Chord Quality Identification Using Convolutional Neural Networks

Zeamanuel Zeweldu

Introduction

Analysing musical content requires identifying certain attributes. One of these attributes is the “quality” of a given chord - i.e, whether it is major or minor (or one of a set of more complex chord types). Typically, chord identification on the part of a listener requires a degree of musical theory knowledge and ear training, and is considered an important skill for anyone looking to become an advanced musician. The chord identification problem lends itself well to a classification problem, as suitable features (such as pitch-class profiles or harmonic spectra) can be extracted from audio signals and used to train a model to recognize patterns that distinguish one chord type from another. By framing chord identification as a supervised learning problem, we can teach an algorithm to label the “quality” of chords from numerous examples, thus streamlining a process that would otherwise depend on extensive musical training and experience.

In order to achieve this, I used CNNs on a pruned and diverse dataset of labeled audio clips, each representing a sample of a major or minor chord played on a guitar or a piano. These clips were first converted into log-mel spectrograms, a format that is well suited for processing by CNNs. The network was then trained to learn discriminative features between major and minor chords. Finding a suitable dataset of sufficient size was initially challenging, but I was able to enlarge my dataset by merging two separate datasets.

Data

The first dataset I used was GUITAR CHORDS V3, which contains guitar chords played in different strumming patterns. In order to use this data, I relabeled the folders with “_major” or “_minor” depending on the chord quality. The Bdim data was not used, as it falls outside of the major/minor chord binary. The length of .wav clips in this dataset also varied, which necessitated some trimming during pre-processing.

The second dataset I used was Musical Instrument Chord Classification (Audio), which contains both piano and guitar chords in a more standardized format. After primarily starting training on

the first dataset, I introduced this dataset to provide a more diverse set of instrument and audio types.

Methodology

Feature extraction was done by converting the waveform into a log-mel spectrogram.

The network consists of two convolutional layers, each followed by a ReLU activation and a max pooling operation. The first convolution maps the spectrogram input to 16 feature maps, and the second expands to 32 feature maps in order to capture more complex local patterns. A dropout layer with probability 0.3 is included to reduce overfitting.

The size of the flattened output is dynamically computed during the first forward pass, and it is then used to initialize the first fully connected layer. After flattening, the output passes through a 64-unit dense layer, followed by a final sigmoid-activated output neuron.

After defining model architecture, I trained the network using supervised learning over several epochs. Each epoch involved iterations over mini-batches of spectrogram-lael pairs from the training set. The objective function used for optimization was binary cross-entropy loss, gradients of the loss were computed and updated using the Adam optimizer, an adaptive learning rate adjustment algorithm based on historical gradients. Training loss per epoch is tracked through a printout to monitor convergence. After this, the model is evaluated on a binary prediction, where it predicts whether a chord is major or minor based on whether the probability is less than or greater than 0.5, and the validation accuracy is printed as a percentage afterwards.

Implementation

Preprocessing the data involved resampling to 22050 Hz, trimming all audio to 2 seconds, normalizing peak levels, and converting any stereo audio to mono. The dataset was split into 70 percent training set, 20 percent validation set, and a 10 percent testing set.

The project was implemented in Python, using the PyTorch framework and the torchaudio library for audio processing. Training was conducted in a Google Colab environment with GPU acceleration enabled. Spectrograms were generated using the MelSpectrogram transform from torch audio, with 128 mel bins, an FFT window size of 1024, and a hop length of 512.

The model was trained using the Adam optimizer with a learning rate of 1e-3, a batch size of 16, and a dropout probability of 0.3 on the fully connected layer. Training was carried out over 30 epochs, with performance monitored using the average training loss.

Results

The trained CNN consistently achieves between 80 to 85 percent accuracy rates consistently, which is below the 90 percent target I set in the project proposal. Initial trials with the first dataset by itself did occasionally produce accuracy rates above 90 percent, suggesting that the diversity of the combined dataset may contribute to the somewhat lower than expected accuracy. However, this also implies that the model trained on the combined dataset is less fitted to the dataset features, and would therefore be able to identify chords outside the data more accurately. More epochs were also promising in reducing training loss, although that did not necessarily always translate to marked improvements in binary prediction accuracy.

Training loss steadily decreased during training, and there were no signs of overfitting.

In order to improve the model, the use of more advanced architectures (such as CRNNs) can be considered, as well as more robust normalization techniques to ensure features are consistent across the board. Acquiring and training the model on a more diverse dataset, containing more instrument types, would also produce a more generalized and functional model.