Music Classification

Mehmet Fatih AYTEN
VUB-Electrical Engineering
mehmet.fatih.ayten@vub.be

Giovanny RICO
VUB-Electrical Engineering
giovanny.vicente.rico.soria.galvarro@vub.be

Mohamad SAAB
VUB-Electrical Engineering
Mohamad.Saab@vub.be

Jialun KOU
VUB-Electrical Engineering
jialun.kou@vub.be

***Index Terms*—MFCC, NN, CNN, RNN, LSTM, Spectrogram, Optimizer, Music Classifier**

I. Introduction

The classification of music by genres plays a fundamental role in music information retrieval systems, especially in the context of digital music on the Internet, as mentioned in [1], [2] and [3]. This report aims to provide a comprehensive understanding of training a Neural Network (NN) capable of effectively classifying music genres.

To begin with, we present a survey of the existing literature in Section II. This literature review highlights the research conducted thus far, giving insights into the progress made in content-based music genre classification. By understanding the current state of the field, we can identify gaps and opportunities for further exploration.

Before delving into training the NN, it is essential to preprocess the raw audio data and extract relevant features. In Section III, we introduce the concept of Mel Frequency Cepstral Coefficients (MFCC), a widely used technique for extracting feature coefficients from audio data. The principles and significance of MFCC in music genre classification are discussed in detail. These extracted features serve as inputs to the NN, enabling it to learn and make accurate genre predictions.

The choice of optimizer plays a crucial role in training NNs effectively. In Section IV, we present an overview of different optimizers, including SGD, Momentum, RMSprop and Adam. The principles underlying each optimizer are explained, providing insights into their strengths and limitations. Understanding these optimizers will help in selecting the most suitable one for our music genre classification task.

Section V focuses on the principles of various neural network architectures, including the standard Neural Network (NN), Convolutional Neural Network (CNN), and Recurrent Neural Network with Long Short-Term Memory (RNN-LSTM). These architectures are widely utilized for music genre classification tasks, and their distinctive characteristics and advantages are discussed. By exploring different models, we aim to identify the most effective architecture for our specific objective.

In Section VI, we present our experimental results, where we compare the performance of different activation functions, optimizers, neural network architectures (CNN, NN, LSTM), and alternative feature extraction methods. Through experimentation and evaluation, we aim to determine the optimal combination of techniques that yield the highest classification accuracy for music genre identification.

II. State of the Art

Within the domain of music classification, several distinct categories of music features have been identified, as outlined in the works of [4], [5]. These feature classes can be broadly categorized into three main groups: Timbral Textural Features, Rhythmic Content Features, and Pitch Content Features. In addition to the various classes of music features, there exist diverse methods for feature extraction. One such method employed by researchers, as described in [4], is the utilization of Daubechies Wavelet Coefficient Histograms. Also, it is a prevalent approach in academic research to combine multiple extraction methods in order to generate feature vectors [6]. The purpose of combining these methods is to enhance the comprehensiveness of the feature representation.

Gaussian Mixture Models (GMMs), Support Vector Machines (SVMs), K-Nearest Neighbor (KNN), and Convolutional Neural Networks (CNNs) have emerged as popular classifiers in the field of music classification. Nonetheless, research findings from academic literature, including the study referenced in [7], support the idea that combining multiple classifiers can lead to better performance in music classification.

In the realm of music genre classification, numerous notable papers have emerged, employing diverse extraction techniques, classifiers, and datasets. Some noteworthy studies with high accuracy rates include: The researchers in the study [8] presented a music genre classifier that relies on Convolutional Neural Networks (CNNs). They employed the GTZAN dataset as their primary source of data and utilized Mel-spectrogram and MFCCs as the feature vectors for classification. In their experiments, they evaluated the performance of the classifier on 10 different music genres. The results indicated an accuracy of 76% and 47% for the Mel-spectrogram and MFCCs feature vectors, respectively. In [7], three datasets (Latin Music

Database, ISMIR 2004, and African music database) were used and different classifiers were compared. They obtained the highest accuracy of 87.4% using Support Vector Machine (SVM) as the classifier. Following that, the Convolutional Neural Network (CNN) achieved an accuracy of 83%. In the study conducted by the authors of [9], an initial Convolutional Neural Network (CNN) model was trained on a dataset comprising 995 songs, encompassing 27 different genres. The resulting accuracy rate achieved by this model was 47%. Subsequently, the dataset was expanded to include 1880 songs, albeit with a reduced number of genres (7), and the model's performance was evaluated. The test accuracy for this modified dataset was determined to be 62%. In the study conducted by the authors of [10], they employed a Gaussian Mixture Model (GMM) to classify 10 distinct music genres from four different datasets, namely ballroom, SLAC, Codaich, and Bodhidharma. As an extraction method, they employed a combination of several features including time domain zero crossings, Mel-frequency cepstral coefficients (MFCC), spectral rolloff, and flux. The highest accuracy rate achieved using the GMM classifier was 73.8% on the SLAC dataset.

III. Data Pre-processing

First of all, what is meant by data pre-processing is preparing a file containing the labels and the feature values. They will be extracted from whatever the data set of songs we are considering. These labels represent the input of our training model and the feature values for whatever feature we choose to use represent the target output we expect for those inputs.

In traditional machine learning for sound fields, the data pre-processing was about feature engineering, the sound waveform itself or its fourier spectrum was used to extract some features (amplitude envelope in time domain, spectral centroid in frequency domain, etc...), and then they would be used in designing algorithms to extract specific characteristics from the audio signals and then feeding those engineered features into machine learning models.

However, with the rise of deep learning and the availability of large-scale annotated datasets, there has been a shift towards automated feature learning. Instead of manually designing and extracting features, deep learning models are capable of learning meaningful representations directly from the raw audio data. Spectrograms or MFCCs "Mel-frequency cepstral coefficients", which are essentially visual representations of the audio signal's frequency content over time, serve as input features to these models.

To show how our data pre processing was done we will use MFCC's extraction method as the same principle applies for other features extraction.

