Music Classification With Deep Learning

James Smith
University of Massachusetts Amherst
jamwsmith@umass.edu

Abstract

Using a data set in which I cultivated and structured, the goal of this system is to try a similar, yet different overall approach to the music classification problem. The code I've written is to allow a user to not only to classify songs, but also to train a network on their own collection of music, building on a network with up to 80.06% test accuracy already. The library is also structured to take in additional genres over time, to add to the data used in the over data set. My intention with this is to add to this network in future works and try to see if it can handle both additional genres, and also sub-genre classes, while still having accurate results.

1. Introduction

When listening to music, not only do we have an intimate understanding of what we like, it is also very simple differentiate songs by genre. The more we listen to a type of music, the better we get at determining the subtle differences between these genres. In general, these differences can be described, for example, post-rock music is known to have more smoother, building tones, while something like hardcore is known for more fast paced, harsh tones. In other instances, trying to categorize a song can be very difficult though. Genres titles like "experimental" often have vague definitions, thus are more difficult to describe. This project's goal is to use a deep learning technique in order to more accurately categorize songs by their genres. Specifically, my goal is to train a convolutional neural network to classify songs. The type of data available changes how this network is constructed, so it important to understand the domain.

1.1. Digital Representations of Sounds

A song is represented by data points denoting the pitches over the time length of the song, at a defined rate, often called the sampling rate. This sample rate for most songs is either 44.1 KHz or 96KHz. There are two channels of these pitch sequences, which are the left and right channels of the audio. In your headset, this mixing of audio from both speakers create the experience of the song we know. This effect is similar to the RGB channels of a photograph, where each pixel is a 3-dimensional value that completes the representation of the image. This comparison to images will be used a later part of this paper to explain the use of a CNN for this classification. The accuracy of these samples are denoted by the bit-depth value for the song, which is telling of how accurate of a bit representation it is.

1.2. Potential Data Issues

One issue with this type of data is also part of its strength, which is the overwhelming amount of features to deal with (3 minute song at 44.1KHz has about 10^7 features). This potentially gives a lot of insight into each sample, if these features contain important trends, but can also create insanely high running times for the algorithm which don't end up helping the learning. Another major issue is the input size for each song. Between different sampling rates and song time, the sequence lengths of each song are greatly different, which most networks can't handle. Usually, networks that do handle this (commonly recurrent neural networks), would have the greatest issue with high feature amounts without any additional dimensional reduction. The aim for this CNN training is to keep a high amount of features available, while still being able to train on large sets of data.

2. Processing the Song Data

2.1. Song Clips

For this project, my goal was not only to make and train a convolutional network capable of classifying music genre, but also working at every level of the data construction and cultivation. Since I fundamentally created the process this way, I also incorporated the possibility of expanding genres so that this problem could get expanded overtime. For now, I focused on 6 genres; Classical, Rock, Rap, Folk, Electronic and Jazz, and how to turn them all into samples which a network can understand. Since the songs can range from a minute to a half hour, creating some predetermined

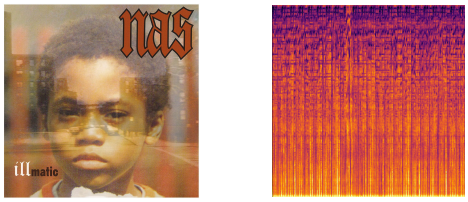

Figure 1. In this example, I took the song "The World Is Yours" by Nas [4] and clipped a random 30 second portion. From this using the librosa melspectrogram feature function [2] I created the right image, which is 323x323x1. These are the features I use as an input to my system

size to clip the song to is important. Even though we lose some features in this process, intuitively it seems like the right way to reduce data dimensionality. If I was to reduce the data points over the whole set, the data would have to be reduced by different amounts depending on the overall size of the song. This means the longer the song, the more "choppy" the signal would become. Each song would therefore be dramatically changed to the point where it would no longer be an accurate representation of the genre, which could negatively impact the network's learning. By taking an x many features from some portion of the song (in my case somewhere halfway through the song), the input size is set and every sample has the same sequence length. The only issue with this technique is if two songs use different sampling rates, the length of the clip in seconds for them is different. To fix this, a conversion to a file type that uses the same same default sampling rate, without making the sample seem as though any song is slower or faster than any other song. Though this is an issue, if you listen to a song for 30 seconds, it is equally easy to classify a song. I created a sub-directory for each genre, and in each of those folders contains a range of 300 to 400 full songs, in various different formats, giving a song library of around 2400 songs. In order to make sure test and train data is separated, so that I didn't accidentally train on some test data, I took some of those songs and stored them in a separated sub-directory labeled as test data. The songs are extracted from the genre folders, reduced to a random 30 second clip, converted to wav format and exported to another directory for training or testing. For each genre this was done about 2000 times so that each song could be used a few times, with different 30 second clips, in order to train for that genre label. This method is analogous to the method of rotating images in order to increase the sample size for a network. The separation of train and test a step earlier now makes it so that the system never tests on a clip from a song that is used to train the network. This means the testing is more robust for generalization.

