Modeling Guitar Pickup Audio

Aron Connors

Introduction: Electric guitar pickups

The electric guitar is an innovation on a centuries old instrument that occurred within the past 100 years. The addition of electronic devices called *pickups* transform the string vibrations into an electric signal which can be amplified (and also manipulated with a variety of effects).

Two main pickup form factors have prevailed, the tall skinny single coil pickup and the short wide humbucker pickup. The difference in shape created different levels of induction in the pickups, which affects the tone (specifically, the resonant peak of the instrument). While the tonal difference between the two pickups might not be obvious to most, an experienced guitar player certainly can tell the difference. This project aims to learn the features of each tone using various neural network architectures.

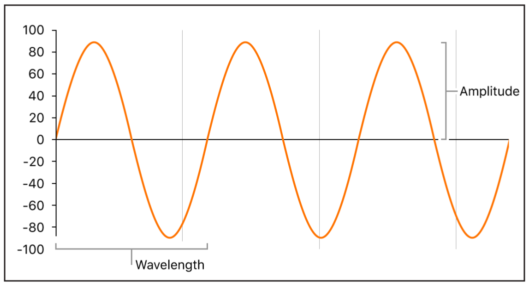
Single Coil Pickups on a Fender Stratocaster	Humbucker Pickups on a Gibson Les Paul

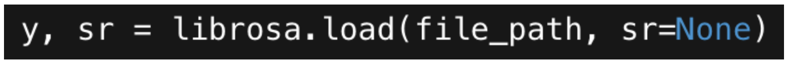
Dataset: Direct input audio from each pickup class

I searched for a suitable dataset for this ML task and came across the IDMT-SMT Audio-Effects database of audio effects for electric guitar and bass. The dataset was very large, and was a challenge to download and transfer between my laptop and erdos. I found the command rsync worked best because I was able to resume without restarting when there were errors.
Since the dataset was mostly effects (pedals) on guitars, I had to trim the dataset down to only direct input (clean sound, no effects) recorded on guitars with either humbucker or single coil pickups. I was able to successfully group a perfectly balanced 313 recordings of each class, each using a variety of pickup settings (to ensure our dataset covers a wide range of the sounds these pickups can possibly make). The dataset file structure is a master monoGuitarDataset directory and two subdirectories (humbucker and singleCoil) which house the respective .wav files that constitute the dataset.

Exploratory Data Analysis: Digital signal processing

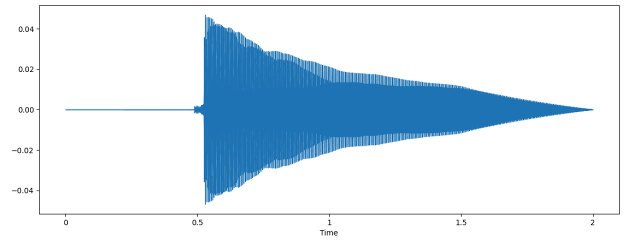
This project employs various digital signal processing concepts, domain knowledge that I was not previously familiar with. The .wav files we're working with contain waveform data. A waveform is a series of sound wave amplitude values that can be plotted over time (as seen in an example to the right). The number of times we sample from this soundwave is called the sample rate and is measured in Hz. A CD typically has 44,100 Hz, nice headphones can have 48,000 Hz. Sample rate is the audio processing equivalent of a framerate. A few python libraries make this information very easily extractable from .wav files, specifically for this project I used librosa. The small code example to the left shows how you can extract the amplitude values (y), the sample rate (sr) from a specific file_path. The sr=None argument preserves the original sample rate of the file we're using, but this can be set to a specific value as well.

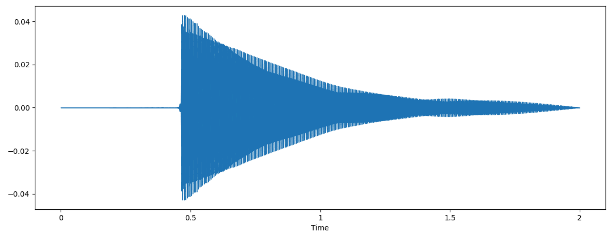




```
y, sr = librosa.load(file_path, sr=None)
```

Using these tools, lets visually explore the differences between the two pickup classes. Below is the note G played on humbucker (left) and single coil (right) pickups.





Although the plots look quite similar, we can see some subtle differences. Lets see what tactics we can use to learn the subtle nuances of the two pickup classes.

