular and successful research area, it is perhaps surprising that personal music generation is such an understudied problem.

We have proposed a first prototype pipeline that tackles this problem through the generation of covers in the favorite genres of a user. The pipeline starts out with genre classification. This is used to process the users music library and find which genres they enjoy listening to most.

We then take any random song and create a cover in one of the user's favorite genres. For that we use many different techniques, namely source separation, timbre transfer, and heuristic-based automatic mixing. The music generation has been implemented for translations from pop to jazz and can be extended for different translations with reasonable effort. Further research may focus on ways to make this process more universal.

Lastly for our own reference to evaluate the quality of our covers quantitatively, we made use of cover song classification. A broader qualitative approach such as a larger crowd sourcing survey to establish a human quality perception score may further build trust in the robustness of our method and should be an additional goal of any future research.

REFERENCES

[1] Thierry Bertin-Mahieux, Daniel PW Ellis, Brian Whitman, and Paul Lamere. 2011. The million song dataset. (2011).

[2] Gérard Biau and Erwan Scornet. 2016. A random forest guided tour. *Test* 25, 2 (2016), 197–227.

[3] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. [n.d.]. A training algorithm for optimal margin classifiers. In *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*. 144–152.

[4] Leo Breiman. 2001. Random forests. *Machine learning* 45, 1 (2001), 5–32.

[5] Alexandre Défossez, Nicolas Usunier, Léon Bottou, and Francis Bach. 2019. Music Source Separation in the Waveform Domain. *arXiv preprint arXiv:1911.13254* (2019).

[6] Jesse Engel, Lamtharn (Hanoi) Hantrakul, Chenjie Gu, and Adam Roberts. 2020. DDSP: Differentiable Digital Signal Processing. In *International Conference on Learning Representations*. https://openreview.net/forum?id=B1x1ma4tDr

[7] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. 2016. *Deep learning*. Vol. 1. MIT press Cambridge.

[8] Philippe Hamel, Matthew EP Davies, Kazuyoshi Yoshii, and Masataka Goto. 2013. Transfer learning in mir: Sharing learned latent representations for music audio classification and similarity. (2013).
[9] Yaru Hao, Li Dong, Furu Wei, and Ke Xu. 2019. Visualizing and understanding the effectiveness of BERT. *arXiv preprint arXiv:1908.05620* (2019).
[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.
[11] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size. *arXiv preprint arXiv:1602.07360* (2016).
[12] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. 1989. Backpropagation applied to handwritten zip code recognition. *Neural computation* 1, 4 (1989), 541–551.
[13] Tom LH Li, Antoni B Chan, and Andy HW Chun. 2010. Automatic musical pattern feature extraction using convolutional neural network. *Genre* 10 (2010), 1x1.
[14] Yin-Jyun Luo, Li Su, et al. 2018. Learning Domain-Adaptive Latent Representations of Music Signals Using Variational Autoencoders.. In *ISMIR*. 653–660.
[15] Noam Mor, Lior Wolf, Adam Polyak, and Yaniv Taigman. 2018. A universal music translation network. *arXiv preprint arXiv:1805.07848* (2018).
[16] J Anthony Movshon, Ian D Thompson, and David J Tolhurst. 1978. Spatial summation in the receptive fields of simple cells in the cat's striate cortex. *The Journal of physiology* 283, 1 (1978), 53–77.
[17] A Olteanu. 2020. "GTZAN Dataset-Music Genre Classification. *Kaggle. com* (2020).
[18] Jiyoung Park. 2018. Representation learning of music using artist labels= . (2018).
[19] Hendrik Schreiber. 2015. Improving Genre Annotations for the Million Song Dataset.. In *ISMIR*. 241–247.
[20] Andree Silva-Reyes, Fabiola Martinez-Licona, and Alma Licona. 2018. Cover Song Recognition Using Machine Learning Techniques. *Research in Computing Science* 147 (04 2018), 9–21. https://doi.org/10.13053/rcs-147-4-1
[21] sklearn. 2021. A scikit learn intro to KNeighbours. https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
[22] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. 2016. WaveNet: A Generative Model for Raw Audio. arXiv:1609.03499 [cs.SD]
[23] Daniel van Strien, Kaspar Beelen, Mariona Coll Ardanuy, Kasra Hosseini, Barbara McGillivray, and Giovanni Colavizza. 2020. Assessing the Impact of OCR Quality on Downstream NLP Tasks.. In *ICAART (1)*. 484–496.

