Music Genre Classification using Artificial Neural Networks

Andreas Buchmüller
a.buchmueller@stud.uni-goettingen.de
University of Göttingen

Christoph Gerloff
christoph.gerloff@stud.uni-goettingen.de
University of Göttingen

April 6, 2020

Abstract

Music genre recognition is a promising field of research in the area of music information retrieval (MIR). Genre classifiers have many real world applications, e.g. as a way to automatically tag large data sets suited as inputs to recommender systems. In this paper we propose a way to sample song data with the Spotify API and create a music genre classifier using artificial neural networks. We compare different feature sets to each other and evaluate their performance and accuracy using confusion matrices and more sophisticated metrics like F_1 scores. We show that convolutional neural networks using timbre values perform well on this task and also propose ways to handle class imbalance.

Keywords: Music, Genre, Classification, CNN, Spotify, Echonest

1 INTRODUCTION

The analysis and classification of sound signals has become of increasing importance, as not only Amazons Alexa or Windows Cortana show how it can be used to create a verbal interface, but also music streaming services rely on sound analysis in order to recommend their users music based on their listening history. These technological improvements are especially advantagous for people with impaired senses such as blindness or deafness. In both cases speech and sound recognition can assist those people in communication with others. Apart from those applications, sound analysis can be used to enhance the users experience. Genre classification is used for recommendation, as genres are a simple way to find similar music, which can then be suggested to individuals. With the onset of music streaming services, this task seems more relevant than ever. However due to the characteristics of audio data, retrieving information from sound signals and classifying them is a non-trivial task. Recent efforts in the area of music genre classification have been fueled by the availability of large data sets for analysis. Due to copyright constraints however, source audio files cannot be freely distributed. Large pre-sampled data sets like the *Million Song Dataset* [4] or the *Free Music Archive* [9] have existed for years now but have have obvious drawbacks such as lack of sampling freedom, genre choice or the ability to generalize results onto commercial music.
In music genre classification two components play a major role. Feature extraction, which is the art of finding a suitable representation for music samples, as well as the choice of classifier. Efforts led by the Laboratory for Recognition and Organisation of Speech and Audio (LabROSA) [1] have made feature extraction of musical data accessible for researchers worldwide. Their python package `librosa` [22], is equipped with functions specifically tailored to extract features of musical data. For classification tasks, there has been a plethora of supervised and unsupervised machine learning algorithms suitable for large data sets like the k-nearest neighbors algorithm or support vector machines, both of which have been a popular choice for a while now. Recently, advances in neural networks and deep learning research demonstrated the power of convolutional neural networks at image recognition tasks, outperforming even the best machine learning algorithms. In the past, researchers like [19] have successfully demonstrated that music genre classification may be approached as an image recognition task by feeding spectrograms into convolutional neural networks.
In this paper we are proposing a way to create a genre classifier on the basis of spectrogram-like features, namely pitch and timbre, of the Echo Nest application from the popular music streaming service Spotify. The Echo Nest is a music intelligence and data platform owned by Spotify, describing itself as the "Industry's leading music intelligence company, providing developers with the deepest understanding of music content" [29]. It is responsible for Spotify's curated personalized music recommendations. Both timbre and pitch are used individually as well as in conjunction within convolutional neural networks and then evaluated in terms of performance and different accuracy measures, such as confusion matrices and F_1 scores in balanced and unbalanced samples.

[1]https://labrosa.ee.columbia.edu/

2 METHODOLOGY

2.1 Data acquisition

There are numerous methods to obtain musical datasets, differing by the features that can be extracted. One simple method to create a dataset is extracting features from ones personal collection. The `librosa` package [22] provides a framework to extract meta and audio features of sound signals and in particular music. It works with most of the common sound data formats. This includes waveplots and spectrograms allowing to exactly analyse audio signals. This method however is limited by the size and diversity of your collection and the lack of a target label. A music recommender for example would need some kind of information, indicating what a user wants to hear (target). This could be achieved by giving songs your own review on a scale (very labor intenstive) or extracting user reviews from external data bases. For our genre classifier we need songs that are already labeled with their corresponding genres.

2.1.1 Spotify API

Spotify has one of the largest collections of music in history. This abundance of songs comes with the added cost of exact measures, as Spotify is careful concerning what parts of the songs are accessible. The *Spotify Web API* gives access to the Spotify library of song data. The `spotipy` package allows us to use the *Spotify Web API* with Python. Before this, a developer account and project have to be created on the *Spotify Dashboard*, where the `client_id` and the `client_secret` can be copied and then used in Python.
Spotify has grouped its songs within playlists, that are created by users or Spotify itself. This is especially useful when searching for songs of a specific genre, as for a multitude of them, numerous playlists exist and are relatively easy to extract. Unfortunately the playlists and their contents are changed weekly, with varying degree. This results in code that has to be adjusted from time to time in order to get working playlists. This potentially complicates replication of results.
`Spotipy` includes functions as `audio_features` and `audio_analysis`, that can pull meta and song data, provided by Echo Nest [15] from Spotify tracks. The former of these functions takes the `track_id` or `playlist_id` and returns features describing the music in high-level metadata, like danceability or energy in values, ranging from $[0, 1]$.
The `audio_analysis` function returns a number of features describing the song. Songs are divided into individual segments, which are varying in range and are generally shorter than one second. For each of those segments a timbre or pitch vector is given, describing the fraction of the song. For a more detailed description a what a segment is refer to the Echo Nest documentation [29].

2.1.2 Million Songs Dataset

In addition to the data Spotify provides, datasets incorporating Echo Nest metadata as well as audio features taken with `librosa` already exist. The *Million Songs Dataset* [4] is a freely available collection of audio features and metadata for one million songs. It is also possible to extract snippets of audio samples as files, not just as descriptive data, which is needed to create spectrograms with `librosa`. The MSD has been widely used and analyzed, therefore it appears not that compelling to build models, as plenty of methods have already been used on the exact same data.

2.1.3 Free Music Archive

Another readily available collection of audio data, is the *Free Music Archive* data set. The FMA [9] provides up to 917 GB of data containing 106,574 tracks from 16,341 artists and 14,854 albums, arranged in a hierarchical taxonomy of 161 genres. The data set provides full-length(copyright free) and high-quality audio, pre-computed features, together with track- and user-level metadata, tags and free-form texts such as biographies or lyrics. As with the MSD, the FMA is widely used.

2.2 Feature Engineering

There are a number of possible attributes that are extractable from the three aforementioned music collections. The pre-computed dataset of the FMA for example contains 518 different features. This results in a multitude of possibilities for analysis. We are creating a genre classifier, which can classify music genres from input songs and as such focus on timbral coefficients.

