A Deep Learning Approach to Musical Chord Recognition

Michael Kruppa

**Abstract**

I present a deep network model for automated chord recognition in musical recordings. Audio files are split into multiresolution STFT frames and converted to a “wrapped spectrogram” format to preserve octave information. Training was done on an annotated set of Beatles songs publicly available, and the model achieved a frame accuracy of 70% on the validation set. Furthermore, the model has reasonable success when tested on completely foreign audio recordings, showing good generalization. I conclude that deep network approaches are effective at solving this problem, and that with more development, even more accuracy can be achieved.

**Introduction**

Chord recognition is a fundamental task in music processing with many applications. Perhaps the most obvious use case is for arranging a piece of music or for making a cover; it would be infeasible to try to transcribe long pieces by ear, so having an automated tool would facilitate this. Similarly, if one wanted to play along with a recording, for example by strumming chords on a guitar, it would be useful to have a program to read out the chords. Finally, chords (and their progressions) form an important aspect of music in general, and they can be useful in analyzing other features of a song, such as structural segmentation, style analysis, and content IDing.

There are several approaches to chord detection, including template-based matching, Hidden Markov Models, and deep networks. Template matching utilizes chromagram “templates” of each chord class based on expected intensities and harmonics. Each unknown chromagram is then compared to each template to find the closest match. Christoph Hausner was able to achieve 72.4% weighted average overlap ratio in the 2009 MIREX competition using a template-based approach with harmonics [1]. Hidden Markov Models combine information from the previous frame’s estimation as well as transition and emission probabilities to compute the optimal chord sequence through a song (Viterbi Algorithm). Meinard Müller argues in his textbook, *Fundamentals of Music Processing*, that “the concept of HMMs has been widely used in applications such as speech recognition and also constitutes the de facto standard method in most automated procedures for chord recognition” [2]. Nevertheless, deep learning techniques have recently been the focus of intense research due to their ability to model complex relations between variables and incorporate large amounts of training data. J. Osmalskyj et al. was able to create a deep network model to detect ten chords played in isolation using a small neural network[3]. Finally, I referenced the work of Christopher Harte, since the format of the chord annotations in the dataset I used was based on the notation he devised [4].

I was interested in pursuing a deep network approach because it seemed like it had the best possibility to perform well in real world testing. Some chord recognition models were only trained and tested in isolated environments and are only able to run inference on a recording of a single chord, as opposed to a full recording. Moreover, there are variations on how expressive models are, with some being able to describe only a limited set of chords. As a starting point, I wanted to describe all 24 major and minor chords, which would allow me to examine nearly any song in the Western Classical Music canon.

In order to do this, I used a dataset of annotated Beatles songs[5], which provided the training data for my model. I achieved an accuracy of 70% on the validation set, and even with this accuracy, I was able to produce a useable chord recognition script that could render labeled videos ready for play along.

**Background and Theory**

In audio processing, use of the Short Time Fourier Transform (STFT) is a common technique to extract frequency information from a relatively localized time segment. However, there is tradeoff between frequency resolution and time resolution depending on the width of the localizing window – a longer window results in better frequency resolution and worse time resolution and vice versa. Especially in image processing, multiresolution features are often extracted, which attempt to get around this tradeoff by taking multiple transforms of an input at different scales. Focusing on music signal processing, the STFT is often further processed into a spectrogram, which pools frequency intensities common to a single MIDI pitch together, resulting in a single vector representing strengths for each MIDI pitch. This is often further processed into a chromagram, which combines all octaves of the same pitch together, resulting in a single vector of length 12.

**Methods**

Since I was using a supervised learning approach, I needed training data, and I used a publicly available dataset containing annotations of Beatles songs [5]. The annotations contained information on beats, chords, segmentations, and keys, but I only used the chord annotations. Next, I needed the actual songs, which I found on Youtube, and using the python package youtube-dl, I wrote a script to download 87 songs. This use falls under Fair Use, since I am using these recordings for my own purposes as a free, educational, and research-oriented usage.

