Light It Up
Predicting Song Features to Motivate Lighting Choice
Project Category: Audio and Music

Ahmed Abdalla, Mark Bechthold, and Tristan Saucedo
Departments of Computer Science and Electrical Engineering, Stanford University
(Dated: June 28, 2024)

I. Introduction

A staple of any college dorm room is the programmable LED lighting strip. These lights often come with the supposed ability to sync their hue and intensity to the ambient soundscape. But, as any college student can tell you, this preprogrammed function is often too inaccurate and awkward to be of any use. We applied machine learning methods to help solve this problem.

We predict qualitative classifications (genre) using K-Means classification on a wide set of features. Then, we train an algorithm to predict the quantitative audial characteristics (pitch and timbre) at any point in a song using what the algorithm has previously heard. The recursive nature of this problem naturally led us to explore recurrent neural networks (RNNs). We benchmark this approached against a fully connected neural network (FCNN) and logistic regression (LR) to confirm that a RNN is well-suited to tackling this problem.

Armed with this predictive ability, we built an open source pipeline to translate our model outputs into RGB color sweeps on an LED strip light controlled by a Raspberry Pi. We combined Western notions of tonality with our own subjectivity to systematically extract relative notions of tension and movement in a piece. For a given genre, these values map to specific locations on predetermined color bars. The lighting design anticipates upcoming musical changes before they occur. The model adjusts using the new audio data allowing us to accomplish accurate song feature prediction and solve a perennial problem for college students everywhere.

II. Related Works

Music Information Retrieval (MIR) is a field of research that focuses on the development of algorithms and methods for automatically extracting, analyzing, and representing musical data. Some of the most common tasks in MIR include audio segmentation, audio classification, music generation, music recommendation, music similarity, and music structure analysis. We explored genre prediction and focused on music generation (pitch and timbre prediction using time-sequenced data).

Previous research in MIR has included the use of machine learning techniques such as Support Vector Machines (SVMs) and Neural Networks (NNs) to classify different types of audio data [1, 2]. These techniques were limited in their ability to recognize complex musical structures, like harmony and melody and when used for genre prediction, face difficulty distinguishing between similar genres.

A. Genre Prediction

Modified versions of the K-Means algorithm have been used to understand and predict musical genres [3]. In addition to audio features such as pitch and timbre, authors have also used lyrical data and sound intensity to achieve higher fidelity classification [4]. Such work used augmented versions of The Million Song Dataset, the same dataset we use in our training [5].

We noted that the most accurate genre prediction software uses random forest algorithms, but elected to pursue a different classification method because these results were reported on much smaller datasets [6, 7]. It was our hope to build a more accurate classifier using a fundamentally different approach.

B. Audial Prediction

Deep learning techniques have also been used to address the task of MIR. Convolutional Neural Networks (CNNs) have been used to recognize musical patterns from spectrograms [8], while RNNs have been used to generate new music from a dataset of existing musical pieces [9]. Researchers have also used deep neural networks to augment digital musical instruments, such as the electric keyboard [10]. Additional work leverages RNNs to generate music in different styles or genres [11]. These generative models have a similar goal to ours; we are given the part of the song that has been played and predict the upcoming pitch and timbre features while they predict a whole new set of pitches and features from an existing song.

The best generative music models use RNNs but have yet to be leveraged for generative lighting design [12].

III. Dataset and Features

We trained our models using subsets of the Million Song Dataset [5]. We were drawn to this dataset because it contained features that we hypothesized could be naturally translated into aesthetic lighting. We specifically focused on two $N \times 12$ arrays which contained pitch and timbre information. Here N refers to the number of segments in a song, where songs are broken up

into segments using an unknown algorithm that identifies melodic changes. These segments have a mean time interval of 0.33 and a standard deviation of 0.23. The 12 columns refer to the 12 pitch classes in Western equal temperament. For a given octave, playing one note from each pitch class yields a chromatic scale. As such we refer to the pitch matrix, where values are normalized such that the greatest entry per row is 1, as the chroma matrix.