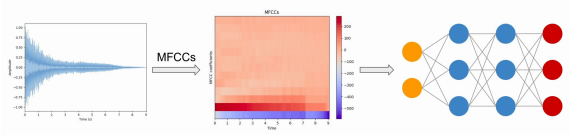


Fig. 1. Data Pre Processing with MFCC feature. Image taken from Here

A. MFCC Feature Extraction

MFCC's or Mel-frequency cepstral coefficients are widely used feature extraction method in the audio field. They are derived from the mel-frequency spectrum as will be explained here. From the early 2000s published research showed experimental relationship between pitch in mels and frequency using subjective listening tests! As shown by [11] a non linear relation between the pitch in mel and frequency was deduced. Using this relation a mel frequency scale was created that include more information about the human auditory system. This reason the mel frequency scale has an advantage is that humans are much better at discerning small changes in pitch at low frequencies than they are at high frequencies. This can be shown in Figure 2.

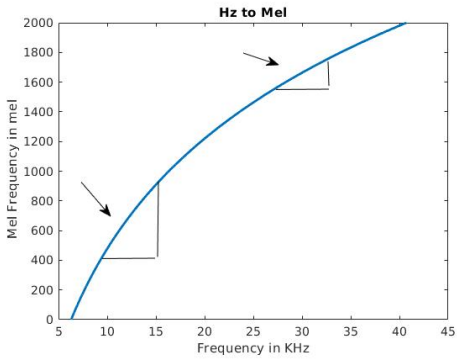


Fig. 2. Mel scale, done by matlab using the mel to freq relation

Now that the reasoning behind going to the mel scale is clear, we show the steps followed in order to extract the MFCC feature and by referring to [12]:

1) Frame the signal into short frames.
2) For each frame calculate the periodogram estimate of the power spectrum.
3) Apply the mel filterbank to the power spectra, sum the energy in each filter.
4) Take the logarithm of all filterbank energies.
5) Take the DCT of the log filterbank energies.
6) Keep DCT coefficients 2-13, discard the rest.

To avoid implementing an algorithm from scratch, we used librosa library built in function *librosa.feature.mfcc* [13].

B. Building the JSON file

The json file will contain all the training data, it will contain the genres that will be mapped to labels which is another part of the json file. For example classical music will be mapped into label 0, rock to label 1, etc. This is just for ease of comparison and evaluation while training the model as we can not use text labels. And the other part of the json file will be list of arrays containing the MFCC's. So, finally we would have a file that contains the inputs "MFCC's array" and the target "labels". This file is fed to the training model later. To build this file an algorithm was used in which it loops through the genre sub-folders, extract the semantic label, iterate through

the audio files in the current genre sub-folder, load and process audio segments using librosa built in function here Librosa-load, and finally save the current label with its extracted MFCC using also librosa library as previously explained.

IV. Optimizer Choice

Optimizer is a component in machine learning that determines how a model's parameters should be adjusted during training to minimize the loss function or maximize performance. In this project, we will compare the performance of four different optimizers. [14]

Note: In the formulas below, θ_t represents the parameters at iteration t, $\nabla J(\theta_t)$ represents the gradient of the loss function with respect to θ_t, η represents the learning rate.

1. SGD (Stochastic Gradient Descent): - Update Parameter Formula:

$$\theta_{t+1} = \theta_t - \eta \cdot \nabla J(\theta_t) \tag{1}$$

- Description: SGD is a basic optimization algorithm that updates the model's parameters based on the gradient of the loss function with respect to the parameters. It performs parameter updates for each training example individually, making it computationally efficient. However, it can exhibit slow convergence and can get trapped in a local minima.

2. Momentum: - Update Parameter Formula:

$$v_{t+1} = \gamma \cdot v_t + \eta \cdot \nabla J(\theta_t) \tag{2}$$
$$\theta_{t+1} = \theta_t - v_{t+1} \tag{3}$$

- Description: Momentum is an extension of SGD that introduces a momentum term to speed up convergence and escape local minima. It accumulates a fraction of the past gradients and uses it to update the parameters, leading to faster convergence. The momentum term helps the optimizer to continue moving in the direction of previous gradients.

3. RMSprop (Root Mean Square Propagation): - Update Parameter Formula:

$$v_{t+1} = \beta \cdot v_t + (1 - \beta) \cdot (\nabla J(\theta_t))^2 \tag{4}$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{v_{t+1} + \epsilon}} \cdot \nabla J(\theta_t) \tag{5}$$

- Description: RMSprop is an adaptive optimization algorithm that modifies the learning rate based on the magnitudes of recent gradients. It uses an exponentially decaying average of past squared gradients to adjust the learning rate for each parameter individually. This helps to scale the learning rates differently for different parameters, leading to faster convergence. [14]

4. Adam (Adaptive Moment Estimation): - Update Parameter Formula:

$$m_{t+1} = \beta_1 \cdot m_t + (1 - \beta_1) \cdot \nabla J(\theta_t) \tag{6}$$
$$v_{t+1} = \beta_2 \cdot v_t + (1 - \beta_2) \cdot (\nabla J(\theta_t))^2 \tag{7}$$
$$\hat{m}_{t+1} = \frac{m_{t+1}}{1 - \beta_1^{t+1}} \tag{8}$$
$$\hat{v}_{t+1} = \frac{v_{t+1}}{1 - \beta_2^{t+1}} \tag{9}$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_{t+1}} + \epsilon} \cdot \hat{m}_{t+1} \tag{10}$$

- Description: Adam is an adaptive optimization algorithm that combines the benefits of both Momentum and RMSprop. It maintains adaptive learning rates for different parameters and stores exponentially decaying averages of both past gradients (momentum) and past squared gradients (RMSprop). The algorithm uses bias-corrected first and second moment estimates to update the parameters. [15]

A. Dropout

Dropout is a regularization technique commonly used in deep learning to prevent over fitting and improve generalization of the model. It works by randomly setting a fraction of the input units (neurons) to zero during training, which helps to reduce interdependent learning among neurons. The idea is that by randomly dropping neurons, the model becomes more robust and less reliant on specific neurons for making predictions [16].

Mathematically, the dropout process can be represented using the following formula:

$$\mathbf{r} \sim \text{Bernoulli}(1 - p) \tag{11}$$
$$\mathbf{h}_{\text{out}} = \frac{\mathbf{h}_{\text{in}} \odot \mathbf{r}}{1 - p} \tag{12}$$

where:
- $\mathbf{r}$ is a binary mask vector of the same shape as the input $\mathbf{h}_{\text{in}}$, where each entry is independently set to 0 or 1 with a probability of p and $1 - p$, respectively. This mask determines which neurons are dropped.
- $\mathbf{h}_{\text{in}}$ represents the input to the dropout layer.
- $\mathbf{h}_{\text{out}}$ is the output of the dropout layer.
- $\odot$ denotes element-wise multiplication.