A picture containing chart

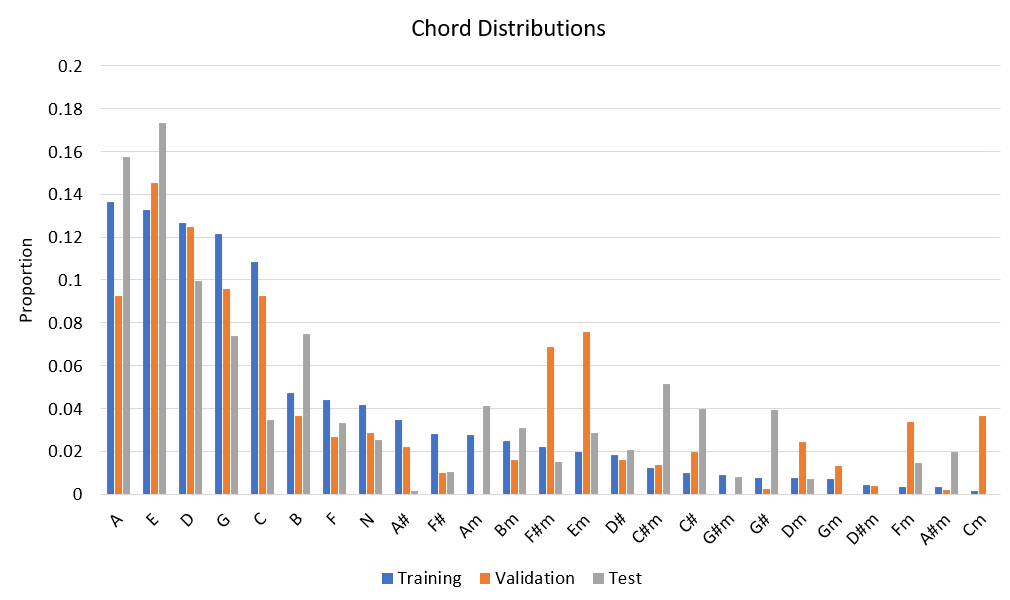
Description automatically generated After the songs were downloaded, I preprocessed them to extract the relevant data so that training would be as fast as possible. I made several decisions to try to improve the amount of useful information I could give to the network. First, when calculating the STFT, I used a multiresolution approach, by computing it with H = 0.1s, window = 0.2s and 0.8s. With the shorter window, I was hoping to capture more time locality information, at the expense of frequency resolution. With the longer window however, I can capture more frequency information over a longer time, but this actually has the additional benefit of being less susceptible to being confused by broken chords. Code from Meinard Müller’s FMP Python notebooks was adapted and used for this processing[2].

Secondly, I decided to not use a chromagram and instead use a wrapped spectrogram. Chromagrams effectively lose information by combining octaves, and this is undesirable. Instead, I wrap the spectrogram into a 10x12 “image”, representing the MIDI notes [0, 120), which preserves all its information while suggesting structure. For example, in the figure on the right, this is a wrapped spectrogram showing a D major chord.

For the sake of a reasonably simple yet functional first model, I chose to only capture the 24 major and minor root chords, as well as a “no chord” label for silence or applause. The Beatles annotations had more information, including 7th chords, inversions, adds, sus, etc, so I had to simplify that to the base chord. In addition, since I am not estimating the overall key, I lose the enharmonic information, so I might call an A# chord B flat, for example.

The Hidden Markov Model derives a lot of its accuracy from the fact that it considers the previous chord as influencing its decision on the current chord, and I wanted to adopt that idea, so as part of each labeled data pair, I have 4 wrapped spectrograms interpreted as channels of an image, being the short window current frame, long window current frame, short window previous frame, and long window previous frame.