2.2.1 Genres

The musical genre of a song often is interesting when analysing and working with music. Genres are groups of similar music divided into (for humans) distinguishable classes. Not every song is classifiable into a well known genre, especially as the boundaries are fuzzy sometimes. For example the genre term *Rock* invokes different songs for different people, as some use the term interchangeable with metal, punk or just as a general overall term, for music with an electric guitar.
For our project we chose to use the following genres: Blues, Classical, Country, Hip-Hop, Jazz, Rock, Metal and Pop. For most common genres Spotify provides numerous playlists. One thing to be noted there, is that the playlists in the genres are individualised, depending on the country, from which Spotify is accessed. We accessed the collection of songs from Germany, which results in the inclusion of songs of the sub-genre 'Schlager' in the pop-playlists, which are actually more related to german country music than pop music in general.

2.2.2 Spectrogram

The spectrogram can be seen as a basic tool for sound and speech recognition analysis and can be defined as an intensity plot of the "Short-Time fourier Transformation" magnitude of an audio wave signal [27]. A spectrogram is constructed from the time, the frequency in hertz and the amplitude of the frequency at time t of an audio signal. In practice the spectrogram is corresponding to the matching sound extremely well.

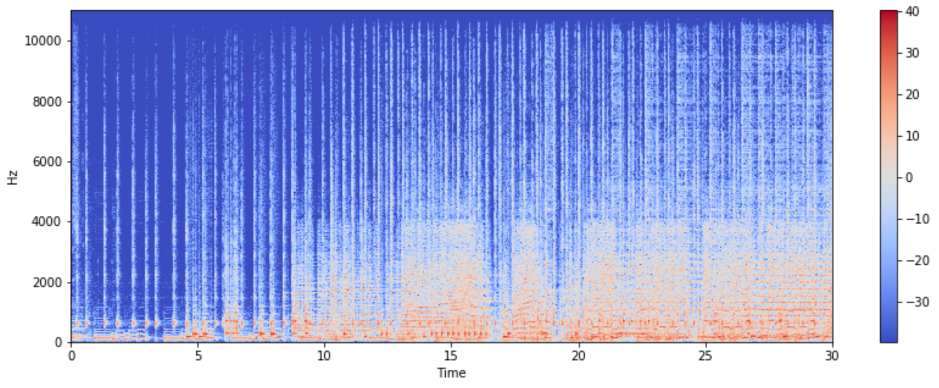


Figure 1: Spectrogram

The spectrogram is then often logarithmized, rendering the plot more interpretable. This log-spectrogram can then be converted onto the Mel-scale and transformed by the the Discrete Cosine Transformation (DCT). The Mel-frequency Cepstral Coefficients (MFCCs) [20] are the amplitudes of the resulting plot as seen in Figure 2. Spotify currently does not provide a way to extract or visualize spectrograms of songs in the Spotify library. It is possible to get audio samples, but even those are hard to extract and not that very accessible in practice. On the other hand Spotify does offer a handful of different features that are closely related to MFCC features.

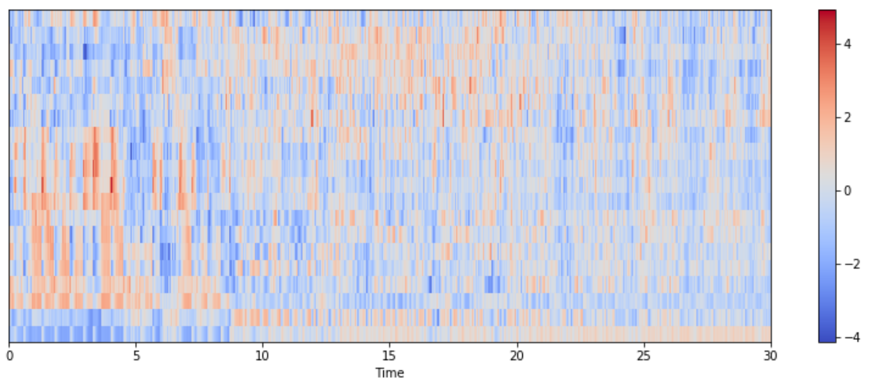


Figure 2: Mel-frequency Cepstral Coefficients (MFCCs)

2.2.3 Echonest Features

Spotify created high-level variables, which can be extracted with `spotipy`. These variables mainly try to quantify subjective attributes as ”danceabilty”, ”acousticness” or ”speechiness”. Numerical values were assigned for these indirectly measurable attributes. Each song is given an average value for each attribute meaning these attributes cannot be analyzed by their variance within each song.

2.2.4 Timbre

As mentioned before, is not possible to get real spectrograms from the data extracted from Spotify. Instead the timbre and the pitch can be used, making it possible to use spectrogram-like representations of songs for our model. The `audio_analysis` function gets analysis features from Echo Nest [15], of a specific track in the Spotify library. This returns a list of `segments` of differing length and quantity for each song. One segment is usually a fraction of a second and contains (besides the start point and duration) information about the loudness, pitch and timbre. For our model the last two are of interest. The timbre is defined, as the quality of a sound, distinguishing different types of musical instruments or voices [26]. It is also referred to as tone quality or sound color. Each segment contains a timbre vector of 12 unbound values centered around 0. These values are corresponding to the twelve principal components derived from MFCCs.

$$\text{timbre} = \begin{Bmatrix} PC1 \\ PC2 \\ \vdots \\ PC12 \end{Bmatrix}$$

The timbre vectors in combination with the segment information can be used to get pseudo-spectrograms, having the segments on the x- and the PCs on the y-axis. Higher values of the timbres vectors are represented by a higher contrasted color.

