Beatbox to Analog Drum Translation Using a Convolutional Neural Network

Joe Bumpus
University of Rochester
jbumpus2@ur.rochester.edu

Noah Miller
University of Rochester
nmill15@ece.rochester.edu

Julia Weinstock
University of Rochester
jweins10@ece.rochester.edu

ABSTRACT

Beatboxing is a vocal performance technique in which a vocalist imitates percussive and melodic instrument sounds. This project aims to translate the recording of a percussive beatboxing performance into a corresponding audio track, matching the sound and timing of the performance using preloaded drum samples. This is achieved using a convolutional neural network (CNN) in Python. The mel spectrograms of thousands of labeled beatboxing sounds were used to train the CNN, making this audio recognition task actually one of image recognition. Testing the model on unique recordings, the model was able to achieve over 95% accuracy in percussive element recognition. With this, in tandem with determining the onset locations within the input recording, a very accurate translated audio file is able to be created through triggering analog drum samples.

1. INTRODUCTION

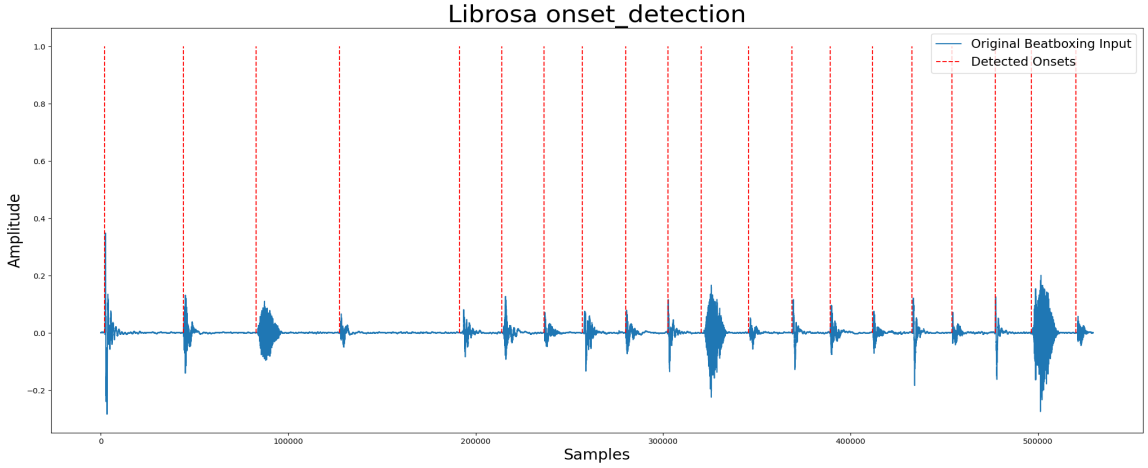
Beatboxing can be anything from a personal pastime to a live musical artform; a staple of the casual, impromptu musical jam or the driving rhythmic force of an a cappella performance. From a technical standpoint, beatboxing is an increasingly popular contemporary singing style where the vocalist imitates percussive drum and pitched musical instrument sounds. In this project, we focus specifically on the percussive elements of beatboxing and attempt to turn this purely vocal art into a digital sample controller using a convolutional neural network.

A convolutional neural network (CNN) is a type of deep learning model most commonly used for image recognition tasks as it is particularly effective in pattern recognition. Here, a recording of human beatboxing is parsed and separated into individual drum sounds. The mel spectrogram of each is then taken, giving us a representative image for each sound. With a CNN model trained on the mel spectrograms of thousands of labeled beatboxing drum sounds, we can predict the label of each individual recorded sound. This, in combination with detecting the onsets of each sound in the recording, allows us to compose an audio track by triggering corresponding analog drum samples in the same timing as the original recording.

This concept could prove extremely useful for musicians and producers with limited time or resources, facilitating quick form musical idea creation in a way that is far more natural, intuitive, and inspiring than manual drum programming in typical digital audio workstations.

In the following sections, we will describe the dataset used in training this model, as well as the data preparation necessary for training and predicting validation/test data.. We will then discuss the architecture of our specific model before detailing exactly how onsets are detected and the input recording is divided, necessary for creating the final audio file translation. Finally, we will discuss the results of this methodology, as well as some future considerations in implementing this as a possible product.

