Classification of Spotify features using various CNN architectures

Link to supplementary materials:
https://drive.google.com/drive/folders/1sA9N3xVADW44kdq3fsgAyO1VN856rGCa?usp=sharing

1. Abstract

The goal of my project was to see if song-feature ratings from Spotify could be effectively predicted by a convolutional neural network. Specifically, this project focuses on Spotify's danceability rating. Input data was also tested in the form of direct amplitude value as well as Mel Spectrogram frequency expression. From there, it was tested on convolutional networks of varying dimensionality. Overall, the results generated were intriguing, but not particularly conclusive. And while there certainly is potential in rating songs according to convolutional network predictions, larger datasets and further parameter experimentation are likely needed in order to produce useful results.

2. Introduction

Over the past couple of decades, Spotify's user base has risen starkly to "over 350 million" [1], and is arguably the most popular music streaming service in the world. Many would accredit this to its proprietary algorithm, which offers personalized song recommendations to anyone with a profile. And while this algorithm is certainly vast and depthful in the amount of individual factors that it considers, an integral part of it involves feature ratings that Spotify gives to each song that is uploaded to the streaming service. These feature ratings, or at least the ones available to the public, include but are not limited to valence (emotional affect), danceability, acousticness, and energy. Keeping this in mind, modeling Spotify's feature ratings seems particularly enticing considering that they help to power what is possibly the most successful music recommendation algorithm in the world. Too, modeling these features with a convolutional neural network would be a powerful proof of concept towards their effectiveness. And this project specifically focuses on 'danceability', for no particular reason other than it appears to be a subjective adjective and that modeling more than one feature simultaneously is outside of the scope of this project.

It is likely that most, if not all, of Spotify's features are automatically generated through analysis of songs' audio data [2], so in theory a strong enough neural network should be able to extract these features with enough training. Spotify's own documentation says that danceability is "based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity" [3]. Keeping these in mind, and the fact that timing is extremely important in music, a convolutional network appeared as the most suitable choice to model dependencies that related to danceability.

3. Related Works

According to initial research, there are not many instances online of people trying to predict Spotify's feature ratings according to audio data. This is likely due to a combination of the fact that all of these features are already publicly available (so there isn't much need to be able to determine them) and that the search for related works was not particularly extensive due to the time constraints when conducting this project. Of the related works found, one the most relevant was a Medium article by Tobi Sam [4]. In it, he tries to use other features, such as danceability or valence, to predict popularity. Notably, there were many more complex instances of this same concept that were found too, but since it is only the general concept that relates, this one was chosen to be mentioned. And while classification of Spotify features from the other features is still vastly different from classifying according to audio signal data, it does imply correlations between various features and suggests that there may be more similarities than one might initially think.

4. Architecture

4.1) Data collection and processing

One of the first challenges for this project was finding suitable data to analyze. Because most songs are not actually legal to download without paying a fee to the artist, the amount of available songs was actually quite limited. Luckily however, I happened to know that one of my favorite artists, Nicolas Jaar, actually has a significant amount of his discography available to download for educational use. Uncompressed, it includes 3.67 Gigabytes, which was plenty for the scope of this project. Too, this data offered an interesting challenge for the model due to the variation in content, ranging from a dance album to one intended as an "alternate soundtrack" for a movie.

Next, the audio data needed to be transformed so that it could be processed by the convolutional network. Both waveform/amplitude data and spectrogram data were analyzed. Initially, I had planned to use just amplitude data, but shortly into testing the network, it became clear that using the entire set of uncompressed .wav files was extremely RAM intensive and slow to train. Too, the initial network architecture that modeled this data needed to be extremely large to capture any complexities contained within the 3.67 GB. Thus, I decided to also test transformed frequency data in order to compress and possibly reveal various frequency features that may have been harder for the model to extract from purely the waveform of the songs. For this, a Mel Spectrogram was used due to the fact that it emulates human perception of frequency, which would presumably be desirable when modeling a heavily perception-based trait such as danceability. Additionally, the stereo audio was transformed into mid/side in effort to aid with extraction of track width.

