Development of a Neural Network-Based Music Genre Classification Tool

Mohammed Ali El Adlouni[1*]

[1*]Master M2 MALIA, Université Lumière Lyon 2, Lyon, France.

Corresponding author(s). E-mail(s): ma.el-adlouni@univ-lyon2.fr;

Abstract

This project focuses on developing a neural network-based music genre classification tool using a dataset with five music genres. The dataset is analyzed through descriptive statistics and baseline classification using baseline Machine Learning models. A neural network model is then designed and optimized by experimenting with different hyperparameters, using heuristics like batch normalization, dropout, and gradient clipping to improve performance. Predictions are made for the test set, and results are exported. The model is extended by incorporating textual information, and the neural network is retrained with a new dataset containing an additional genre, using transfer learning.

Keywords: music genre classification, neural networks, machine learning, transfer learning, textual data

1 Introduction

This project aims to develop a robust music genre classification system using machine learning techniques, focusing on a neural network approach. The project begins with a detailed analysis of the provided dataset, which includes five music genres. Descriptive statistics are performed to better understand the data, followed by the implementation of baseline classification algorithms like Random Forest and XGBoost. These serve as benchmarks for evaluating the performance of more complex models.

A neural network architecture is then developed, and various hyperparameters, such as batch size, number of layers, and learning rates, are experimented with to optimize the model. Heuristics such as batch normalization and dropout are implemented to improve training and prevent overfitting. The best model is used to predict genres for the test dataset, and results are exported.

Further experimentation involves enriching the model by incorporating textual data from song titles using sentence embeddings. A combined model, utilizing both numerical and textual features, is trained and evaluated. The model is then tested on a new dataset, which includes an additional genre, and the effect of this new data is explored using transfer learning and continued training. Results are compared to assess the impact of transfer learning on performance and convergence speed on the new model.

2 Dataset and Classification Task

2.1 Data Acquisition and Description

The dataset used for this analysis consists of 21,428 rows and 20 features. The features include track details such as track name, popularity, album name, release date, playlist details, and various audio features like danceability, energy, loudness, speechiness, acousticness, instrumentalness, liveness, valence, tempo, and duration.

Variable	**Class**	**Description**
track_name	character	Song Name
track_popularity	double	Song Popularity (0-100)
track_album_id	character	Album unique ID
track_album_name	character	Song album name
track_album_release_date	character	Album release date
playlist_name	character	Name of playlist
playlist_id	character	Playlist ID
playlist_genre	character	Playlist genre
playlist_subgenre	character	Playlist subgenre
danceability	double	Suitability for dancing (0.0-1.0)
energy	double	Intensity and activity (0.0-1.0)
key	double	Overall key of the track
loudness	double	Overall loudness in decibels (dB)
mode	double	Modality (major=1, minor=0)
speechiness	double	Presence of spoken words (0.0-1.0)
acousticness	double	Confidence of being acoustic (0.0-1.0)
instrumentalness	double	Presence of vocals (0.0-1.0)
liveness	double	Presence of live audience (0.0-1.0)
valence	double	Musical positiveness (0.0-1.0)
tempo	double	Estimated tempo in BPM
duration_ms	double	Duration in milliseconds

Table 1: Dataset Variables and Descriptions

2.2 Exploratory Data Analysis

2.2.1 Distribution of Genres

The dataset includes a variety of music genres. The distribution of genres is visualized in the pie chart below. We can see that the dataset is more or less balanced between different genres.

