

# Major or Minor

## Audio Classification Project





By: Ben McPeck





# Background

In music there are four main categories of chord qualities: Major, Minor, Diminished, and Augmented. Each of these chords have a specific notation shape that determines its class, however the order of the notation can vary incredibly. For this reason in this project we will be exclusively focusing on Major and Minor chords.

triad type	example	symbol(s)
major triad		Cmaj, C
minor triad		Cmin, c, Cm, C-
diminished triad		Cdim, C°
augmented triad		Caug, C+

# Background: Inversions

Fundamentally a chord is constructed out of the first, third, fifth, and seventh note of a seven note scale. Inversions are different ways to order a chord. Here are the most basic examples of chord inversions.

7ths

	Root	1st	2nd	3rd	Root
	7	6	6	6	7
	5	5	4	4	5
	3	3	3	2	3
Simplified:	7	6 5	4 3	4 2 (or just 2)	7



## Background: Challenges with Data

Chords in our audio data contained between 3 - 6 notes per file. Chords containing more than three notes can have cross relationships with other chord qualities within their inversions. Here is an example showing that C6 and it's relative minor Am7 contain the exact same notes. Time constraints did not allow to remove these files however this will impact the performance of our models



Diagram illustrating the relationship between C6 and Am7 chords, showing they contain the same notes.

**C6** and **Am7** are shown side-by-side. The C6 chord is represented by a treble clef, 4/4 time signature, and a first finger position (1). The Am7 chord is represented by a treble clef and a first finger position (1). Lines connect the notes of the C6 chord to the notes of the Am7 chord, demonstrating that they share the same notes.

**TAB** (Tuning) is shown below the chords, indicating the fret numbers for each string:

String	C6 Fret	Am7 Fret
8	8	5
9	9	5
7	7	5
8	8	5



# Problem Statement

We are looking to build a CNN model that can classify the difference between a Major and Minor chord by processing the audio files as spectrogram images. Using spectrograms is the most common current method for audio classification.

This project aims to determine:

- Which formatted spectrogram performs best for a CNN model?
- Which model performs best to determine audio binary classification?

In this project we will test different augmented spectrograms on CNN models and determine our success based on how it compares to our null\_baseline of 58.4% accuracy. This project can give insight to what models and which spectrograms should be used for future audio classification projects.



# Data Summary

Audio Files (Major): 502 | Audio Files (Minor): 357 | Audio File Type: .wav

Total Audio Files: 859

Percent of Major: 0.5844 | Percent of Minor: 0.4155

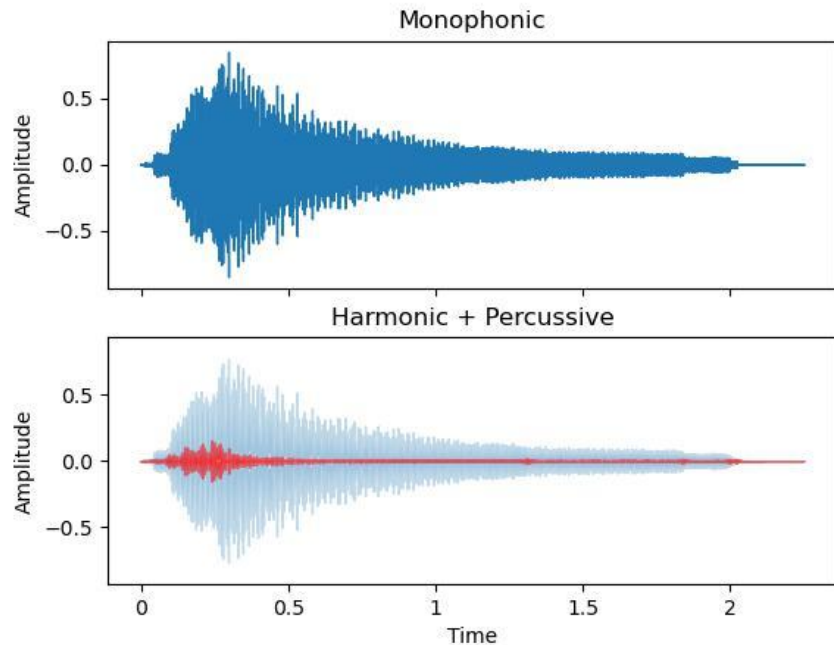
Null-Baseline = 58.4%

We will create a dataframe composed of the file path names of each audio file and its corresponding target class {'major': 0, 'minor': 1}. This dataframe will be used in a for loop to preprocess the audio data into spectrograms.