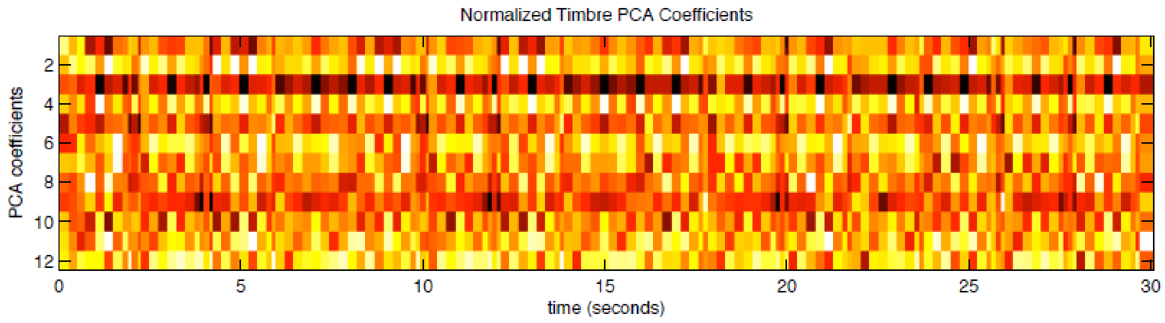


Figure 3: Spectrogram of Timbre averages [15]

2.2.5 Pitch

Another feature divided into segments by the Echo Nest is the pitch. The keys are track-leveled, ranging from 0-11 and are corresponding to the 12 musical keys, C, C#, D, up to B. The value is -1 if no key was detected, the mode equals 0 for a minor" or 1 for a "major" note.

$$\text{pitch} = \{C \quad C\# \quad D \quad D\# \quad \ldots \quad G \quad G\# \quad A \quad A\# \quad B\}$$

This allows for sounds to be scaled in relation to the frequency. It has to be noted, that the major key is more likely to be confused with the minor key three semitones lower, as both keys carry the same pitch. The structure of the pitch is described as a chroma-like vector, in which the values represent relative dominance of every pitch in the chromatic scale. Noise is represented in this by all values of every pitch being near 1, whereas the pure tones are described by one key at value 1 and every other key close to 0. Like the timbre, the pitch cab be displayed in a Chroma-like representation, with the musical notes on the y-axis.

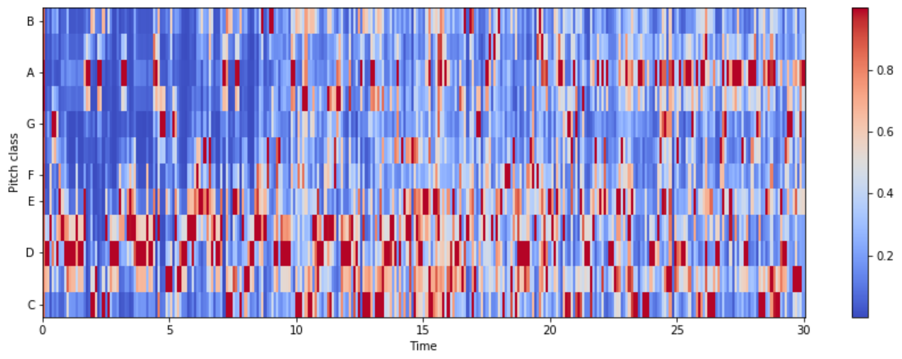


Figure 4: Chroma of the Pitch

2.2.6 Our Choice

The MSD and FMA are both easier to implement and can contain the actual audio signal, making the deep learning implementation straight forward. The downside of these data sets are their pre-made structures. These two collections are widely used, therefore they have been subject to a lot of analysis. We therefore chose the more difficult route and use the Spotify Web API to sample our own data for analysis. This way we do not have access to audio signals itself as with the other two data sets and are limited to spectrogram-like representation of our data but the data collection is highly flexible.

2.3 Data pre-processing

Before we go into the architecture of our neural networks it is worth mentioning how we created our input data, i.e. how we sampled from the raw JSON files we created using `spotipy`.

The data pulled from Spotify comes in JSON format, which is very machine learning friendly as Python reads JSON files like hierarchical structured lists. We pulled audio features and analysis data from the 20 most popular playlists of 8 arbitrarily chosen genres resulting in 10115 tracks total. The JSON file comes with various track metadata but for our analysis only two features are of interest: timbre and pitch vectors. The raw Spotify data contains between 42 and 9000 segments per track. A segment is defined as "a set of sound entities (typically under a second) each relatively uniform in timbre and harmony" [15]. Using the data we pulled from Spotify we created three samples out of which we

created 3 input arrays each. One sample was unbalanced, as it contained between 1100 and 1400 songs depending on the genre. This is due to different playlist lengths and leads to class imbalance across train/- validation/- and test sets. Our second sample was balanced by downsampling. It consists of 1100 tracks per genre, that were randomly drawn without replacement, if the genre originally had more than 1100 tracks. However, the train/- validation/- and test sets were also drawn randomly leading to slight imbalance in the validation and test set. Lastly we created a balanced sample, where we also made sure all subsets contain exactly the same amount of genres. We choose a 8:1:1 train/- validation/- test split, which is typically used in machine learning problems [10], resulting in a 880 / 110 / 110 split for each genre in our balanced set and slightly different sizes for each unbalanced set. It was interesting to compare the performance of our network across these samples for two reasons. First, in the case of the completely unbalanced sample, splitting the set into our three subsets results in approximately proportional genre sizes across all subsets i.e., if Jazz is heavily over represented in the training set, it is also likely over represented in the validation and test set. In the semi-balanced sample, this does not hold true any more. Because they are both relatively small (remember 880 tracks each means 110 tracks per genre at an 8:1:1 split) and we've drawn randomly, this results in a considerable variance in the validation and test set, as we may have 100 tracks of Jazz tracks versus 120 Rock tracks in the validation or test set. Now this has an obvious drawback: Sampling this way, we cannot rely on simple accuracy measures like global accuracy on the test set any more and need to use weighted averages or compare the accuracy for each genre individually. This stems from the fact that it may be that our network performs well on classical music and poorly on blues. If the test set now contains 120 tracks of classical music and only 100 blues songs, one may falsely conclude that the accuracy of the network is higher than it truly is, so well performing genres might be under or over-represented in the test set used to measure accuracy. Also one must consider that if the train set is imbalanced, better performance of the network in a single genre might stem from the fact that there is more training data for this genre.
From the three samples, we created three different versions of an input array that goes into our models resulting in nine input arrays total. One 3D array of size (batch, segments, 12) for timbre and pitch each, and a 4D array of size (batch, segments, 12, 2) where we combined timbre and pitch vectors into a 12×2 matrix to see if we can increase accuracy of our networks by adding additional data. Because of the added dimension 1D convolution is not suited for this input so we used 2D convolution layers and 2D max-pooling in models where the input is 4D.

2.4 Model architecture

All models have been implemented using Keras [8] with the TensorFlow backend [1]. Other important packages used in this paper are NumPy [28], Pandas [25], scikit-learn [11], matplotlib [5] and seaborn [31].

For our genre classifier we experimented with convolutional neural networks. CNN's are widely used in image recognition tasks [7] and since our input data is derived from MFCC-like features it can be expressed as a kind of spectrogram, so using CNN's seemed the natural choice. CNN's have also been successfully demonstrated to be well suited for this task before [6].

