– Independent Work Report Spring 2023 –

The Development of a Convolutional Neural Network for the Classification of Drum Sounds

Louis Larsen '24
Advisor: Brian Kernighan

Abstract

Music is a universal language, and people can easily decipher between many different types of instruments. While recent developments in machine learning have allowed computers to distinguish between multiple types of instruments, trying to classify instruments with similar timbral features remains a significant problem. Therefore, this paper implements a Convolutional Neural Network (CNN) to classify drum set sub-instruments, like kick drums, snare drums, tom drums, and cymbals. These instruments were chosen because although they are different in character, they share many timbral qualities. Overall, the CNN scored a testing accuracy of around 94% when distinguishing these four sounds, proving that a CNN is a viable algorithm for classifying instruments with similar timbral features.

Keywords: Music Instrument Classification, Convolutional Neural Network, Deep Learning, MEL Spectrogram, Music Information Retrieval

1. Introduction

Music is an art form that allows people to express their creativity and emotions using various sounds produced by voices, musical instruments, and technology. Once a captivating piece of music falls upon a listener's ears, it holds a unique ability to stimulate their feelings and heighten their state of mind. Hearing these sounds over and over can cause people to start to remember the characteristics and qualities of each instrument, which not only allows people to learn how each instrument sounds but also how to classify them quickly. [2,3]

Due to recent advances in machine learning, computers can now almost distinguish between multiple different groups of instruments successfully. In most of these studies, classification is performed in a specific way: using a training dataset of audio segments consisting of a few seconds of each instrument with distinguishable timbral features or characteristic sound qualities, like drums, piano, bass, guitar, violin, and so forth, the program extracts important features from that data and then trains a machine learning model upon those features to see which ones are most closely associated with each sound. [3] Many machine learning algorithms could be used during this step, but some of the most common ones include K-means, Random Forest, and Naïve Bayes classifiers. [2,3,5,7]

However, the traditional model described above struggles with identifying instruments with similar timbral features. [3] For example, distinguishing between a trumpet and a saxophone may be relatively easy due to their different timbres, but differentiating between two saxophones with slightly different timbres could be problematic. Additionally, the traditional machine learning algorithms listed above are not theoretically the strongest algorithms one could use for machine learning. In the last decade, there has been enormous progress in deep learning techniques with algorithms that crush the accuracy of previously dominant ideas. One of these deep learning techniques is called a convolutional neural network, or a CNN for short. One could think of a CNN as a collection of filters that are applied to an image to extract different features. These filters first learn simple patterns, like edges and corners, then piece those patterns together to understand more complex features. After the CNN extracts all the information it can get from the filters, it attempts to predict the image. By showing a CNN a lot of images, it gets better and better at recognizing the differences between them.

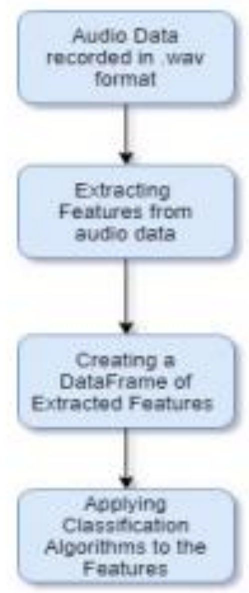
Audio Data recorded in .wav format
Extracting Features from audio data
Creating a DataFrame of Extracted Features
Applying Classification Algorithms to the Features

Figure 1: Flowchart for understanding the workflow step-by-step. Adapted from "Drum Instrument Classification Using Machine Learning" by Chhabra et al, 2020, IEEE. [2]

The main focus for the rest of this paper is to build a CNN which can classify multiple different drum kit sub-instruments: a kick drum, a snare drum, a tom, and a cymbal. The reason for choosing these instruments is that while they are distinct, they share similar timbral qualities. A kick drum, also known as a bass drum, is a drum that is played with a foot pedal, which produces a deep, low-frequency sound. By contrast, a snare drum is a drum that produces a sharp, cracking sound when hit with a drumstick due to the snares stretched across the bottom of the drum. Next, a tom drum, also known as a tom-tom, is a drum that comes in various sizes and shares aspects between a snare drum and a kick drum. Like a snare drum, it is usually hit with a drumstick, but unlike a snare, it does not contain snares on the bottom head. The resulting sound is more like a kick drum: deeper, fuller, and more resonant. However, a tom's sound is typically higher-pitched and more melodic than a kick drum. Finally, a cymbal is a percussion instrument that produces a ringing, metallic sound when hit with a drumstick.

Regarding my experimentation, an initial input is a dataset consisting of these four sub-instruments in a WAV audio format. All the files in the dataset are first converted into MEL spectrograms to make the input suitable for the CNN. To understand what a MEL spectrogram is, think of a graph with two axes – the horizontal axis shows time, and the vertical axis shows different sound frequencies. A MEL spectrogram divides these frequencies into smaller, more specific sections that our ears can distinguish. Now that all the dataset is turned into MEL spectrograms, a training dataset comprising around 80% of the original dataset is then used to train the CNN to predict the drum sub-instruments. The final 20% is then used for testing.