Preprocessing

Before jumping into classification tasks, there are a few preprocessing steps required. Firstly, we aim to normalize the amplitude values between +/-1 during preprocessing. Librosa does this by default. In addition to normalization, we will also transform our data to Mel-Frequency Cepstral Coefficients (MFCCs). MFCCs essentially separate our single amplitude value into a feature set that represents how humans hear the sounds. The number of features is tunable, I found that 40 features was a good representation of the data.

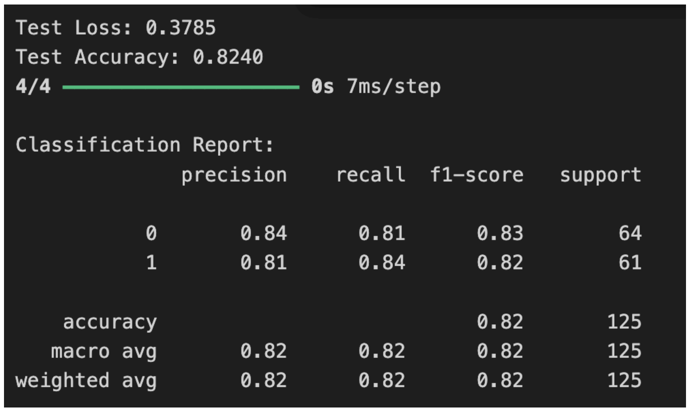
I set up a loop to process the training data into a dataframe with MFCCs for each file for each pickup. A train test split was performed with 20% of the data to be used as a test set.

Classification: Attempt 1 (MFCCs)

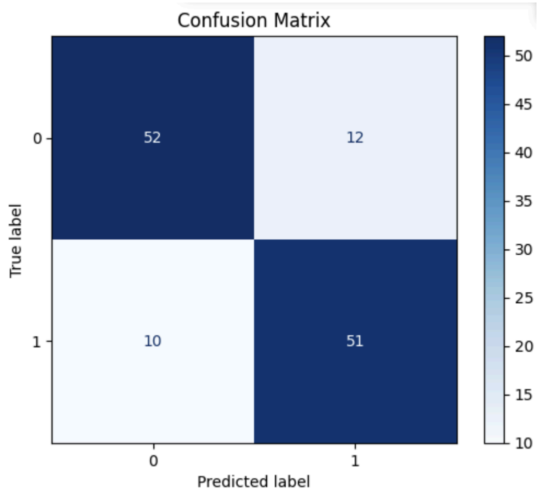
Using our preprocessed data, we can set up a neural network classifier to learn the subtle differences between pickup tones. The input layer of this neural network is size (40,), as this is the number of cepstral coefficients we chose. This architecture is easily adjustable to a different number of coefficients by adjusting the size of the input layer. The 3 fully connected layers of the neural network sequentially decrease in size. After each layer, ReLU activation is utilized and a dropout of 0.2 is applied to avoid overfitting.

As this is a binary classification task, there are 2 classes in the output layer ([‘humbucker’, ‘singleCoil’]). The code utilizes a softmax function for 2 one-hot-encoded class values, which allows for easy reconfiguration to analyze multiple pickup types or guitar audio effects.

After training, the model certainly learned some differences in pickup tone. The model was able to achieve 82.4% accuracy.



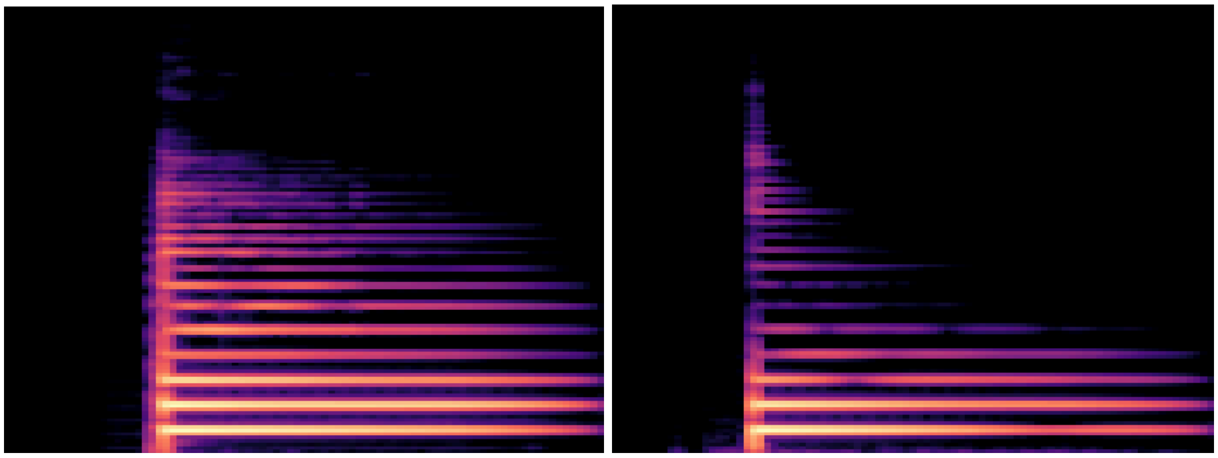
```
Test Loss: 0.3785
Test Accuracy: 0.8240
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step

Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.81      0.83        64
           1       0.81      0.84      0.82        61

    accuracy                           0.82       125
   macro avg       0.82      0.82      0.82       125
weighted avg       0.82      0.82      0.82       125
```