Finally, for data segmentation, I initially segmented it “cyclically”, with a ratio of 8/1/1 for train/validation/test. In other words, going through a song’s frames sequentially, I would give the first 8 label pairs to the training set, then the next one to validation, then the next to test, and all over again. However, I realized that, although I wasn’t explicitly training on the validation or test sets, I was effectively doing so. This is because consecutive label pairs don’t change too much, and so whatever label pairs appeared in the validation/test sets, a very similar pair would have appeared in the training set. Therefore, I changed my segmentation to be by song, and in the end, I had 71 training songs (108123 label pairs), 8 validation songs (10999 label pairs), and 8 test songs (11444 label pairs).



It is also interesting to consider the distribution of chords in the data, as this could suggest that the model would be more sensitive to certain chords. We can notice several things immediately. First, the distribution between training, validation, and test sets is very roughly the same, as it should be, since they were randomly assigned. This is good, since the network will be training more on chords it will be tested on more. From a purely musical perspective, it’s interesting to see that the Beatles favored mostly the common major chords of A, E, D, G, and C, with noticeably fewer minor chords. Thus, we can interpret this graph as some measure of style, typical of the Beatles. This might also suggest that the model, trained on this data, would have a harder time generalizing to other styles of music. One way to address this is to transpose each piece to every key (implemented as rotating the spectrogram vector), so that all major keys and all minor keys will have the same amount of data.

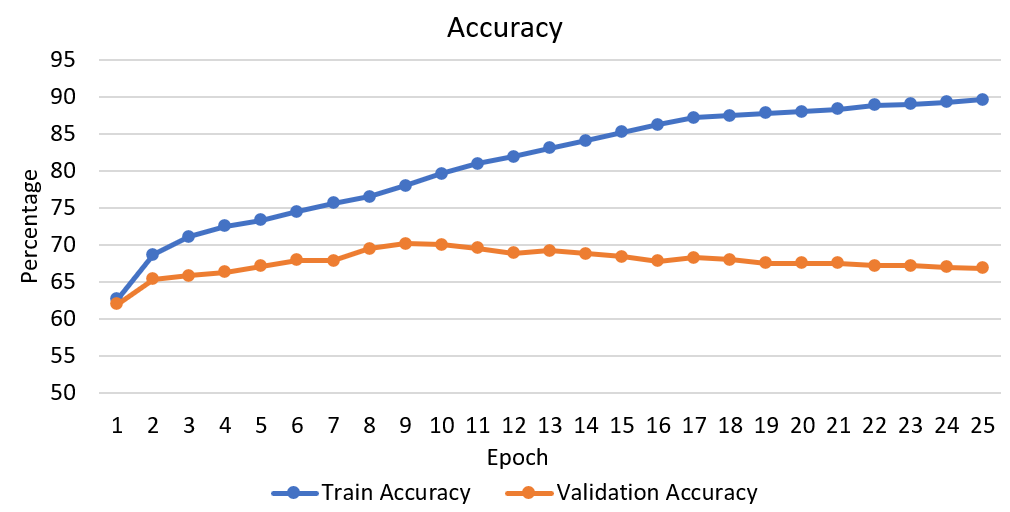
With all the preprocessing done, I simply save the data in a pickle file to be read into a lookup table by the network. Fundamentally, this is a classification problem with 25 classes, and so the last layer in the network has 25 neurons with one-hot encoding and softmax. At the front of the network, we have essentially an input image from which to extract data from. Convolutional layers are often used with images, and I used them with the hopes that they would pick out certain shapes of the wrapped spectrogram according to the harmonics series. However, there are some more subtle details. First, convolutional layers often have a padding scheme of zeros or a constant, but this would bias the resulting activations since the wrapped spectrogram really needs to be interpreted cyclically, as a cylinder. Therefore, I had to cyclically pad before every convolution, but only along the last axis, since the data is not cyclic along the second to last axis. In addition, convolutional layers attempt to leverage spatial invariance, especially for detection-type problems. In this case, a major chord pattern is theoretically spatially invariant, but where it manifests is extremely important, since its precise location determines the chord type. Therefore, I pad after every convolution and do not use pooling layers.