We tried our architectures on varying levels of sample balance. We first tried our neural networks with imbalanced classes, then used balanced classes with slightly unbalanced train/- validation/- and test sets (i.e. randomly drawn from balanced sample). Lastly a sample where we additionally made sure the train/- validation/- and test sets are balanced. As we sampled songs from 8 genres our baseline accuracy was initially set to 12.5% meaning the first goal was to achieve a higher accuracy than a random draw. Since this is a very low threshold and our dataset is very similar in size and complexity to the famous GTZAN [30] data set our next goal was to achieve a better accuracy than the authors of GTZAN paper. Its averages range between 44% and 62% overall and 40% to 75 % for each genre with well established unsupervised machine learning methods like k-nearest neighbors.

Figure 5 shows the architecture of our CNN model without the input layer (starting at the first convolution layer). It consists of six layers in total, including input and output layers. The input layer is a 512 × 12 map, that hosts either 12 MFCC-like timbre values or 12 pitch classes (corresponding to one of the 12 keys C, C#, D, etc.) across 512 segments of one track. It is followed by a 1D convolution layer with 48 filters of size 5 (first layer in Figure 5).

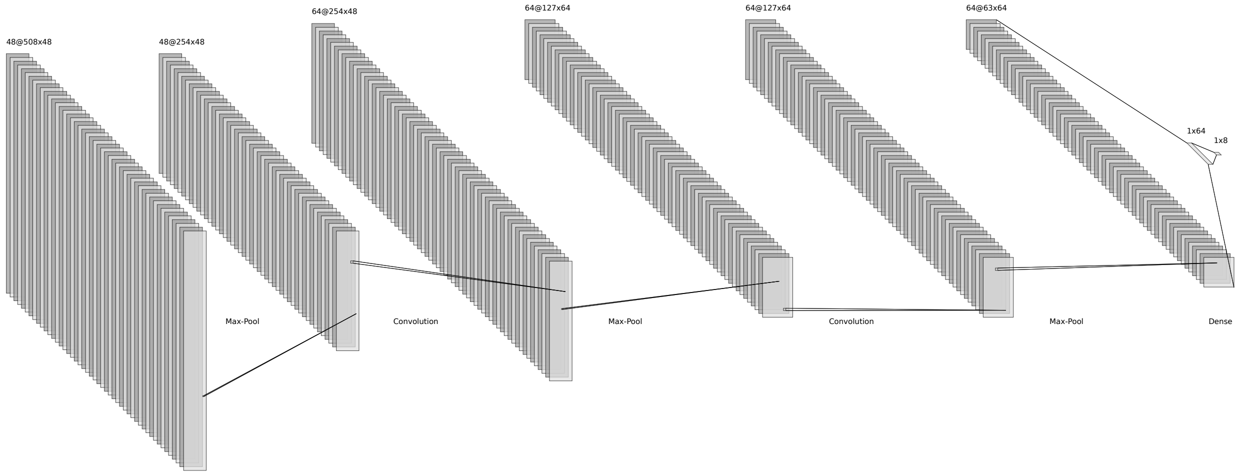


Figure 5: Schematic of our CNN

After each convolution layer, max-pooling and dropout is applied before the results are fed into the next layer. The first convolution layer is followed by two additional convolution layers, each with 64 filters and filter size 2. The output from the last convolution layer is then flattened and fed into a dense layer with 64 perceptrons before going into the output layer with 8 perceptrons - one for each genre. All layers used the ReLU activation function, except the output layer which is softmax [10] activated and acts as a our classifier. The ReLU (Rectified Linear Unit) activation function introduced by [13] is a popular choice. It sets a threshold at 0 meaning it will set the output to 0 if $x < 0$ and produces a linear function with slope 1 when $x > 0$.

$$f(x) = \max(0, x) = \begin{cases} x_{i,} & \text{if } x_i \geq 0 \\ 0, & \text{if } x_i < 0 \end{cases}$$

The softmax function at the output layer, related to the sigmoid function, is often used in binary classification tasks. In contrast to the sigmoid function the softmax function is capable of multi-class classification such as music genre classification see [24]. It computes a probability distribution over over a vector of real numbers, i.e., the output and its weights from the last dense layer and outputs values in the range between 0 and 1 for each class, with the sum being equal to 1.

$$P(y = i|\mathbf{x}) = \frac{e^{\mathbf{x}^\top \mathbf{w}_i}}{\sum_{c=1}^{M} e^{\mathbf{x}^\top \mathbf{w}_c}}$$

The predicted class $\hat{y}$ then equals the class with the highest probability.

Since we approached the task of genre classification as a multi-class single label problem (we assume one track can only belong to one genre) our loss is defined by categorical cross entropy. A mathematical definition, where y_0 is the ground truth, p_0 the corresponding prediction and we have M classes is given by:

$$CCE(p, y) = -\sum_{c=1}^{M} y_{o,c} \log\left(p_{o,c}\right) \text{ for } c = 1, \ldots, M$$

In our network however we used a slightly different version called sparse categorical cross entropy. Sparse categorical cross entropy is more memory efficient (especially when dealing with a lot of classes), since it takes a vector of integer target labels instead of a dense matrix of one-hot encoded target labels.

All networks in this paper used the same optimisation technique called Adam. Adam is a stochastic gradient descent type algorithm and an extension to classical stochastic gradient decent. Instead of maintaining a single learning rate for all weight updates, Adam computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients. [17] An explanation of Adam in pseudocode is given below:

Algorithm 1 Adam: Adaptive moment estimation. Default parameters are $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999$ and $\epsilon = 10^{-8}$

```
  *Initialize parameters for stepsize, exponential decay rates for the moment estimates and
  a small integer to avoid division by zero for use in stochastic objective function*
Require: $\alpha$, $\beta_1, \beta_2 \in [0, 1)$, $\epsilon$
  $\theta_0$ {Initial parameter vector}
  $m_0 \leftarrow 0$ {Initialize $1^{st}$ moment vector}
  $v_0 \leftarrow 0$ {Initialize $2^{nd}$ moment vector}
  $t \leftarrow 0$ {Initialize timestep}
  while $\theta_t$ not converged do
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ {Get gradients w.r.t. stochastic objective at timestep t}
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ {Update biased first moment estimate}
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ {Update biased second raw moment estimate}
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ {Compute bias-corrected first moment estimate}
    $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ {Compute bias-corrected second raw moment estimate}
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ {Update parameters}
  end while
  return $\theta_t$ {Resulting parameters}
```

