IML - S. Juneja : CS-1390/ PHY-1390-2
Final Project Report - Divyam Chaudhary (solo)

Topic: Music Genre Classification + Content-Based Recommendation System

GitHub Repo: https://github.com/xanfow/dc_cs-1390-2-iml.git

Abstract: Traditional music recommendation systems rely heavily on collaborative filtering, which suggests music based on the listening behavior of other users with similar tastes. While this method can be effective, it often fails to consider the actual audio characteristics of songs, potentially limiting the diversity and accuracy of the recommendations. With this project, I aim to develop a machine learning model (a CNN in particular) to classify music based on genre and build a recommendation system that suggests songs to users based on similarities in the songs they listen to. I aim to improve recommendation accuracy by focusing on content-based similarities such as rhythm, mood, harmony, associated emotions, genres, etc. These features are aptly captured in the MFCCs (mel-frequency cepstral coefficients) of each song's audio file, which can be extracted to form the basis for both the genre classification and the content-based song recommendations.

Dataset:

1. Classifier model training samples borrowed from GTZAN Dataset @ https://www.kaggle.com/datasets/andradaolteanu/gtzan-dataset-music-genre-classification
2. Recommendation System testing samples borrowed from Free Music Archive @ https://freemusicarchive.org/home

Project Structure:

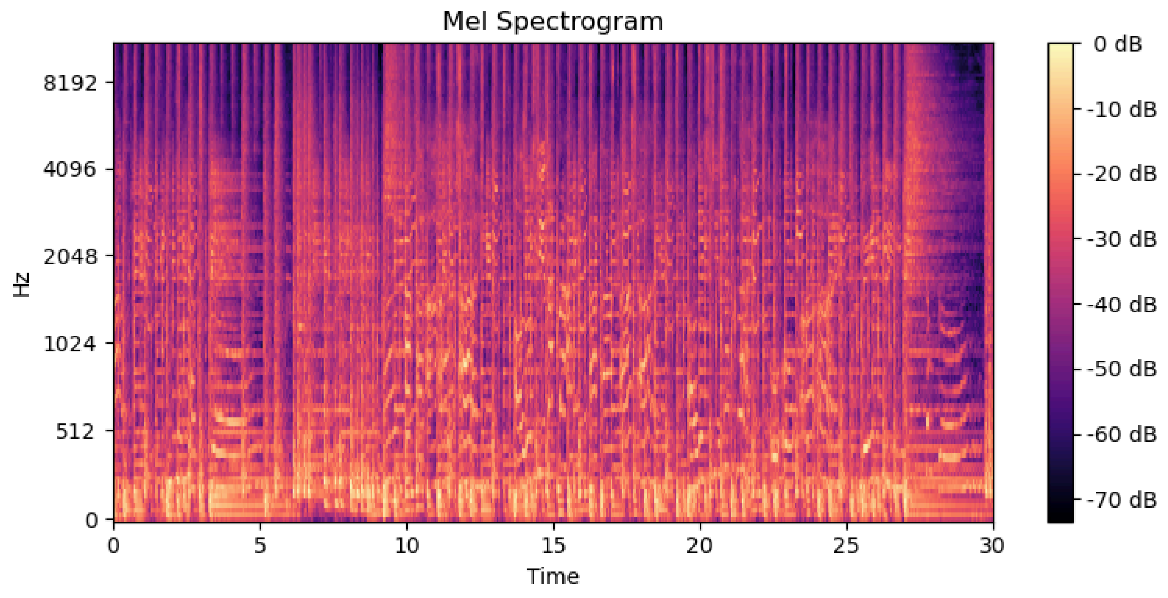
1. Music Genre Classification: I first trained a convolutional neural network to classify songs into different genres based on extracted audio features.
2. Content-Based Recommendation System: Then, I built a recommendation system that suggests songs based on their audio feature similarities. The app first predicts the genres of the inputted songs using the pre-trained model in (1.), then recommends tracks from the GTZAN dataset that closely match these predicted genres.