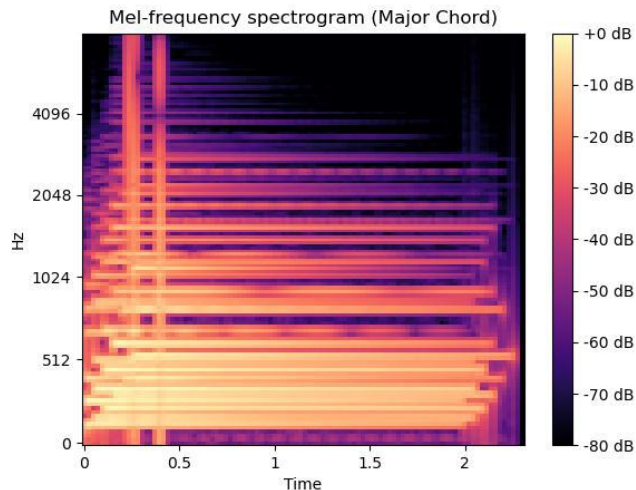
Attribute Name	Data Type	Description
<code>chord_qual</code>	[object]	[the file path name of the audio file]
<code>target</code>	[int64]	[a boolean of 0 or 1 determining the target class {'major': 0, 'minor': 1}]

# EDA: Raw Audio Data

We can see that all transients have one initial peak and wane from there. This indicates that there is only one strike of the chord for each file. We can also see little percussive elements in our audio. Opposed to the initial strike of the chord there are no follow up indications of rhythm.



# EDA: Image vs. Spectrogram



For images, the weights on the x and y axis are the same. Every point on an image represents a pixel intensity. For spectrograms, the x and y axis are fundamentally different. The x-axis represents time and the y-axis represents amplitude and frequency. This means that models will respond differently if a spectrogram is rotated.





## EDA: Misabeled Data

Part of the EDA for this project was to randomly sample audio from both classes and test if they were properly labeled. The first mislabeled class I found was this F Diminished Triad which does not match either chord quality classes. This is a mislabeled class and will impact the quality of our model. Future consideration will be to locate other data that is properly labeled or create an original dataset with simpler chord qualities.



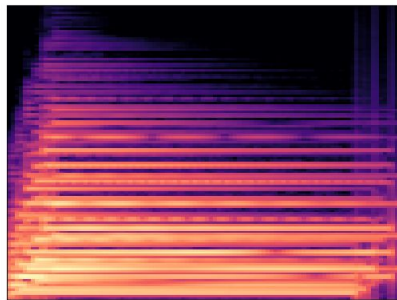


# Mel/Spectrograms

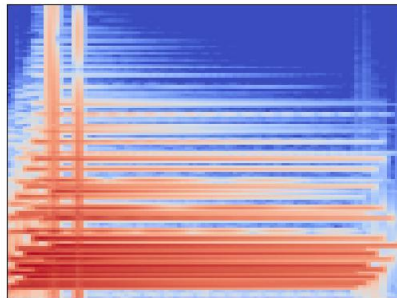
A Mel-Spectrogram uses a fourier transformer to convert the frequencies to the mel\_frequency scale. This can help the resolution with lower frequencies that otherwise would be harder to visually represent.

This project tested mel-specs and scaled mel-specs. We scaled the signal array of our mel-specs using StandardScaler as a method of keeping the input data format as consistent as possible.

Original Mel-Spec



Scaled Mel-Spec





# Preprocessing Summary

We created the class, Audio Transformer, to load in an audio .wav file with a set sample rate of 22050 and setting the file to mono. The Audio Transformer class will be run through a for loop that reads each 859 unique audio files from the dataset created in the eda\_audio notebook.