Diagram

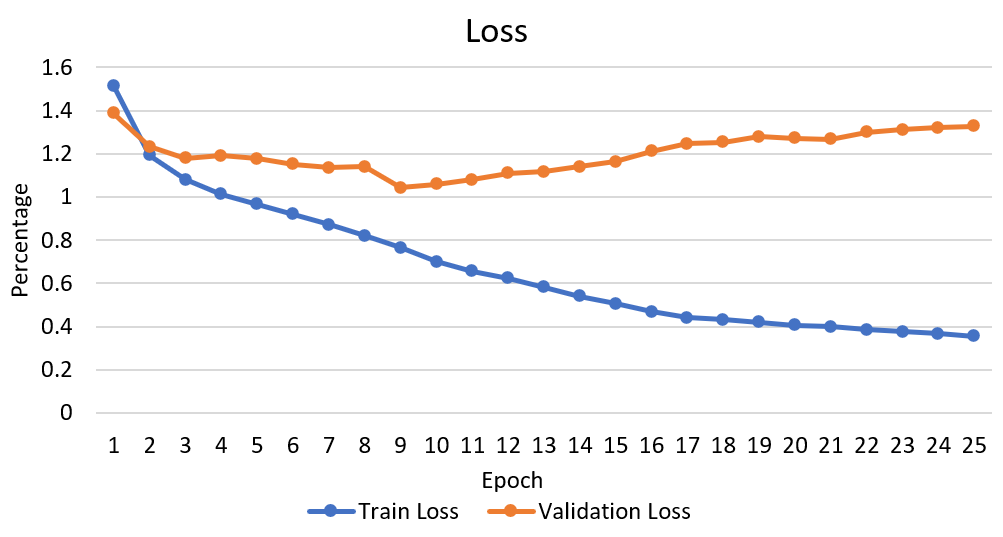
Description automatically generated

Above is the network architecture; it features four convolutional layers and then two fully connected layers. Notice that for the entire convolutional section, the last axis always has length 12 to respect the number of notes in the octave. In addition, the convolutional kernel is 3x9, since horizontal features are “wider” than vertical features. For example, to reach a perfect fifth (3rd partial) across the spectrogram, we need a length of 8 pixels, but the information is denser in the vertical direction. The batch size was 128, learning rate was 0.0005, and the Adam Optimizer was used with L2 weight decay of 0.001.