I attempted this classification task on my own to see how human results compare to the neural network we trained. I was able to successfully guess 75% of pickup tones, so 82% accuracy from the machine learning model is not particularly impressive. My suspicion as to why this was happening was because the MFCCs were not capturing the subtle differences in tone the way a more detailed dataset would. In order to account more closely for these subtle nuances in tone, we will upgrade to using Mel Spectrograms.

Classification: Attempt 2 (Spectrograms)

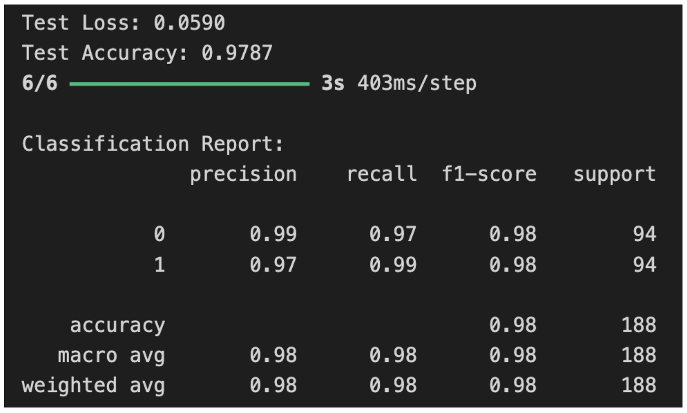
Mel Spectrograms are a visual representation of sound. Pixel width represents time, pixel height represents frequency, and the 3 RGB values represent the amplitude or power of the signal at each point. Below are the same 2 notes G represented as spectrograms. It is clear that this data contains more information about subtle tonal differences by comparing them (humbucker on left, single coil on right).



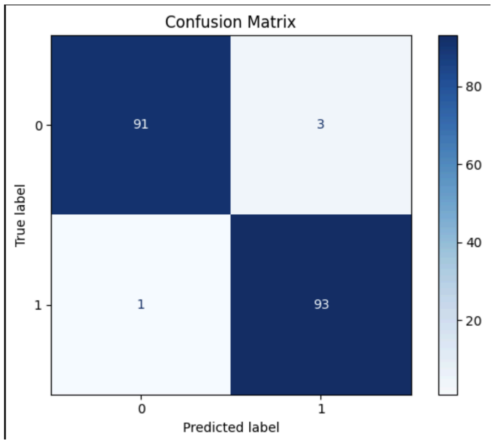
We perform a similar preprocessing step as before where the spectrogram images are created from the .wav files. The dataset is created in the same format as the original directory structure.

With this type of data, we can use a convolutional neural network to classify pickup type. The size of our CNN input is (224, 224, 3), the same size as our spectrogram images. The images are passed through 4 convolutional layers. Each layer uses a 3x3 kernel, max pooling of 2x2, and ReLU activation. The filters increase from 32 to 128. The feature map is flattened and passed to a fully connected layer of 1024 neurons. The same output softmax function is used for this binary classification task for one-hot-encoded class values.

The results of this network were far better than using MFCCs, achieving nearly 98% accuracy and making very few classification mistakes.



```
Test Loss: 0.0590
Test Accuracy: 0.9787
6/6 ━━━━━━━━━━━━━━━━━━━━ 3s 403ms/step

Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.97      0.98        94
           1       0.97      0.99      0.98        94

    accuracy                           0.98       188
   macro avg       0.98      0.98      0.98       188
weighted avg       0.98      0.98      0.98       188
```