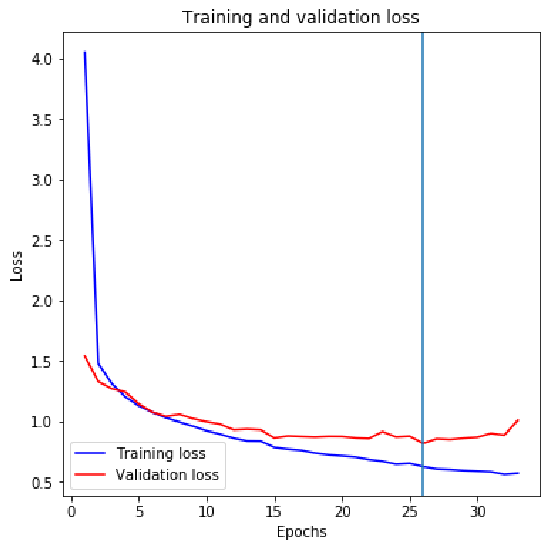
Compared to stochastic gradient decent it works well without much tuning of hyper-parameters and is computationally efficient, making it more suited towards local machines without a dedicated graphics card. We tried different learning rates and found that the default learning rate of $\alpha = 0.001$, as well as default values for $\beta_1 = 0.9$ and $\beta_2 = 0.999$ work well with batch sizes of 32 and 64 respectively.
Dropout rates vary depending on input data. In case of the fully balanced input array, we found a weak dropout of 0.25 to be the best rate for timbre and pitch input respectively. Lower rates lead to early overfitting, higher rates to a flat learning curve. When using our 4D input array, where we combined timbre and pitch values into a 12×2 matrix for each segment, a slightly stronger dropout between 0.3 and 0.4 works well. If the data is unbalanced or semi-balanced, strong dropout rates between 0.3 and 0.5 work best. Table 1 summarizes the complete CNN models we used for timbre and pitch input, respectively.

Layer (type)	Output shape	Param #
conv1d_7 (Conv1D)	(None, 508, 48)	2928
max_pooling1d_7 (MaxPooling1D)	(None, 254, 48)	0
dropout_9 (Dropout)	(None, 254, 48)	0
conv1d_8 (Conv1D)	(None, 254, 64)	15424
max_pooling1d_8 (MaxPooling1D)	(None, 127, 64)	0
dropout_10 (Dropout)	(None, 127, 64)	0
conv1d_9 (Conv1D)	(None, 127, 64)	20544
max_pooling1d_9 (MaxPooling1D)	(None, 63, 64)	0
dropout_11 (Dropout)	(None, 63, 64)	0
flatten_3 (Flatten)	(None, 4032)	0
dropout_12 (Dropout)	(None, 4032)	0
dense_5 (Dense)	(None, 64)	258112
dense_6 (Dense)	(None, 8)	520

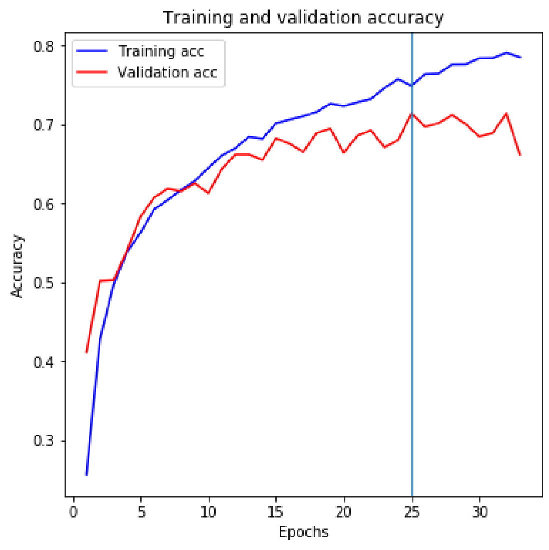
Table 1: Model summary of our CNN (Timbre/Pitch input)

The training time and required number of epochs varies considerably across our models. To reduce training time and overfitting we used callbacks in our training process, so that the training stops early if the accuracy on the validation set does not increase by at least 1% for 8 successive epochs. Experimenting a bit we found that for some models (especially pitch input models) this threshold is a little bit too weak to prevent overfitting and better suited for timbre models when using the same hyperparameters but still decided to use this threshold for two reasons: First, it makes comparison between the models easier and more meaningful and secondly overfitting does not worsen validation / test accuracy but underfitting does (a stronger threshold for early stopping leads to underfitting of timbre and mixed models).

Figure 6 show the training process for our timbre model. We can see first signs of overfitting after epoch 15. From figure 6 (b) we can see our timbre model begins overfitting after epoch 20. The vertical bars show the epoch, where the validation loss reaches its minimum and the accuracy its maximum. The model reaches its highest validation accuracy at epoch 25 and its lowest validation loss at epoch 26. It stops training at epoch 34 due to early stopping. We conclude that, with the hyperparameters we set, our timbre model generalizes poorly after epoch 25 and so training should not continue much further. If we continue training, the validation and test accuracy stays at around 70% while the training accuracy climbs up to over 90% if we train our model long enough. This finding is also confirmed by repeated experiments, where our model always reaches its validation maximum between epoch 19 and 29. As mentioned earlier our pitch model reaches its maximum accuracy a bit earlier (around 5 epochs in repeated measurements) and overfits much quicker. Our mixed model trains a bit slower and reaches its validation maximum generally 5-10 epochs later than the pure timbre model. Training curves for pitch and mixed model can be found in the appendix.



(a)



(b)

Figure 6: Training process of our timbre model

3 RESULTS

To evaluate the results of our analysis, we used the following metrics.

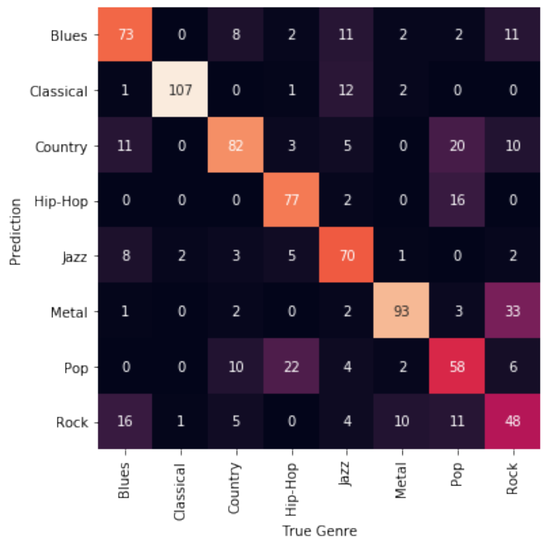
1. Accuracy on a test set.
2. Confusion Matrices. Tables describing classification performance in multi-class problems.
3. Precision, Recall, and F_1 score for each genre. Metrics defined in true and false positives and true and false negatives.