The timbre matrix contains Mel-Frequency Cepstral Coefficients (MFCCs). The human ear is an imperfect sound sensor making raw sonic intensity a poor metric for perceived loudness. The Mel scale corrects for this by adjusting intensity for human perception. The Fourier transform power spectrum is mapped to the log-Mel scale for each of the twelve chromatic pitches. The inverse Fourier transform of these values gives the MFCCs in our dataset. These type of cepstral features are common in speech recognition and instrument classification research, often referred to as quefrency analysis [13, 14].

For our FCNN and RNN we ignored metadata and only used these time-series features. For K-Means genre prediction, we included metadata (song length, tempo, time signature, and key), but removed smaller genres with overlap to make the problem more tractable, keeping only hip-hop, punk, dance and electronica, jazz and blues, soul and reggae, classic pop and rock, folk, classical, pop, and metal. We chose these 10 genres based on previous work [4, 12]. Additionally, we normalized the genre size and explored smaller subsets of genres. Some challenges we faced were the subjective nature of genres ("pop" vs "classic pop and rock" vs "rock and indie"). This data was also averaged over the entire length of the song, and thus not discretized over time.

IV. Methods

A. Model Design

There were two distinct parts of our analysis. First, we used K-means and Principle Component Analysis (PCA) to explore our data, and better understand differences in genres of music. Then we used two neural networks to predict pitch and timbre features of songs in real time.

K-Means is a clustering algorithm that can be used to group data points into clusters based on their similarity. In music genre prediction K-means can identify patterns in the data that are indicative of certain genres. For example, by using k-means, a model could be created that identifies patterns in the frequency, intensity, and timbre of different songs to identify what genre they belong to. Using metadata of songs, such as length, tempo, and time signature, can also help us classify different genres.

Principal Component Analysis (PCA) is a dimensional reduction technique that can be used to reduce the number of variables in a dataset, or to find the most explanatory vector multidimensional data. In music genre prediction PCA can help to identify the most important features in the data that are indicative of different genres. We used PCA to explore the variance of genres, specifically on chroma and timbre features. Additionally, we used the songs metadata, including length and tempo, to augment the classification.

A **Classical Neural Network** was used to predict the upcoming pitch, and timbre, in our time sequence data. To do this we created a sliding window of 20 segments as our input example and tried to predict the next segment in the song. Effectively our input was a 480 dimensional vector of inputs comprised of the the 20 sets of 12 chroma and 12 timbre values flattened together. We then predicted the output which was a 24 dimensional vector corresponding to the 12 chroma and 12 timbre values of the next segment. We picked a sliding window of 20 because it encoded $\sim$5 seconds of a given song and seemed a relevant amount of data for predicting the next value in the sequence. Using this data we then split it into a training, dev, and test set comprising 70%, 20%, and 10%, respectively. With this architecture we started by establishing a baseline where we performed linear regression without any intermediate layers in our network. Then we created a neural net by adding intermediate layers to the regression of varying size using ReLU between layers. Using the python package Ray tune, we sampled various hyperparameter configurations for the learning rate, batch size, and the number of intermediate layers and their associated sizes. We sampled learning rates from a uniform log from 0.1 to 0.0001 and our batch size from 32, 64, or 128-element batches. We ran 5 such samples for 100 epochs, and to assess our performance we used Mean Square Error, according to the following equation:

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

where n is notably not the sequence length but the size of the batch.