V. Training model introduction

We used different types of training models or networks to see how they perform on the same data set. In this project we used a simple Neural Network (NN), a NN with dropout (see the previous section on that), a Convolutional Neural Network (CNN) and finally a Recursive Neural Network - Long Short-Term Memory (RNN-LSTM).

1. Neural Network: We use the Multi Layer Perceptron (MLP) Neural Network (NN) here; first a NN is described by its own name, is a Network composed by Neurons, the Neurons are represented as nodes, and each node is connected to others, but they have a special "shape", as shown in Figure 3. The NN's are composed by 2 main layers and one or

more intermediate layers: Input layer, Hidden layer(s) and Output layer. Each layer has a determined amount of Neurons (nodes), which depends on the problem to solve (loss function to minimize), sometimes we have many nodes in the hidden layers, other times less, it depends on how you change the Network to optimize the performance, the number of Neuron in the Input layer normally goes related to the features we extract from the image to be analyzed (in our project the spectrogram with MFCCs), the hidden layers can be 1 or more, again to make complex or not so complex the network, that can or not improve the performance of the NN, increasing the amount of neurons and hidden layers leads to an increase in time for the training model. Finally, the Output layer's number of Neurons depends on the application, in our example is 10 nodes because we have 10 genres to classify the different samples of music. The more genres the more nodes. The MLP is a fully connected NN, it means that each node from one layer is connected to all the nodes from the next layer, and so on and so far. Like in the Figure 3 we can see all the nodes connected to each other.

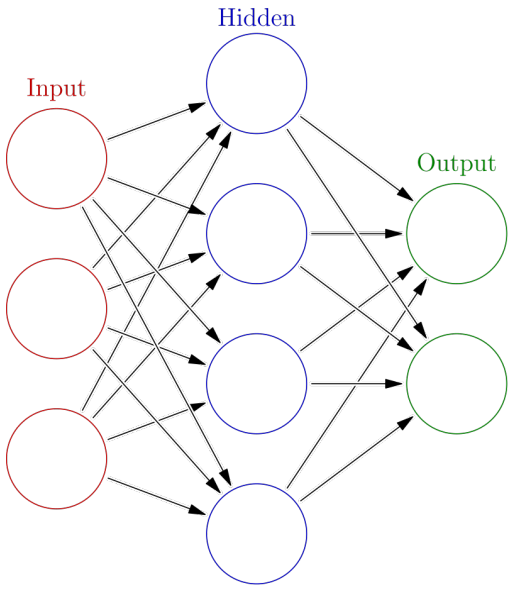


Fig. 3. Neural Network

2. NN with Dropout: This is mentioned again in the previous section, is just the same NN with some Neurons discarded during the training with the dropout technique.

3. Convolutional Neural Network: The CNNs are an special case of NNs, where we applied some operations before the normal NN (the MLP or a Dense Layer, like is the name used in the python code), this operations are Convolution (that's where the name comes from) and Pooling (where we can apply Max, Min or Average Pooling). In the lab sessions we've seen this type of NN's, the idea of them is to extract all the features from the data set, what will improve the performance of the model, in this case the Music Classifier, also they are less complex and more efficient than the normal NNs with Dense Layers. The Convolution is applied between the image and a kernel, the kernel is shifted over the whole image to obtain the convolution of it. The problem with this is if we are in the edges or borders of the image, in that case we can do 2 things, we ignore the edges, so the result will be less in dimensions than the original image, or we apply a padding so we can compute the normal operation including those edges, this can be done by zero padding (this can be seen in Figure 4 taken from an exercise to calculate the dimension from the output based on the Kernel size and the input image with the 0's around it or zero padding). The equation for the Convolution is given by:

$$(I * K)(i,j) = \Sigma_m \Sigma_n I(i+m, j+n) \cdot K(m,n) \qquad (13)$$

where I stands for the image and K for the Kernel (equation from the slides of the course, authors the Teaching Assistants Remco and Yuqing, lab 6 pdf). This is a dot product or inner product.

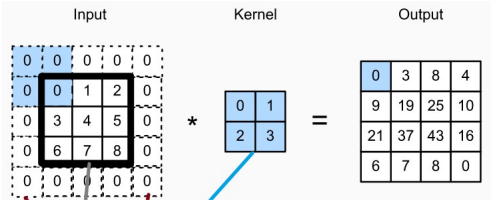


Fig. 4. Zero Padding. Image taken from: CNN. Padding and Strides

After doing this convolution, also we can have some Poolings, normally Max Pooling, is just a simple Math operation so it does not count parameters in the Network (this will be shown later in the results of our training methods), what you do in Max Pooling, is also having a Kernel, but instead of doing a dot product, the only thing to do is to get the Maximum value (that's where the name comes from) between all the values taken in that Kernel matrix, is like taking the most representative value between all of them, sometimes we don't want many Max Poolings because you lose some information doing this (some pixels are going to be omitted) but it helps in accelerating the process of the training because you reduce significantly the dimensions of the data to be processed. An example of Max Pooling can be seen in the Figure 5.

Other concept/component in the Convolution and Pooling operations apart from the Kernel size (dimensions of the Kernel matrix, normally square shaped, but can be rectangular, so not just 2x2 or 3x3 but 2x3 for example) and padding (to solve the edges issue), is the stride, that is the shift amount that the Kernel will have during the operations, normally is taken as 1, so the Kernel goes step by step through all the image, as shown in both Figures 4 and 5 already mentioned, but it could have a value of 2 or more, in that case the Kernel will be calculated after shifting 2 columns to the right, and so on (we can also talk about horizontal and vertical strides, but we just mention them as it's not so important).