The rest of the paper will proceed as follows: after describing related work which goes more in-depth about some experiments already done in the realm of instrument classification, I will touch more in-depth about my approach, my implementation, and the results from the experimentation as shown through the accuracy of the testing rounds.

2. Related Work

Research in general-purpose musical instrument classification goes back to the mid-1990s, stemming from the fields of speech classification and music transcription. [6] Early studies on this topic dealt with trying to classify instruments of wildly different timbres. For instance, one of the earliest studies dealt with trying to classify piano, marimba, guitar, and accordion. [7] Another early study tried to classify eight instruments: bagpipe, clarinet, flute, harpsichord, organ, piano, trombone, and violin. [6] The algorithms and input variables used during this time varied quite widely, with algorithms like Support Vector Machines and K-Means, and input types ranging from MEL frequency coefficients to MPEG-7 characteristics to even raw audio. [5,6,7,8]

Before continuing, it might be necessary to stress how each of the algorithms the other researchers used works. A Support Vector Machine, otherwise known as an SVM, tries to find the best boundary to separate two different groups of data points by maximizing the distance between the boundary and the closest data points on either side. These closest data points are called "support vectors" because they are the most important in determining where the boundary should be placed. In a K-Means system, a random number of "cluster centers" from the data points are initially chosen. Next, each data point in the dataset is assigned to the cluster with the closest cluster center based on the distance between the data point and the cluster center. Once every data point is assigned to a cluster, the mean value of each cluster is calculated, and the cluster center is moved to that new mean. This process is repeated until there is no change in the assignment of the data points to the clusters or a maximum number of iterations is reached.

However, as far as instruments with similar musical characteristics are concerned, much less work has been done. Tan et al. (2023) attempted to train a program to classify violin and viola pieces. After gathering a dataset of Violin and Viola pieces from throughout history, they used a "statistical feature approach," which involved running the violin and viola pieces through a short time Fourier Transformation and an amplitude to dB-scaled spectrogram to train their classifiers. Subsequently, they applied machine learning algorithms like random forest, support vector machines, and k-means to perform the classification task. In a Random Forest System, many decision trees are created, and then the results are combined to come to a final conclusion about the data. By the end, their model achieved an accuracy of 94% when classifying all the violin and viola pieces they collected and using a combination of the three algorithms listed above. [3]

Regarding classification with only percussion and drum sub-instruments, Herrera et al. (2002) used a set of 20 features to classify drum sub-instruments. Using a dataset of 634 sounds from different commercial CDs, they used multiple different descriptors, including attack, decay, relative energy, and MEL frequency descriptors, to attempt to classify the drum sounds. As far as the algorithms they used, they tested a variety of algorithms, including K-Means and a variant of such, Canonical Discriminant Analysis, which is a method that could be used to identify linear combinations of variables that can best discriminate between multiple groups, and a Decision Tree algorithm. They found that the best algorithm to classify the drum sounds was a variant of a K-Means algorithm that classified different objects using a type of entropy measure, which had an accuracy rate of 97% when performing a basic-level classification between different drum sounds. [5]

Another study by Chhabra et al. (2022) used a set of 3 features to classify four categories of drum instruments. They used a dataset of 160 self-recorded sounds and multiple features like Zero-Crossing Rates, or the number of times the sound wave crosses the x-axis, Spectral Centroids, which correlates to the brightness of the sound, and the Root Mean Square Energy, which is the "square root of the summation of distance from the Zero Reference Line" reflecting the energy found in the sound wave. Regarding the algorithms used, they compared the results of K-Means, Random Forest, and Naïve Bayes Classifiers. A Naïve Bayes Classifier calculates the probability that an input belongs to a particular class by analyzing the occurrence of features in the training data, assuming that they are independent. Their results found that the Naïve Bayes Classifier performed the best from the algorithms listed above, classifying the drum sounds with around 97% accuracy. [2]

Finally, over the past decade, more studies have started to use CNNs for instrument classification. For instance, Han et al. (2017) trained a CNN to classify between 11 instruments found in the IRMAS dataset. After training the CNN on roughly 10 thousand MEL frequency spectrograms generated from the audio excerpts, they found that neural networks were more robust than conventional methods, achieving a 60% F-micro score, which was 20% better than the conventional methods described above. [4] Blaszke and Kostek (2022) also trained sets of CNNs to detect between 4 different types of instruments. Using the Slakh dataset, which contained 2100 polyphonic audio tracks, meaning that different instruments were playing simultaneously, aligned with MIDI files and separate instrument tracks, they trained their CNN to have an accuracy of around 72%. While their accuracy may have been lower than traditional models, they achieved an AUC ROC of approximately 0.96 and an F1 score of about 0.93, which dominate over other more traditional methods, which receive AUC ROC scores of around 0.91 and F1 scores of 0.64. [1]

3. Approach

Unlike conventional methods, my program plans to use a Convolutional Network to classify between different sub-groups of percussion instruments. While all the papers listed above concluded that the conventional methods produced accurate results by using many different features, they are not theoretically the strongest algorithms one could use to classify instruments. In the last decade, there has been enormous progress in deep learning techniques with algorithms that crush the accuracy of previously dominant ideas. One of these deep learning techniques is a CNN. So, theoretically, a CNN should produce at least comparable results to the highly optimized conventional algorithms described in the previous section. Additionally, the studies that used a CNN to classify instruments used a range of different instruments, many of which have easily distinct timbres. Therefore, the question arises, "To what extent can a CNN assist in classifying instruments with similar timbral structures, such as drum set sub-instruments?"

