Automated Classification of Drum Sounds for Modern Music Production

Nathan Heck
Department of Electrical and Computer Engineering
University of Florida
Gainesville, FL
nathan.heck@ufl.edu

***Abstract*— Similar to voice recognition, musical instrument timbre recognition has been a challenge in machine learning research for several decades. For these audio signals, we can use the Mel Frequency Cepstrum (MFC) to extract important features about their frequency content, which is a large part of what makes up an instrument's timbre. There have been several attempts to train neural networks to classify monophonic recordings of Western classical instruments. In this project I will combine the techniques from a few of the most successful papers I read to get a neural network to distinguish short audio recordings of various percussion instruments that might be used in contemporary electronic music.**

I. Introduction

Now more than ever, it is important for a computer to be able to recognize patterns in the voice of different musical instruments because of the way contemporary music is being made. Instead of recording instruments with a microphone, songwriters and music producers are increasingly moving towards using prerecorded samples of these instruments. They might have a lower budget studio environment to work in, so it would be difficult to recreate the professional quality samples that are so widely available in the online marketplace. They might want to use exotic instruments or sound effects in their music which they do not have physical access to in their studio. These developments in the music production industry have spawned an adjacent marketplace where producers are exchanging curated collections of sounds they recorded in the past that they find inspirational in some way.

Many music professionals amass a huge library of sounds that they downloaded from the internet or recorded themselves to reuse in other productions. The problem with these large sample collections is that it takes so long to parse through them to decide which sounds to use and which to ignore. Navigating this library is usually done by scrolling down a long list or clicking into different folders, and this can be very tedious when you just want to focus on creating new compositions. If the audio files were not named properly, then this process takes much longer. We need a machine learning solution that can recognize which instruments are present in each file and label them accordingly.

II. Data Set

To reduce the complexity of my learning model, I will only try to learn a few categories that represent some pieces of a drum set rather than trying to categorize everything in a whole band. I read other papers that tried to categorize an entire orchestra and the training took a very long time. For my initial training and validation data I collected many free sample packs online that were made publicly available by either individual producers or audio software companies. Each sample pack is made up of several .wav type files which are one shot recordings of various percussion instruments. After about a week of foraging on the internet for free samples, I amassed a collection of about 3,000 files, but after some experimentation I realized it would be very difficult to classify all of them. Originally, I set up my model to learn 7 classes, but after some initial testing I realized it was going to be tough to distinguish between some of those classes. I decided to just focus on the 3 most important parts of the drum kit: the kick drum, snare drum, and hi-hat cymbals. With that in mind, I whittled down the training data set to 918 samples. For the testing data, I put together another collection of 61 samples that the model had never seen before.

There is a nearly infinite number of ways you can strike the instruments on the drum kit to bring out wildly different sounds. I had to control what percussion techniques were represented in the dataset so that I would be able to reliably classify between the different instruments. Across all the instruments, I only included recordings where the drummer performs a single strike. For the snare drum, this can be done either in the center of the head, or on the side where the stick will simultaneously hit the rim of the drum in a maneuver that is called a "rimshot." I chose to exclude recordings where the drummer only hits the stick on the rim without touching the head, and where the drummer was playing with brushes. For the kick drum, I did not really have to filter out any recordings because this instrument has less expressive capability compared to the others. All the recordings were made with the drummer pressing down their foot on the kick pedal.

For the hi-hat, these recordings could be expressed in 3 categories: closed, open, and pedal. The hi-hat instrument is actually 2 cymbals on a stand, where the drummer's foot controls how far apart the two cymbals are from each other. A closed hi-hat sound is when the two cymbals are pressed up against each other, and the drummer strikes the top one with a stick. An open hi-hat sound is the same thing, except the drummer releases their foot so the cymbals ring out a little more. A pedal hi-hat sound is when the drummer does nothing with their hands but uses their foot to forcefully close the cymbals against each other. Some of the variables I did not control were the microphone placement used in the recordings, the microphone type, the acoustic properties of the studio room, the material and construction of the instruments, the intensity of each strike from the drummer, and any post processing effects used by the recording engineer.

