CS 4771

Music Genre Classification

Final Project Proposal

Thomas Schmidt

October 31, 2025

Contents

1 Problem Statement and Motivation

Music streaming platforms hold ever growing music libraries that have created the need for automated tools to organize and recommend content. The tools described are favored by users because of the new song discovery offered. Labeling songs into genres is time consuming, subjective, and inconsistent across platforms. This project addresses the challenge of building an automatic system to classify music tracks into genres based on audio content.

The goal is to develop a classifier that accepts a short audio segment, at least half a minute, and predicts its genre from a predefined set (rock, pop, jazz, classical, blues, hip-hop. country, etc.). The applications are clear to Music recommendation engines that aim to curate user playlists, user song discovery, and content tagging. Using traditional audio signals and modern deep learning, the system will demonstrate how machine learning can extract patterns from raw acoustic data with meaning. Patterns that correlate with predetermined genre distinctions from existing classifications, seen from rhythm, timbre, harmony, and instrumentation.

This application is non-trivial from genre boundary ambiguities, such as indie rock versus alternative, variations within genres, and potential noisy or low-quality tracks. Success relies on model accuracy as well as feature resilience to real audio variability.

2 Data Specification

The dataset of choice for this project is the GTZAN Genre Collection. It is sourced from http://marsyas.info/downloads/datasets.html (public domain, available via Kaggle or direct download). A widely used benchmark in music information retrieval. Consisting of 1,000 audio files, with 10 genres having 100 tracks each; blues, classical, country, disco, hip-hop, jazz, metal, pop, reggae, and rock. The files are 30-second ẇav clips sampled at 22,050Hz in mono format. Features prudent to this project include raw waveform data for spectrogram generation, extracted acoustic features using librosa, chroma features, spectral contrast, tonal centroid, zero-crossing rate, spectral roll-off, RMS energy, and tempo in BPM (beats per minute).

The preprocessing plan takes several steps, from loading all the 30-second clips (verifying integrity). Splitting each track into non-overlapping 3-second segments (augmenting data), yielding approximately 10 segments per track (around 10,000 samples total). Form baseline models by mean, variance, and min/max over segments via computing the frame-level features. For CNN (Convolutional Neural Network) models, generating the log-mel spectrograms with 128 mel bands over the 3-second segments and a hop length of 512, normalizing to [0,1]. Preforming a stratified train (70%), validation (15%), and test splits (15%) whilst still preserving the genre balance. Then lastly storig the preprocessed features into HDF5 formaat for the CNN inputs and CSV (with NumPy) for baselines in the processed data folder.

For further validation, around 20 selected tracks from Free Music Archive will be used, and preprocessed identically to the primary dataset.

3 Method

3.1 Technical Approach

3.1.1 Phase 1: Feature-Based Classification (Baselines)

Statistical Summaries of audio features for each 3-second segment are extracted. Baseline models are trained with scikit-learn. Logistic Regression with L2 regularization and the liblinear solver included, as well as, k-Nearest Neighbors with $k = 5$ and cosine distance. Performance of the model will be assessed with 5-fold cross-validation, using hyperparameter tuning conducted via grid search.

3.1.2 Phase 2: End-to-End Deep Learning

Each 3-second clip is turned into a 128×130 log-mel spectrogram. I'll build a CNN in PyTorch with three 3×3 conv layers (32, 64, 128 filters), each followed by ReLU and 2×2 max pooling, but the last, which uses global average pooling. After that, a 128-unit dense layer with 0.5 dropout feeds into a 10-class softmax.

The model will be trained with cross-entropy loss, the Adam optimizer (learning rate of 0.001), a batch size of 32, and early stopping. Data summaries will include time stretching (±10%), pitch shifting (±2 semitones), and noise injection. Following the CNN architecture, with some tuning for real-world audio with noise injection.

3.1.3 Evaluation Metrics

The primary metric will be macro-averaged F1-score to account for class imbalance. Secondary metrics will include overall accuracy, per-class precision and recall, and visualization of the confusion matrix. Statistical significance between the best baseline and the CNN will be evaluated using McNemar's test.

3.2 Timeline (5 Weeks, Starting November 4, 2025)

Wk	Dates	Milestones	Hrs
1	Nov 4–8	Repo setup, GTZAN download, feature extraction, initial EDA	8
2	Nov 11–15	Baseline models, tuning, `baseline_models.ipynb`	9
3	Nov 18–22	Spectrogram pipeline, initial CNN, debug overfitting	12
4	Nov 25–29	Hyperparameter tuning, ablation (MFCCs vs. full), final eval.	10
5	Dec 2–6	Report, demo script, docs, README	8
		Total: ∼47 hrs	

Table 1: Project Timeline

All code will be versioned on GitHub with clear commit messages and branch protection enabled. https://github.com/TWi5td/cs4771-music-genre-classification

4 Justification and Novelty

Even though music genre classification has been around for years, this project is far from simple. It's a real test of machine learning in messy, real-world conditions. First, bridging two very different worlds: turning raw audio into usable features (signal processing) and then training models that actually process genre differences (pattern recognition).

Second, I'm working with just 1,000 labeled tracks. A tiny dataset by today's standards. So, splitting each 30-second clip into ten 3-second segments, adding noise and pitch shifts, and using early stopping to avoid overfitting. This mirrors what happens when you don't have millions of labeled songs lying around.

And finally, I'm building a complete system: from loading .wav files to spitting out a genre prediction. That means writing robust code for audio I/O, preprocessing, training, evaluation—and packaging it all in a way that someone else could run it tomorrow.

5 References

1. Tzanetakis, G., & Cook, P. (2002). Musical genre classification of audio signals. *IEEE Transactions on Speech and Audio Processing*, *10*(5), 293–302.

2. GTZAN Dataset: http://marsyas.info/downloads/datasets.html

3. McFee, B., et al. (2015). `librosa`: Audio and music signal analysis in Python. *Proceedings of the 14th Python in Science Conference*.

4. Choi, K., et al. (2017). Convolutional recurrent neural networks for music classification. *Proceedings of ICASSP 2017*.

5. Scikit-learn documentation: https://scikit-learn.org

6. PyTorch documentation: https://pytorch.org