Fig. 1. Librosa Onset Detection



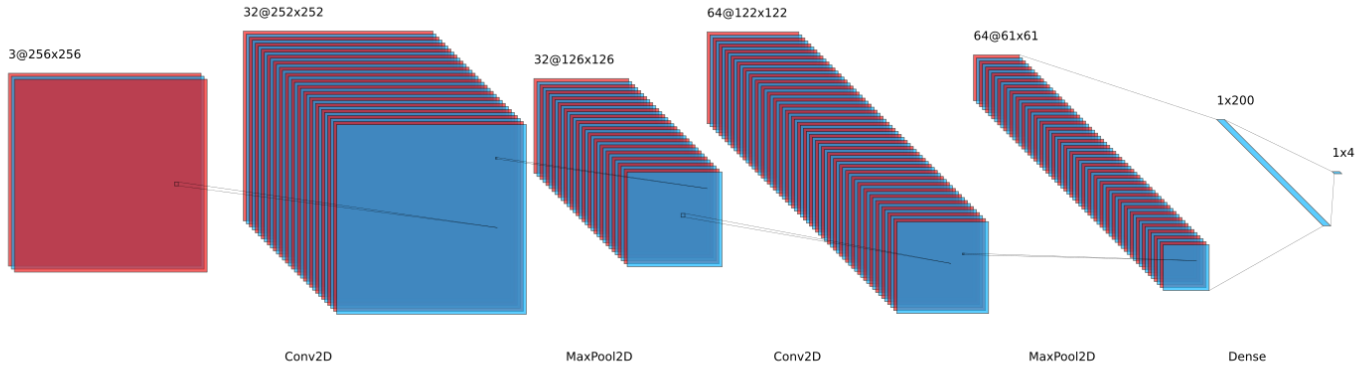


Fig. 2. Convolutional Neural Network Model Architecture

2. METHOD

2.1 Dataset

This project uses the Amateur Vocal Percussionist (AVP) dataset for testing, training, and validation data [1]. The AVP dataset consists of data from 28 participants with four labels: kick drum, snare drum, closed hi-hat, and open hi-hat. Each participant folder contains 5 .wav files. Four files are each of the labels vocalized 30-40 times, and the fifth file is an improvisation track. Overall, there are 4873 total sound events. Along with each .wav file is an accompanying .csv file which contains the onset location in seconds and label of each vocal percussion sound event.

2.2 Data Preparation

2.2.1 Data Extraction

Before feeding the samples into the network, each individual vocal percussion event needs to be extracted and saved as a mel spectrogram (see section 2.2.3). The samples of one event are taken from the onset in the .csv to the onset of the following sound. It is then buffered in the beginning by 1000 samples to center the event and prevent smearing in the spectrogram. Finally, each sample is set to a fixed length of 11025 samples for consistent spectrograms.

2.2.2 Data Augmentation

To increase the robustness of the network, the clean samples are augmented with samples that have added noise, pitch shift, time shift, and gain change. After a clean sample is extracted and added to the dataset, augmentations are applied with a random probability, and the resulting augmented audio is also added as an independent entry to the dataset. In this way, the number of samples is doubled from 4873 to 9746 samples.

2.2.3 Mel Spectrograms

After extracting each individual sound event, the audio files are then transformed into mel spectrograms. Mel spectrograms are used over spectrograms because they provide more rich information for audio data. As seen in Fig. 3, the mel spectrograms for a kick and a high hat closed sound are more visually differentiable than the spectrogram.

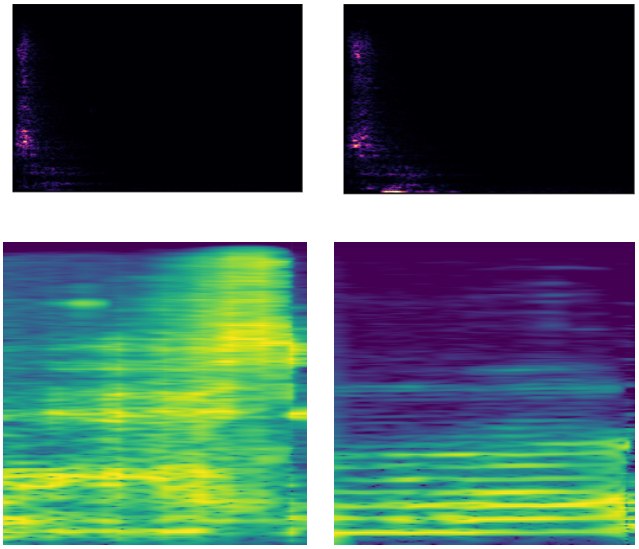


Fig. 3. Comparison between a spectrogram of a high hat closed (top left), spectrogram of a kick (top right), mel spectrogram of a high hat closed (bottom left), and a mels spectrogram of a kick (bottom right)

2.3 Model Architecture

As a result of the translation from audio signal to a visual representation, the percussion identification task can be treated as more conventional image recognition. Now that we are focusing on image recognition, we can use the more specialized convolutional neural network framework, as it is well suited to computer vision problems. CNNs use convolutional filters to extract patterns and features from image data, allowing fully connected classifiers to differentiate between those feature maps. Specifically, our model will use a multi-layer convolutional design, introducing further

specificity within the feature maps, taking advantage of pooling layers to reduce the spatial size of each layer.