I feel drum set sub-instruments are appropriate for this task because while each drum has its unique sound and character, they share similar timbral qualities. For example, a snare and tom drum can have similar timbral qualities depending on their size and tuning. Both drums can produce a dry, focused sound when struck in the center of the drumhead and a more open and resonant sound when struck closer to the edge. Similarly, a kick drum and a tom drum can have similar qualities due to their deep, booming sound and low-frequency range.

Additionally, the question might arise that since a CNN is designed for classifying images, how could a CNN be used to process audio files? To solve this problem, before training the CNN, all the drum samples must be converted into a visual format, specifically MEL spectrograms, and then running the CNN on the generated MEL spectrograms. Compared to other methods of visualizing audio, a MEL spectrogram is the most helpful because it allows us to see which frequencies are the most important for understanding a piece of audio. We can see which frequencies are loudest at each particular point of the sample and use that data to predict the drum sound.

To create the CNN to classify the drum sounds, wav files consisting of one-shot, or single-sound, drum samples must be found. These drum samples would consist of kick drums, snare drums, tom drums, and overhead cymbals, or crash and ride cymbals.

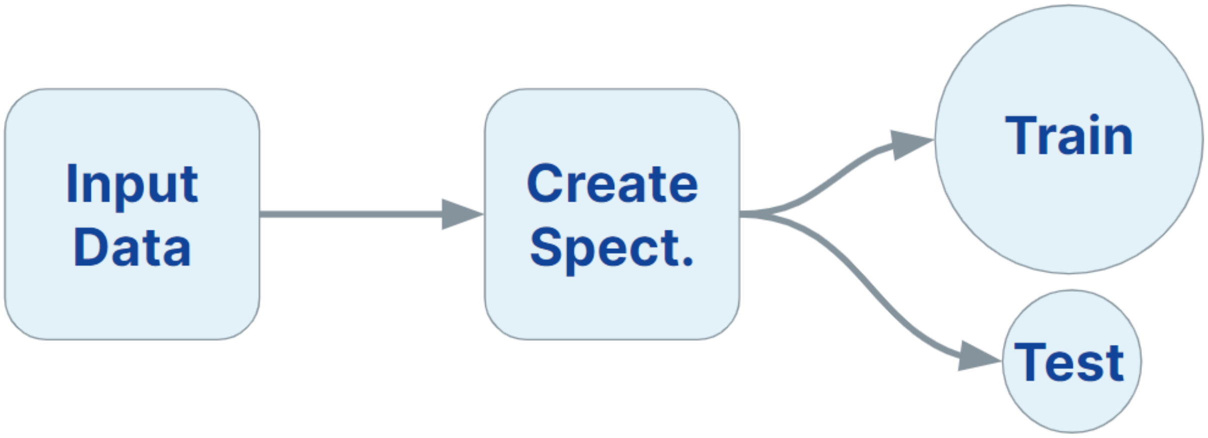

Figure 2: Flowchart for the structure of the program to create the MEL Spectrograms, showing an 80/20 split between training and testing data

Then, all the raw audio files would be turned into MEL Spectrograms. When the program starts, all the audio files are stored in folders corresponding to which drum sub-instrument they are. I propose a system consisting of 4 folders: kick, snare, cymbals, and toms. The program goes through each of these folders and turns the audio file into a MEL Spectrogram. These spectrograms are stored in either a training folder with an 80% probability or a testing folder with a 20% probability. Additionally, the code to create the spectrograms outputs multiple lists telling the CNN where to find the images and the original classification of the audio files.