2.2. Melspectrogram and Features

Now that we have every song as an equally sized sequence, the question remains if there is anything left to do that would create a better representation of the data set for the network. Earlier due to the multi-channel nature of the data, this sequence was compared to idea of an unfolded image. So, to create the image version of this data, it could make sense to fold the sequence into an image by cutting the signal every x amount of points (if x divided by the length of the sequence is an integer y), and creating a x by y by 2 channel image. Although this could potentially work, shapes and other regions created by this are most likely not consistent over the course of the genres, meaning that the network has nothing to learn. For this reason, I chose to turn an image into a melspectrogram [2]. The process of making a spectrogram starts with moving a window over the sequence and take a Fourier transform of the windowed area. This give an amplitude value for every frequency, and x many of these sequences if there are x many windows used on the sequence. This creates an image of frequency vs. time, with a 2-dimensional value for every "pixel", for the amplitude in each channel. For this particular network, I wanted to work with square images, so the window size was picked in order created a 323x323x1 (since the melspec feature extraction was used on a mono signal). All of these images are then appended to a list, with was saved as a pickle file through python, one for testing, another for training. This way, If I ever wanted to expand the genre amount the network handles, I could simply add a saved pickle of the test and train for whatever genre is necessary. Since these files are relatively big, I have a simple script that runs through each main processed genre file and splits it into more manageable chunks. Using these chunks, I've trained separate models, that I'll discuss further in the results portion of this paper.

3. The Convolutional Neural Network

All of the network implementation comes from the python package PyTorch [3], which I felt contained the best balance of control over the network and not too convoluted. For the duration of the experiment, I had no GPU with CUDA support, except for one day. All of this was streamlined to be calculated on a CPU.

The basis for this network comes from the idea that every vertical strip in the image set represents a Fast Fourier Transform (FFT) on a certain window size of the signal. So, all of these strips represent that window of time in the signal well. Thus, with this idea and intuition from other papers [1], the net was constructed to have an initial kernel size of (height, 1), which means the convolutional movement through the image generates a sequence of time based features. In order to increase this dimensionality slightly, I

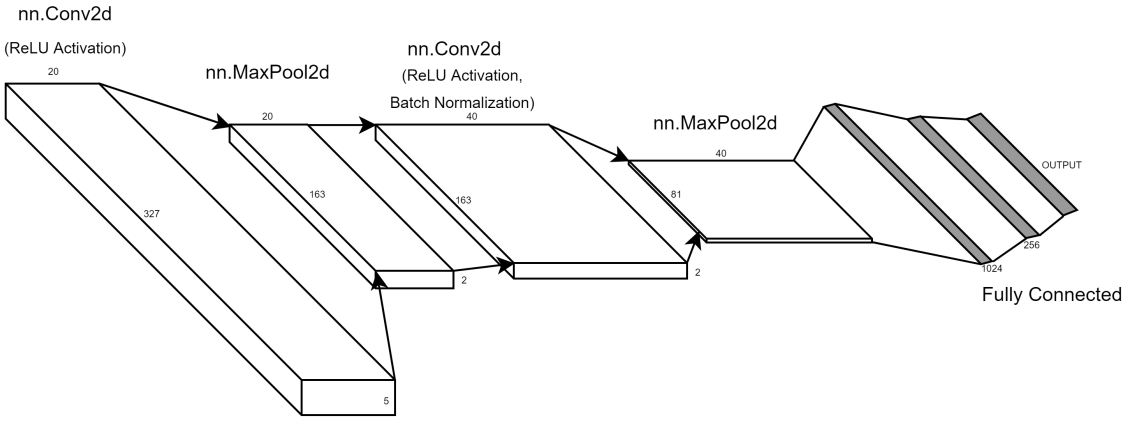

Figure 2. This image visualizes the changes the spectrogram undergoes as it is input into the convolutional network. Although the construction isn't very deep, the way information is passed across layers is based intimately on the features of the data set, due to convolutional windows based on height of the images.

added a padding of 2 to each side of the image. This lets me use a max pooling layer on after activation, with a 2x2 filter, in order to reduce image values for the second convolution. All max pooling and convolutions are 2d, in order to get the best results possible, along with the dimensions necessary to implement a system of this architecture.

The second level of the convolutional layers are constructed in a different way than before. Also using a padding value of 2, this time I used a convolution of kernel size 5 in order to image of width 2, in order to let one more pass of max pooling occur with the same filter size as before. Before either the activation or the max pooling, the filters run through a batch normalization, in order to minimize the total values in the later tensors. The system doesn't implement this after the first convolution in hopes that the difference between values is more distinct. At this point, I wanted the data to be more manageable for the linear layer once the max pooling and flattening of the tensor. To avoid over fitting I tried to keep the amount of nodes lower (1024 as the max hidden layer size), and in some experiments implemented a dropout after the flattening of the tensor. For activation in all pre-fully connected layers, I used a basic ReLU function, which utilizes the max function; $f(x) = max\{0, x\}$.

To train this network, I used different chunks of the larger 12000 sample data set, to check for any sets that would train more effectively, and how well this would generalize to the full test set, which is about 2000 samples. I did this for five chunks, each with a 90/10 split for train and validation. Each time I checked how well they could train on the data set by looking at validation accuracy and comparing to training accuracy, as well as overall loss trends.