A FULL LIST OF FEATURES FOR THE SUPERVISED GENRE CLASSIFICATION

1	length	1000 non-null	int64
2	chroma_stft_mean	1000 non-null	float64
3	chroma_stft_var	1000 non-null	float64
4	rms_mean	1000 non-null	float64
5	rms_var	1000 non-null	float64
6	spectral_centroid_mean	1000 non-null	float64
7	spectral_centroid_var	1000 non-null	float64
8	spectral_bandwidth_mean	1000 non-null	float64
9	spectral_bandwidth_var	1000 non-null	float64
10	rolloff_mean	1000 non-null	float64
11	rolloff_var	1000 non-null	float64
12	zero_crossing_rate_mean	1000 non-null	float64
13	zero_crossing_rate_var	1000 non-null	float64
14	harmony_mean	1000 non-null	float64
15	harmony_var	1000 non-null	float64
16	perceptr_mean	1000 non-null	float64
17	perceptr_var	1000 non-null	float64
18	tempo	1000 non-null	float64
19	mfcc1_mean	1000 non-null	float64
20	mfcc1_var	1000 non-null	float64
21	mfcc2_mean	1000 non-null	float64
22	mfcc2_var	1000 non-null	float64
23	mfcc3_mean	1000 non-null	float64
24	mfcc3_var	1000 non-null	float64
25	mfcc4_mean	1000 non-null	float64
26	mfcc4_var	1000 non-null	float64
27	mfcc5_mean	1000 non-null	float64
28	mfcc5_var	1000 non-null	float64
29	mfcc6_mean	1000 non-null	float64
30	mfcc6_var	1000 non-null	float64
31	mfcc7_mean	1000 non-null	float64
32	mfcc7_var	1000 non-null	float64
33	mfcc8_mean	1000 non-null	float64
34	mfcc8_var	1000 non-null	float64
35	mfcc9_mean	1000 non-null	float64
36	mfcc9_var	1000 non-null	float64
37	mfcc10_mean	1000 non-null	float64
38	mfcc10_var	1000 non-null	float64
39	mfcc11_mean	1000 non-null	float64
40	mfcc11_var	1000 non-null	float64
41	mfcc12_mean	1000 non-null	float64
42	mfcc12_var	1000 non-null	float64
43	mfcc13_mean	1000 non-null	float64
44	mfcc13_var	1000 non-null	float64
45	mfcc14_mean	1000 non-null	float64
46	mfcc14_var	1000 non-null	float64
47	mfcc15_mean	1000 non-null	float64
48	mfcc15_var	1000 non-null	float64
49	mfcc16_mean	1000 non-null	float64
50	mfcc16_var	1000 non-null	float64
51	mfcc17_mean	1000 non-null	float64
52	mfcc17_var	1000 non-null	float64
53	mfcc18_mean	1000 non-null	float64
54	mfcc18_var	1000 non-null	float64
55	mfcc19_mean	1000 non-null	float64
56	mfcc19_var	1000 non-null	float64
57	mfcc20_mean	1000 non-null	float64
58	mfcc20_var	1000 non-null	float64

Figure 4: Full list of the 60 computed GTZAN features.

1	familiarity	6240 non-null	float64
2	danceability	6240 non-null	float64
3	duration	6240 non-null	float64
4	loudness	6240 non-null	float64
5	year	6240 non-null	int64
6	tempo	6240 non-null	float64
7	analysis_rate	6240 non-null	int64
8	end_of_fade_in	6240 non-null	float64
9	key	6240 non-null	int64
10	key_confidence	6240 non-null	float64
11	mode	6240 non-null	int64
12	mode_confidence	6240 non-null	float64
13	start_of_fade_out	6240 non-null	float64
14	time_signature	6240 non-null	int64
15	time_signature_conf	6240 non-null	float64

Figure 5: Full list of the 15 provided MSD features.

B GENRE CLASSIFICATION - ADDITIONAL CONFUSION TABLE

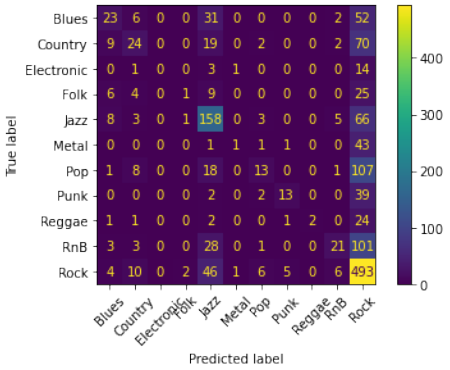

Figure 6: Confusion tables for the random forest algorithm on the MSD dataset.

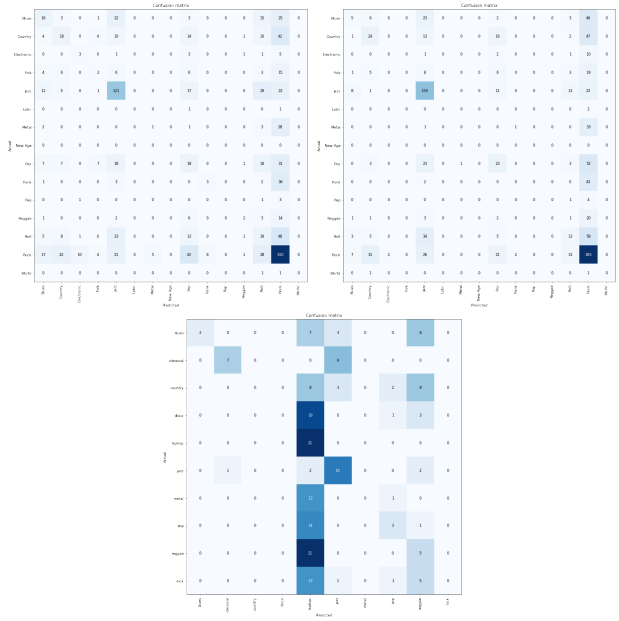

Figure 7: Confusion tables for the ResNet34,ResNet50 and Squeeznet1_0 respectively on the MSD dataset.