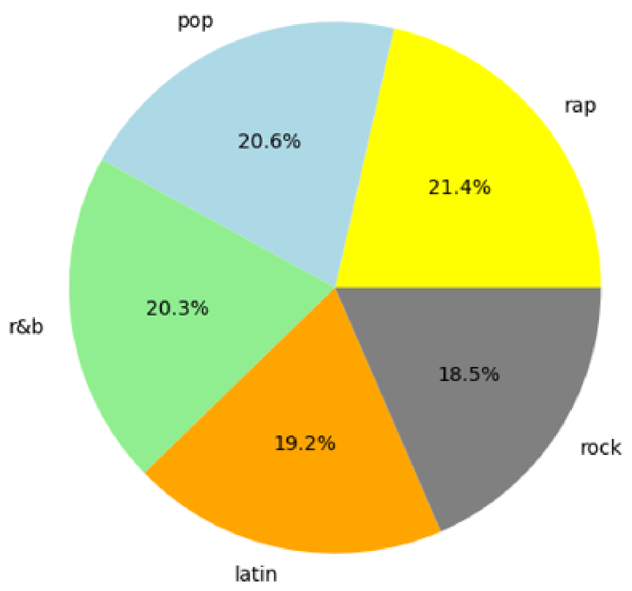


Fig. 1: Distribution of Genres

2.2.2 Distribution of Audio Features

The distribution of various audio features such as danceability, energy, loudness, speechiness, acousticness, instrumentalness, liveness, valence, and tempo is analyzed.

	track_popularity	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo	duration_ms
mean	44.28	0.66	0.67	5.38	-7.02	0.58	0.11	0.20	0.05	0.19	0.54	119.73	226668.72
std	24.99	0.15	0.18	3.62	3.05	0.49	0.11	0.23	0.18	0.15	0.23	28.71	57745.82
min	0.00	0.08	0.00	0.00	-46.45	0.00	0.02	0.00	0.00	0.01	0.00	35.48	29493.00
25%	27.00	0.56	0.55	2.00	-8.51	0.00	0.04	0.02	0.00	0.09	0.36	96.54	190191.25
50%	48.00	0.68	0.69	6.00	-6.46	1.00	0.06	0.10	0.00	0.12	0.54	117.94	218933.00
75%	64.00	0.77	0.81	9.00	-4.89	1.00	0.14	0.30	0.00	0.24	0.71	137.31	255392.50
max	100.00	0.98	1.00	11.00	1.27	1.00	0.92	0.99	0.99	0.99	0.99	239.44	517810.00

Table 2: Descriptive Statistics of the Dataset

2.2.3 Multivariate Analysis

A correlation matrix is created to understand the relationships between different numerical features. Overall, we can see that there is no big correlation between our variables.