I also decided to include samples recorded from several electronic drum machines. These generally fall into 2 categories: either they use synthesizers to generate sound from scratch that approximates the sound of a real drum, or they start with an audio recording of a drum and apply heavy post processing. While there is also a near infinite number of sounds these drum machines could produce, there are a few

that have been used so much in popular music throughout the years that they have become a part of our shared musical vocabulary and it would make sense to include them in my training set.

III. Related Work

When I started doing research on the topic of musical instrument timbre recognition, one of the first papers I came across was using a Convolutional Neural Network (CNN) to classify audio samples from a dataset recorded by the London Philharmonic Orchestra. The authors were using Mel Frequency Cepstrum Coefficients (MFCCs) extracted from each audio file as features to train the neural network. I had some experience working with each of these techniques separately, so I thought this might be a good path to follow. Their dataset was similar to mine, since each audio file only had one instrument playing one note, but they had many more classes represented in their data. [3]

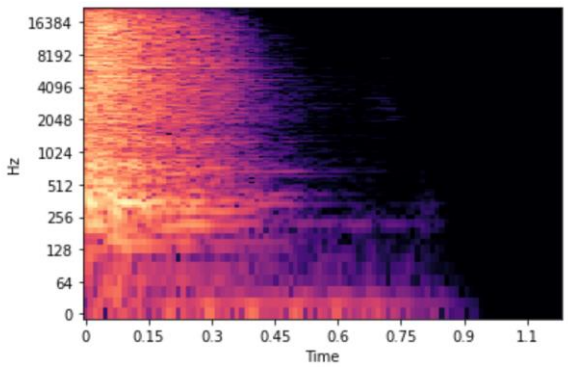
I thought that I might have to do something differently in my model, since my data is all inharmonic percussion instruments. So I found this other paper proposing a new kind of audio feature extraction called nontonal MFCC. It was an interesting read, but I could not find any other mentions of this technique in the literature, and I was a little confused about how to implement the math that they were discussing in actual Python code. [2] However, I did come across a dissertation by K.D. Martin that gave me a much fuller understanding of some of the fundamental problems in sound-source recognition. His work informed a lot of my decisions when it came to pruning my dataset, and it also gave me the idea to mix supervised and unsupervised learning in my model design. [1]

IV. Method 1: CNN Trained On MFCC Features

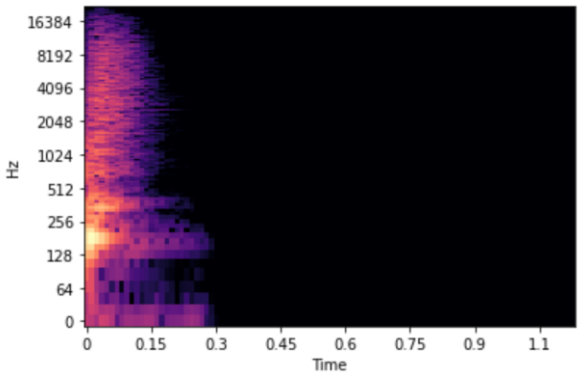
All the code I wrote for this experiment is in Python—contained in a set of Jupyter notebooks for training and testing the different models. The first model I implemented is a CNN trained on MFCC features. I used the Keras library for the neural network design, and the Librosa library for the audio processing and feature extraction. All the audio files were loaded in at a sample rate of 44,100 Hz, and I forced them all to be in mono. When recording a drum set, it is common practice to use multiple microphones to capture one sound source but depending on the song you might want to have the drums spread out more or less across the stereo field. Since we are dealing with only the core elements of the drum kit in this experiment, we want them to sound like they are right in the center of the mix, so if there are any stereo audio files in the dataset I will sum them to mono. To ensure each audio file produced the same number of features, I kept the length of the waveform at 50,000 samples. Any less, and it would be padded with zeros, any more and the ending would be trimmed off. I then calculated the MFCCs for each waveform using 40 Mel-frequency bands, an FFT window size of 2048 samples, and a hop length of 512 samples. This results in a feature matrix with size 918 x 40 x 98 that we can use as input to the neural network.

