629 Hardware Project

Yadi Eslami

Nicholas Azadnia and Parth Bhangla

05-09-2025

Song Classification Using MLP Implemented With PyTorch

This project explores how machine learning (ML) can be applied to classify songs based on short audio clips, aiming to simulate real-world conditions such as background noise and low-quality recordings. The goal was to develop a system that could accurately recognize songs from audio recorded through a microphone, even in imperfect scenarios—a significant challenge due to the noisy and unpredictable nature of real-life audio.

We aimed to combine digital signal processing (DSP) with machine learning to build a full pipeline. By taking on this project, we also gained insight into how popular apps like Shazam and Alexa recognize music. While these services are highly effective, it's worth noting that Shazam, for example, does not use machine learning. Instead, it relies on audio fingerprinting to match recorded clips to a database. This project takes a different approach, offering a proof of concept that ML can be applied to song recognition tasks.

Shazam identifies songs using audio fingerprinting. The app analyzes the spectrogram of a song to extract unique time-frequency pairs, similar to star constellations, and stores them as fingerprints. A user’s recorded clip is then matched against these fingerprints. This method, which does not rely on machine

learning, remains the industry standard, underscoring the novelty of exploring ML-based approaches.

Dataset Creation and Processing

No existing dataset fit the project's specific needs, so a custom dataset was built using a Spotify playlist of the top 60 songs in the United States. Songs were downloaded using SpotDownloader, resulting in clean MP3 files. To simulate noise, a Kaggle dataset of environmental sounds was used, featuring seven types of noise such as traffic, crowded places, and rain.

The dataset preparation was automated with a Python script, *build_dataset.py*, which included functions to clean song names, convert formats, split audio into 10-second clips, create ID mappings, rename clips, and overlay noise. The noisy dataset included over 9,500 clips, totaling around 17.87 GB. Clean and noisy datasets were combined and split 80/20 for training and testing.

Figure 1 shows the entire workflow of the project, from building the dataset to generating predictions. It starts with downloading the MP3 files and cleaning their names using a mapping. The files are then converted to WAV format and split into 10-second clips. Noise is overlaid using environmental sound clips to simulate real-world conditions, and the noisy dataset is then ready for training. The model is trained using PyTorch and saved, along with the scaler, to maintain consistency during prediction. For testing, a 10-second audio clip is recorded through a microphone, passed through the model, and the predicted song is output. This

diagram helps visualize how the different steps connect and how the model transitions from training to live prediction.

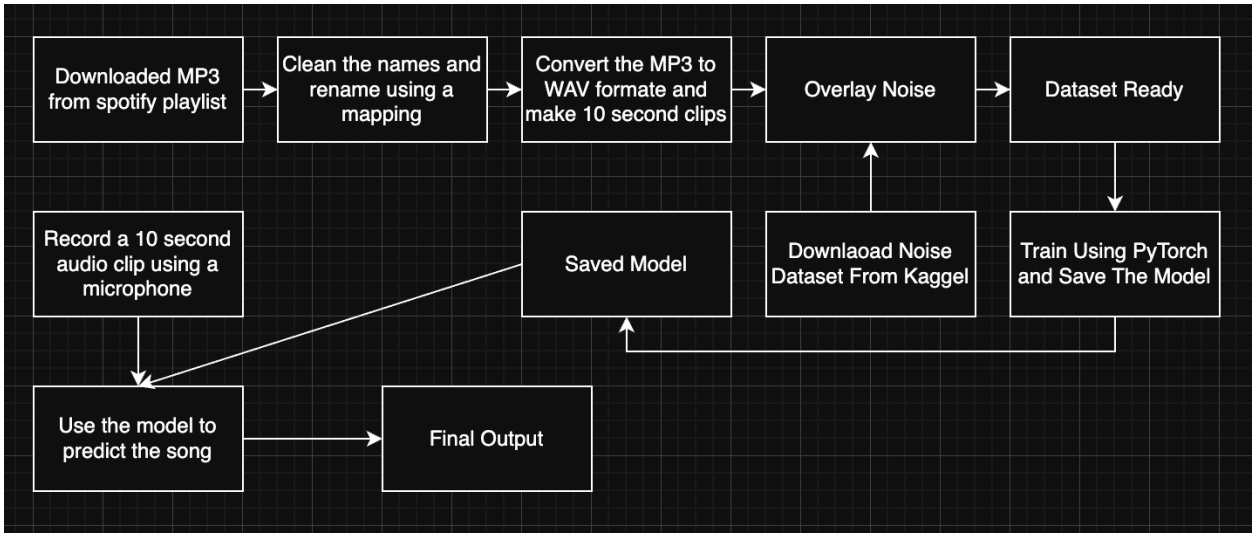

Figure 1: Software Flow Chart

Model Development

The model is a simple Multi-Layer Perceptron (MLP) built with PyTorch, chosen for its fast training speed and efficiency on the Jetson Nano. The model includes six layers: three trainable (input, hidden, and output layers) and three non-trainable (activation and dropout layers). Dropout was used for regularization.

Audio features were extracted using Librosa, including 13 MFCCs, 12 chroma features, 7 spectral contrast measures, the zero-crossing rate, and RMS energy, for a total of 34 features. These features were scaled using StandardScaler, and the scaler was saved to ensure consistency for future predictions.