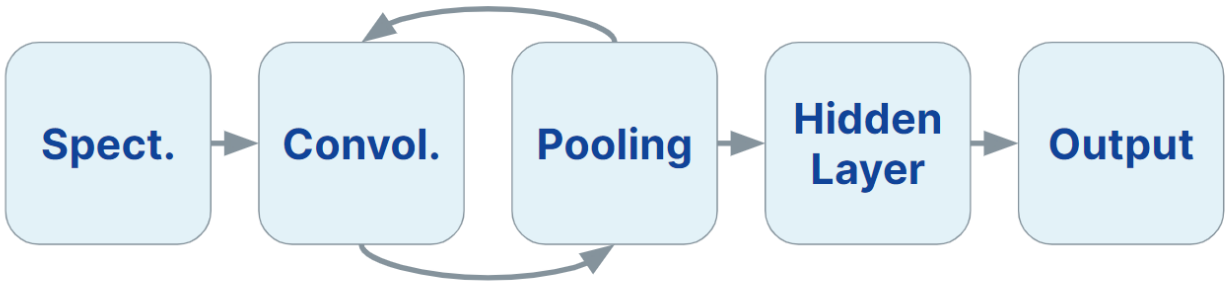

Figure 3: Flowchart for the structure of the CNN classifier used to predict the drum set sub-instruments

After the spectrograms are created, my spectrograms are run through a CNN classifier. To build a CNN, several factors must be considered. For example, the spectrogram images need to be normalized and converted into grayscale to simplify the data and ensure all features have similar scales and distributions, helping to improve the model's performance. As far as the architecture of the model is concerned, two convolutional and pooling layers and two fully connected layers are used to reduce the output to the four defined classes of drum instruments. Two phases of experimentation will be used: the first outlining the CNN and the second involving testing the CNN. During the training phase, the CNN is trained on a training dataset by adjusting the weights of the neurons in the network using optimization algorithms and backpropagation. In the testing phase of the experiment, the CNN is then evaluated on a separate testing dataset to determine its accuracy and any potential overfitting.

Finally, to evaluate the function, I will look at how accurate the CNN was overall at classifying each of the drum sounds. Using the MEL Spectrograms combined with the power of the CNN, I aim to achieve an accuracy of above 90%.

4. Implementation

4.1 – Dataset collection

My database consists of 1440 drum samples in a wav file format divided into four categories: kick, snare, cymbals, and toms. The dataset was compiled from many different accessible sources I found online. For instance, one of the datasets was Drum Kit Sound Samples found on Kaggle, containing 160 high-quality audio samples of kick drums, snare drums, tom drums, and cymbals. [9] Another one of the datasets was SampleRadar's 1000 free drum samples, containing kicks, snares, cymbals, toms, and more. [10] There are both live-recorded and simulated sounds included in the dataset as well. After skimming all the datasets for specifically kick drums, snare drums, tom drums, and cymbals, I was initially left with more snare and kick sounds than any other drum. This was a problem since neural networks work best when there are equal numbers of each sound. So, to maximize the efficiency and accuracy of the CNN, I ensured that each sound had an equal number of samples attached to it: 360.

Table 1 provides a visual representation of the database.

Table 1

Samples	**Number**	**Percentage**
Kick	**360**	**25%**
Snare	**360**	**25%**
Cymbals	**360**	**25%**
Toms	**360**	**25%**
Total	**1440**	**100%**

Table 1: A visual layout of the compiled database

4.2 – Design of Spectrogram Generator

I used the SciPy and the Librosa Python libraries to create the spectrograms. SciPy is an open-source scientific library built on top of the NumPy library. I chose SciPy because of its wide range of features for audio processing, including Fourier Transforms and Spectral Analysis. Secondly, Librosa is a Python package specifically designed for analyzing and processing audio with comprehensive audio analysis features. In this report, Librosa is the main package that generates the spectrograms. Therefore, SciPy handles the mathematical signal-processing tasks while Librosa is used for the more specialized music analysis tasks.

After reading the drum sample, if the program detects that the audio clip is in stereo, the average of the left and right tracks is taken and put into one mono track. Now that all the audio files are in mono, the program takes the audio samples and the sample rate and computes a power spectrum using a short-time Fourier Transform. The window size, or the length of each segment to analyze, used to create the spectrogram is 1024 samples, while the hop size, or the number of samples shared between each analysis, is 512. This power spectrum shows how much power, or energy, is contained within each signal frequency component.

A MEL spectrogram is then generated from the power spectrum by taking a logarithm of the magnitudes from the power spectrum to convert the magnitudes into decibels and applying a MEL filter bank to the decibel power spectrum matrix. Regarding the number of MEL frequency bins, 128 was used because it is a reasonable setting that preserves the timbral characteristics of the audio. This spectrogram is either saved in a training folder with an 80% probability or a testing folder with a 20% probability. The labels for each spectrogram, which in this case is the folder where the original drum sample came from, are stored as well. The spectrograms generated here will be used later on to train the CNN.

4.3 – Implementation of the CNN

I used the PyTorch python library to build the CNN for several reasons. PyTorch allows for easy debugging and more flexibility, allowing for a quick change to the network if bugs were to develop. Secondly, PyTorch is easy-to-use with an excellent user-friendly interface. Finally, PyTorch is built with GPU acceleration in mind, which greatly helped to speed up the calculations used in the network.

After the spectrogram image is output by the spectrogram generator, it is not yet in a format that machine learning algorithms can understand. As a result, the images are first loaded into an image loader, where they are turned into grayscale, converted into tensors, and then normalized. I convert the spectrogram images into grayscale to simplify the model. A grayscale image only has one channel, whereas a colored image has three channels for red, green, and blue. Consequently, a grayscale image has fewer input features, making it easier for the machine learning algorithm to train on the data. Therefore, removing color can remove unnecessary information and noise from the image, improving the model's accuracy.