Feature augmentation to create more data:

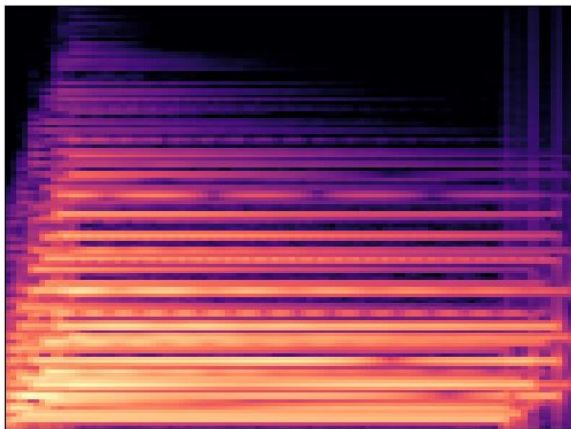
- `time_shift`: shifts the audio to a random starting point within the set signal length of 50,000
- `transpose`: takes the audio signal and doubles it to be exactly on higher than its original
- `aug_spectrogram`: masks a bar of the y and x axis and random widths by setting those axis areas to 0. This is intended to prevent overfitting but did not perform well in my models.

By creating a set of original spectrograms, a set with only `time_shift`, a set with only `transpose`, and a set with both `transpose` and `time_shift`, enabled me to quadruple my data.

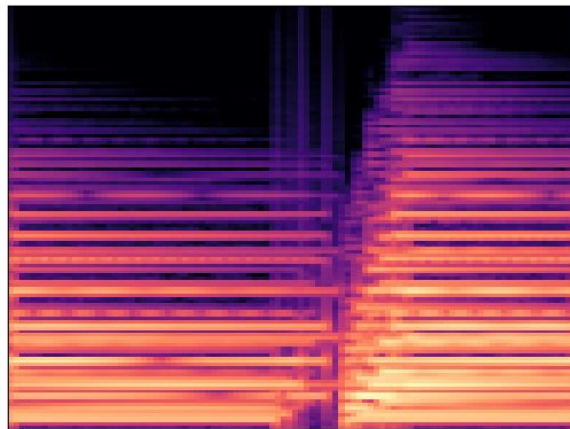


# Example of Time-Shifting

Original Mel-Spec



Time-Shifted Mel-Spec

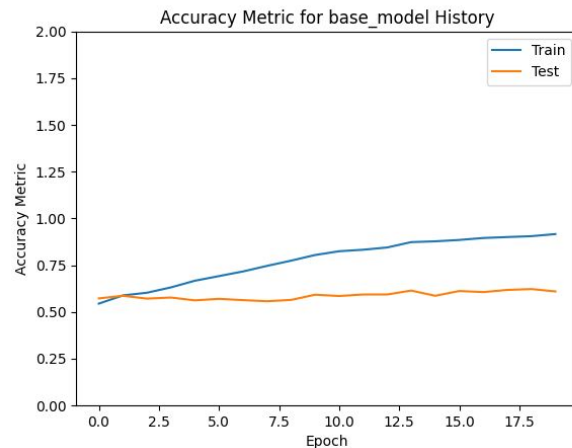
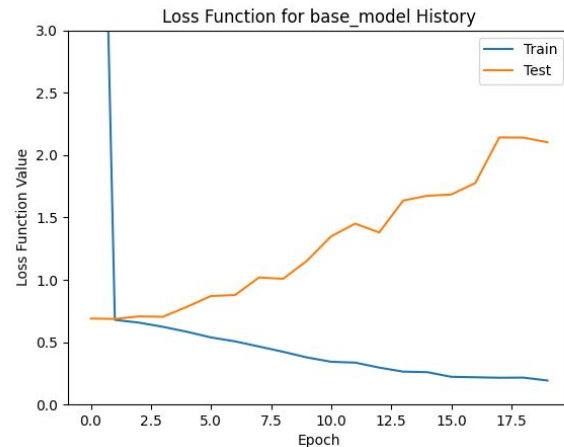




# Base Model

Accuracy	Loss
60.89%	2.1

Our model is severely overfit because our training data scored an accuracy of 91.64% which is too great of a distance. It is also clear that our validation data loss value grew exponentially over the epochs. This means the difference between the predicted probability and the actual binary label was getting worse and our model is not learning the different classes very well.