First we will compare how timbre performed against pitch in the same model. Then we will look at how both compare against a model, where the input array consists of timbre and pitch combined. We did this for all our samples, but will focus on our balanced sample, since comparison is easier here. To avoid confusion our CNN with timbre input will be referred to as c1m1, the model with pitch input as c1m2 and our mixed model with timbre and pitch input combined c2m1 (e.g. c1 for 1D convolution, c2 for 2D and m1 timbre input and m2 pitch input). Table 2 displays the accuracy of each model on our validation and test sets.

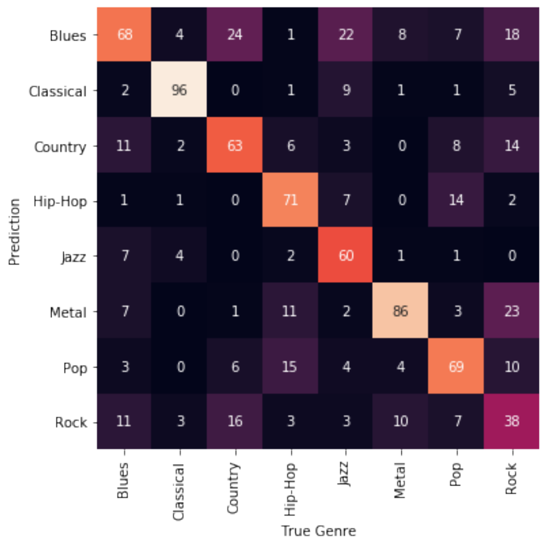
Model	Validation accuracy	Test accuracy
c1m1 (Timbre)	71.36%	68.18%
c1m2 (Pitch)	65.68%	62.61%
c2m1 (Mixed)	67.95%	67.84%

Table 2: Overall test accuracies for balanced sample models

From the table we can see that all of these perform similarly well with the timbre model performing best overall. However from this table, we cannot see which genres cause the models to perform better or worse. To see what makes one model perform better than the other it is worth taking a look at their respective confusion matrices.



(a) c1m1 (Timbre)



(b) c1m2 (Pitch)

Figure 7: Confusion matrices for c1m1 and c1m2

Figure 7 shows confusion matrices for timbre and pitch models trained on the balanced sample. Comparing them side by side, we can see that four genres seem particularly difficult so classify: Blues, Country, Pop and Rock. Our timbre model does a better job at most of them (except Pop/Hip-Hop), while the pitch model has most false positives. This makes sense intuitively since the boundaries between Rock and Country, Hip-Hop and Pop or Blues and Jazz are fuzzy and not easily distinguished even by human listeners. This is confirmed by looking where most false positives are: its exactly these genres where the models misclassify whereas, e.g. classical music, is easily distinguished with almost no false positives by all models.

From figure 8 we see that the addition of pitch does not necessarily increase the accuracy of the model. For most genres the addition of pitch decreases accuracy except for Hip-Hop and Blues so we can conclude that increasing the dimensionality of a model does not necessarily yield increased accuracy. A reason for why this might be was given earlier. In section 2 we noted that a major key is likely to be confused with a minor key three semitones lower, as both keys carry the same pitch. This could lead to reduced accuracy in models where pitch was used as input.

To dive even deeper into the results we can look at the classification reports for each model. A classification report mainly consists of three metrics for each genre: Precision, Recall and F_1 score. Before we discuss the results it is important to know what each of these means. Precision is the ability of a classifier not to label an instance positive that is actually negative and described by the ratio of true positives to the sum of true and false

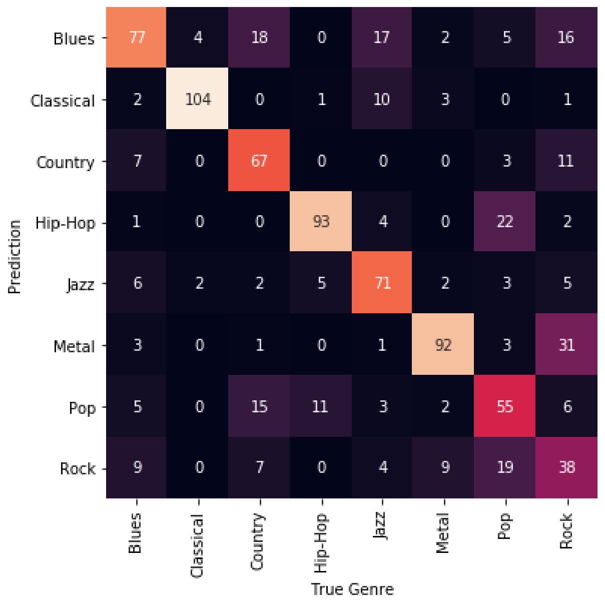


Figure 8: Confusion matrix for c2m1 (Mixed)

positives. Recall is the ability of a classifier to find all positive instances and defined as the ratio of true positives to the sum of true positives and false negatives. Lastly, the F_1 score is the harmonic mean of precision and recall ranging from 0 (worst) to 1 (best). It is typically lower than global accuracy as it embeds both precision and recall into calculation thus is more suited to compare classifiers.

Genre	c1m1 (Timbre)			c1m2 (Pitch)		
	precision	recall	f1-score	precision	recall	f1-score
Blues	0.543	0.691	0.608	0.447	0.618	0.519
Classical	0.860	0.945	0.900	0.835	0.873	0.853
Country	0.573	0.782	0.662	0.589	0.573	0.581
Hip-Hop	0.833	0.727	0.777	0.740	0.645	0.689
Jazz	0.686	0.755	0.719	0.800	0.545	0.649
Metal	0.765	0.800	0.782	0.647	0.782	0.708
Pop	0.684	0.355	0.467	0.622	0.627	0.624
Rock	0.550	0.400	0.463	0.418	0.345	0.378

Table 3: Classification for reports c1m1 and c1m2

Table 3 again shows that the timbre model performs better on almost every genre except the for Pop genre, independent of which metric you choose to compare. The classification table for c1m2 can be found in the appendix (see table 4). It can be seen that our mixed model is somewhat in between our timbre and pitch models w.r.t. F_1 score with the exception of Hip-Hop where it beats both other models. We also see the recall for our timbre model in the Pop genre is particularly low meaning that for all instances that were classified as Pop only 35.5% were actually Pop songs. Considering the state of contemporary commercial music and that Pop was the dominating genre in charts for years now this finding makes sense intuitively since Pop has had a huge influence over all other genres over the last few years such that even though a song is labeled as Hip-Hop it has a spectrogram that looks very much like that of a Pop song. The addition of the pitch truly increases model accuracy here indicating that the PCA components derived from MFCCs are missing information related to the pitch of the song. However we also clearly see that the timbre values capture many features of a song well, since they classify more accurate than the pitch, for most genres even without much hyperparameter tuning or complicated model architectures.

