Final Project: Music Genre and Composer Classification Using Deep Learning

Fernando Calderon

University of San Diego

AAI511 - Neural Networks and Deep Learning

Professor: Kahila Mokhtari

August 4th, 2024

Introduction

The objective is to create a model that allows novice musicians, listeners, and music enthusiasts to accurately identify the composer of a musical piece. The dataset employed, known as the "MIDI Classic Music" dataset from Kaggle, comprises 3,929 MIDI files featuring classical works by 175 composers, including prominent names like Bach, Beethoven, Mozart, and others. This rich dataset provides a diverse range of classical music compositions, ideal for training deep learning models. Calderon's approach involves meticulous data pre-processing, feature extraction, model building, training, and evaluation, aiming to harness the full potential of deep learning in music classification.

Methodology

This project employs machine learning techniques to predict the composer of a musical piece from MIDI files. The selected composers for this study include Bach, Beethoven, Chopin, and Mozart. The methodological approach encompasses data preprocessing, feature extraction, model architecture design, training, and validation phases, which are systematically executed to ensure model reproducibility and effectiveness.

Data Pre-processing

The initial phase of the project involved extensive data preprocessing to prepare the MIDI files for effective learning. Given the nature of MIDI files, which encode music as digital information, the preprocessing step was crucial to convert these files into a format suitable for machine learning models. Each MIDI file was parsed to extract musical features such as pitch, velocity, and duration of each note. This conversion was necessary to transform raw MIDI data into structured sequences that represent musical compositions.

To standardize the data, all sequences were padded or truncated to ensure uniformity in sequence length across the dataset. This uniformity is vital for training neural networks, which require consistent input sizes. Additionally, the data was normalized to scale the numerical values used in the model, improving the convergence speed during training and preventing issues related to varying scales of input features.

Feature Extraction Techniques

Feature extraction was a critical step in capturing the essence of musical compositions. The primary features extracted from the MIDI files included melodic intervals, harmonic intervals, and temporal features like note duration and time between notes. Melodic intervals, representing the pitch difference between consecutive notes, and harmonic intervals, capturing the vertical aspect of music when notes are played simultaneously, were both quantified. Further, a Fourier Transform was applied to transform these sequences into the frequency domain, enabling the model to understand and capture the dominant frequencies, which are indicative of the musical style and can be linked to specific composers. This transformation was particularly useful for identifying unique signatures in the compositions of different composers.

Model Architecture

The model architecture consisted of a combination of LSTM and CNN layers, leveraging the strengths of both to process sequential and spatial patterns in data. The LSTM layers were designed to capture the long-term dependencies in musical sequences, which is essential for understanding the structure of compositions. In contrast, the CNN layers were utilized to detect patterns across these sequences, providing a robust mechanism for feature detection in the context of music analysis. The network included dropout layers to prevent overfitting, and batch normalization layers to ensure that the model generalizes well to new, unseen data. The final layer of the model was a softmax layer, which classified the output into one of the predefined classes, each corresponding to a composer in the dataset.

Training Process

The training process involved several steps to optimize the model's performance. The model was trained using a categorical cross-entropy loss function, which is standard for multi-class classification problems. An Adam optimizer was chosen for its efficiency in handling sparse gradients and its adaptive learning rate capabilities, which are suitable for this type of data. The model was trained in batches to optimize the computational resources and to ensure that the model updates its weights effectively after each batch. Early stopping was implemented to halt the training if the validation loss did not improve for a set number of epochs, preventing unnecessary computations and overfitting. During training, the model's performance was evaluated using a hold-out validation set, which helped in monitoring the model’s ability to generalize beyond the training data. The accuracy and loss metrics were logged for each epoch to track progress and make necessary adjustments to the training regime.

Conclusion

The initial results, derived from implementing LSTM and CNN models, have shown promising accuracy in classifying composers from musical scores. This success could transform how individuals interact with music, providing an educational tool for understanding and appreciating classical music compositions.

For future improvements, the project could explore the integration of additional types of neural network architectures, such as Transformer models, which have shown great success in handling sequential data. Moreover, expanding the dataset to include more contemporary and less represented composers could enhance the model's robustness and applicability to a broader spectrum of musical pieces. Another potential extension could involve the development of a user-friendly application that allows real-time composer prediction, thereby making classical music more accessible to the general public.

References

- Humphrey, E., Bello, J. P., & LeCun, Y. (2013). Moving beyond feature design: Deep architectures and automatic feature learning in music informatics. Proceedings of the 13th International Society for Music Information Retrieval Conference (ISMIR), 403-408.
- Krogh, A., & Hertz, J. A. (1992). A simple weight decay can improve generalization. Advances in Neural Information Processing Systems, 4, 950-957.
- Tzanetakis, G., & Cook, P. (2002). Musical genre classification of audio signals. IEEE Transactions on Speech and Audio Processing, 10(5), 293-302.

Python Libraries | Reference List

- NumPy (van der Walt, S., Colbert, S. C., & Varoquaux, G., 2011). NumPy is a fundamental package for scientific computing in Python, providing support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays. NumPy.
- pandas (McKinney, W., 2010). pandas is an open-source data analysis and manipulation tool built on top of the Python programming language, providing data structures and operations for manipulating numerical tables and time series. pandas.
- pretty_midi (Raffel, C., 2014). pretty_midi is a library for analyzing and modifying MIDI files. A useful tool for extracting information from complex MIDI files and manipulating musical data. pretty_midi.

- music21 (Cuthbert, M. S., & Ariza, C., 2010). music21 is a toolkit for computer-aided musicology. It allows for easy manipulation and analysis of musical scores, especially useful for extracting features from music files. music21.
- Matplotlib (Hunter, J. D., 2007). Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. It's used extensively for plotting graphs and charts. Matplotlib.
- Seaborn (Waskom, M., 2021). Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics. Seaborn.
- Keras (Chollet, F., 2015). Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation. Keras.
- Scikit-learn (Pedregosa et al., 2011). Scikit-learn is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms. Scikit-learn.
- TQDM (da Costa-Luis, C., 2021). TQDM is a fast, extensible progress bar for Python and CLI that allows you to see the progress of your loops. TQDM.
- Adam Optimizer. Adam is an optimization algorithm that can be used instead of the classical stochastic gradient descent procedure to update network weights iterative based in training data (Kingma, D. P., & Ba, J., 2014). Adam Optimizer.