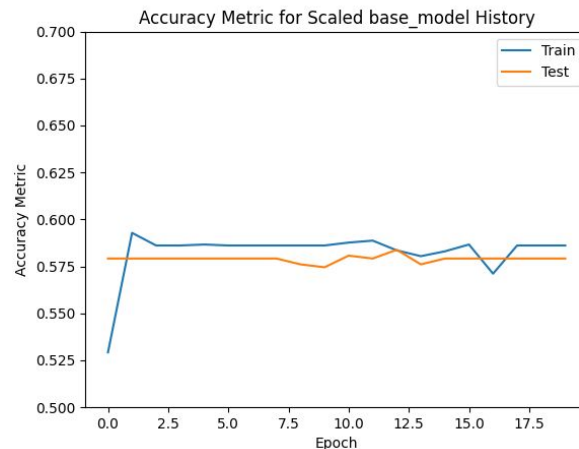
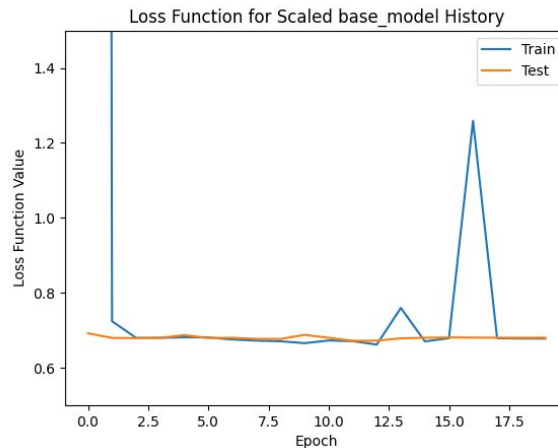




# Scaled Base Model

Accuracy	Loss
58.77%	0.75

This model also did not perform well to our defining success. As we can see our validation data performed with an accuracy of 58.77% which is the closest to our null baseline (58.4%) so far. Although our overall loss has decreased and improved from our first model (0.75 compared to 2.1), it is clear this model is stagnant. We can see no development or increase in accuracy and little decrease in overall loss.

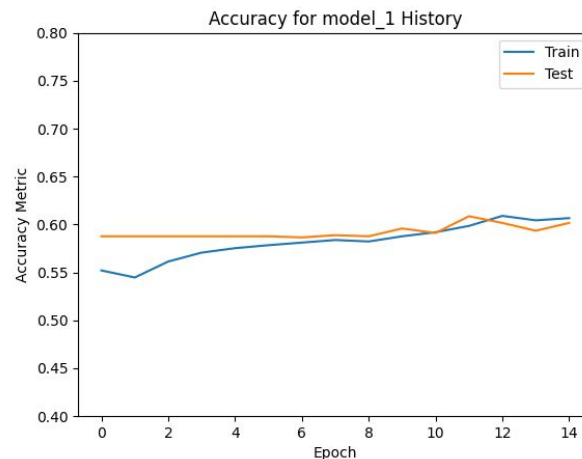
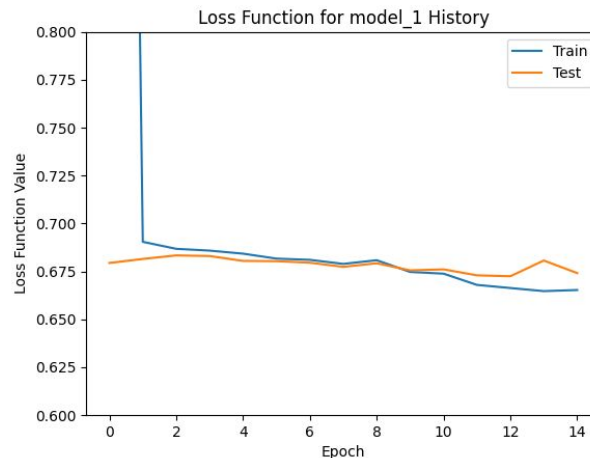




# Layered CNN Model

Accuracy	Loss
60.16%	0.67

So far this is our best performing model but only by a fraction. Visually we can see a subtle increase in accuracy over our epochs. Similarly we can see a gradual decrease in our overall loss. So far this model has the lowest overall loss at 0.67. Although this passes our null-baseline this is still an underperforming model.