4. Recurrent Neural Network and Long Short-Term Memory: The RNNs are another special case of NNs, where we

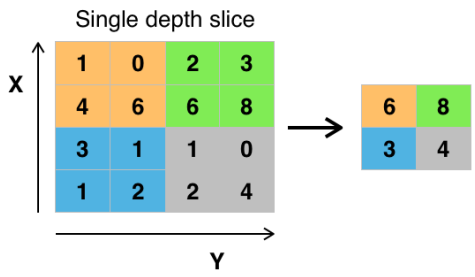


Fig. 5. Max Pooling. Image taken from: CNNs

use the concept of recurrent neurons, in the sense that these networks can process sequential data, what is ideal for music, as waveform is like a time series of data. Also MFFCs can be treated as multivariate time series, because for each interval of time we get 13 coefficients, more measurements for each interval. In an RNN the prediction depends on the previous state as well. The architecture in simple words of an RNN is the following: It has an input, a recurrent layer, a dense layer and an output, as shown in Figure 6, here the input X has a 3D shape, the Recurrent layer process the data in a sequential way, the Dense layer is a classification one like Softmax in our project, and Y the output. In the 3D shape of X we have the Batch size as 1st dimension (number of samples or time windows), the 2nd dimension is the number of steps in the sequence and the 3rd is the number of dimensions or different measures that we have in the sequence itself, e.g. in the case of a waveform this is equal to 1, in case of MFCCs is equal to 13 in our project. In the RNN the important element is the cell because that's the one that processes the sequential data. We have an input, output and a state for the cell, that is like a hidden (H) state for the next step. This can be seen as many cells in chain, as shown in the Figure 7.

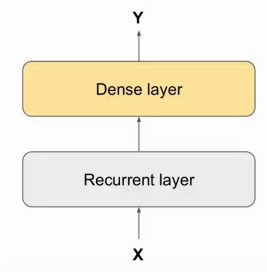


Fig. 6. RNN simple architecture

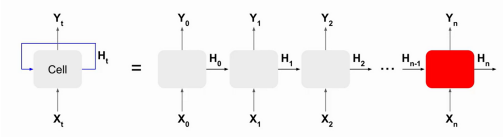


Fig. 7. RNN cells

We can have different configurations of RNNs, like sequence to sequence (where all the outputs at each time step are kept), sequence to vector or vector to sequence, or combinations of those like in the last image (right below) shows a sequence to vector + a vector to sequence architecture, all of this shown in Figure 8, in Keras the default value is a sequence to vector architecture as is more common.

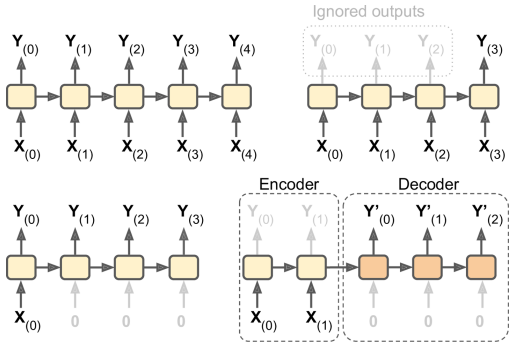


Fig. 8. RNN types. Image taken from Sorts of RNNs

The memory cell in simple RNNs is a simple NN, where you have a Dense layer with input having the input data (at the current time step) + state vector (at the previous time step) and as activation function tanh (hyperbolic tangent), why is the case of using this type of function? and not ReLU? (Rectified Linear Unit that is 0 for values minor than 0 and the same value for values bigger or equal to 0), because with ReLU as the function grows with bigger values, we can have exploiding gradients, as it is not bounded, but tanh constraints the values between -1 and 1, so avoids the exploding issue. And what about vanishing gradient? When we have deep networks, in the backpropagation of the error, travels through more neurons so it will vanish, it comes smaller and smaller. This can be seen in Figure 9, to fix this we can calculate the errors at each time step by using the sequence to sequence RNN, but we are interested at the end in the last output, so we can drop later the rest of the sequence, this is shown in Figure 10.

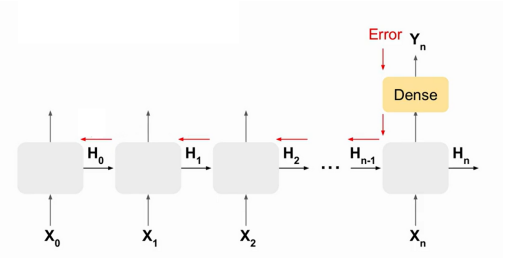


Fig. 9. RNN Backpropagation Through Time

Focusing on one time step we can analyze the math behind the RNNs. Input at time t is xt, ht is the state vector at time t and the state vector at t-1 ht-1, yt the output; U, V and W are the weight matrixes conntected to their respective inputs/outputs. ht is a function of xt the input state and the previous step ht-1, and the output is just an activation function of ht with weight V. The weight matrixes are the ones the RNN

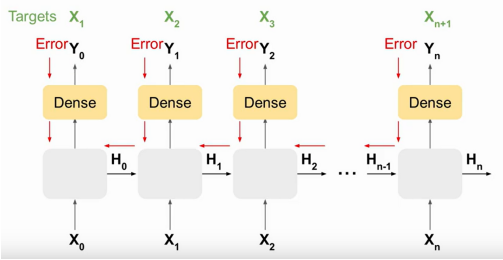
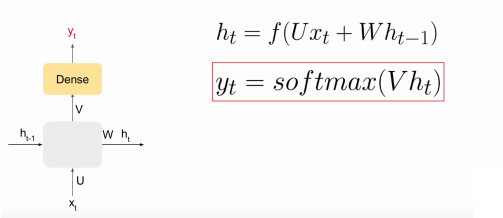


Fig. 10. RNN Errors at each time step

is going to learn, all of these is depicted in Figure 11, where the activation function used is a softmax.



$$h_t = f(Ux_t + Wh_{t-1})$$

$$y_t = softmax(Vh_t)$$

Fig. 11. RNN mathematical expressions

The problem with normal RNNs is that they don't have long term memory, and the music is structured in that way, the verse and the chorus for example, are in some times of the song, and they can be related, but without long term memory is a problem to train audio data. Therefore we use a special case of RNNs called Long Short-Term Memory RNNs, or LSTM-RNNs.

The LSTMs can learn pattern with long dependencies. Works well with 100s of steps but struggle with hundreds and thousands of steps. The LSTM changes in the cell itself, where it has more than just an activation function (tanh in the example shown in the Figures 12 and 13). The LSTM has some components: an LSTM cell contains a simple RNN cell, and second state vector or cell state also known as long-term memory, three gates: forget, input and output gate (these work as filters).