Looking at Figure 1 on the right, we can see how our 3 classes—hi-hat, snare drum, and kick drum—appear in the frequency domain. I am using the Mel spectrogram for these visualizations rather than the MFCCs because those are not as human readable. You can see that the kick drum at the bottom has the most low frequency information and the least high frequency information. The snare drum has most of its frequency content in the mid-range. The hi-hat at the top of Figure 1 has a lot of high and mid-range frequency information, and the sound also rings out a lot longer over time when compared to the kick and the snare drum.

Hi-Hat



Snare Drum



Kick Drum

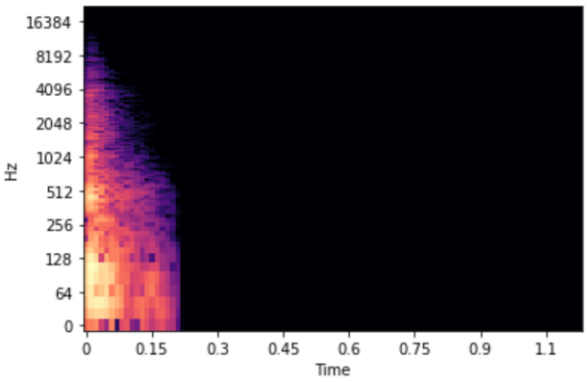


Figure 1: Mel spectrogram of data from 3 classes

For the CNN architecture, I used a similar design to Mahanta et al. except I had to make a couple adjustments to the dimensions at the input and output layers since I am using a different number of features and only 3 classes in my data. The last layer uses Softmax to provide a confidence percentage for each class, so the CNN outputs a matrix sized N x 3 where N is the number of audio files. The highest number in each row determines to which class that audio file belongs. There are 3 hidden layers in between and they are all using the ReLU activation function. Figure 2 below will illustrate the entire structure of this neural network. To train the CNN for Model 1, I used the Adam optimizer with a learning rate of 0.0001, sparse categorical entropy for my loss function, a batch size of 32, and 30 epochs. [3]