4. Results

After the implementation of this network multiple times on different amounts of this data set, the most accurate model I could train had accuracy values of 100% train, 77.00% validation and 80.06% test. Based off one of the chunks of train data, this model had a dropout on the flattened tensor of probability 0.2. Using another script I wrote, I checked how well this generalized outside of train, validation and test samples. For the few samples I selected, it performed as well or better than expected. Every model fell between 60 - 80% test accuracy, which confirms that this architecture is overall adequate at solving this music classification problem. More graphs and images are available on my git, as well as the code I used to make all of this possible. https://github.com/jommysmoth/ECE_Music

5. Conclusions

Though this network already generalizes well, in order to account for additional labels joining the data set, a few changes could be necessary. I want to increase the amount of fully connected layers, as well as their size, since these were limitations of the system that I had at my disposal. Though this will increase the chances of over fitting, it also has the potential to squeeze out those last few percentages of accuracy. If the over fit does become a concern, more dropout could be added, as well as setting up an ensemble of other classifiers for the last few layers.

References

[1] Lee, Hwaran, et al. Deep CNNs Along the Time Axis With Intermap Pooling for Robustness to Spectral Variations. IEEE Signal Processing Letters, vol. 23, no. 10, 2016, pp. 13101314., doi:10.1109/lsp.2016.2589962. 2

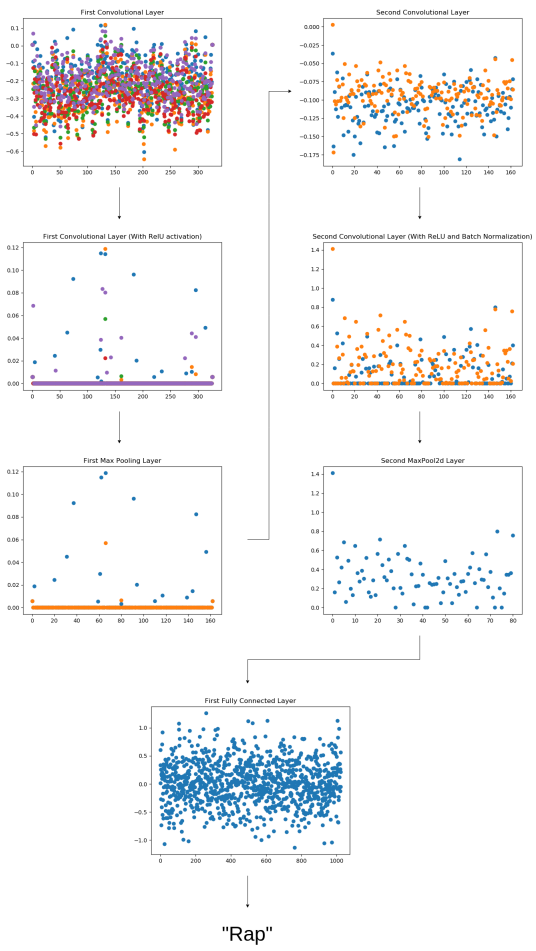

Figure 3. This diagram shows the flow of the data input through the network, with all graph titles showing which portion of the network that they represent. As the diagram progresses, features pop out and become more apparent, which is how the network selects how to classify inputs.

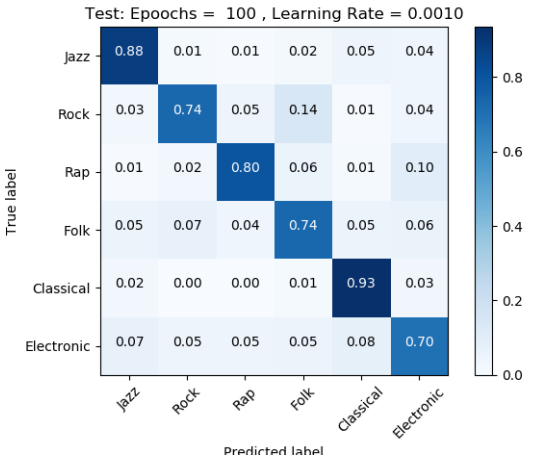

Figure 4. This confusion matrix shows the how the best trained model generalized to the test set (2000 samples). The overall accuracy is 80.06%

[2] B. McFee, M. McVicar, O. Nieto, S. Balke, C. Thome, D. Liang, E. Battenberg, J. Moore, R. Bittner, R. Yamamoto, D. Ellis, F.-R. Stoter, D. Repetto, S. Waloschek, C. Carr, S. Kranzler, K. Choi, P. Viktorin, J. F. Santos, A. Holovaty, W. Pimenta, and H. Lee. librosa 0.5.0, Feb. 2017. 2

[3] Paszke, Adam and Gross, Sam and Chintala, Soumith and Chanan, Gregory and Yang, Edward and DeVito, Zachary and Lin, Zeming and Desmaison, Alban and Antiga, Luca and Lerer, Adam "Automatic differentiation in PyTorch". 2

[4] Nas. "The World Is Yours" Illmatic.