Advanced Audio Course - Chord classifier project

Project report

Kalle Rantalainen & Julian Juujärvi

GitHub: https://github.com/KalleRantalainen/ChordClassifier

Problem definition

We wanted to create a model that can recognize chords from songs. In addition to detecting which songs are present in the song we wanted to detect when these chords appear in the song. So, our problem is a multiclass classification problem for each time-step of a song. This means that there can be only one chord for each time-step.

External resources

We used multiple popular python libraries for this project and a dataset we found from the internet.

Dataset

Finding the correct dataset took some time. We needed a dataset that would contain the chords for the songs, the timestamps for each chord in a song and the actual song files. This was quite challenging as many of the datasets did not contain the songs. However, we found a dataset called MusicBench which we ended up using in our project [1].

This dataset contained all the required information. However, it contained way too much data for us. It contained about 52 thousand 10 second songs or clips of songs. Many of these songs were also augmented versions of the original songs. So there were about 5000 songs in the dataset and then 9 augmented versions for each song. We decided to use only a subset of the original songs as we did not know if the chord labels were still valid for the augmented ones. Also, we have limited computing power so in the end we ended up using 800 files for training, 100 files for validation and 100 files for testing. These 1000 files were randomly selected from the non-augmented files and a new json file was created to describe the data. This json file was created based on file provided by the dataset, but the new file only contained the songs and datasplits we used. We also tried using all the 5000 files for training the model, but it was quite slow and did not improve the accuracy significantly.

Other resources

We also used many common python libraries to help with data processing and visualization and model creation. We used PyTorch to create our CNN model. Seaborn, matplotlib and sklearn were used for data visualization. Librosa was used for feature extraction and numpy was used to store the features.

Methods used

In this section we describe how we processed the data. We also describe what kind of model we used and how we trained our model.

Data processing

Once we had our audio files, we could look at the statistics of the data. We checked which chords every data split contains. From this we could extract “common” chords that is all the chords that are present in every data split. After we had done this, we could also plot the chord distributions in each data split as histograms. After we had trained the initial model with all the common chords, we decided to drop some of the chords from the dataset. We noticed that some of the chords were much rarer than others. For example, there were about 3700 occurrences of C chord in the training dataset while some chords only had dozens of occurrences. As a result, the model never classified a window as belonging to those less frequent chord classes.

We decided that it is better to drop these less frequent classes. To determine which chords to discard, we set a threshold: a chord must occur more than 10% of the frequency of the most common chord. For example, if C is the most frequent chord with 3,700 occurrences, any chord that appears fewer than 370 times in the training dataset will be discarded from all data splits. Now this still results in great class imbalances but provides much better accuracy than leaving all the common chords in the data splits. To get completely rid of class imbalances we could set up a higher threshold, but this would get rid of most of the chords. We could also set up a hard limit that each chord needs to have 500 occurrences. If the chord has less, it is dropped from the datasets or if it has more than some of the data for that chord is deleted. This would also get rid of many of the chords and also reduce the amount of training data radically. We also need to consider the fact that chord classifier with 5 possible chords is not very useful.

Feature extraction

We used constant Q spectrograms as our features. We calculated the spectrogram for a whole song. Then we split the spectrogram into windows. Each of these windows contained 6 time-bins and all the time vectors had 84 frequency bins. With our parameters these windows were about 100ms long. We wanted to use quite short windows so that it would be unlikely that we would have multiple chords active during one window.

Now that we had the windows, we checked from the json file which chord was present in the song during these windows. If two chords overlapped within a window, we determined which one was active for a longer duration based on the json timestamps

and selected it as the target value. Once we had the target chords for the windows, we one-hot encoded those. This way we got multiple feature-target pairs for each song. We repeated this for every song. At the end we were left with about 56000 training features and 7000 validation and 7000 testing features. For each song all of the song's features and target vector were stored in .npz format to be later used.

Model architecture

We decided to use CNN as our model since CNNs work well with spectrograms. We experimented a lot with different amount of layers and many different hyperparameters. At the end we decided to use 3 convolutional layers, adaptive average pooling and two linear layers. We used dropout layer for each of the convolutional layers and after the first linear layer to prevent overfitting. We also used batch normalization for two of the convolutional layers.

We defined the convolution kernels to be non-symmetrical, so those only affected the frequency dimension. This was because we thought that reducing the frequency dimension was more beneficial than reducing the time dimension, and because our features contained 84 frequency bins and only 6 time steps. These non-symmetrical kernels were also proven to work better when we tested the model.

We also tried using max pooling in our convolutional layers but it did not improve the accuracy. We also tried using 2 and 4 convolutional blocks but found out that the model performed best with 3 blocks.

Model training

We used categorical cross entropy as our loss function since we are dealing with multiclass classification problem. We used Adam optimizer with a learning rate of 1e-4. Our batch size was 32 and we had 20 epochs with early stopping to prevent over fitting. Our model reached the best validation loss in 13 to 16 epochs. We used GPU for the training since we had quite a lot of data.

Evaluation setup

We evaluated the model couple different ways. Firstly we used validation data during training to evaluate the model. We calculated the validation loss using category cross entropy. If the validation loss stopped decreasing for 5 epochs, we ended the training and reverted back to the best model i.e. the model with the smallest validation loss.