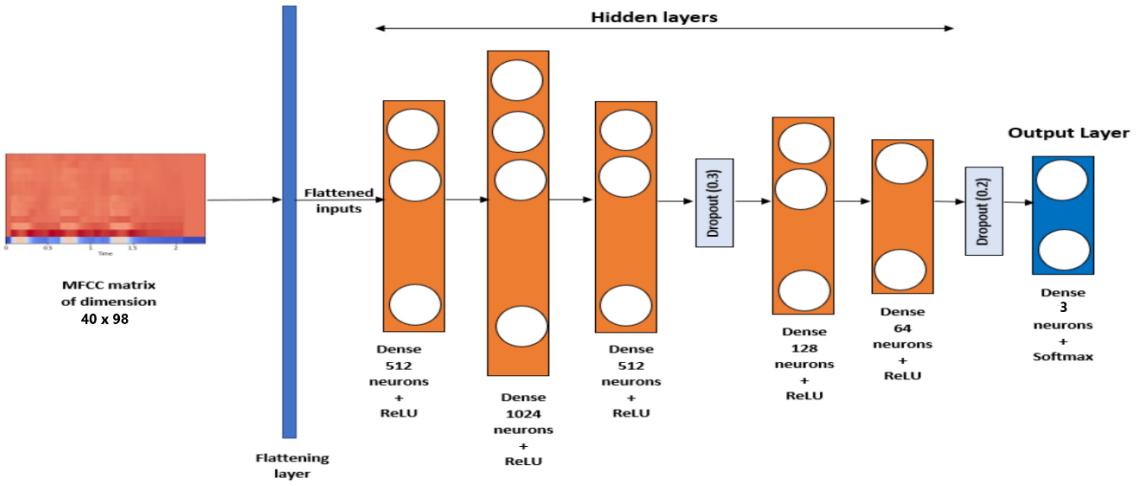


Figure 2: Full Model Architecture Using Method 1

V. Method 2: Mixing Supervised & Unsupervised

The second model I chose to implement uses a similar CNN classifier from Method 1, but instead of the MFCCs, it uses features generated by a Convolutional Autoencoder (CAE). All the audio files were loaded in at a sample rate of 22,050 Hz, and I forced them all to be in mono. To ensure each audio file produced the same number of features, I kept the length of the waveform at 32,600 samples. While the signal is still in the time domain, we can use the Librosa library to extract the percussive parts of the signal and the harmonic parts. Then using only the percussive part of the waveform, I calculated the Mel spectrogram for each waveform with 128 Mel-frequency bands, an FFT window size of 512 samples, and a hop length of 256 samples.

At this point, I am representing each input data point with a 128x128 spectrogram image. Now we pass these images through an autoencoder to select the most relevant features we will need to train our CNN classifier. The design for my CAE comes from a blog post on the Keras website on building autoencoders. I had to modify the input dimensions to accommodate my larger images because they originally wrote that code to use with the MNIST dataset. I trained the CAE using the Adam optimizer with a learning rate of 0.0001, mean squared error for the loss function, batch size of 128, and 10 epochs. Then I encoded all the images using 2048 latent variables. In Figure 3, I took two of these latent variables as the x and y axis and plotted all the training data on a scatter plot so you can see how the different classes are spread across the latent space. Each class cluster is represented by a different color. [6]

Now we have a feature matrix with size 918 x 16 x 16 x 8 that we can use as the input to the neural network, which has the same architecture as the one from Method 1. The training process for the Method 2 CNN was the same as for Method 1: an Adam optimizer with a learning rate of 0.0001, sparse categorical entropy for my loss function, a batch size of 32, and 30 epochs.

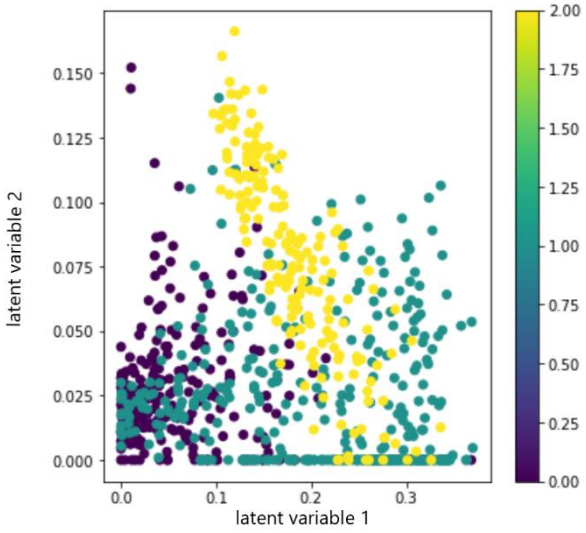


Figure 3: Plotting training data in the Method 2 latent space

VI. Model Evaluation

Let us now take a look at the results of these experiments to see how well the different models performed. On the next page I will show some figures to visualize my results, but here I will give a brief analysis. Method 1 had a 93.4% accuracy rating on the testing dataset, and the only class it had trouble identifying was the hi-hat. About 25% of the time it saw a hi-hat recording, it classified it as a snare drum instead. Method 2 also had 93.4% accuracy on the test dataset. It made some of the same mistakes as Method 1, and a couple more. I did a few trials with Method 2 by commenting out the preprocessing step where I separate the percussive part out of the original waveform for each data point. When I just use the percussive waveform, the model seems to have a little more confusion between the snare and kick drum, but when I use the full waveform that confusion only lies between the snare drum and the hi-hat. I tried playing around with the learning rate for both models too, but if I made it any higher than 0.0001, the performance would drop on the testing dataset.