Once the data was read and processed from the .wav files, each song was divided into 30 second long windows. The idea here was to balance context and network size/processing with amount of data and data readability. So while some valuable context is sacrificed by slicing the audio into these shorter windows, it also allowed me to work with 650 data samples instead of 66. Furthermore, it allowed the networks to contain more depth because the architecture did not have to account for larger input sizes. This being said, the context sacrificed by splitting window sizes here cannot be overstated. Because labels (danceability ratings) could not be changed in correspondence with their individual windows, the model now had a classification-type

problem of needing to label vastly different sonic parts similarly. This concept will be touched upon more in the discussion.

Finally, before being fed to the networks, the audio data was normalized by zeroing the mean, adjusting the standard deviation to 1, and changing it to range from 0 to 1. The output labels (feature data) were not normalized because Spotify's danceability ratings are already normally distributed from 0 to 1 [5].

4.2) Network architectures

1DWav: This was the first attempted network architecture. However, it was quickly moved on from considering that it was not producing very meaningful results, and it was tough to even get it to learn on the training data. This being said, it laid a solid foundation for the architecture that was developed in later networks.

The architecture consisted of two convolutional blocks in parallel, one for the mid information and the other for the side. Within each block, there was a convolutional layer (kernel size 100), followed by a max pooling layer (pool size 2), and a batch normalization layer, twice in succession (6 layers for each block). The blocks are then concatenated and fed through two fully connected layers before being sent through the output layer with a sigmoid activation function. Each other layer contained a Relu activation function, as did all of the models tested for this project.

While this model's architecture is woefully oversimplified for the data it is trying to model, which was reflected by its training accuracy, it served as a means to scaffold the ideas behind the architectures to come.

2DWav: This architecture was created in attempt to capture more scope than 1DWav by stacking the audio data on top of itself. It does this by slicing the 661500 sample long input into 100th's, then stacking them on top of each other. While it was more successful than 1DWav at modeling the data, it also was extremely difficult to train. Because the dataset alone was 3.67 GB and this network has almost 500x as many parameters, it took at least 5 minutes to train for each epoch, which made it very difficult for me to test longer training iterations due to the time constraints of this project.

This being said, the architecture follows the same parallel flow as 1DWav, except now with (45, 10) sample convolutional kernels and (3, 2) max pools.

2DSpec: After running into constant issues with data size and training duration, I decided to pivot the data being processed into the spectral domain. The idea was that by throwing out phase data and dividing the waveforms into discrete frequency intervals, the data could be compressed without throwing away too much valuable information. Furthermore, it could possibly help the model to generalize more because there is less specific information that it can focus on. For this process, the Mel Spectrogram transform was chosen for reasons aforementioned.

The first iteration of 2DSpec differed greatly from that of the Wav architectures. Firstly, I came to realize that it might be worth processing mid/side data concurrently, instead of in isolation, and so inputs were fed as a [x, y, 2] matrix instead of 2 [x, y] matrices. Additionally, because there was so much less workload/space taken up by the data, I was able to make the network far deeper while still within the scope of RAM available to me. Thus, the 2DSpec network (which took inputs of size 24,2584)

processed input through 4 (3,4) kernel size convolutional layers, one (2,2) max pooling layer, another 3 (3,4) kernel size layers, another (2,2) max pooling layer, then finally 3 fully connected layers. Eventually, with most testing and troubleshooting, numerous dropout and batch normalization layers were included as well, the specifics of which are contained within the project's Jupyter notebooks.

3DSpec: Finally, in effort to leverage the advantages presented by 2DWav and 2DSpec, a 3DSpec model was implemented. This took the already 2 dimensional spectrogram data and sliced it on top of itself, much like 2DWav, into 38th's. From there, a very similar sequential order to 2DSpec was followed, including two (3,2,2) convolutional layers, followed by a (2,2,2) pooling layer, followed by a (3,2,3) and (2,3,3) layer, a (1,2,2) pooling layer, and two more (2,2,2) convolutional layers and three fully connected layers. Again, batch normalization and dropout were added later on, and the architecture can be observed in-depth in the Jupyter notebook.