Additionally, we used a **Recurrent Neural Network** (RNN), leveraging the sinusoidal nature of music to make better predictions. Using the same 12 features for timbre and 12 features for pitches to represent a single time segment, we padded our inputs to the maximum length of a song and for a song with n time segments we passed in the first n-1 times segments into the RNN to predict time steps 2 through n. We randomly shuffled the data, and divided our data set into a training, dev, and test set consisting of 70%, 20%, and 10% of the data set, respectively. Due to time considerations we were forced to run our model on approximately an eighth of the total Million Song subset. (Certain songs were too short to compose a sliding window of twenty, so we filtered these out of the data set). We further tuned our model using the python package Ray tune to select hyperparameters. We sampled randomly hidden layers with dimensions between 2^2 and 2^9, and we chose randomly between 1 and 6 RNN layers to stack. Finally we sampled a uniform

log between 0.1 and 0.0001 for our learning rate and our batch size from 32, 64, and 128-element batches. We only ran three such samples due to time considerations. We ran the samples for 50 epochs, and used Mean Squared Error to assess our performance.

B. Lighting Design

Even given perfect knowledge of a songs qualitative and quantitative audial features, generating an accompanying color sweep is not a deterministic problem. Qualification of the final result will depend on an individual's subjective taste meaning that there are a variety of acceptable paths. This extra room for artistic interpretation encouraged us to pursue novel and creative methods for generative lighting design.

We began by explicitly formulating the problem we hoped to solve. For a given musical genre g pulled from a set G and a time series of musical notes $\{x^{(i)}\} \in X = [0,1]^{12}$ and timbres $\{y^{(i)}\} \in Y = \mathbb{R}^{12}$, we would like to generate an accompanying time series of RGB values $c \in C = [\texttt{0x00}, \texttt{0xFF}]^3$ (note that all $x \in X$ are normalized such that $||x||_1 = 1$). That is we seek to design some function f_g such that

$$f_g : \quad X^{n-1} \times Y^{n-1} \to C \tag{1}$$

$$\{x^{(i)}\}_{i=1}^{n-1} \times \{y^{(i)}\}_{i=1}^{n-1} \mapsto c^{(n)}. \tag{2}$$

Note that c_n is only a function of the first $n-1$ element of $\{x^{(i)}\}$ and $\{y^{(i)}\}$. This is a reflection of the fact that our approach hopes to anticipate the next best color in the time series. In retrospect there exists and ideal function

$$\tilde{f}_g : \quad X^{n} \times Y^{n} \to C \tag{3}$$

$$\{x^{(i)}\}_{i=1}^{n} \times \{y^{(i)}\}_{i=1}^{n} \mapsto \tilde{c}^{(n)} \tag{4}$$

where

$$\begin{aligned} &f_g(\{x^{(1)}, \cdots, x^{(n-1)}\}, \{y^{(1)}, \cdots, y^{(n-1)}\}) \\ &= \tilde{f}_g(\{x^{(1)}, \cdots, x^{(n-1)}, \tilde{x}^{(n)}\}, \{y^{(1)}, \cdots, y^{(n-1)}, \tilde{y}^{(n)}\}) \\ &\approx \tilde{f}_g(\{x^{(1)}, \cdots, x^{(n)}\}, \{y^{(1)}, \cdots, y^{(n)}\}) \end{aligned} \tag{5}$$

and $(\tilde{x}^{(n)}, \tilde{y}^{(n)})$ is the output of our model of choice. Our hope is to design f_g to minimize $||c^{(n)} - \tilde{c}^{(n)}||_1$.

A core difficulty with this approach is that there is not one choice of $c^{(n)}$ that a viewer might find aesthetically complimentary to the musical features. To simplify our problem we impose some restrictions on the the range of f_g by conditioning on the genre g (hence the notation). Specifically we restrict our values to a color bar like those shown in Fig. 1. The mapping of color bars to genres was purely subjective and reflects the inherently ill-defined nature of our minimization problem. That said, we did roughly base the design of these color bars on music theoretic concepts.

In Western music theory, a cadence is a sequence of chords which ends a musical phrase. Cadences can be roughly categorized as either perfect or plagal depending on the degree to which they invoke a sense of resolution. Roughly speaking, perfect cadences give a strong sense of finality while plagal cadences imply a soft ending. Both naturally lead back to the tonal center, or key, of a specific piece. For a given key, we assign a number in $[-5, 5]$ to each of the 12 scale degrees. Negative number correspond to notes which occur in chords implying plagal cadences relative to the key while positive number correspond to notes occurring in chords implying perfect cadences.