For my experiments, I use a model with two 5x5 convolution layers, each with a stride of one and padding of two. The first layer has 32 filters, and the second has 64. There are also two 2x2 max-pooling layers, which pull the most critical information from the previous convolution layer forward. After both convolution layers are complete, the convolution results are run through a flattening layer and two feedforward hidden neural networks, which allow the CNN to output a prediction for the classification of the sound. The first feedforward layer flattens the convolution model and reduces the number of channels to 512. The second reduces the number of channels again to 4, representing the total number of classes the program can classify. Finally, after each round, a ReLU activation function is applied to every layer except for the flattening layer and the second feedforward layer. ReLU, which stands for Rectified Linear Unit, is a simple function. For any given input value, if the input is positive, the output is equal to the input. However, if it is negative, the output is zero. So, ReLU chops off any negative values and leaves the positive values as they are. This may seem like a simple idea, but it is very effective in helping neural networks to learn. When the network encounters negative values, ReLU sets them to zero, which means the network is not wasting any computation on them. This can speed up the training process and make the network more efficient.

A graphical way of illustrating the proposed CNN model is shown in Table 2.

Table 2

Layer	Description
Layer 1	**5 x 5** 2D convolutional layer with 32 filters, a stride of 1, and a padding of 2
	ReLU activation function
	2 x 2 Max Pooling
Layer 2	**5 x 5** 2D convolutional layer with 64 filters, a stride of 1, and a padding of 2
	ReLU activation function
	2 x 2 Max Pooling
Layer 3	Flattening layer
Layer 4	Feedforward Neural Network reducing the number of outputs to 512
	ReLU activation function
Layer 5	Feedforward Neural Network reducing the number of outputs to 4

Table 2: A layout of the proposed structure of the CNN

My program implements a batch size of 256 and employs Stochastic Gradient Descent, otherwise known as SGD, to optimize each of the weights for the program to make the best possible prediction. 256 was chosen as the batch size because it allows for both computational efficiency and stability during training. Additionally, my program calculates the loss from each round by using an L2 regularizer. This regularizer helps to prevent overfitting, or fitting the training data too closely, leading to poor performance on new, unseen data, in the training data. The L2 regularizer works by adding a penalty term, proportional to the sum of the squared magnitudes of the weights, to the loss function during training, which encourages the model to have smaller weights.

To sum up, the program looks at 256 training examples at once, computes the error or loss based on the model's predictions for those examples, and adjusts the model's parameters to minimize the error. The training lasts for 100 epochs, or rounds, with a learning rate of 0.005 to ensure the training loss is as low as possible. So, throughout all the training rounds, the CNN gets to study each of the images in the training folder and starts to become more accurate in its prediction.

Finally, to evaluate the CNN, I noted how many testing cases it got correct compared to the total number of testing cases. Therefore, I computed the accuracy of the CNN as defined below:

$$A = \frac{tp}{tp + tn}$$

where tp stands for true positives, or the number of times the CNN guessed the drum sound correctly and tn stands for true negatives, or the number of times the CNN guessed the drum sound incorrectly.