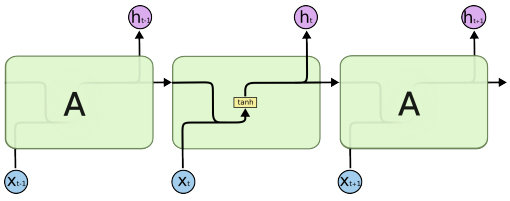


Fig. 12. Simple RNN. Image taken from: Understanding LSTMs

Here we have the cell state (that's why is represented as ct and ct-1), where the operation of multiplication decides what to forget, and the sum what information to remember. This shown in Figure 14. In the first stage shown in Figure 15 the input vector is concatenated with the previous state vector, applied a weight to them and adding a bias term, all that goes

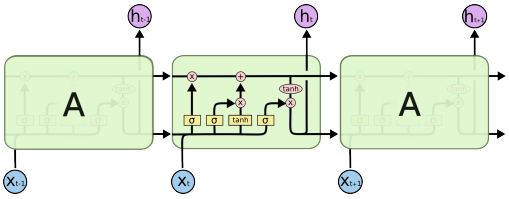


Fig. 13. LSTM. Image taken from: Understanding LSTMs

through a Sigmoid activation function represented as σ, that keeps the values between 0 and 1, 0 to forget and 1 to keep, and the element wise multiplication allows that to happen.

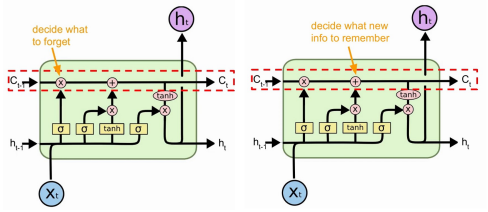
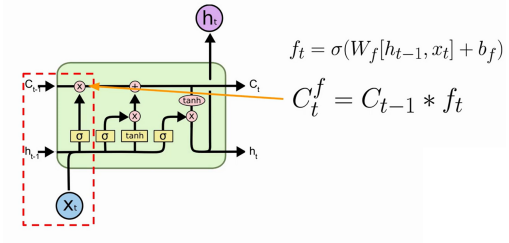


Fig. 14. LSTM cell state. Image taken from: LSTM easily

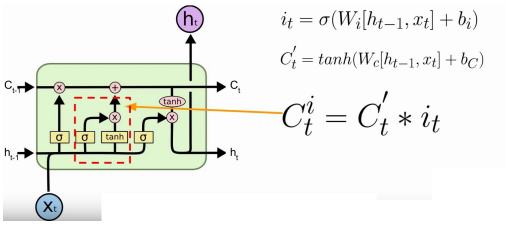


$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f)$$

$$C_t^f = C_{t-1} * f_t$$

Fig. 15. LSTM forget gate. Image taken from: LSTM easily

In the second stage, the input goes through the tanh function and a Sigmoid one, here shown in the Figure 16 the term it is which decides what is important from Ct' (new cell state).



$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$$

$$C'_t = tanh(W_c[h_{t-1}, x_t] + b_C)$$

$$C_t^i = C'_t * i_t$$

Fig. 16. LSTM keeping important information gate. Image taken from: LSTM easily

Then both Ct's (what to forget+what to keep/important) are sumed by the element wise sum, to get the new cell state Ct, as depicted in Figure 17.

And the last stage or gate, where we have a Sigmoid function, and then that filtered input (output) is wise multiplied by a tanh from the cell state Ct, to constrain the value from -1 to 1. The output is ht used as current hidden state and output for the dense layer, as can be seen in Figure 18.

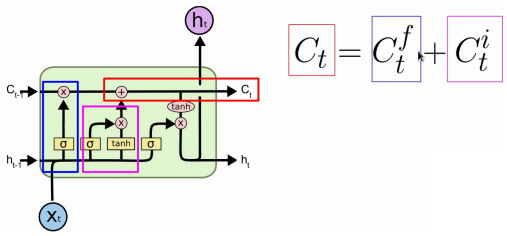


Fig. 17. LSTM, forget+important step. Image taken from: LSTM easily

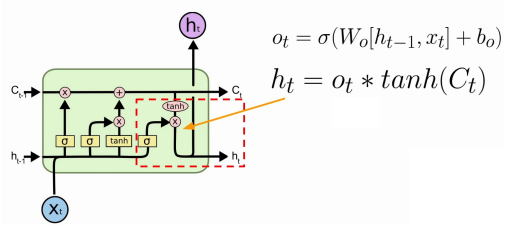


Fig. 18. LSTM output gate. Image taken from: LSTM easily

With all the theory behind explained we applied it to the data set we got in the preprocessed music files (*.json) and we get some results, taking the Loss or error and the Accuracy as measures of performance of each training model (NN, CNN, LSTM). The results are shown in the Figures 19, 20 and 21.

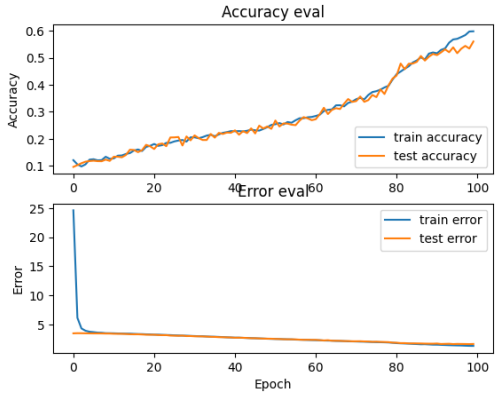


Fig. 19. NN Result

In case of LSTM we compare the model trained with 30 epochs and with 60 epochs to see how much improved the accuracy/loss.

These results are also included in the repository that we have in github, as mentioned in the acknowledgment part. The percentages for the models are the following:

56% test accuracy and 1,66 of Loss for NN with dropout
69% test accuracy and 0,76 of Loss for CNN
62% test accuracy and 1,11 of Loss for LSTM

VI. Experimental Results

A. Choice of Activation function

Keras library provides several activation functions to be chosen within the dense layer. Those are:

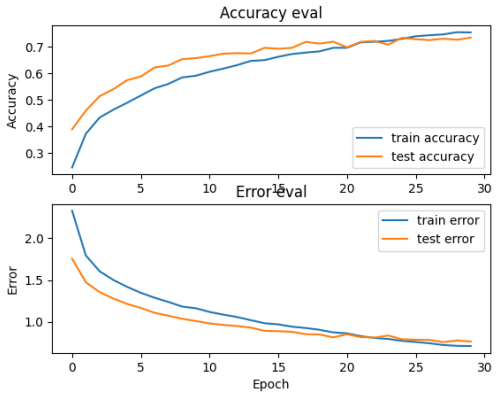


Fig. 20. CNN Result

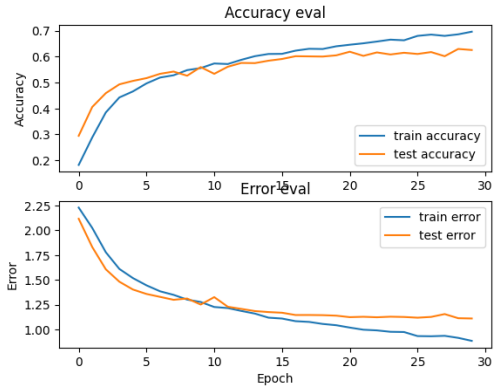


Fig. 21. LSTM Result

1) ReLU: Rectified Linear Unit, which returns the input if it is positive and 0 otherwise.
2) Sigmoid: Sigmoid function, which squeezes the input values between 0 and 1.
3) Tanh: Hyperbolic tangent function, which squeezes the input values between -1 and 1.
4) Softmax: Softmax function, which converts the input values into a probability distribution over multiple classes.
5) Linear: Linear activation function, which applies a simple identity function and does not introduce non-linearity.

Softmax activation function was fixed for the last layer as it converts the output of the network into a probability ditribution which is most convenient for making predictions. For the rest of the previous layers the other functions where tested and the following results were observed in Figure 22.

It was noticed that the accuracy reached with ReLU was the best and hence we chose it for the rest of our project. However it is worth to mention that although higher accuracy was reached, but only after 80 epochs. This can be reasoned due to gradient saturation in the other activation functions cases that ReLU does not suffer from, since its gradient is constant for positive inputs. And regarding the error analysis,

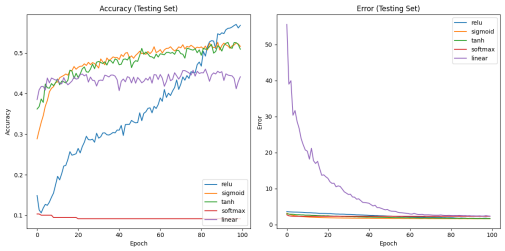


Fig. 22. Network Accuracy and Error as function of activation functions

it was observed that the linear activation function resulted in the highest error which took longer time to decrease. This can be interpreted due to not introducing of non linearity which mitigates the ability to model complex relationships and patterns in the data.

B. Compare the performance of different optimizers in Neural Networks

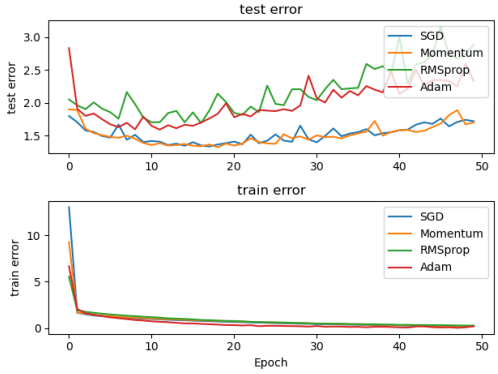


Fig. 23. Neural Network Without dropout

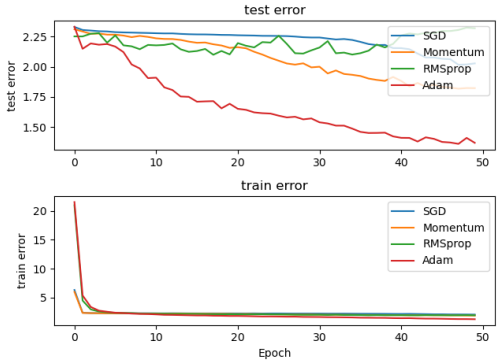


Fig. 24. Neural Network applying dropout

In this section, we delve into a comparative analysis of various optimizers, including Adam, RMSprop, SGD, and Momentum. These optimizers play a pivotal role in updating the model's parameters during the training process. Before exploring their impact, it is important to note that we have selected the rectified linear unit (ReLU) activation function for our neural network, as it has proven effective in the previous section.

When comparing different optimizers, it's important to consider their underlying algorithms and how they update the model's parameters during training.

1. Adam and RMSprop: Adam and RMSprop are adaptive optimization algorithms that adjust the learning rate for each parameter based on the magnitude of past gradients. They incorporate momentum and adaptive learning rate techniques to converge faster and handle different data distributions. These optimizers tend to perform well in terms of training error because they effectively navigate the optimization landscape, quickly reaching regions with lower training loss. However, they can be more prone to overfitting on the training data, as they may adapt too quickly to the specific patterns or noise in the training set. This can lead to worse performance on unseen data (test data) due to over-optimization as illustrated in Figure 23.

2. SGD and Momentum: SGD and Momentum are traditional optimization algorithms. SGD updates the parameters by directly multiplying the gradient with a fixed learning rate, while Momentum adds a fraction of the previous update to the current update. These optimizers have slower convergence but tend to generalize better on unseen data. They are less prone to overfitting because they update the parameters more gradually and smooth out the optimization process. As shown in Figure 23, we could observe that they have slightly higher training error compared to Adam, but they exhibit better performance on test data due to their regularization effect.

When dropout is applied, it introduces a form of regularization by randomly dropping neurons during training. This prevents the model from relying too heavily on specific neurons and encourages more robust representations.

As shown in Figure 24, it can be observed that after applying dropout, Adam has the best performance compared to other optimizers in both the test and train data.

C. Compare the performance of different extraction methods

In addition to the MFCCs, librosa package offers many other extraction methods [13]. In this package, different extraction methods are categorized under two main classes: Spectral features and Rhythm features. The paragraphs below explains some selected extraction methods.

- Chroma extraction: In Western music, 12 pitches are used. Chroma features are obtained by projecting the entire spectrum onto 12 bins representing the 12 distinct semitones (or *chroma*) [17] As explained in detail in [18], there are different methods for computing chroma features:
  - 'chroma_stft' uses STFT (Short-Time-Fourier-Transform) to analyze the spectral content.
  - 'chroma_cqt' computes CQT (Constant-Q Transform) which is a frequency analysis method that provides a frequency representation in the logarithmic scale [19].