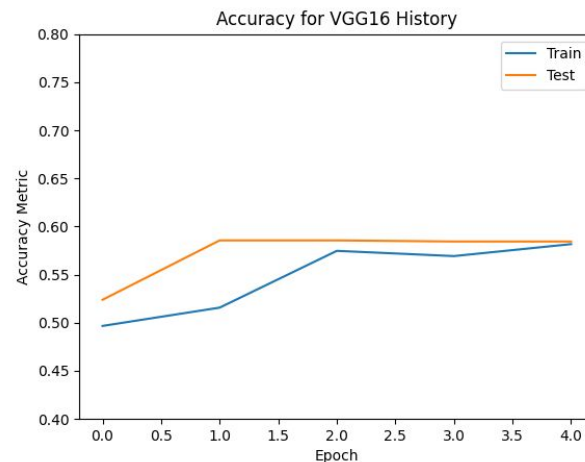
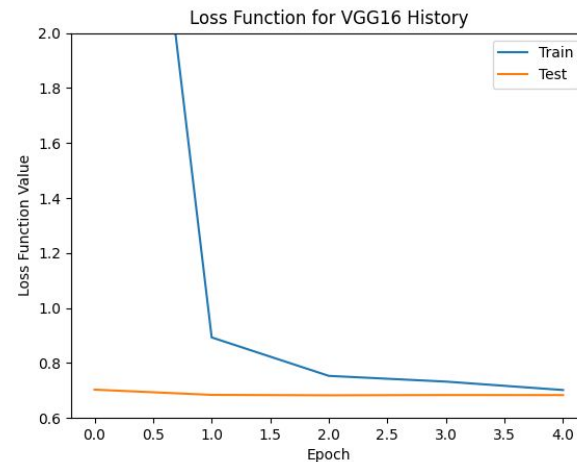
# VGG16 Model

Accuracy	Loss
58.44%	0.68

VGG16 has been trained on an extensive library of images. It's architecture has proven to successfully produce highly accurate image classification models. There could be many explanations for why the VGG16 model did not perform well with our spectrogram dataset.

For future considerations I would like to propose these solutions to increasing the accuracy of our VGG16 model.

- Increase the amount of audio samples from both chords and ensure there is a closer even split of classes.
- Create a large multi-class dataset of detailed categories of chord qualities. VGG16 was trained on a 1000 multi-class dataset.







# Conclusion

Scaling our data using the z-score method stagnated the accuracy and loss results. For future consideration this project will aim to compare the performance of scaled and unscaled spectrogram data by running multiple iterations of CNN models through both to be able to compare a large series of results. Discovering the optimal way to format spectrogram data for a CNN model is still uncertain and this project aims to find a clear answer.

Although our `added_layers_model` performed the best and improved from our `null_baseline`, none of our models show significant clear results. There could be a number of reasons for this but two potential problems this project aims to tackle are the mislabeled class data discovered and the vague classes. Mislabeled data is the most obvious reason why our model could be performing poorly. For future consideration this project aims to generate our own audio data with completely correct class labels. We also aim to simplify the audio context by only recording triad chords with their respective inversions. Triad chords are much more distinguishable from one another than 7th chords. The majority of the samples listened to of the data used in this project were 7th chords causing potential overlap of chord qualities.



# Recommendations

- For audio classification models it is suggested to initially work with a smaller dataset. With a smaller dataset you can ensure each file is correctly labeled and have a better understanding of the different variations within each class. When audio files are turned into spectrograms they can be augmented with shifting and transposing to continually duplicate your input data. This helps compensate for the initial smaller dataset.
- Before scaling or normalizing your mel-spectrograms, run your CNN model using the original graphs. Our model comparison has shown that scaled spectrograms stagnate any increase in accuracy or decrease in overall loss. Future series of model comparisons will help solidify this recommendation.