5. Experiments

There was a lot of difficulty actually getting any of the models to train even in a way that improved training accuracy. Ultimately, I believe these struggles were due to vanishing gradients, which I addressed through batch normalization and He weight initialization. Additionally, testing was cut short for the 2DWav architecture because the system would constantly run out of RAM when trying to scale the training/testing to the full scope of the data. Aside from these challenges and because it was hard enough for the models to learn from the training data, the main experimentation lied in seeking to reduce training error, then seeking to improve testing generalization once a sufficiently powerful architecture was found. That is, hyperparameters and layer placements were first optimized to improve training performance, then tweaked to aid with testing ability.

Of the different factors tested (besides architecture itself), Adam was compared against SGD along with various learning rates for each, levels of dropout inclusion were tested extensively, number of layers and kernel size were considered, as was type and location of pooling layers and type of loss function used (between mean absolute error and mean square error).

The parameters, optimizers, and layers found in the Jupyter notebook are the ones that I found to be the best, and so I used those when conducting my final comparison between architectures.

6. Results

There was really nothing found that is worth writing home about, but some of the results are intriguing and garner more extensive exploration. Overall, spectrogram data seemed to perform better than direct waveform data, which I suspect is because it condensed the data to the point where it could be trained more efficiently and had more generalizability (since overfitting was a significant challenge for the model). It is also possible that spectrogram data performed better because it revealed certain frequencies which existed in both the training and testing data, since windows from the same song were included within each. It is also possible that these architectures were simply more refined, as I had learned from my creation and testing of each architecture.

Specific results can be seen in the Jupyter notebook, but the general trends observed were that while training loss was ultimately able to be reduced consistently by the models, it did not correspond to testing accuracy. Instead, the change in testing accuracy appeared to be rather volatile, seeming to change more according to chance than the respective training loss. 3DSpec appeared to be the least volatile of all of them, but that is not saying much.

7. Conclusion and Discussion

The project was inconclusive, but since each model did produce an effect size and certainly appeared to be learning features, a number of extensions would be worth looking into. First and foremost, I believe that more data would be essential. I don't think it is possible for anything to be able to generalize such a broad term as danceability from only looking at 66 songs. Furthermore, because the songs in the dataset vary so much in content and genre, I think it was all the more difficult for the model to learn. Perhaps it would be worth testing the concept on a limited genre space of say, pop-country, and then seeing how well it can generalize within the same genre before trying to branch out into varying genres.

Other potentially promising paths of exploration include separating training and testing data by song, data augmentation, varying or sequential window sizes (while perhaps calculating total danceability ratings by the combined danceability of each window within a song), identifying first whether the network can identify more straightforward features such as BPM, use of context-based models such as LSTM or transformers, and stacking network inputs in a way that corresponds with song tempo. It also might be worth trying to analyze a compressed audio format such as mp3, because there is limited perceptual difference in comparison to the vast size difference, which would make it easier to train and likely more generalizable.

Ultimately, while I think this project was a very interesting pursuit, I cannot see much use for it in the real world. If these features were determined by humans, a successful implementation may have potential to reduce labor, but since these ratings are already algorithmically determined, this network implementation isn't necessarily offering anything 'new'. That being said, with enough training, I could see using predictions from neural networks as another potential way for people to categorize songs according to the features that it is trained on. The idea of being able to stick a song into a computer and get an accurate rating of how emotional, or danceable, or happy it is is simultaneously exciting and scary.

8. References

https://www.businessofapps.com/data/music-streaming-market/ [1]

https://en.wikipedia.org/wiki/The_Echo_Nest [2]

https://developer.spotify.com/documentation/web-api/reference/get-audio-features [3]

https://towardsdatascience.com/do-hit-songs-have-anything-in-common-37599940590 [4]

https://dhruv-khurjekar.medium.com/investigating-spotifys-danceability-index-other-song-attributes-1983142f7dfd [5]

9. Personal statement

I don't know if this part is relevant/productive to the assignment's rubric at all, but I figured it useful and insightful, at least for myself, to include. I was challenged greatly by this project, but am happy to say that I also learned a lot through it. Never have I experienced an issue where a convolutional network simply doesn't improve with training, so troubleshooting that, as tedious and annoying as it was, was an enlightening and revealing experience. On top of that, this project has helped me to understand that data is everything. Regardless of the quality or quantity, if it is not reflective and encompassing of everything that you are trying to learn, it will likely not be very helpful. Perhaps most important of all, it has only left me more curious, so I am eager to continue playing around with things now that this class is concluded.