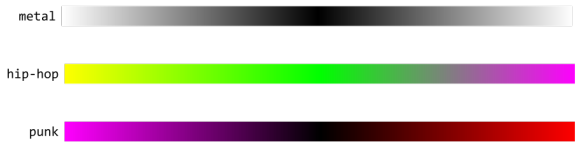

FIG. 1. Color maps for a subset of the genres in the set M

The color bar approach simplifies our problem greatly. Given a note prediction x, we can perform a simple interpolation on the color bar based on the implied cadence to get the associated color. Unfortunately, this simple approach ignores the ambiguity of our program outputs. At each time segment we hear a range of pitches meaning that, generally speaking, x is not a one-hot vector. To account for this we define for each key center $p \in \mathbb{Z}/12\mathbb{Z}$ a movement cadence vector m_p such that $m_p^T x \in [-5, 5]$ gives the expected cadence movement of a time segment. Of course, this assumes that we know our key center. While we could have also learned this feature, such an approach ignores that key centers often change mid-song. Instead we treat the key center as a moving weighted average. This equates to replacing our movement vector with a movement matrix

$$M_{pq}^n := M_{pq}^{n-1} + \alpha \tilde{x}_p^{(n)} (m_p)_q^T \tag{6}$$

where α is a hyperparameter encoding how we weight new notes relative to old ones when approximating the key center. Since pitch vectors are normalized relative to the ℓ_1-norm, we also normalize M^n by the ℓ_1 matrix norm. This means that $M^n x \in [-1, 1]$. To see this recall that

$$||M||_p := \sup_{||x||_p = 1} \frac{||Mx||_p}{||x||_p}. \tag{7}$$

Another important musical feature we can extract from our model outputs is the expected dissonance in the next time segment. Given two notes $p, q \in \mathbb{Z}/12\mathbb{Z}$, the distance between them $d(p, q) = |p - q|$ is a rough measure of consonance. For a given key center p we define a vector d_p where $(d_p)_q = d(p, q)$ and $d_p^T x$ gives the expected dissonance. To account for uncertainty in the key we upgrade this to

$$D_{pq}^n := D_{pq}^{n-1} + \alpha \tilde{x}_p^{(n)} (d_p)_q^T \tag{8}$$

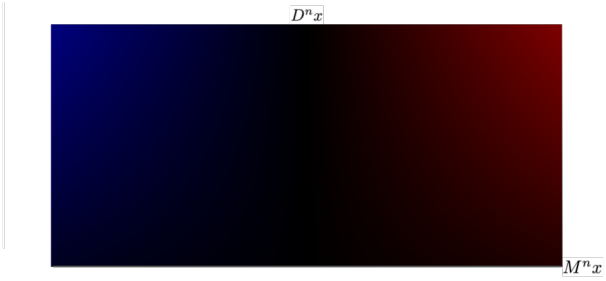

FIG. 2. The color plane associated with the jazz colorbar. The x-axis measures movement cadence and the y-axis measures dissonance.

and again normalize D^n relative to the ℓ-1 matrix norm. This guarantees $D^n x \in [0, 1]$. This dissonance calculation becomes the saturation of our RGB values, mapping our predictions $\tilde{x}^{(n)}$ to points $(M^n \tilde{x}^{(n)}, D^n \tilde{x}^{(n)})$ corresponding to colors.

(see Fig. 2).

C. Hardware and Firmware

As a proof of concept, we implemented the methods in IV-A and IV-B in a Raspberry Pi Model Zero 2 W running Debian Lite. This lightweight computer was wireless connected to Stanford residential networks to allow remote control via SSH and remote firmware uploads and downloads through Github. The Raspberry Pi General Purpose Input/Output (GPIO) pins were used to control Tenmiro LED light strips, however we anticipate that most commonly available LED light strips will be compatible with our implementation. Arbitrary RGB color values were achieved through software and hardware-based Pulse Width Modulation (PWM).