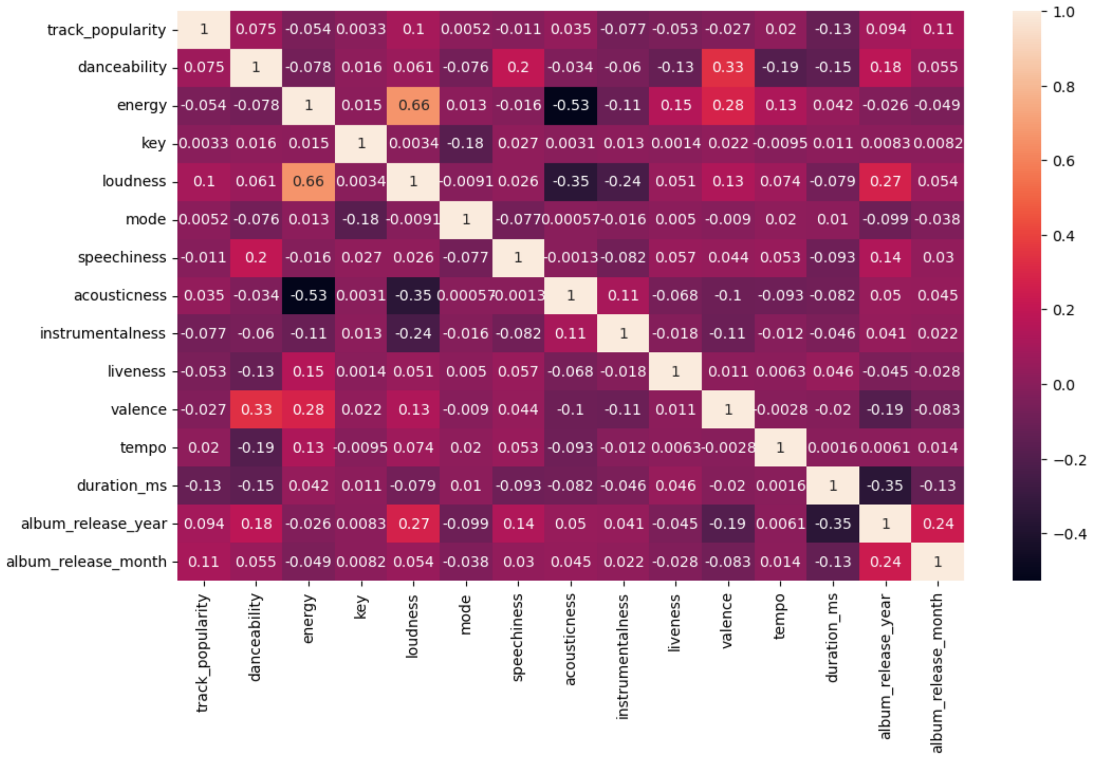


Fig. 2: Correlation Matrix

2.3 Data Preprocessing

The data preprocessing phase involves handling both textual and numerical data, which are processed separately before being combined for model training. The textual and numerical data require different techniques to ensure proper representation and scaling of input features, enabling effective use in the classification model.

2.3.1 Textual Data Preprocessing

The textual data is processed to transform unstructured text into meaningful numerical embeddings. The dataset is first loaded using the pandas library with UTF-8 encoding, ensuring that text data is read correctly. The columns `track_album_name` and `track_name`, which contain the textual information, are combined into a new column named `combined_text`. We used these two textual features because they appear to have the biggest potential to predict the music genre. This new column represents a single sentence for each track, encapsulating its descriptive details.

To prepare this textual information for the model, the `combined_text` column is encoded using the Sentence-BERT model (`all-MiniLM-L6-v2`), which converts the text into high-dimensional numerical embeddings (384-dimension). These embeddings are then standardized using `StandardScaler`, ensuring that the data has zero mean and unit variance.

2.3.2 Numerical Data Preprocessing

The numerical features in the dataset are preprocessed to normalize and scale the values for consistency across features. The dataset includes various numerical attributes, such as track characteristics and release dates, that require different preprocessing techniques. First, the dataset is loaded using pandas, with UTF-8 encoding, to handle any special characters in the data. The `track_album_release_date` column is converted to a datetime object, and the year is extracted from it to represent the release date as a single numerical feature.

Next, the preprocessing focuses on selecting numerical columns while excluding the `mode` and `date` columns. The selected features are normalized using `StandardScaler`, ensuring they are scaled to have zero mean and unit variance. The `mode` column, which is categorical, is not normalized but is retained in its original form for concatenation with the normalized numerical features. Finally, the processed numerical features are combined with the `mode` column to create the final feature set.

3 Proposed Architecture and Learning Protocol

3.1 Baseline Models

Two baseline models were used: XGBoost and Random Forest. Hyperparameter grids were defined for each model to perform grid search cross-validation. The hyperparameters included learning rate, number of estimators, maximum depth, subsample, and column sample by tree for XGBoost, and number of estimators, maximum depth, minimum samples split, minimum samples leaf, and bootstrap for Random Forest. The models were trained and validated using 5-fold cross-validation.

3.2 Neural Network Architecture

The neural network in this study was implemented using PyTorch, allowing flexible architecture and incorporating practices to improve learning stability and generalization. The constructor takes parameters for input size (`input_size`), hidden layer sizes (`hidden_layers`), output size (`output_size`), activation function, batch normalization, and dropout rate.

Hidden layers consist of:

- **Linear transformation** (`nn.Linear`).
- **Batch normalization** (`nn.BatchNorm1d`) for faster convergence.
- **Activation function** (`nn.ReLU` by default).
- **Dropout** (`nn.Dropout`) to prevent overfitting.

The output layer is a single `nn.Linear` layer.

This modular design ensures simplicity and flexibility for various regression and classification tasks.

3.3 Training Process

The training process is implemented in the `PyTorchTrainer` class, which handles key functionalities like forward/backward passes, gradient updates, early stopping, and model checkpointing. It takes inputs such as the model (`model`), loss function (`criterion`), optimizer (`optimizer`), training data (`X`, `y`), and optional parameters like batch size and gradient clipping.

