Music Genre Classification using Convolutional Neural Networks

By Anish Kar

Introduction

Music is a universal language, yet its diverse forms and genres create a rich tapestry of sound. The goal of this project was to bridge the gap between human auditory perception and machine understanding by creating a system capable of automatically classifying the genre of a piece of music. By leveraging the power of deep learning, specifically Convolutional Neural Networks (CNNs), this project explores how digital signal processing and artificial intelligence can be combined to categorize audio files with impressive accuracy. The endeavor serves not only as a technical exercise but also as a fascinating look into how machines can learn to interpret the subtle patterns and textures that define musical styles.

Abstract

This report details the development of a music genre classification system using a Convolutional Neural Network (CNN). The objective was to train a model capable of accurately predicting the genre of an audio file from a set of ten distinct categories: blues, classical, country, disco, hip-hop, jazz, metal, pop, reggae, and rock. The widely-used GTZAN dataset, containing 1,000 audio tracks, served as the foundation for training and validation. The core methodology involved extracting Mel-Frequency Cepstral Coefficients (MFCCs) from audio signals to create a visual, feature-rich representation suitable for a CNN. The model was built and trained using Python with the Keras/TensorFlow library. The resulting system successfully classifies new audio files, demonstrating the efficacy of CNNs for audio pattern recognition tasks.

Tools Used

- **Programming Language:** Python
- **Core Libraries:**
    - **TensorFlow & Keras:** For building and training the deep learning model.
    - **Librosa:** For advanced audio processing and feature extraction (MFCCs).
    - **Scikit-learn:** For splitting the dataset into training and testing sets.
    - **NumPy:** For numerical operations and data manipulation.
    - **Matplotlib:** For visualizing the training process and audio spectrograms.
- **Environment:** Google Colaboratory (Colab) with GPU acceleration.
- **Dataset:** GTZAN Genre Collection Dataset.

Steps Involved in Building the Project

The project was executed in a sequential pipeline, from data acquisition to model deployment for prediction.

Step 1: Data Acquisition and Preparation

The first step involved setting up the environment and acquiring the GTZAN dataset. The dataset, stored as a .zip file in Google Drive, was mounted and unzipped into the Colab environment. This provided access to the genres_original folder containing 1,000 .wav files, organized into 10 subfolders, one for each genre.

Step 2: Feature Extraction (MFCC)

save_mfcc function: This function automates the feature extraction process.

1. It iterates through each genre subfolder in the genres_original directory.
2. For each 30-second .wav file, it uses librosa.load() to load the audio signal.
3. To augment the dataset, each track is divided into 10 non-overlapping 3-second segments.
4. librosa.feature.mfcc() is called on each segment to compute a 2D array of MFCC features.
5. These MFCC arrays, along with their corresponding genre labels (e.g., 0 for blues, 1 for classical), are stored in a dictionary and saved as a single data.json file. This prevents the need for re-processing the audio files every time the script is run.

Step 3: Model Architecture (CNN)

build_model function: The model is a sequential stack of layers built with Keras:

- **Convolutional Layers (Conv2D):** Three Conv2D layers act as feature detectors, learning to recognize patterns like specific harmonic structures or rhythmic motifs within the MFCCs.
- **Pooling Layers (MaxPooling2D):** Each convolutional layer is followed by a max-pooling layer, which downsamples the feature maps. This makes the model more robust to variations in the feature's position and reduces computational complexity.
- **Batch Normalization:** Used after each pooling layer to stabilize and accelerate the training process.
- **Flatten Layer:** Converts the final 2D feature maps into a single 1D vector.
- **Dense Layers:** A fully connected Dense layer acts as a classifier on the flattened features, followed by a Dropout layer to prevent overfitting.
- **Output Layer:** The final Dense layer has 10 neurons (one for each genre) with a softmax activation function, which outputs a probability distribution indicating the model's confidence for each genre.

Step 4: Model Training and Evaluation

With the data prepared and the model built, the next step was to train the network.

1. The data from data.json was loaded and split into an 80% training set and a 20% validation set.
2. The model was compiled using the Adam optimizer and sparse_categorical_crossentropy loss function, suitable for multi-class classification with integer labels.
3. The model.fit() function trained the network for 50 epochs, continuously adjusting its internal weights to minimize the loss on the training data.
4. The model's performance was monitored on the validation set at the end of each epoch. The final accuracy and loss were plotted to visualize the learning process.

Step 5: Prediction on New Audio

predict_genre function: This function was designed to handle audio files of any length.

1. It loads the user-uploaded audio file.
2. It processes the audio in 3-second segments, just like the training data. Crucially, it pads or truncates each segment to ensure it has the exact length required by the model, preventing shape-related errors.
3. MFCCs are extracted from each segment.
4. The model predicts the genre probabilities for each segment.
5. The predictions are averaged across all segments to produce a single, robust prediction for the entire song, which is then presented to the user with a confidence score.

Conclusion

This project successfully demonstrates a complete pipeline for building an effective music genre classifier using deep learning. By converting audio signals into a visual MFCC representation, we were able to leverage the pattern-recognition power of a CNN to achieve reliable classification. The final model performs well on the test data and can classify new audio files in real-time. Potential future improvements could include experimenting with more complex architectures (like Recurrent Neural Networks), incorporating a wider range of audio features, and expanding the dataset to include more genres and a greater diversity of tracks.