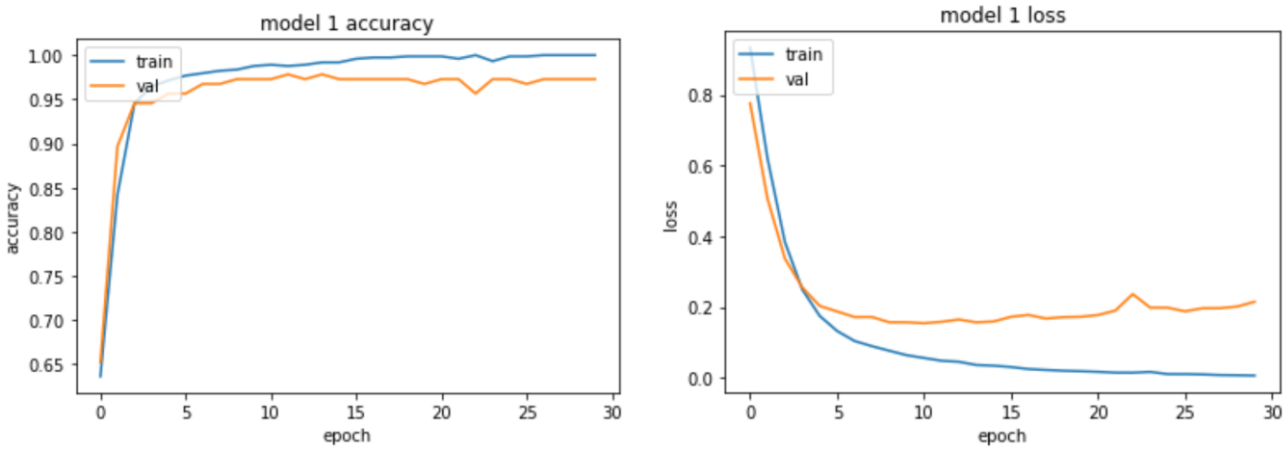


Figure 4: Accuracy and loss plots for the CNN classifier from Model 1

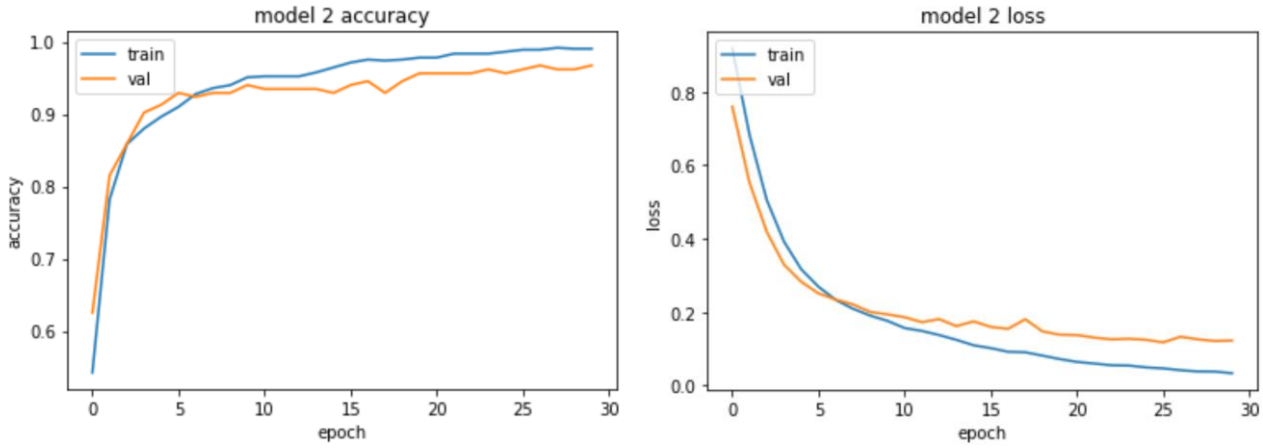


Figure 5: Accuracy and loss plots for the CNN classifier from Model 2

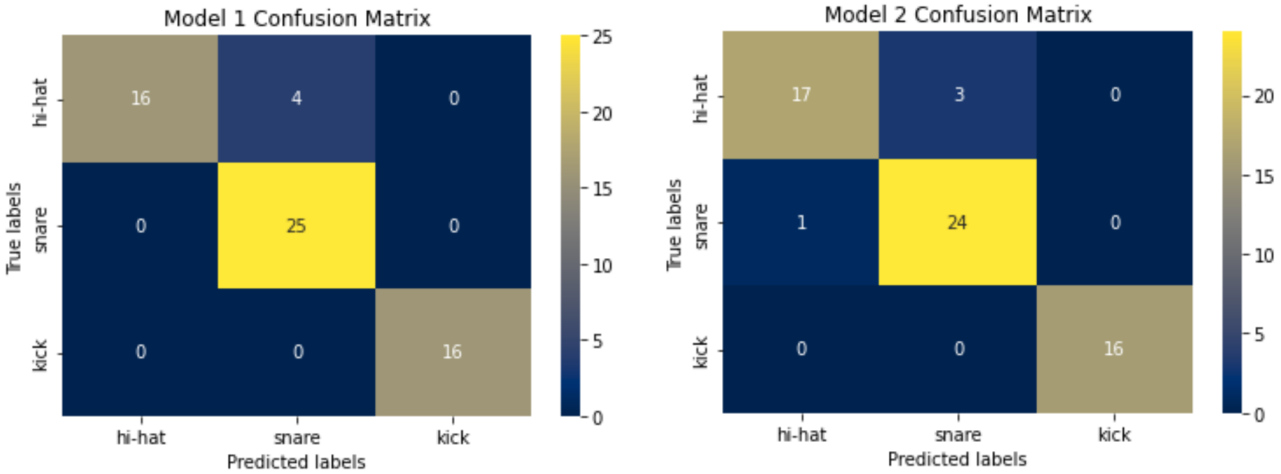


Figure 6: Comparing confusion matrices for two CNN classifiers trained on different types of features

VII. Conclusion

After seeing these results, we can conclude that a CNN will reliably distinguish between 3 classes musical instruments when it is trained on spectral features from audio recordings of those instruments. In Method 1 I showed that the combination of MFCC features and CNN deep learning, which has been proven in the literature to have some success with classifying orchestra instruments, is also useful for classifying parts of a drum kit. In Method 2 I demonstrated that we can instead use unsupervised learning techniques like autoencoders to extract features from audio files. The CAE I designed for this experiment was trained to extract visual features from a spectrogram representation of the audio input. This experimentation with different feature extraction methods did not suggest that either Method 1 or Method 2 was more effective since both had similar performance on the testing dataset.

In the future, I want to try using a Variational Autoencoder (VAE) to do the unsupervised feature extraction rather than a convolutional one. Since a VAE will actually try to learn the probability distributions of the features it selects, it might give the neural network better predictive