Methodology:

1. I begin by importing all the relevant utilities from the following dependencies:
    a. NumPy (for data processing and reshaping)
    b. Pandas (for data processing and reshaping)
    c. Matplotlib (for visualization)
    d. Seaborn (for confusion matrix visualization)
    e. Librosa (for audio feature extraction)
    f. Scikit-Learn (for train-test split API and confusion matrix construction)
    g. TensorFlow (for one hot encodings and implementing the CNN classifier)
2. Next, I define a *hyper* object to store all of the key hyper parameters for the project that can be quickly tweaked in one place for rapid experimentation.
3. MFCCs, Mel Spectrograms & Feature Engineering: A Mel spectrogram is a visual depiction of sound that illustrates how frequencies evolve over time, mapped to the Mel scale to align with human pitch perception. It finds extensive application in areas like speech recognition, music analysis, and audio synthesis. The x-axis represents time, the y-axis corresponds to Mel frequencies (a perceptually scaled version of frequency), and color intensity indicates power.

Mel Frequency Cepstral Coefficients (MFCCs) are compact audio features extracted from the Mel spectrogram. They capture the spectral characteristics of sound in a way that closely reflects human auditory perception, which makes them pivotal particularly for tasks such as speech recognition and sound identification. Below is an example of a Mel spectrogram:

Shape: (128, 1293)



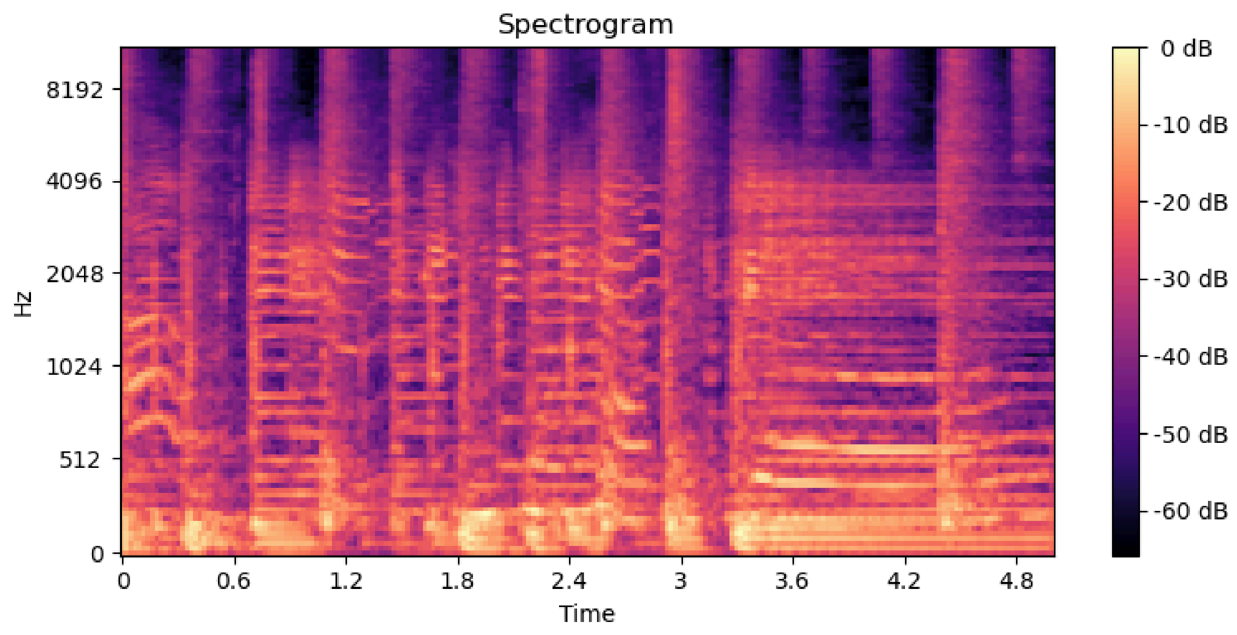
An audio signal is essentially a sequence of air pressure samples (amplitude) that vary over time. By applying a Fast Fourier Transform (FFT), this signal can be transformed into the frequency domain, enabling visualization of its frequency components. Both the frequency and amplitude can then be scaled logarithmically, resulting in a Mel spectrogram.

FFT reveals patterns that differentiate sounds as perceived by the human ear. In music, for example, the twelve semitones serve as the fundamental building blocks, distinguished by differences in frequency rather than amplitude. Amplitude variations affect loudness perception but do not alter harmonic qualities. FFT also simplifies data preprocessing by isolating and reducing unwanted noise frequencies. This isolation benefit is why compression algorithms and codecs rely on frequency representations rather than raw amplitude waveforms.