5. Results

5.1 – MEL Spectrograms

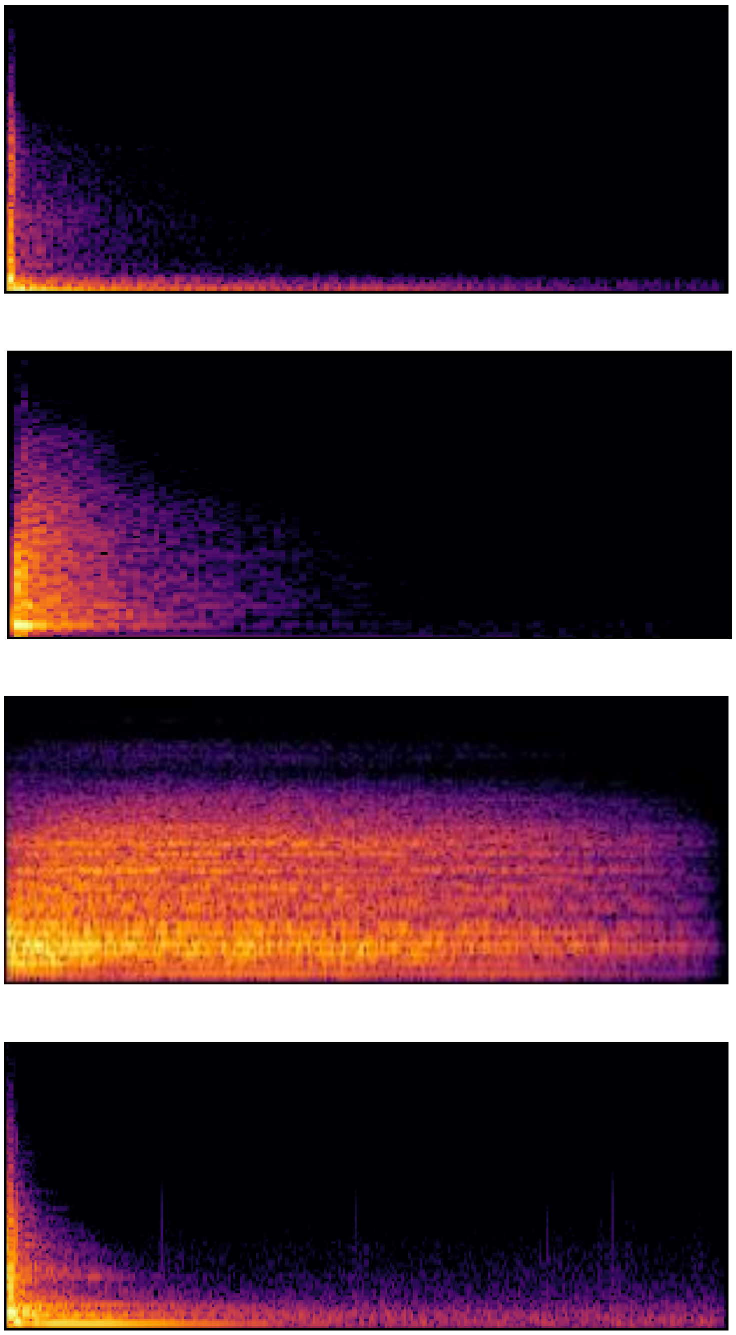

Figure 4: Shows example generated spectrograms for each of the four instrument types. The first shows a kick drum, the second shows a snare drum, the third shows a cymbal, and the fourth shows a tom drum

The previous page demonstrates examples of different MEL spectrograms generated by my program. Spectrograms like these were fed into the application's training and testing phases, turning them into grayscale images to determine the classification of each drum sound.

5.2 – CNN

Below is a graph that shows the training loss over the training period of the algorithm. As one can see, the CNN has learned to classify each of the four different drum sounds due to the low training loss by the end of the training period.

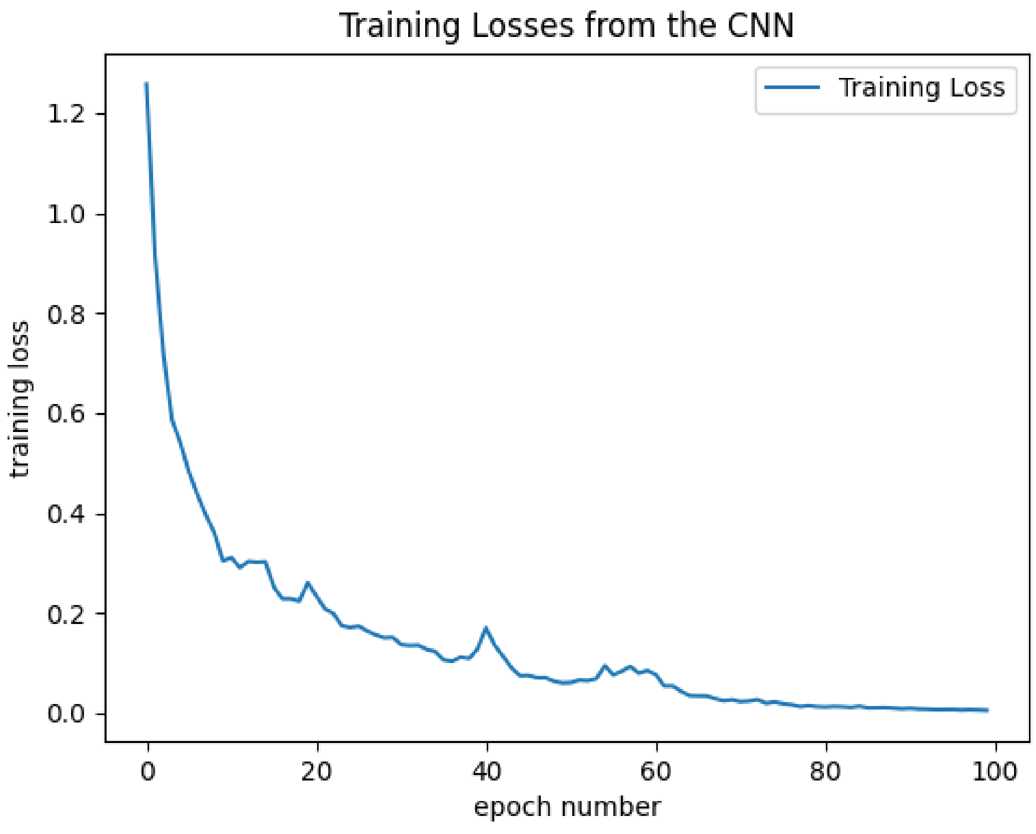

Figure 5: Shows Training Loss per epoch number, with a 1/x relationship.