Key features include:

- **Batch Processing:** Mini-batches are created using `DataLoader` for efficient gradient computation.
- **Gradient Clipping:** Prevents exploding gradients by normalizing them to a maximum value.
- **Early Stopping:** Stops training if validation loss doesn't improve within a specified number of epochs.
- **Pre-Trained Weights:** Loads weights from pre-trained models, excluding incompatible layers.
- **Validation:** Computes validation loss and accuracy during training to monitor generalization.

The `train` method processes training in epochs, performing forward passes, loss computation, backpropagation, and model updates. The `evaluate` method tests the model on unseen data, while `save_model` saves the model's weights.

This efficient pipeline addresses common challenges like overfitting, gradient instability, and resource utilization.

3.4 Hyperparameter Tuning

A grid search was conducted to explore various configurations for both network architecture and training. The parameters tested included:

- **Hidden Layers:** Configurations like $[64, 32]$, $[16, 16, 16]$, and $[128, 16]$ were evaluated.
- **Batch Normalization:** Tested with `True` and `False`.
- **Dropout Rate:** Values of 0.3 and 0.7 were considered.
- **Learning Rate:** Rates of 10^{-3} and 10^{-4} were tested.
- **Batch Size:** Sizes of 16 and 32 were evaluated.
- **Epochs:** Training durations of 20 and 50 epochs were explored.
- **Optimizer:** Comparison of Adam and SGD.
- **Gradient Clipping Value:** Tested at 0.5 and 1.5.

4 Results

This section presents the experimental outcomes of the study. The performance of baseline models, the simple neural network, and the hyperparameter-tuned neural network are evaluated and compared. Additionally, results from textual and combined data inputs, as well as transfer learning, are analyzed.

4.1 Baseline Models

The performance of the baseline models, including Random Forest (RF) and XGBoost, was evaluated using the dataset. The results provide insights into the effectiveness of these traditional models in comparison to the neural network. Since XGBoost demonstrated slightly better results than Random Forest, we only present the results for the XGBoost model.

After fine-tuning the XGBoost model, the best achieved accuracy was 0.572. The optimal hyperparameters after fine-tuning were as follows:

- **colsample_bytree**: 0.9
- **learning_rate**: 0.1
- **max_depth**: 10
- **n_estimators**: 300
- **subsample**: 0.9

4.2 Performance of the Simple Neural Network

The simple neural network's performance was compared with the baseline models using accuracy as the primary evaluation metric. This comparison demonstrates how the neural network performs relative to more traditional machine learning models.

Results and Performance Metrics

The evaluation of the trained neural network yielded the following results:

- **Accuracy:** 0.546
- **Precision:** 0.538
- **Recall:** 0.546
- **F1-Score:** 0.531

Table 3: Model Architecture and Training Configuration

Category	**Details**
	Model Architecture
Input size	14
Hidden layers	[64, 32, 16]
Output size	5
Batch normalization	Enabled
Dropout rate	0.5
	Training Configuration
Loss function	CrossEntropyLoss
Optimizer	Adam
Learning rate	0.001
L2 regularization (weight decay)	1×10^{-4}
Batch size	32
Number of epochs	100
Gradient clipping	Enabled (clip value: 1.0)

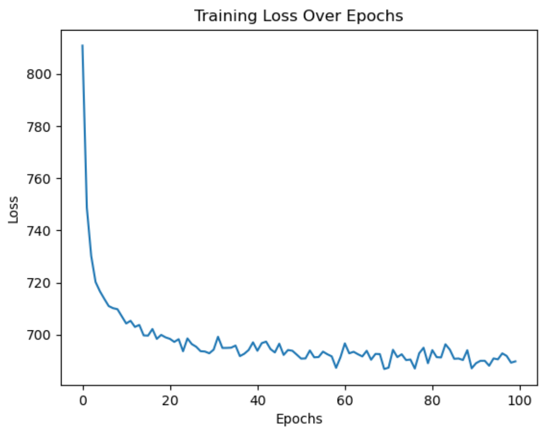


Fig. 3: Learning Curve for Simple Neural Network

In summary, the model shows moderate performance with room for improvement. The learning curve indicates that the model learns quickly but stabilizes, suggesting that further training with the current setup may not yield significant gains. Adjusting hyperparameters, modifying the model architecture, and implementing techniques like early stopping could help enhance the model's performance.