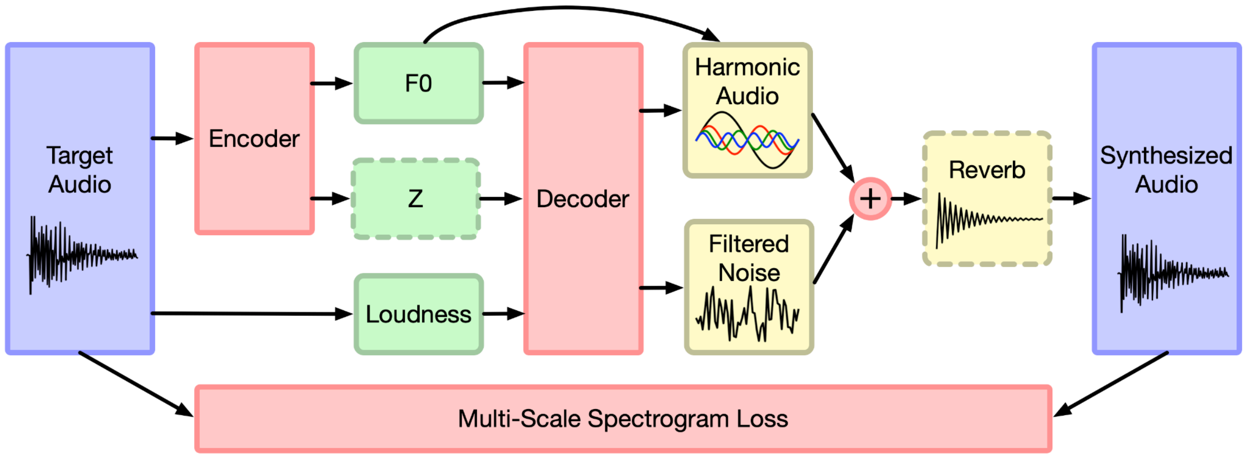


Judging from these results, we can confidently say that Mel Spectrograms are far better at detecting the subtle nuances between guitar pickup tones than MFCCs.

DDSP: Differentiable Digital Signal Processing

After successfully building a classifier for the pickup tones, I began a 3rd (ambitious) task to try and synthesize specific pickup tones using Google's Differentiable Digital Signal Processing (DDSP) library. I was unable to get this to work by the due date, but I certainly put a lot of effort into making it work and learned a lot despite negative results.

DDSP works by taking into consideration loudness, pitch, and filter noise in training an autoencoder on specific audio. DDSP has reshaped old physics equations (DSP) to enable them to be differentiable and usable to train neural networks. Google has published colab demonstrations on how to train the tone transfer autoencoder and synthesize new tones using DDSP. Below is a diagram of the training process of a DDSP autoencoder.



The main learning curve I faced at this part of the project was getting the necessary libraries installed. Google colab uses a newer version of python, so it is not possible to run these demonstrations in colab anymore. I was forced to use erdos and install the packages manually. I learned that the DDSP installation would also install other libraries like tensorflow and numpy at newer versions than intended to be used with DDSP, so this process needed to be manual as well. CUDA also posed challenges in this way. I was able to get code to run on a few separate equations, but each change to a package version would break many other parts of the code, so this was a very slow process that required multiple restarts. For this reason, even after adjusting the environment for days straight, I was not able to perform a tone transfer before the due date. It was a pleasant experience to learn how audio synthesis works, and I certainly learned a lot about managing packages and how to install required dependencies in a stable fashion.

Final Thoughts

I think this project can be used to aid in replicating the guitar tones of another artist, and laid foundations for synthesizing their tone. Not being able to perform a tone transfer in time was disappointing, but I'm proud of the classification techniques and results I achieved. I think I will revisit a tone transfer in the next few weeks. I underestimated the amount of digital signal processing domain knowledge necessary to perform this sort of task from scratch. This project reinforced both the machine learning concepts from class as well as DSP concepts.

References

The dataset I used to obtain the direct input audio for the pickup classes:
https://www.idmt.fraunhofer.de/en/publications/datasets/audio_effects.html

An educational youtube video detailing the difference between the pickups that I was trying to learn:
https://www.youtube.com/watch?si=WxvqLAPJYFOteDFm&v=5JMsRX6SGlw&feature=youtu.be

The neural network and convolutional neural network github pages from Sundeep Rangan:
https://github.com/sdrangan/introml/tree/master/unit09_neural
https://github.com/sdrangan/introml/tree/master/unit10_cnn

Krish Naik's introductory video to digital signal processing:
https://youtu.be/mHPpCXqQd7Y?si=HTNvRrvywQSZYJ0N

Google's colab demos for a DDSP tone transfer:
https://colab.research.google.com/github/magenta/ddsp/blob/master/ddsp/colab/demos/timbre_transfer.ipynb
https://colab.research.google.com/github/magenta/ddsp/blob/main/ddsp/colab/demos/train_autoencoder.ipynb

The DDSP paper published by google magenta:
https://openreview.net/forum?id=B1x1ma4tDr