The high false positive rate between Country and Pop music might be due to our sampling process mentioned earlier. Since we sampled from Germany our Pop playlists contain a lot of 'Schlager' music, which to non-native speakers can loosely be described as a Country/Pop hybrid genre. For other genres the fuzzy boundaries have existed forever. Blues and Jazz for example are notoriously difficult to classify as even human listeners might struggle to identify the genre correctly. We were surprised however that our timbre model and even the pitch model did so well in the Metal genre, as the spectrograms of Metal and Rock are closely related to each other as well.

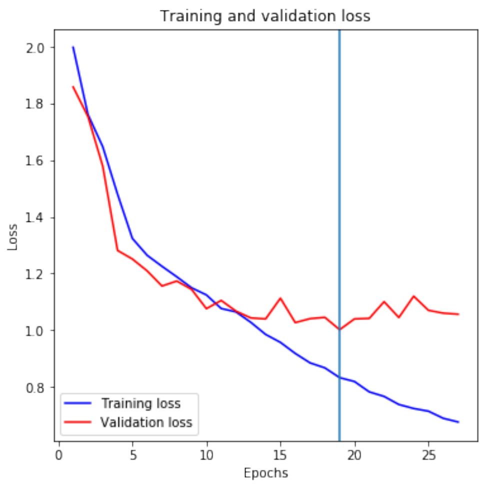
4 CONCLUSION

In this research paper we explained how we sampled track data using the Spotify Web API with `spotipy` and how we used this data to create a music genre classifier with artificial neural networks. From the data we pulled we created balanced and unbalanced samples that we used as an input and compared their respective performance and accuracy. We showed that, although not exactly equivalent to spectrograms and chroma plots, timbre and pitch values from the Echo Nest can be used as an input to CNN's in order to classify their genres with high accuracy. Especially timbre values are suited for this task since they take over a considerable part of the feature extraction, which otherwise the neural net would have to do. Although the pitch of a track generally does not perform as well, we found evidence that its addition can be beneficial for the accuracy of certain genres. We also found that when working with imbalanced samples, using higher dropout rates is necessary but also highly effective against overfitting, without impacting the performance significantly, and that these models provide similar accuracies to balanced sample models, although you need to be careful with interpretation if the validation and test sets are imbalanced. Weighted averages can be useful for a meaningful interpretation and comparison of imbalanced samples. With our simplistic CNN model, we achieved a global accuracy of about 70% on our balanced sample, although the difference between some genres is substantial.

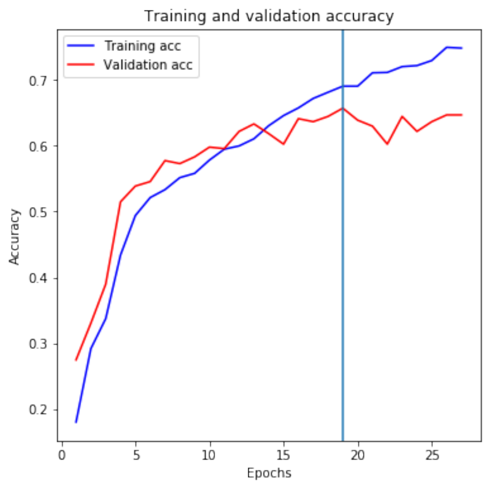
Due to the simplicity of our model we conclude that higher accuracies in the realm of 80% can be achieved with for example a larger sample size, additional convolution layers, sophisticated hyperparamter tuning (e.g. using Gridsearch), different activation functions (e.g. using SReLU instead of ReLU, which was tested on award winning CNN architectures for image recognition tasks like CIFAR-10, ImageNet etc. see [16]), the addition of more genres to capture hybrid genres better or trying a different architecture like CRNN [6] altogether.

A Appendix

A.1 Training processes of our models

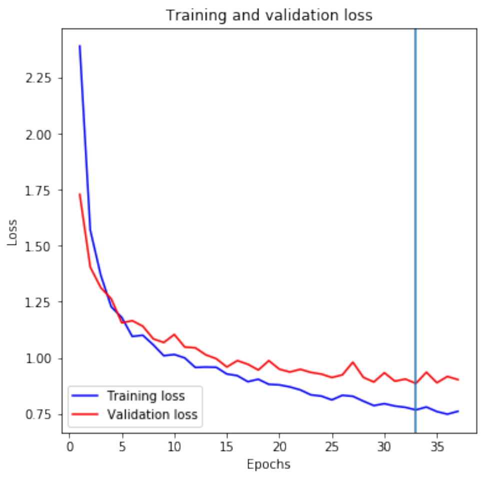


(a)

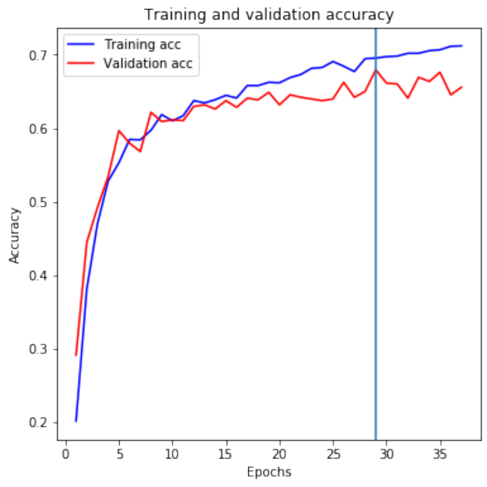


(b)

Figure 9: Training process of our pitch model



(a)



(b)

Figure 10: Training process of our pitch model

A.2 Full classification report of c2m1

	precision	recall	f1-score	support
Blues	0.554	0.700	0.618	110.000
Classical	0.860	0.945	0.900	110.000
Country	0.761	0.609	0.677	110.000
Hip-Hop	0.762	0.845	0.802	110.000
Jazz	0.740	0.645	0.689	110.000
Metal	0.702	0.836	0.763	110.000
Pop	0.567	0.500	0.531	110.000
Rock	0.442	0.345	0.388	110.000
accuracy	0.678	0.678	0.678	0.678
macro avg	0.673	0.678	0.671	880.000
weighted avg	0.673	0.678	0.671	880.000

Table 4: Classification results of our mixed model

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] A. F. Agarap. Deep learning using rectified linear units (relu). *CoRR*, abs/1803.08375, 2018.