Our model takes in the RGB spectrogram of dimension 3x256x256. The first convolutional layer translates our spectrogram into 32 feature maps of dimensions 252x252, followed by a max pooling layer to reduce the spatial size by half. The second convolutional layer doubles the filters, with dimensions of 122x122, followed once again by a max pooling layer, resulting in dimensions of 64x61x61 [2]. This data is flattened, before being passed through two fully connected layers, resulting in a 4-class classifier. No activation is used, as training utilizes cross entropy loss, which has a softmax function built-in. Training also uses the AdamW stochastic optimization method, with a learning rate value of 0.0001. The CNN model was built using the Torchvision package library, allowing for seamless integration, and removing any issues with redundancy by utilizing the Torch DataLoader class. Fig. 4 shows the PyTorch summary output of our neural network model.

```
==========================================================================================
Layer (type:depth-idx)                   Input Shape               Output Shape
==========================================================================================
Network                                  [1, 3, 256, 256]          [1, 4]
├─Sequential: 1-1                        [1, 3, 256, 256]          [1, 4]
│    └─Conv2d: 2-1                       [1, 3, 256, 256]          [1, 32, 252, 252]
│    └─MaxPool2d: 2-2                    [1, 32, 252, 252]         [1, 32, 126, 126]
│    └─ReLU: 2-3                         [1, 32, 126, 126]         [1, 32, 126, 126]
│    └─Conv2d: 2-4                       [1, 32, 126, 126]         [1, 64, 122, 122]
│    └─Dropout2d: 2-5                    [1, 64, 122, 122]         [1, 64, 122, 122]
│    └─MaxPool2d: 2-6                    [1, 64, 122, 122]         [1, 64, 61, 61]
│    └─ReLU: 2-7                         [1, 64, 61, 61]           [1, 64, 61, 61]
│    └─Flatten: 2-8                      [1, 64, 61, 61]           [1, 238144]
│    └─Linear: 2-9                       [1, 238144]               [1, 200]
│    └─ReLU: 2-10                        [1, 200]                  [1, 200]
│    └─Dropout2d: 2-11                   [1, 200]                  [1, 200]
│    └─Linear: 2-12                      [1, 200]                  [1, 4]
==========================================================================================
Total params: 47,683,500
Trainable params: 47,683,500
Non-trainable params: 0
Total mult-adds (M): 965.08
==========================================================================================
Input size (MB): 0.79
Forward/backward pass size (MB): 23.88
Params size (MB): 190.73
Estimated Total Size (MB): 215.40
==========================================================================================
```

Fig. 4. Torch Summary of CNN Model

2.4 Input Onset Detection & Sample Triggering

A user may input a beatboxing recording of their own for translation. Their recording may consist of any number of the four sound classes (‘hi-hat open’, ‘hi-hat closed’, ‘kick’, and ‘snare’, as previously mentioned) in any order. This recording is then broken down by onset to create a set of individual sounds with which the model can predict the respective labels. The onsets are detected using Librosa’s onset_detection function, returning an array containing the sample index position of each onset in the audio file, as shown in Fig. 1 using an example recording containing 20 total utterances.

Just as with the training data, each individual drum sound is shortened to a uniform length starting with 1000 samples before the onset. The mel spectrogram of each of these sections is then taken and passed into the model for prediction. The model returns the predicted label of each sound. An array of zeros is created equal in length to the input recording. Based on the predicted label and array of onset locations, the corresponding analog drum sample to each onset is appended to the array of zeros.

3. RESULTS

Training our model on 10 epochs, we were able to achieve a peak of 97% training accuracy (as seen for the particular run reflected in Fig. 5) and a peak validation accuracy of 94%. At 10 epochs, based on the learning curve, accuracy is likely still improving. This shows promise for expanded training, which may include more epochs or more diverse training, and being able to achieve even more accurate classification.