4.3 Results of Hyperparameter Tuning

Hyperparameter tuning was conducted to optimize the neural network configuration. The dataset was split into 80% training and 20% validation. A grid search approach was used to explore the following hyperparameter combinations, resulting in 192 configurations. To speed up computations, the number of epochs was fixed at 20, which provided a balance between computational efficiency and model performance.

Table 4: Hyperparameter Grid and Best Configuration

Hyperparameter	**Grid Values**	**Best Value**
Hidden Layers	[64, 32], [16, 16, 16], [128, 16]	[128, 16]
Batch Normalization	True, False	False
Dropout Rate	0.3, 0.7	0.3
Learning Rate	10^{-3}, 10^{-4}	10^{-3}
Batch Size	16, 32	32
Epochs	20	20
Optimizer	Adam, SGD	Adam
Gradient Clipping Value	0.5, 1.5	1.5
Accuracy	-	**0.574662**

Why This Configuration Yielded the Highest Accuracy

The best configuration, featuring two hidden layers with [128, 16] neurons, effectively captured feature hierarchies while promoting generalization through dimensionality reduction. Disabling batch normalization indicated the model's ability to train stably without activation normalization, while a moderate dropout rate of 0.3 mitigated overfitting without hindering learning. A learning rate of 0.001, combined with the Adam optimizer, ensured efficient and adaptive weight updates, while a batch size of 32 balanced gradient estimation and computational efficiency. Gradient clipping at 1.5 further stabilized optimization by preventing gradient explosions. This synergy between depth, regularization, and training dynamics struck an ideal balance between model complexity, stability, and performance, leading to the highest validation accuracy.

4.4 Neural Network Results with Textual Data (and Numerical Data)

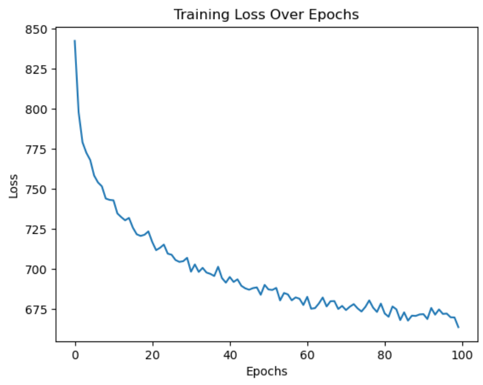
The neural network's performance was evaluated under two configurations: one where the model was trained solely on textual data, and another where both textual and numerical data were used. The same model architecture and training hyperparameters were applied in both configurations. These results highlight the network's ability to extract meaningful patterns from both types of data.

The evaluation scores for both configurations are summarized in Table 5. The table shows the loss, accuracy, precision, recall, and F1-score for each configuration.

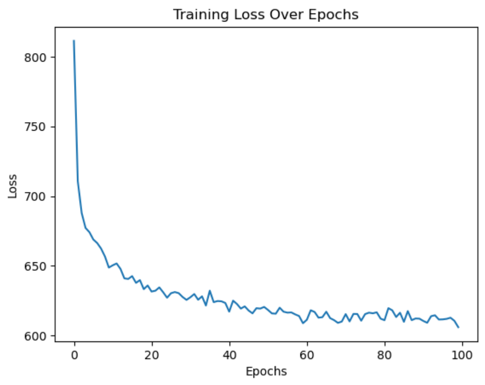
Table 5: Model Evaluation Scores (Textual and Textual + Numerical Data)

Metric	**Textual Data**	**Textual + Numerical Data**
Accuracy	0.4536	0.6174
Precision	0.4568	0.6209
Recall	0.4536	0.6174
F1-Score	0.4430	0.6116

Figures 4a and 4b show the learning curves for both configurations. Figure 4a illustrates the evolution of the loss over 100 epochs when the model was trained solely on textual data, while Figure 4b shows the learning curve for the configuration where both textual and numerical data were used.