[3] C. F. Allaire, J. J. allaire j. 2018. deep learning with r, 2018.

[4] T. Bertin-Mahieux, D. P. Ellis, B. Whitman, and P. Lamere. The million song dataset. In *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)*, 2011.

[5] T. A. Caswell, M. Droettboom, J. Hunter, E. Firing, A. Lee, J. Klymak, D. Stansby, E. S. de Andrade, J. H. Nielsen, N. Varoquaux, B. Root, P. Elson, R. May, D. Dale, T. Hoffmann, J.-J. Lee, J. K. Seppänen, D. McDougall, A. Straw, P. Hobson, C. Gohlke, T. S. Yu, E. Ma, A. F. Vincent, S. Silvester, C. Moad, J. Katins, N. Kniazev, F. Ariza, and E. Ernest. matplotlib/matplotlib v3.0.3, Feb. 2019.

[6] K. Choi, G. Fazekas, M. B. Sandler, and K. Cho. Convolutional recurrent neural networks for music classification. *CoRR*, abs/1609.04243, 2016.

[7] F. Chollet. *Deep learning with Python.* Manning Publications Co, Shelter Island, New York, 2018. OCLC: ocn982650571.

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

[9] M. Defferrard, K. Benzi, P. Vandergheynst, and X. Bresson. Fma: A dataset for music analysis. In *18th International Society for Music Information Retrieval Conference*, 2017.

[10] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning.* MIT Press, 2016.

[11] O. Grisel, A. Mueller, Lars, A. Gramfort, G. Louppe, P. Prettenhofer, M. Blondel, V. Niculae, J. Nothman, A. Joly, J. Vanderplas, manoj kumar, H. Qin, N. Varoquaux, R. Layton, L. Estève, J. H. Metzen, N. Dawe, R. (Venkat) Raghav, J. Schönberger, W. Li, G. Lemaitre, C. Woolam, K. Eren, Eustache, A. Fabisch, A. Passos, bthirion, V. Fritsch, and H. Alsalhi. scikit-learn/scikit-learn: Scikit-learn 0.21.3, July 2019.

[12] A. Gullì and S. Pal. *Deep learning with Keras: implement neural networks with Keras on Theano and TensorFlow.* Packt, Birmingham Mumbai, 2017.

[13] R. Hahnloser, R. Sarpeshkar, M. Mahowald, R. Douglas, and H. Seung. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, 405:947–51, 07 2000.

[14] T. Hope, Y. S. Resheff, and I. Lieder. Learning TensorFlow. 2017.

[15] T. J. Jehan T., DesRoches. *Analyzer Documentation*. the echonest, 2014.

[16] X. Jin, C. Xu, J. Feng, Y. Wei, J. Xiong, and S. Yan. Deep learning with s-shaped rectified linear activation units. *CoRR*, abs/1512.07030, 2015.

[17] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2014.

[18] M. Kirk. *Thoughtful machine learning with Python: a test-driven approach*. O'Reilly, Beijing ; Boston, first edition edition, 2017. OCLC: ocn908375399.

[19] T. Li, A. Chan, and A. Chun. Automatic musical pattern feature extraction using convolutional neural network. *Lecture Notes in Engineering and Computer Science*, 2180, 03 2010.

[20] B. Logan et al. Mel frequency cepstral coefficients for music modeling. In *Ismir*, volume 270, pages 1–11, 2000.

[21] M. Lutz. *Learning Python*. O'Reilly, Beijing, fifth edition edition, 2013.

[22] B. McFee, C. Raffel, D. Liang, D. P. Ellis, M. McVicar, E. Battenberg, and O. Nieto. librosa: Audio and music signal analysis in python. In *Proceedings of the 14th python in science conference*, volume 8, 2015.

[23] W. McKinney. *Python for data analysis*. O'Reilly, Beijing, 2013. OCLC: ocn794362690.

[24] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *CoRR*, abs/1811.03378, 2018.

[25] J. Reback, W. McKinney, J. V. den Bossche, jbrockmendel, T. Augspurger, P. Cloud, gfyoung, Sinhrks, A. Klein, J. Tratner, C. She, M. Roeschke, T. Petersen, W. Ayd, A. Hayden, S. Hawkins, J. Schendel, M. Garcia, V. Jancauskas, P. Battiston, S. Seabold, chris b1, h vetinari, S. Hoyer, W. Overmeire, M. Mehyar, behzad nouri, T. Kluyver, C. Whelan, and K. W. Chen. pandas-dev/pandas: v0.25.2, Oct. 2019.

[26] A. Schindler and A. Rauber. Capturing the temporal domain in echonest features for improved classification effectiveness. In *International Workshop on Adaptive Multimedia Retrieval*, pages 214–227. Springer, 2012.

[27] J. O. Smith. *Mathematics of the Discrete Fourier Transform (DFT)*. W3K Publishing, http://www.w3k.org/books/, 2007.

[28] S. C. C. Stéfan van der Walt and G. Varoquaux. The numpy array: A structure for efficient numerical computation. *Computing in Science & Engineering*, 13:22–30, 2011.

[29] The Echo Nest. The Echo Nest Company Description, 2020.

[30] G. Tzanetakis and P. Cook. Musical genre classification of audio signals. *IEEE Transactions on Speech and Audio Processing*, 10(5):293–302, July 2002.

[31] M. Waskom, O. Botvinnik, D. O'Kane, P. Hobson, J. Ostblom, S. Lukauskas, D. C. Gemperline, T. Augspurger, Y. Halchenko, J. B. Cole, J. Warmenhoven, J. de Ruiter, C. Pye, S. Hoyer, J. Vanderplas, S. Villalba, G. Kunter, E. Quintero, P. Bachant, M. Martin, K. Meyer, A. Miles, Y. Ram, T. Brunner, T. Yarkoni, M. L. Williams, C. Evans, C. Fitzgerald, Brian, and A. Qalieh. mwaskom/seaborn: v0.9.0 (july 2018), July 2018.

[32] N. Zumel and J. Mount. *Practical data science with R.* Manning Publications Co, Shelter Island, NY, 2014. OCLC: ocn862790245.