**Experiments and Results**



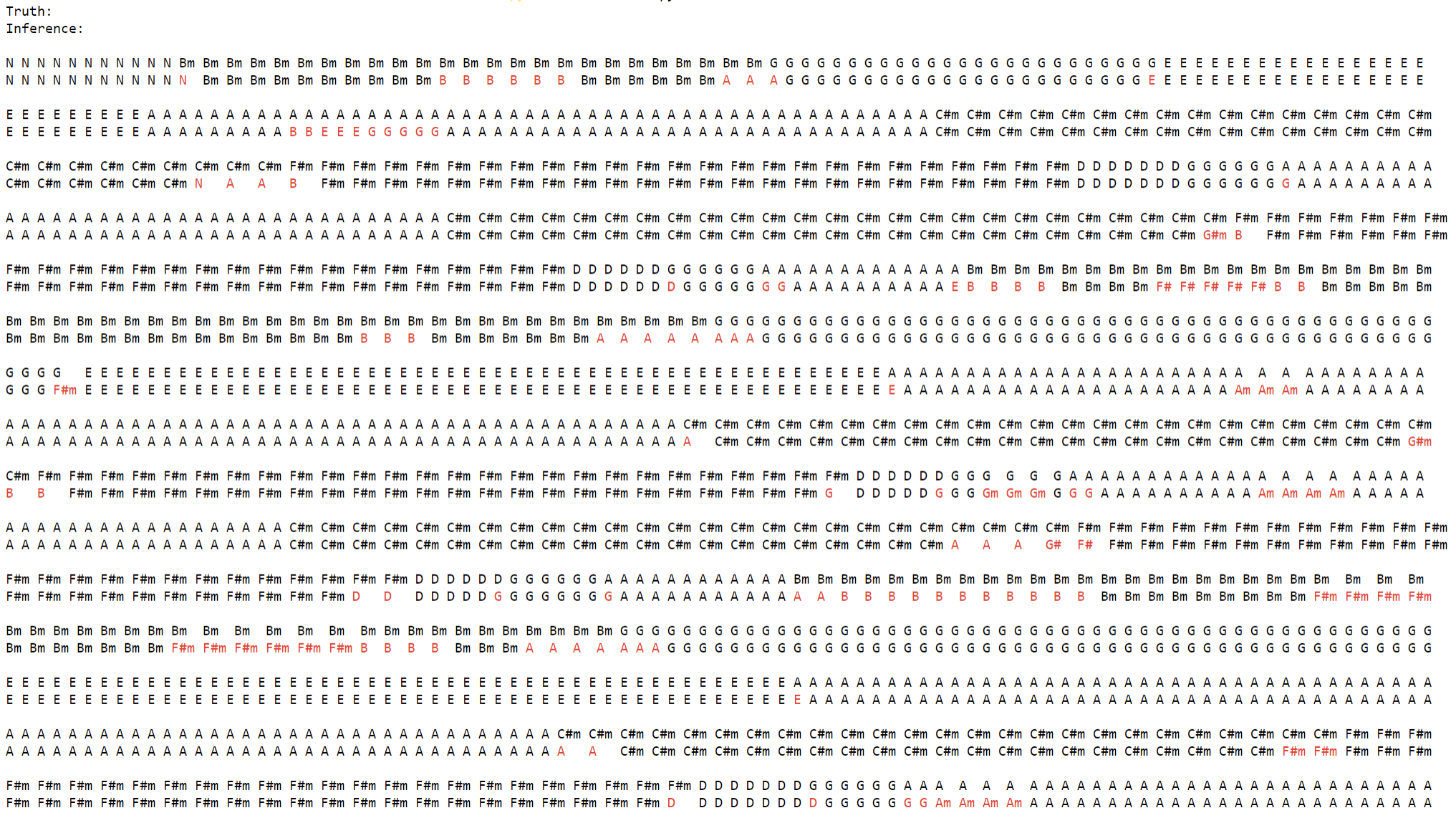
The model was trained for 25 epochs, but I stopped early once I saw overfitting; this occurred after roughly nine epochs. After the ninth epoch, the model achieved an accuracy of 70% on the validation set. I tried a variety of changes to the model, including changing the architecture, kernel size, learning rate, learning rate scheduling, batch size, and this was the best combination of hyperparameters I could find.