(a) Learning Curve for Neural Network (Textual Data)



(b) Learning Curve for Neural Network (Textual + Numerical Data)

Fig. 4: Learning Curves for Neural Networks with Different Data Types

The model trained with Textual + Numerical Data consistently outperforms the model trained with only numerical data (done before). It also outperforms model trained with only Textual Data across all evaluated metrics (Accuracy, Precision, Recall, and F1-Score). This suggests that incorporating textual data significantly enhances the model's performance, making it more accurate, precise, and better at identifying relevant instances.However, as we have seen, only textual data is not enough to get better performance scores, we still need the numerical data.

4.5 Transfer Learning Results

To enhance model performance, transfer learning was applied using pre-trained parameters from numerical and textual data. This approach adapted the model to a new dataset with an additional class, increasing the classification task from 5 to 6 classes. The evaluation scores after transfer learning were as follows: accuracy of 0.572, precision of 0.571, recall of 0.572, and F1-score of 0.561. These results are good but didn't improve in comparison with previous model and classification.

However, a key observation is that transfer learning significantly speeds up convergence compared to previous configurations. Starting with pre-trained parameters near the global minimum reduces the number of training steps required, leading to faster convergence and more efficient learning.

Figure 5 illustrates the loss over epochs during training with transfer learning, highlighting the faster convergence compared to training from scratch.

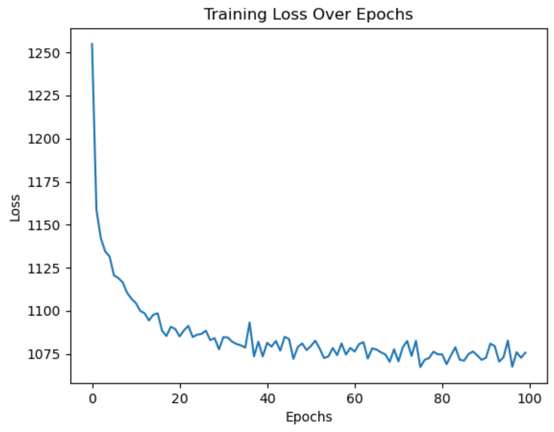


Fig. 5: Learning Curve for Neural Network with Transfer Learning (Textual + Numerical Data)

5 Conclusion

In this project, we developed a neural network-based music genre classification tool, leveraging both numerical and textual data to enhance classification performance. XGBoost and Random Forest models served as benchmarks for evaluating the performance of more complex neural network architectures.

The neural network model was designed and optimized through a hyperparameter tuning process over 192 different configurations, including batch size, number of layers, learning rates, and the use of heuristics like batch normalization, dropout, and gradient clipping. The best-performing model achieved an accuracy of 0.574, outperforming slightly the XGBoost model's accuracy 0.572.

Further experimentation involved integrating textual data from song titles using sentence embeddings, which markedly enhanced the model's performance. The combined model, using both numerical and textual features, showed better performance scores compared to models trained solely on numerical or textual data. Specifically, the combined model achieved an accuracy score of 0.617, surpassing the 0.574 accuracy obtained with numerical data alone and the 0.453 accuracy achieved with textual data alone.

The model was extended to a new dataset containing an additional genre, using transfer learning. While the transfer learning approach did not significantly improve the accuracy=0.572, it accelerated the convergence speed, highlighting the efficiency of transfer learning at reaching faster convergence.

Overall, this study underscores the potential of neural networks in music genre classification and the benefits of integrating textual data and transfer learning techniques to enhance model performance and efficiency. Future work could explore more advanced neural network architectures, additional textual features, and further optimization techniques to improve classification accuracy and robustness.

References

[1] Gabriel, J. (2023). *Spotify Songs Music Genre Predictor Part I.* Kaggle. Available at: https://www.kaggle.com/code/jgabrielsb/spotify-songs-music-genre-predictor-part-i/notebook.

[2] Chen, T., & Guestrin, C. (2016). *XGBoost: A Scalable Tree Boosting System.* Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 785-794.

[3] Breiman, L. (2001). *Random Forests.* Machine Learning, 45(1), 5-32.

[4] Reimers, N., & Gurevych, I. (2019). *Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks.* arXiv preprint arXiv:1908.10084.

[5] Yosinski, J., Clune, J., Bengio, Y., & Lipson, H. (2014). *How Transferable are Features in Deep Neural Networks?* Advances in Neural Information Processing Systems, 27.