V. Experimental Results and Discussion

We started by quantifying our dataset using unsupervised methods to better understand our predictive power. Fig. 3 contains a PCA analysis which highlights that the expected variance is poorly captured by the first few principle components. This implies a weak relationship between the variables, and prohibits PCA from adequately reducing the dimensions of the data without significant loss of information.

Genre prediction was implemented using K-Means. Recall that the genre is used to define the overall lighting palette of a song. We found that some genres were more distinguishable than others and updated these color palettes accordingly. K-Means provided accurae prediction on subsets of the data (Fig 4) but struggled to achieve such high accuracy when we included all 10 genres. Our model does about 3 times better than random prediction. Given the similarity of the features, explored in Fig. 3, this result is compelling.

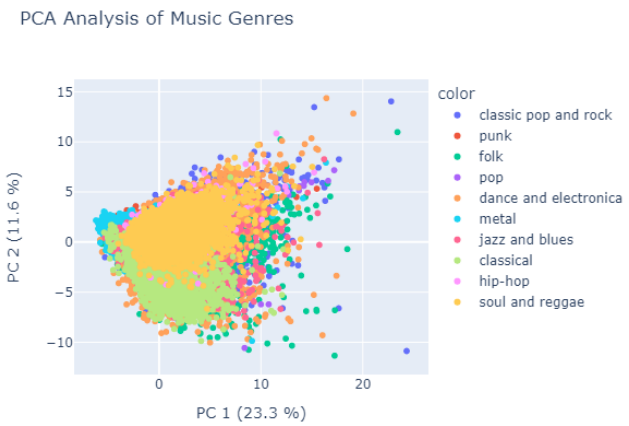

FIG. 3. The first two principle components account for 34.9% of the variance, and the first 10 PC account for barely 70% of the variance. No individual components are able to explain the differences between genres. Notice the large overlap between all genres, indicating timbres low explanatory power.

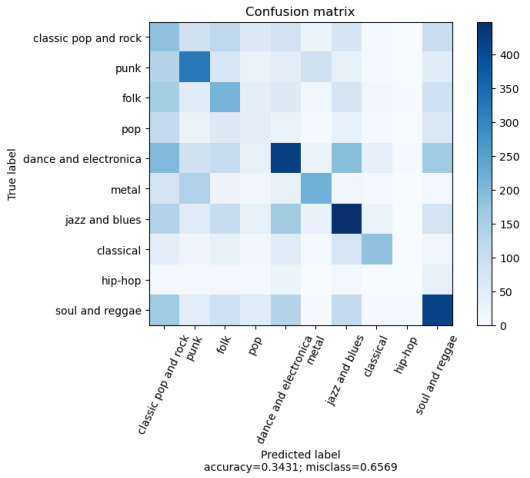

FIG. 4. Predictions for genres proved challenging. Testing was done with roughly 800 samples per class, so 400 corresponds to 50%.

Our baseline linear regression model (learning rate of 0.0001) achieved training and dev losses that vastly outperformed a random guess from the distribution of our data, confirming our conclusions from the literature review: namely, this is a tractable problem (Fig 5). Our test loss for linear regression was 343.955 while random guesses achieved a 1445.886. To further improve on these results, we attempted to create a fully connected neural network with hyperparameters tuned to a learning rate of 4×10^{-6}, a batch size of 128, and two fully connected intermediate layers of 512 neurons with comparable training and dev loss (Figure 5). This model only achieved a test loss of 350.715, which prompted us to pursue the RNN. After parameter tuning, we arrived at