The model architecture had around 16,600 trainable parameters:

Layer 1 (34 → 128): 4,480 parameters

Layer 2 (128 → 64): 8,256 parameters

Output Layer (64 → 60): 3,900 parameters

The model was trained for 20 epochs, stopping early to prevent overfitting. Unlike Keras, PyTorch requires saving both the scaler and model weights for future predictions, which makes the process slightly more complex.

Results and Testing

The validation accuracy, tested on audio files (not microphone input), reached 97.52%. During testing with *test.py*, the same scaler and feature extraction methods were applied. The model definition had to be recreated to load the weights, which is a typical requirement in PyTorch.

For real-time prediction, a new script, *record_and_predict.py*, was created to capture audio via a microphone and make predictions. Testing with pre-recorded clips (such as 006_011_n1.wav for *All The Stars*) showed high accuracy, correctly identifying songs more than 95% of the time, as seen in Figure 2.

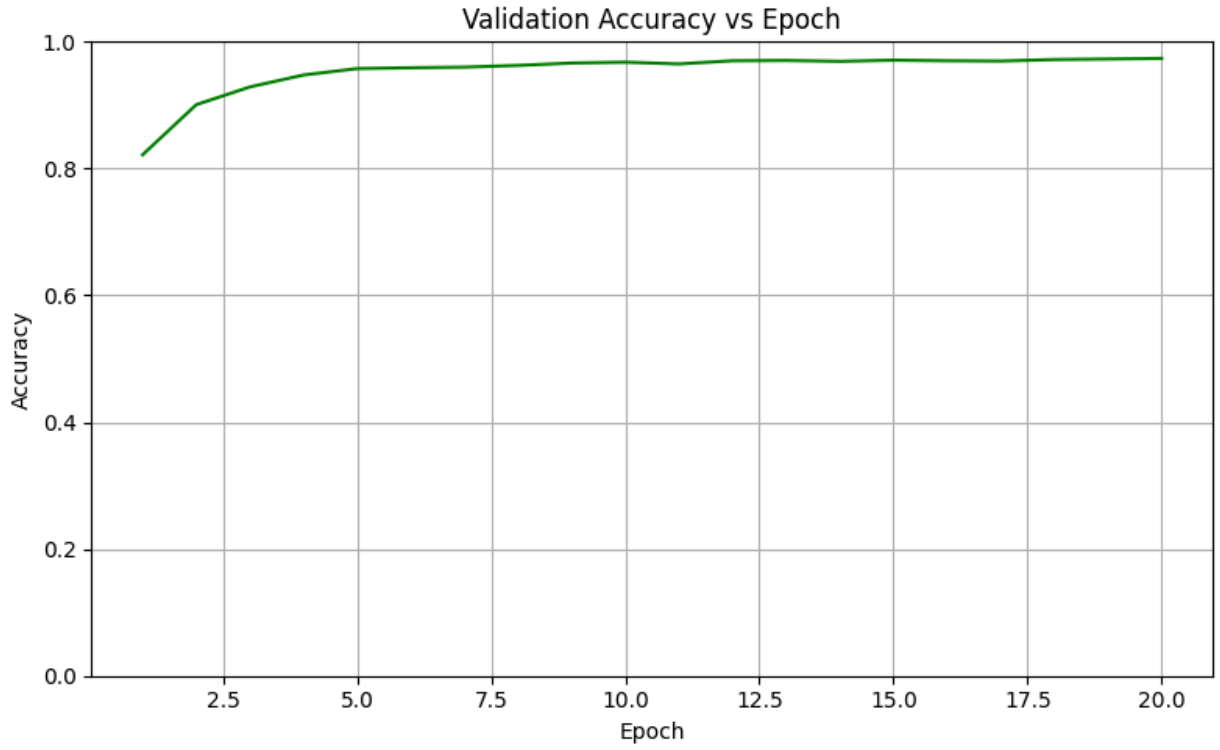

Figure 2: Validation Accuracy vs. Accuracy per Epoch

Real-World Performance and Challenges

However, live microphone tests revealed challenges: out of 10 trials, only 4 were fully correct, and 2 were close (for example, misclassifying *Money Trees* as *Bitch Don't Kill My Vibe*—both by Kendrick Lamar, showing the difficulty of distinguishing similar songs).

Despite high validation accuracy, real-world results were less reliable. This highlighted a key limitation: real-life audio involves unpredictable noise and conditions not fully captured by the training data. Additionally, the dataset's imbalance led to bias toward classes with more data. High similarity between many

of the songs also increased false positives, and a lack of real-world data during training limited generalization.

Reflections and Next Steps

This experience revealed why machine learning is more commonly applied to broader audio tasks (such as genre classification) rather than precise song identification. Related research, such as the Audio MNIST competition, successfully used spectrogram images and convolutional neural networks (CNNs) for classification, but worked best with fewer than 20 classes.

For future improvements, collecting more real-world audio data, especially recordings in varied environments, would likely help the model generalize better. Exploring alternative models, like CNNs applied to spectrogram images, might improve accuracy, especially for complex tasks. Focusing on broader categories like genre or language classification may also produce more practical results. Addressing class imbalance with data augmentation or oversampling smaller classes would improve fairness and performance. Finally, deeper research into audio-focused machine learning techniques could uncover better strategies for handling noisy, real-world audio more effectively. This project highlighted the complexity of audio classification and provided valuable lessons in combining DSP and machine learning for real-world applications.