Next, here are the learned weights of the first layer of the CNN to determine which drum sound each of the spectrograms were.

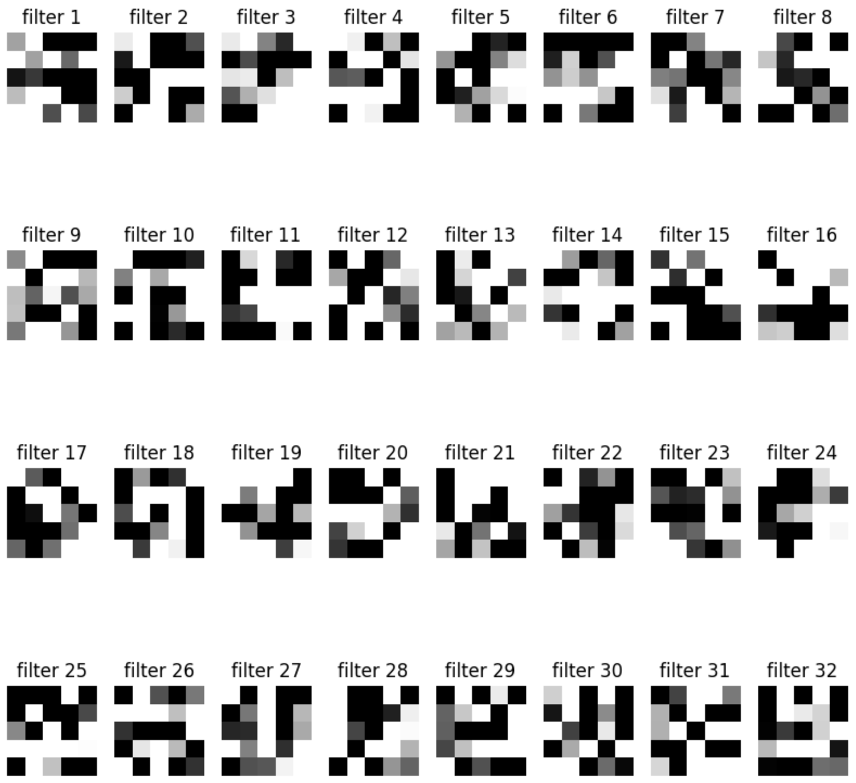

Figure 6: The weights assigned to each pixel of every filter in the first convolutional layer

Finally, my CNN received a testing accuracy of roughly 94.05%, correctly identifying 269 out of the 286 samples in the testing dataset. This accuracy is very high for this task and outperforms my expectations of at least 90% accuracy.

6. Limitations

While a testing accuracy of around 94% exceeded my expectations, it does slightly worse when compared to the highly optimized conventional methods, some of which received accuracy scores as high as 97%. Future research must be done to understand how to improve the accuracy even further, but a great place to start would be to understand the spectrograms that my CNN miscategorized. Two such cases are shown below:

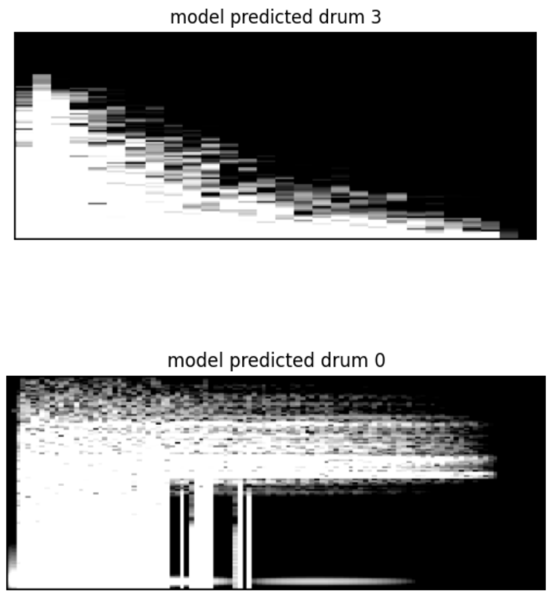

Figure 5: Shows edge cases of spectrogram failures, such as misclassification or corruption.

The first case depicts a genuine misclassification by the program. The drum in the first picture was a kick drum. However, the CNN predicted that the drum was a tom drum. In this case, the kick drum was slightly higher in frequency and had a more resonant sound, much like a typical tom drum. In this way, this kick drum was difficult to distinguish from a tom drum. This illustrates one of the challenges of using machine learning for audio classification - sometimes, subtle differences in frequency or other characteristics can lead to misclassifications. To improve the model's accuracy, it may be necessary to provide more training data that covers a wider range of sounds or to adjust the parameters of the CNN to capture each sound's unique features better.