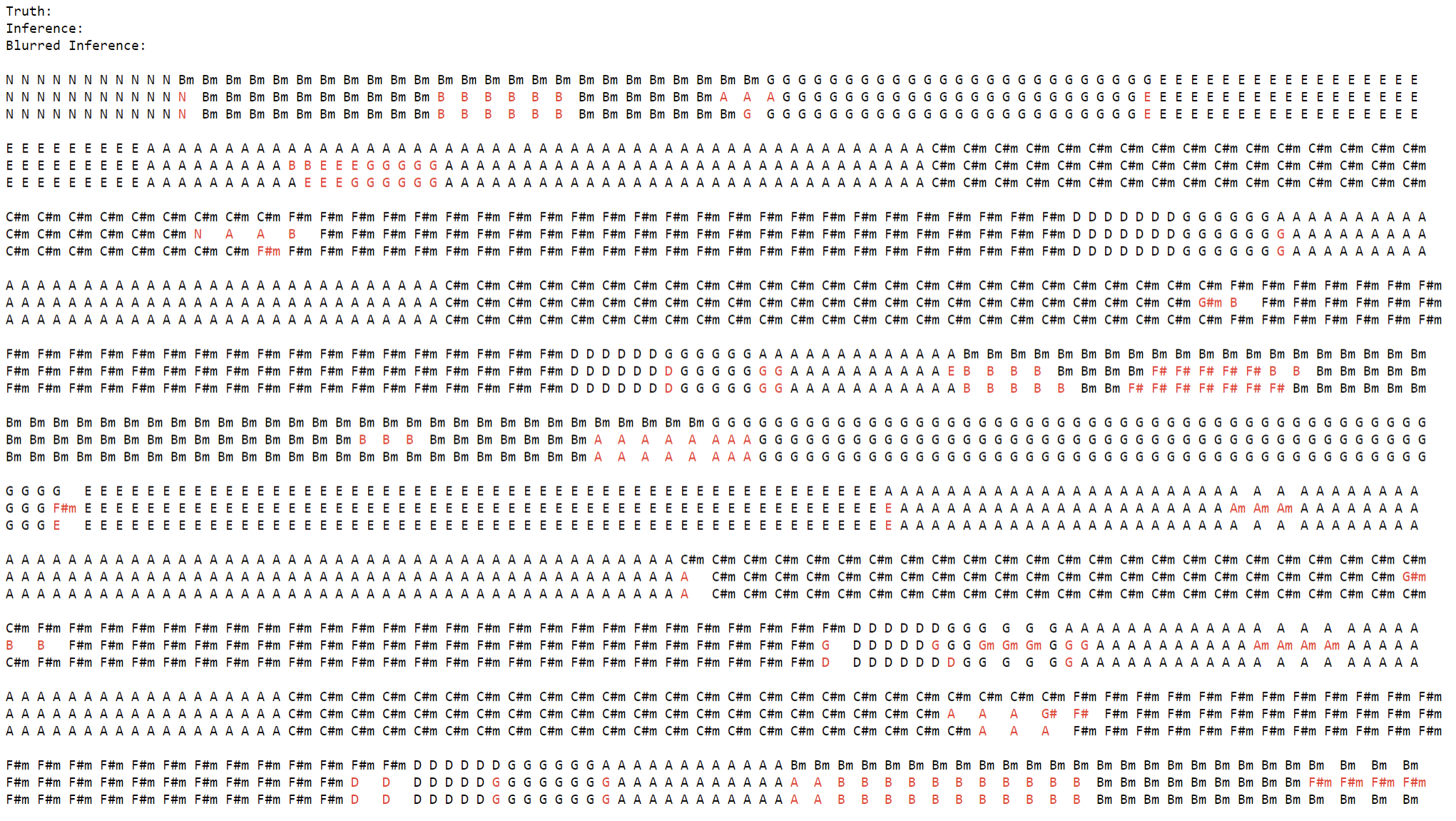
I next ran inference on the song “Help!”, one of the songs in the validation dataset, to get a better understanding of what was going wrong. Below is the confusion matrix for chord labelling. The overall F metric is 0.899, this is specifically the average of all the F values for each of the non-zero classes. We can see large values down the main diagonal indicating generally correct inferences, but we can also see that there is some major-minor confusion; for example, 45 chords were labeled B major instead of B minor, similarly with A major and A minor.

A picture containing text, white

Description automatically generatedWe can get an even better idea of what is going on by looking at the frame-by-frame annotation comparison to ground truth. There are multiple types of errors that appear. Very commonly, there is a boundary error where the estimated and ground truth boundaries are off by a couple frames. This is probably the least blatant mistake, since, for the common use cases of chord recognition, this doesn’t pose much of a problem.



A more significant error type is major-minor confusion, which is also somewhat common. Major and minor chords of a particular key share two notes, and if the determining third is weak, there can often be confusion. More generally, chords that share notes can be mistaken for each other, like C# minor and A major, which both share C# and E. Finally, there is the mistake type where there are entirely wrong chords for a frame for no good reason, like having a “no chord” label in the middle of a song. Notice that these mistakes are sparse, in that they are generally surrounded by more correct chords. This motivates a post-processing step involving a mode blur, as used by Hausner[1], which will replace a chord with the most common chord in a small window around it. Mode blurs are edge-preserving and should remove the occasional wrong chord. Below is the same frame-by-frame comparison as above except showing both the original inference and the mode blurred inference.