The Mel scale plays a crucial role in this process by providing a logarithmic frequency scaling. It spaces out lower frequencies while compressing higher ones, aligning with human auditory sensitivity. Humans can discern subtle differences in lower frequencies more effectively than in higher ones, even when the frequency ratio remains the same.

In the GTZAN dataset, each audio sample spans 30 seconds, which can be too dense to analyze directly. To address this, I divide each sample into overlapping 5-second segments with a 2-second overlap. This segmentation maintains continuity between chunks while expanding the dataset virtually—from 1,000 entries to 9,990 (10 chunks per song, excluding one corrupted file). Using the Librosa Python library, I directly extract Mel spectrograms without manually performing FFT and logarithmic scaling.

Shape: (128, 216)



The GTZAN dataset provides both audio files and pre-extracted MFCCs, either for the full 30-second tracks or for each track split into ten 3-second segments. However, these pre-split segments lack continuity between chunks, which is why I manually calculate the chunks and extract the time-series data from the audio files myself.

Once the chunks are extracted as Mel spectrograms, I resize them into square images. This resizing introduces minimal pattern distortion while ensuring uniformity across the data tensor, facilitating efficient convolution operations. This approach is the rationale behind the *shape_length* hyperparameter.

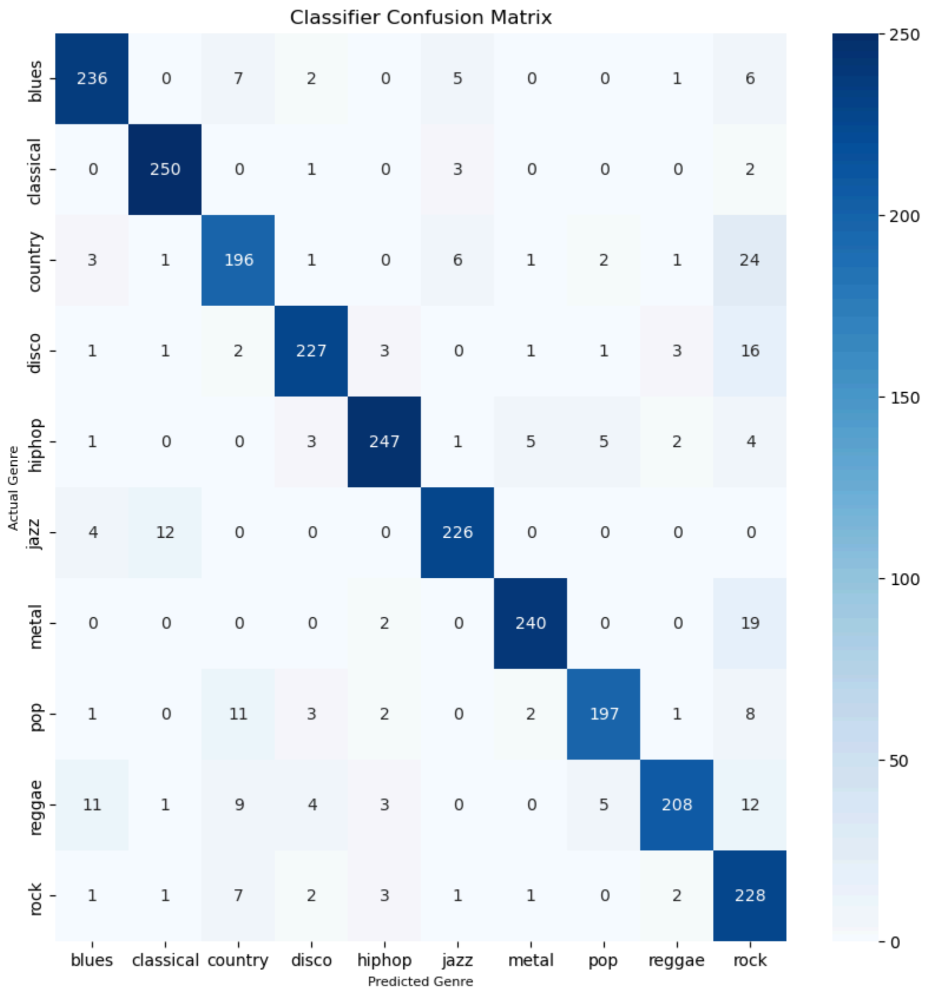
Interestingly, this transformation gives us an image representation of the audio. Consequently, the problem shifts from audio classification to image classification from this points on.