power on a wider variety of sounds. I was a little disappointed that I did not see any performance improvement when using the Librosa function of separating the input waveform into a percussive part and a harmonic part. This might have had more of an effect if I had kept the recordings in the time domain, rather than taking spectrograms of all of them. Maybe if I investigate more on how to extract time domain features from the audio, it might give better separation between some classes in the latent space. Then I could also use other neural network architectures like Recurrent Neural Networks (RNN), which could give better audio classification performance.

VIII. References

[1] K.D. Martin, Sound-Source Recognition: A Theory and Computational Model, May 1999.

[2] Y. Wu, Q. Wang, R. Liu, “Music Instrument Classification using Nontonal MFCC,” in Advances in Engineering Research, vol. 130, 2017

[3] S. Mahanta, A. Khilji, P. Pakray, “Deep Neural Network for Musical Instrument Recognition Using MFCCs,” in Computation and Systems, vol. 25, pp. 351–360, Nov. 2021

[4] D. Fragoulis, C. Papaodysseus, M. Exarhos, G. Roussopoulos, T. Panagopoulos, D. Kamarotos, “Automated Classification of Piano-Guitar Notes”, in IEEE Transactions on Audio, Speech, and Language Processing, vol. 14, no. 3, May 2006

[5] P. Cosi, G. De Poli, and P. Prandoni, “Timbre characterization with Mel-Cepstrum and neural nets,” in Proc. International Computer Music Conference, pp. 42–45, Aarhus, Denmark, 1994.

[6] F. Chollet, “Building Autoencoders in Keras,” *The Keras Blog*, 14-May-2016.