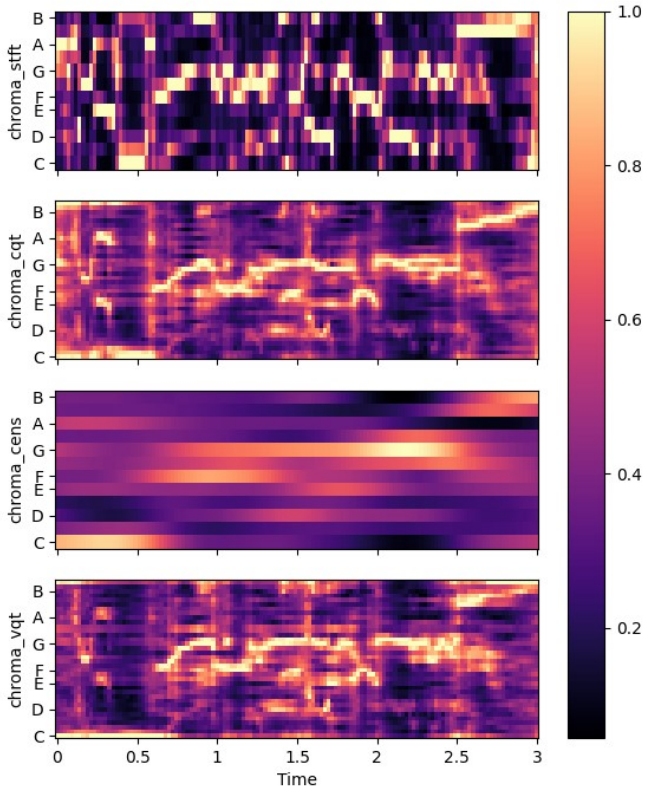


Fig. 25. Chromagram of a blues song using different methods

- 'chroma_cens' provides CENS (Chroma Energy Normalized Statistics) features which add a further degree of abstraction [18].
- 'chroma_vqt' implements VQT (Variable-Q Transform). VQT is a similar method to CQT but VQT allows for different filterbanks to be used in each downsampled octave [20].

Figure 25 shows the chromagrams of a blues song computed using four aforementioned approaches.

- Tempogram and tempogram ratio: Computes tempogram and tempogram ratio features, alternatively named spectral rhythm patterns [21], [22]. The comparison of tempogram and tempogram ratios features of two music genres are illustrated in Figure 26.
- RMS: Calculates the root-mean-square (RMS) value for every frame, which can be derived either from the audio samples or the spectrogram. In this work, the first one is implemented. Figure 27 depicts spectrogram and its RMS for two genres.
- Spectral constrast: Computes the spectral contrast [23]. Power spectrograms and spectral constrasts of two music genres are shown in Figure 28.
- Spectral flatness: Computes spectral flatness [24].
- Polynomial features: Obtain the coefficients for fitting a polynomial of order n to the columns of a spectrogram.

To assess the accuracy performance of the aforementioned

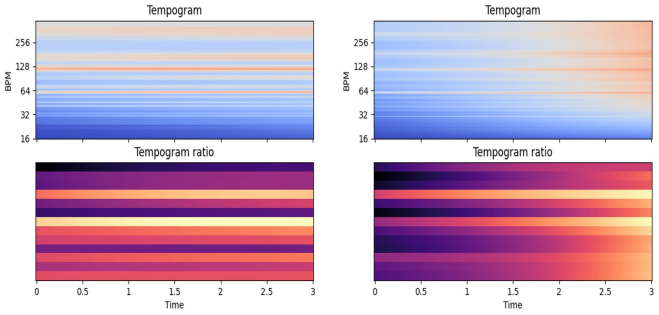


Fig. 26. Tempograms of a blues and a rock music. Left figure: Blues, right figure: Rock

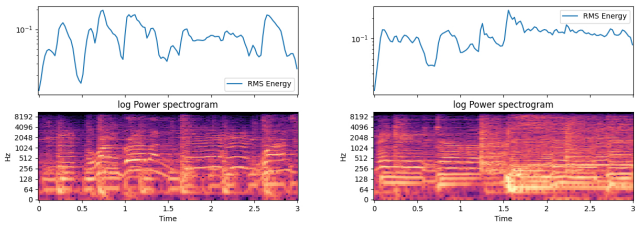


Fig. 27. Tempograms of a blues and a rock music. Left figure: Blues, right figure: Rock

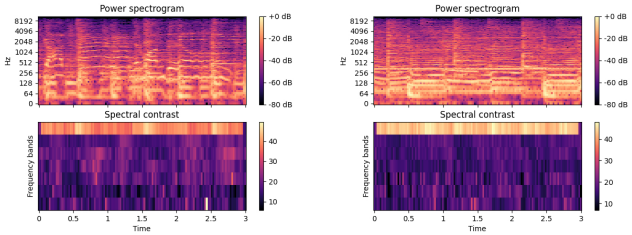


Fig. 28. Tempograms of a blues and a rock music. Left figure: Blues, right figure: Rock

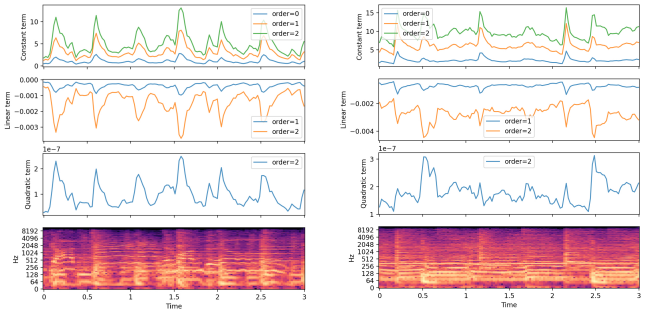


Fig. 29. Tempograms of a blues and a rock music. Left figure: Blues, right figure: Rock

	CNN	
	Validation accuracy	Test accuracy
mfcc	74%	69%
chroma_cqt	71%	54%
chroma_stft	56%	48%
chroma_vqt	53%	48%
chroma_cens	43%	41%
tempogram_ratio	34%	32%

Fig. 30. Performance comparison of different extractors-CNN

	RNN	
	Validation accuracy	Test accuracy
mfcc	66%	62%
spectral_contrast	49%	46%
poly_features	40%	37%
spectral_flatness	33%	31%
rms	24%	23%

Fig. 31. Performance comparison of different extractors-RNN

extraction methods described earlier, a Convolutional Neural Network (CNN) model was employed for the analysis of the 2D dataset, while a Recurrent Neural Network (RNN) model was utilized for the examination of the 1D dataset. The results are shown in Figures 30 and 31. Based on the results obtained, it can be concluded that the MFCC extraction method outperforms the other methods.