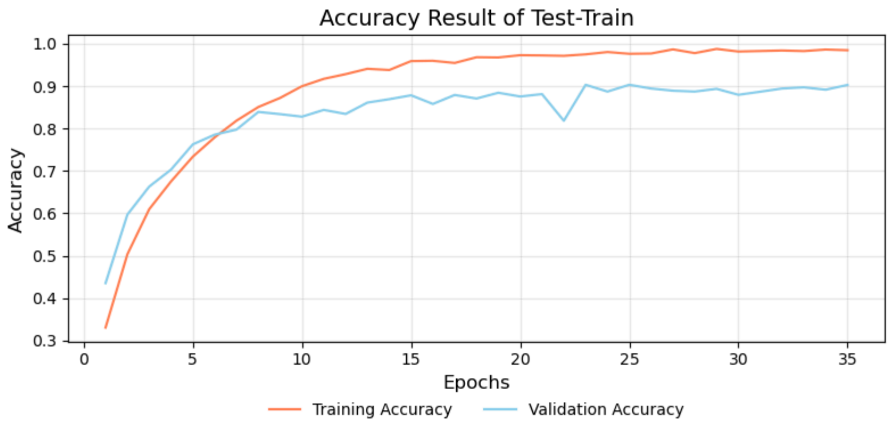
4. Next, I apply one-hot encoding to the dataset labels, enhancing performance and accuracy. I then use the standard scikit-learn utility to split the dataset into training and testing sets, with a 75%-25% split. To mitigate overfitting, I slightly enlarge the test set, ensuring a better evaluation of model generalization.

   For the convolutional neural network (CNN) model, I build it sequentially. The architecture consists of five sets of layers, each comprising two convolutional layers to extract key patterns, followed by a max pooling layer to reduce overfitting and improve computational efficiency. The number of filters increases with each set as early layers with fewer filters focus on simple patterns like frequency bands or time transitions and later ones try to capture more complex and diverse patterns. Dropout layers are added after the first three sets and once again after the final (fifth) set, just before flattening the data. The flattened data is then passed through a fully connected artificial neural network (ANN) layer that employs yet another dropout before propagating to the final output neuron. It is paramount to pay extra attention to overfitting here as many times songs tend to have similar features, tones, or melodies but are largely from two different genres.

   ReLU activation functions are employed in all nodes except for the output layer, which uses a softmax activation function optimized with categorical cross-entropy, appropriate for this multi-class classification problem. Altogether, the model comprises 14,193,162 trainable parameters, designed to effectively capture the nonlinear and intricate patterns of the Mel scale.

5. I then trained the model at a very conservative learning rate (while trying to reproduce my results, it may be required to alter batch sizes a bit to prevent TensorFlow OOM crashes). Next, I saved the trained weights along with the training metrics, allowing the model state to be reloaded efficiently for future use. The training process resulted in impressive accuracy and loss metrics, accompanied by a well-balanced confusion matrix with only a small number of misclassifications.





6. For the recommender system, I curated audio files from various genres within the Free Music Archive dataset. These files were first divided into smaller chunks, which were then processed through my trained model. The genre with the highest prediction frequency across the chunks was assigned as the final prediction for each song. Based on these final predictions, I provided recommendations by selecting four random songs from the GTZAN dataset corresponding to the predicted genres.

Limitations:

1. Limited number of audio samples with short 30-second durations, insufficient to capture unique genre features, especially for complex or evolving genres.
2. Outdated and distorted audio quality affects applicability on modern tracks.
3. Issues with mislabeled, corrupted, and repeated tracks introduce redundancy and noise.
4. Overgeneralized genre categories fail to capture subtle distinctions, niche subgenres, or genre-blending common in modern music.

Failure:

I was eager to compare a content-based recommendation system with a collaborative filtering approach. However, this required an additional layer of attributes in the data—an element that proved difficult to obtain. Most available datasets were either audio feature-focused or user preference-focused but rarely encompassed both, making it particularly challenging to find a suitable dataset that could support the implementation of both models and enable a meaningful comparison of their results.

Verdict:

This project effectively implements a CNN-based music genre classification and content-based recommendation system using Mel spectrograms. Despite dataset limitations and the absence of a collaborative filtering comparison, it achieves its objectives and demonstrates strong technical proficiency, providing a solid foundation for future improvements in music recommendation systems.

* * *