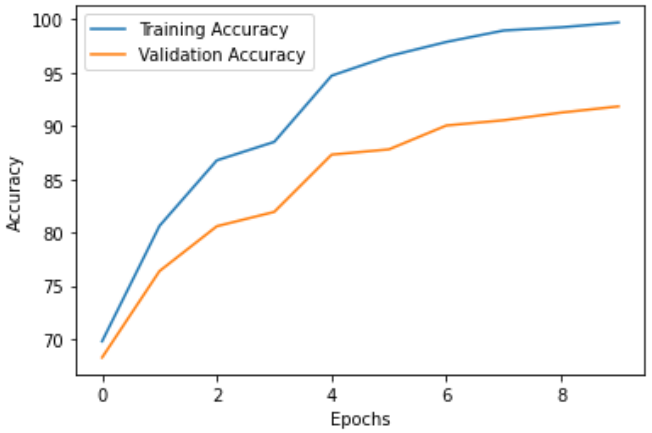


Fig. 5. Learning curves for 10 epochs.

Completing the example shown above Fig. 1, the model achieved 95% accuracy in classifying the input beatbox recording, meaning all but one sound in the audio file was correctly identified. Based on the returned predicted labels, triggering the corresponding analog drum sample at each corresponding onset, we were able to construct a highly accurate translated audio file. The waveform for this translation, in comparison to the original, is as seen in Fig. 6.

4. CONCLUSION

While 94% validation accuracy and 95% test accuracy are certainly impressive, obviously anything less than 100% accuracy would create an incorrect translation. If this were a consumer product, we propose a method for the user to flag incorrectly identified sounds so the model can continue to learn. With continued learning, the model may very well improve to nearly 100% accurate.

This model is clearly more robust, accurate, and efficient than the previous Non-Negative Matrix Factorization based solution. For that reason, we consider this project a success.

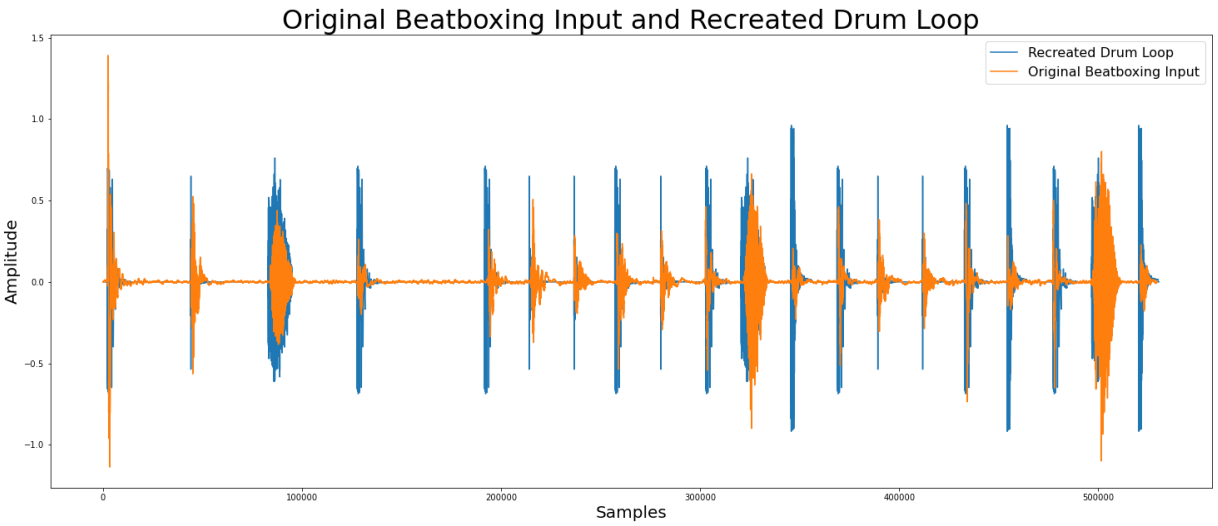


Fig. 6. Recreated drum loop (blue) and ***o***riginal beatboxing input (orange)

5. REFERENCES

[1] A. Delgado, S. K. T. McDonald, N. Xu, and M. Sandler, "A new dataset for amateur vocal percussion analysis," Proceedings of the 14th International Audio Mostly Conference: A Journey in Sound, 2019. DOI: 10.1145/3356590.3356844

[2] "Introduction to Computer Vision with PyTorch." [Online]. Available: https://learn.microsoft.com/en-us/training/modules/intro-computer-vision-pytorch/. [Accessed: 05-May-2023].