The second case, however, does not depict an issue with the CNN but rather the spectrogram generator. It seems that some form of corruption occurred when the audio file was read into the program. Large streaks of data seem to be present where they should not be, leading to a misclassification of the instrument. These streaks of data appear to have misled the CNN during its classification task, and consequently, the CNN misclassified the sound, even though it had seemingly been trained well on clean data. To solve this issue, ensuring that the input data provided to a machine learning model is high quality and free from any significant errors or anomalies that could impact its performance is important. This can involve careful pre-processing and cleaning of the data and ongoing monitoring and validation of the output produced by the model.

Therefore, the results of my experiments have shown great promise in accurately classifying different types of drum sounds. The high accuracy of 94% indicates that my CNN can learn the defining features of each drum sound. However, there is always room for improvement. I feel that with additional refinement and training, my algorithm could easily match or even exceed the accuracy of other, more conventional models. Overall, I am confident in the potential of my approach and excited to continue exploring its capabilities.

7. Conclusions and Future Work

So, drum roll, please, a CNN has proven to be a viable approach when classifying between instruments with similar timbral qualities. My approach of taking drum samples, converting them into MEL spectrograms, and training the CNN on those images has produced an accuracy rate of 94%, which was better than I expected. However, there is still room for improvement. With additional refinements and training, the CNN will be able to distinguish between different drum sounds with an even higher accuracy.

In the long term, this project, when combined with future work in AI music generation, has the potential to open up many different creative opportunities for musicians. By accurately classifying and transcribing drum sounds, musicians can more easily replicate and build on existing rhythms and sounds and sample specific drum sounds in already existing songs. This technology could also be used to generate new drum sounds specifically tailored to the musician's wishes.

Nevertheless, as for now, future work to improve this program would be to work on ways to improve the accuracy even higher, such as providing more training data that covers a wider range of sounds, adjusting some of the parameters of the CNN, or solving the Spectrogram corruption issue. Going forward, adding further classifications, such as high, middle, and low toms, might also be interesting to push the idea of classifying instruments with similar timbral features to its limit. Finally, I could incorporate more drum set sub-instruments, such as a Hi-Hat, which are two cymbals and a pedal mounted on a metal stand.

8. Acknowledgements

As a final note, I would like to thank Dr. Brian Kernigham, Dr. Sierra Eckbert, and the rest of my classmates in my COS IW for helping provide suggestions and feedback on my research as the project evolved.

References

[1] M. Blaszke and B. Kostek, “Musical Instrument Identification Using Deep Learning Approach,” Sensors, vol. 22, no. 8, p. 3033, Apr. 2022, doi: https://doi.org/10.3390/s22083033.

[2] A. Chhabra, A. Veer Singh, R. Srivastava and V. Mittal, "Drum Instrument Classification Using Machine Learning," 2020 2nd International Conference on Advances in Computing, Communication Control and Networking (ICACCCN), Greater Noida, India, 2020, pp. 217-221, doi: 10.1109/ICACCCN51052.2020.9362963.

[3] Chong Tin Tan, K. Wong, V. Monn, Kiki Maulana Adhinugraha, and D. Taniar, “Is it Violin or Viola? Classifying the Instruments’ Music Pieces using Descriptive Statistics,” ACM Transactions on Multimedia Computing, Communications, and Applications, vol. 19, no. 2s, pp. 1–22, Sep. 2022, doi: https://doi.org/10.1145/3563218.

[4] Y. Han, J. Kim, and K. Lee, “Deep Convolutional Neural Networks for Predominant Instrument Recognition in Polyphonic Music,” IEEE/ACM Transactions on Audio, Speech, and Language Processing, vol. 25, no. 1, pp. 208–221, Jan. 2017, doi: https://doi.org/10.1109/taslp.2016.2632307.

[5] P. Herrera, Alexandre Yeterian, and Fabien Gouyon, “Automatic Classification of Drum Sounds: A Comparison of Feature Selection Methods and Classification Techniques,” Lecture Notes in Computer Science, pp. 69–80, Sep. 2002, doi: https://doi.org/10.1007/3-540-45722-4_8.

[6] Marques J., Moreno P.J. A Study of Musical Instrument Classification Using Gaussian Mixture Models and Support Vector Machines. Camb. Res. Lab. Tech. Rep. Ser. CRL. 1999;4:143.

[7] I. Kaminsky and A. Materka, "Automatic source identification of monophonic musical instrument sounds," Proceedings of ICNN'95 - International Conference on Neural Networks, Perth, WA, Australia, 1995, pp. 189-194 vol.1, doi: 10.1109/ICNN.1995.488091.

[8] G. Agostini, M. Longari and E. Pollastri, "Musical instrument timbres classification with spectral features," 2001 IEEE Fourth Workshop on Multimedia Signal Processing (Cat. No.01TH8564), Cannes, France, 2001, pp. 97-102, doi: 10.1109/MMSP.2001.962718.

[9] "Drum Kit Sound Samples," www.kaggle.com. https://www.kaggle.com/datasets/anubhavchhabra/drum-kit-sound-samples.

[10] T. M. team, "SampleRadar: 1,000 free drum samples," MusicRadar, Aug. 17, 2020. https://www.musicradar.com/news/drums/1000-free-drum-samples.