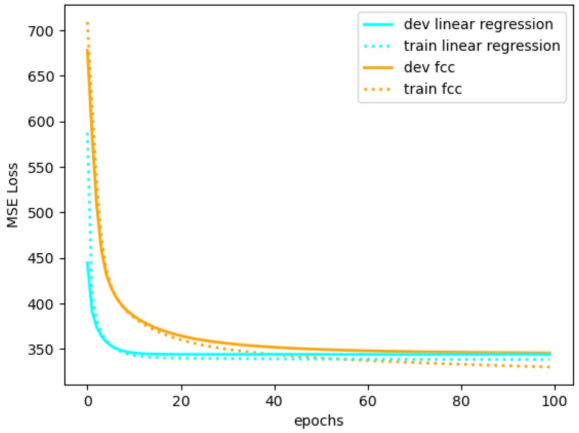

FIG. 5. We plot the training and dev values for both linear regression and a FCNN of two layers over 100 epochs based on our best run of hyper parameter tuning. Both models have comparable performance and final convergence.

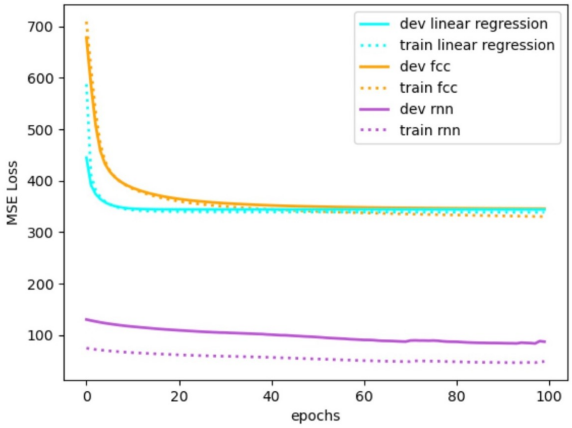

FIG. 6. We plot the training and dev losses for all the linear regression, fully connected and recurrent neural network. We note that the RNN performs significantly better than the other two models.

a learning rate of 0.0161, 16 neurons for the hidden dimension, 5 stacked network layers, and a batch size of 128; this model performed substantially better (Figure 6), achieving a loss value of 88.473 on the test set. To contextualize these loss numbers we also performed an error normalization calculation defined by the following calculation:

$$e = \frac{1}{n}\sum_{i=1}^{n}\frac{|y^{(i)} - \hat{y}^{(i)}|_1}{|y^{(i)}|_1}$$

where $y^{(i)}$ is the true value of the i-th training example and $\hat{y}^{(i)}$ is the prediction and n is the number of examples. We interpret this as the average percent error over all predictions and report normalized error values in Fig. 7. We see that the RNN has by far the lowest test loss and has a much lower normalized pitch error, though its timbre error is slightly larger than the linear regression model. This suggests room for improvement exists in our RNN model.

We then created a new test set of full songs to be fed into the RNN whose accuracy we assessed qualitatively by watching light shows produced on the Raspberry Pi by these predictions. On these data the RNN seemed to improve as the hidden representation of the previous time segments in the song improved. In other words initially our light model had very little information and performed poorly (as the song starts) but as more of the sequence was read in the light show performed better.

	Linear Regression	FCC	RNN
Test Loss	343.955	350.715	88.473
Norm Error Pitches	0.740	1.930	0.624
Norm Error Timbre	0.640	0.656	0.651

FIG. 7. We record the final test performance for each of the models and we see that the RNN vastly outperforms the other two models in terms of test loss. Its pitches error is better than linear regression and the fully connected network, but the timbre error RNN is slightly worse than regression.

VI. Conclusion/Future Work

To better predict genres, we would utilize a novel data processing pipeline. Other papers have found more compelling results utilizing a random forest classifier with more processed and cleansed datasets, and we considered using a semi-supervised k-means classifier. Ultimately, time and compute limited our exploration of genre classification, but a high accuracy genre classifier could be used to select specific models for music prediction and generation. We also used a pre-made dataset, but creating a dataset, or finding a smaller dataset with audiofiles, would allow us to preprocess the data to extract better features.