VII. Conclusion

In this work, a data pre-processing approach is introduced. NN, NN with dropout, CNN, and RNN-LSTM models are trained for the music genre classification. The effect of well-known optimizers and activation functions on the accuracy is compared. Finally, several feature extraction methods are compared. Acccording to our results, CNN with MFCCs feature and using Adam optimizer gave the best accuracy performance for the music genre classification problem.

VIII. Acknowledgments

Jialun Kou led the efforts in the project's introduction, theoretical explanation of optimizers, and Dropout. Additionally, he conducted practical experiments specifically focused on comparing the performance of different optimizers in neural networks.

Giovanny Rico did the abstract (later modified and improved by Jialun Kou into the introduction section), the theory behind each model used and comparison of performances between the NN, CNN and RNN-LSTM models; as well he was in charge of the training of the models NN with dropout, CNN and RNN-LSTM (this last one using a different number of epochs just to show how much changes the Loss/Accuracy with more epochs).

Mohamad Saab was in charge of the preprocessing of the dataset, to get a json file with all the data already preprocessed to be used for training. To do that, the theory behind MFFCs feature extraction was studied, explained and applied in the algorithm. He also was responsible about testing several activation functions to compare and choose the best.

Mehmet Fatih Ayten was in charge of the comparison of different other features apart from the MFCCs and their performances in practice, and he also explained the State of the Art section.

Between all of us we did the final conclusions, after joining our contributions in the whole project/report.

Data Availability Statement: All the code in this project was implemented using TensorFlow (Keras) and Librosa libraries. The implementation was heavily influenced by the labs provided in the course and tutorials available on YouTube https://www.youtube.com/watch?v=fMqL5vckiU0&list=PL-wATfeyAMNrtbkCNsLcpoAyBBRJZVlnf. To access the codes used in this project, please visit the following link: https://github.com/KJialun/Music-classification.

References

[1] Li T, Ogihara M, Li Q. A Comparative Study on Content-Based Music Genre Classification. In: Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Informaion Retrieval. SIGIR '03. New York, NY, USA: Association for Computing Machinery; 2003. p. 282–289. Available from: https://doi.org/10.1145/860435.860487.

[2] Ahrendt P. Music Genre Classification Systems - A Computational Approach; 2006.

[3] van den Oord A, Dieleman S, Schrauwen B. Deep content-based music recommendation. In: Burges CJ, Bottou L, Welling M, Ghahramani Z, Weinberger KQ, editors. Advances in Neural Information Processing Systems. vol. 26. Curran Associates, Inc.; 2013. Available from: https://proceedings.neurips.cc/paper_files/paper/2013/file/b3ba8f1bee1238a2f37603d90b58898d-Paper.pdf.

[4] Li T, Ogihara M, Li Q. A Comparative Study on Content-Based Music Genre Classification; 2003. p. 282-9.

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

[6] Bahuleyan H. Music Genre Classification using Machine Learning Techniques; 2018.

[7] Costa YMG, Oliveira LS, Silla CN. An evaluation of Convolutional Neural Networks for music classification using spectrograms. Applied Soft Computing. 2017;52:28-38. Available from: https://www.sciencedirect.com/science/article/pii/S1568494616306421.

[8] Vishnupriya S, Meenakshi K. Automatic Music Genre Classification using Convolution Neural Network. In: 2018 International Conference on Computer Communication and Informatics (ICCCI); 2018. p. 1-4.

[9] Pelchat N, Gelowitz CM. Neural Network Music Genre Classification. In: 2019 IEEE Canadian Conference of Electrical and Computer Engineering (CCECE); 2019. p. 1-4.

[10] Kaur C, Kumar R. Study and analysis of feature based automatic music genre classification using Gaussian mixture model. In: 2017 International Conference on Inventive Computing and Informatics (ICICI); 2017. p. 465-8.

[11] Pedersen P. The Mel Scale. vol. 9. [Duke University Press, Yale University Department of Music]; 1965. p. 295-308. Available from: http://www.jstor.org/stable/843164.
[12] Davis S, Mermelstein P. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. vol. 28; 1980. p. 357-66.
[13] Librosa. Librosa: Audio and Music Signal Analysis in Python; Accessed: 2023. https://librosa.org/doc/main/feature.html.
[14] Ruder S. An overview of gradient descent optimization algorithms. CoRR. 2016;abs/1609.04747. Available from: http://arxiv.org/abs/1609.04747.
[15] Kingma DP, Ba J. Adam: A Method for Stochastic Optimization. In: 3rd International Conference for Learning Representations; 2015. Available from: https://doi.org/10.48550/arXiv.1412.6980.
[16] Srivastava N, Hinton GE, Krizhevsky A, Sutskever I, Salakhutdinov R. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research. 2014;15(1):1929-58.
[17] Ellis DPW. Chroma feature analysis and synthesis;. Available from: https://www.ee.columbia.edu/~dpwe/resources/matlab/chroma-ansyn/.
[18] Müller M, Ewert S. Chroma Toolbox: Matlab Implementations for Extracting Variants of Chroma-Based Audio Features. In: International Society for Music Information Retrieval Conference; 2011. .
[19] Brown J. Calculation of a Constant Q Spectral Transform. Journal of the Acoustical Society of America. 1991 01;89:425.
[20] Chandrakala S, M V, N S, L JS. Multi-view representation for sound event recognition. Signal, Image and Video Processing. 2021 September;15(6):1211-9. Available from: https://doi.org/10.1007/s11760-020-01851-9.
[21] Peeters G. Rhythm Classification Using Spectral Rhythm Patterns. In: International Society for Music Information Retrieval Conference; 2005. .
[22] Grosche P, Müller M, Kurth F. Cyclic tempogram—A mid-level tempo representation for musicsignals. In: 2010 IEEE International Conference on Acoustics, Speech and Signal Processing; 2010. p. 5522-5.
[23] Jiang DN, Lu L, Zhang HJ, Tao JH, Cai LH. Music type classification by spectral contrast feature. In: Proceedings. IEEE International Conference on Multimedia and Expo. vol. 1; 2002. p. 113-6 vol.1.
[24] Dubnov S. Generalization of spectral flatness measure for non-Gaussian linear processes. IEEE Signal Processing Letters. 2004;11(8):698-701.