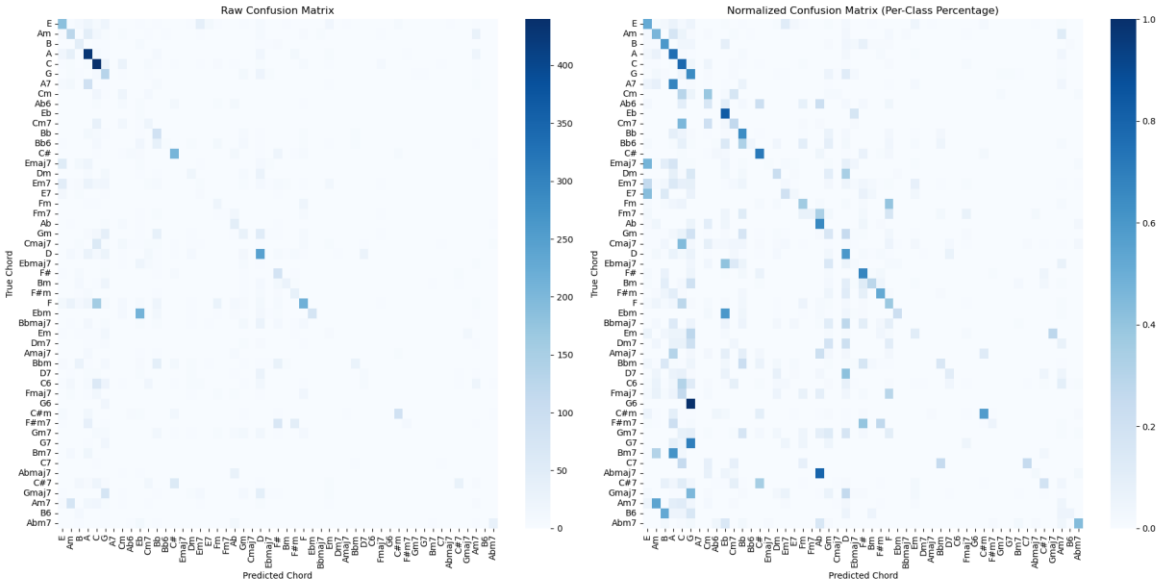
We also evaluated the model using the testing data after the training. With this data we calculated the accuracy and plotted a confusion matrix. These testing and validation sets were taken from the MusicBench dataset but in a way that there is no data leakage between any of the data splits.

In addition to testing the data with the dataset data, we tested the model with our guitar recordings for which we knew the correct chords. Since these recordings had a sample rate of 44000 and the dataset recordings had a sample rate of 16000 we downsampled these guitar recordings before calculating the features to ensure consistency.

Results

Our final model contained 51 output classes, i.e. 51 different chords. Using this model we achieved accuracy of about 37%. We think that this accuracy is ok considering that

a random classifier would reach an accuracy of about 2%. However we do not think that this is anywhere near good enough accuracy to base any application on.



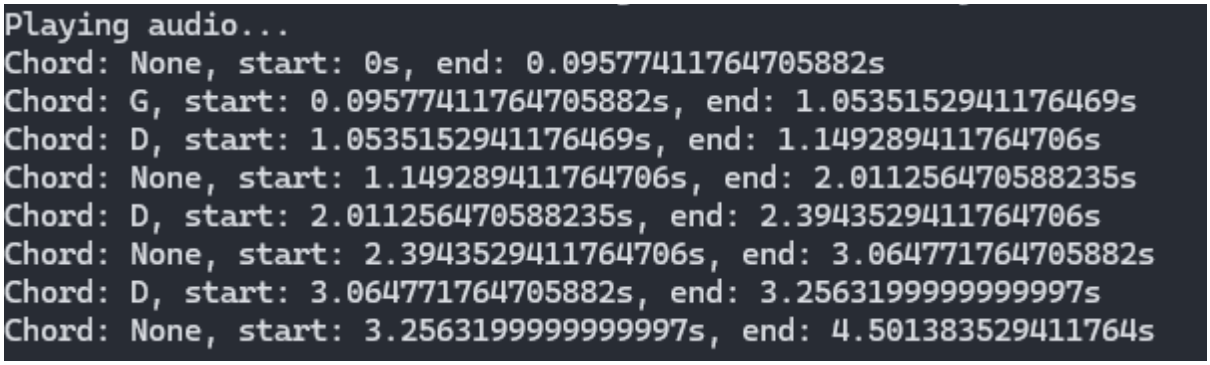
Here we have two confusion matrix of the testing data predictions. The one on the left side is just a normal confusion matrix. The on the right side is normalized in a way that the occurrences in each cell are divided by the total number of occurrences on the cell's row. This better represents the information even for chords that may have less occurrences in the testing data.

From the matrices we can see that the most common chords, such as C and A are very well classified. Actually the class-wise accuracies for these classes are 81% for A chord and 78% for C chord. Then when we look at a chord like Dm7, we can see that it is never predicted correctly.

We can also see another phenomena from confusion matrices. G chord is a pretty common chord that appeared in the training data over a thousand times. G chord variations however are not nearly as common in our dataset. These variations are G6, G7 and Gmaj7. When the target chord was any of these less common G variations it was almost always predicted to be a G chord. This is probably because there are only slight variations between the chords that model can't well detect and because there are more cases for G than for the variations.

When we tested if the model would be able to detect correct chords from our own guitar recordings, the model performance was somewhat similar to the testing results conducted on the dataset testing data. When we tested the model with our own recordings we defined a threshold for the predictions. If the predicted chord had over 50% probability of belonging to some class, it was placed there. If the probability was lower than that it was marked as None. This way we could only get more confident predictions and also display None for empty parts or for chords our model does not

know. The results below were detected from a file where only D chord was played. The D was strummed 3 times on a guitar so the results are actually quite good. The model displays None in between each chord because the chord gets too quiet to detect. The G chord is a misclassification but since it is only 100ms long, it could probably be discarded in a real world situation since it is very unlikely to have only 100ms long chord in guitar track.



List of references

[1] MusicBench datset: https://huggingface.co/datasets/amaai-lab/MusicBench