One can see that the mode blur helps to remove the occasional wrong chords, though it can’t correct major-minor confusion or boundary errors, as expected. With the mode blur, the average F value for Help! rises to 0.914, so there as a small but noticeable improvement.

**Discussion**

Overall, the model performs reasonably well to the point that it could be used for play- along. As mentioned above, some errors like misjudged boundaries do not affect performance that badly, and the mode blur helps reduce the mistakes even further. Most humans can hear the difference between major and minor chords, so if the inference says D major but it sounds minor, there is a good chance it is actually D minor. I also packaged the model for “inference in the wild”, which runs inference on audio completely unknown to it, outside of the Beatles data. I initially wanted to have a script play back the audio while printing out chords at the correct times, but I had trouble synching the audio and printing due to (random) latency in starting the audio buffering for the player and starting a new thread for the printer. Therefore, I instead rendered a video that contains the chords at exactly the right times, as well as a “dial” that shows two past and future chords, and I stitched this together with the audio file using ffmpeg.

During hyperparameter tuning, I tested one variation where I omitted the previous frame information so I only had two input channels. To my surprise, I achieved 69% accuracy on this model, whereas I had expected it to perform more poorly. After all, Hidden Markov Models can perform well using the same sort of prior information. However, I realize that this is not quite the same situation. In HMM, the actual previous chord itself is known (inferred), and this informs the next chord. However, I am not actually giving it the previous chord, just the previous spectrogram, from which the model has to extract the chord out of. But the model itself is trying to learn how to do this, so it’s a bit like circular reasoning; it can either successfully extract chords from a given frame, in which case it won’t need the previous chord, or it can’t, in which providing the previous spectrogram won’t help.

During the live demo on presentation day, my “inference in the wild” script worked well, and I was able to test the model on a variety of songs including Beatles songs, country music, classical music, and pop. As expected, it did worse the more the style deviated from that of the Beatles. It was interesting to test it on the pop songs used by the other project so we could have a comparison of approaches. Based on the discussion, it seemed that my usage of a wrapped spectrogram, instead of a chromagram, was crucial in being able to distinguish between 7th chords that could be interpreted as at least two chords in my 24-chord library. Furthermore, it seemed that my model relied on having the melody line, often sung, contain mostly chord tones, and not straying too far away from them. Once the melody line was not helpful in determining chords, the model made more mistakes.

**Conclusion**

On the whole, the model was successful in attaining reasonable accuracy in inferring chords. Although it only achieved 70% accuracy, certain mistakes are not as important for use cases like live play-along, where the musician is able to combine information from their own understanding of the music with the model’s predictions. Besides those errors however, the model does produce completely incorrect chords sometimes, leaving a lot of room for improvement.

In the future, I would like to expand the chord library to be able to detect 7th chords, especially dominant 7th chords since they are so common in many styles of music. In order to train on an even distribution of chords, I could transpose every training frame to every other key, resulting in 12 times the amount of training data. It would also be very useful to determine what key a piece is in, so that there would not be confusion between enharmonics.

While discussing the project after the demo, I realized that it would also be useful to try source separation to have the ability to ignore certain lines if they aren’t helpful in chord recognition. In addition, instead of having convolutional layers at the start of the network, it would be better to simply have dense layers all the way through, especially since the image size is rather small. Finally, it would be worthwhile to look at the spectrograms that resulted in incorrect classifications in the hopes of discovering why the model made a mistake.

**References**

[1] C. Hausner. Design and Evaluation of a Simple Chord Detection Algorithm. University

of Passau, 2014.

[2] M. Müller. Fundamentals of Music Processing, 2nd ed. Springer, 2021.

[3] J. Osmalskyj, J-J. Embrechts, S. Piérard, M. Van Droogenbroeck. Neural Networks for

Musical Chords Recognition, University of Liège, 2012.

[4] C. Harte. Towards Automatic Extraction of Harmony Information from Music Signals,

University of London, 2010.

[5] <http://isophonics.net/content/reference-annotations>