Predicting the upcoming features of music is a challenging problem, even for most humans with musical training. We found that RNN's were the best for this specific predictive test. While linear regression and our FCNN preformed admirably, the difference in loss between our RNN model and these models shows that RNN's are able to leverage the time series data of music in a way our ordinary neural nets could not. In addition, this method does not enable us to perform true real time prediction for songs we have never fed through our model in advance; nor can we predict the features of new songs to generate light shows if they do not have these audio feature pre-computed.

Regarding the lighting design, there are a few additional features that we believe could have helped us create more pleasing displays. Currently we don't predict any time information and instead use the average segment length to determine the length of each color sweep.

While this works roughly, the weak time resolution has the overall effect of smoothing out our lighting design and rendering us incapable of handling sudden changes in the ambient soundscape. We believe that including the length of each segment as an additional input and output of our RNN could help alleviate this issue. A hardware update we hope to implement in the near future is the addition of a USB microphone to enable live extraction of sound features. Together with improvements to our RNN we can create a better light demonstration from the music.

VII. Contributions

Our names are listed beside the components of the assignment that we contributed to. Contributions are sequential unless otherwise specified. In general, we all worked together, checked each other's code, and supported each other in debugging, writing, and model understanding.

- Report: Ahmed, Mark, Tristan (equal)
- Feature Extraction: Mark, Tristan (equal)
- PCA: Tristan, Mark
- K-Means: Tristan
- Linear Regression: Mark, Tristan
- FCNN: Mark
- RNN: Mark, Ahmed
- Analysis: Ahmed, Mark, Tristan (equal)
- Lighting Design: Ahmed
- Hardware: Ahmed

[1] R. Thiruvengatanadhan, Music genre classification using svm (2018).

[2] N. Pelchat and C. M. Gelowitz, Neural network music genre classification, Canadian Journal of Electrical and Computer Engineering **43**, 170 (2020).

[3] G. Tzanetakis and P. Cook, Musical genre classification of audio signals, IEEE Transactions on Speech and Audio Processing **10**, 293 (2002).

[4] H. G. Dawen Liang and B. O'Connor, Music genre classification with the million song dataset (2011), last accessed 9 December 2022.

[5] 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).

[6] H. Tang, Y. Zhang, and Q. Zhang, The use of deep learning-based intelligent music signal identification and generation technology in national music teaching, Frontiers in psychology **13**, 762402 (2022).

[7] M. Chaudhury, A. Karami, and M. A. Ghazanfar, Large-scale music genre analysis and classification using machine learning with apache spark, Electronics **11**, 2567 (2022).

[8] Y. M. Costa, L. S. Oliveira, and C. N. Silla Jr, An evaluation of convolutional neural networks for music classification using spectrograms, Applied soft computing **52**, 28 (2017).

[9] W. T. Lu, L. Su, *et al.*, Transferring the style of homophonic music using recurrent neural networks and autoregressive model., in *ISMIR* (2018) pp. 740–746.

[10] C. P. Martin, K. O. Ellefsen, and J. Torresen, Deep predictive models in interactive music (2018).

[11] G. Hadjeres and F. Nielsen, Interactive music generation with positional constraints using anticipation-rnns (2017).

[12] Y. Z. Hui Tang and Q. Zhang, The use of deep learning-based intelligent music signal identification and generation technology in national music teaching, Frontiers in Psychology (2022).

[13] I. Maliki *et al.*, Musical instrument recognition using mel-frequency cepstral coefficients and learning vector quantization, in *IOP Conference Series: Materials Science and Engineering*, Vol. 407 (IOP Publishing, 2018) p. 012118.

[14] U. Ubaidi and N. Dewi, Voice pattern recognition using mel-frequency cepstral coefficient and hidden markov model for bahasa madura, in *Journal of Physics: Conference Series*, Vol. 1375 (IOP Publishing, 2019) p. 012057.