CS230

Autogroove: Music for Sharks

Alex J. Bertrand
Department of Electrical Engineering
Stanford University
abert13@stanford.edu

Jordan B. Friedland
Department of Applied Physics
Stanford University
jordanf2@stanford.edu

Abstract

We present a convolutional autoencoder model that takes as input a preprocessed spectrogram of a segment of a song, generates an encoding for it, and subsequently clusters these encodings, all using unsupervised learning techniques. This constitutes a novel approach to music classification, as there has been little to no previous work that utilizes unsupervised learning algorithms to learn representative encodings of songs. We find that the proposed convolutional neural network architecture is appropriate for generating these encodings and shows promise for classification as well.

1 Introduction

In recent years, a great deal of progress has been made in the area of automated music classification. What perhaps started with Pandora's Music Genome Project [1] has since grew into a number of attempts to effectively categorize songs, increasingly with less and less human input. The classifications that have resulted from these efforts have been used in a number of ways and in an increasingly sophisticated manner, with neural network-based approaches now making progress in even composing music themselves.

Despite this, there has been a conspicuous lack of progress in the area of music *recommendations*. Services which claim to be able to find and aggregate songs that are similar to a 'seed' song are, for the most part, largely finding songs that are classified in the same genre. We submit that, in order to effectively group songs, there must be some similarity between songs that is not explicitly classifiable by a human but that results in humans grouping songs together that 'feel' the same.

The input to our model is a preprocessed spectrogram of a song, which is the discrete Fourier transform applied in a sliding window across a raw audio sample. From there, we have two important outputs from our convolutional autoencoder: the first is a reconstruction of the spectrogram, and the second is an encoding of the spectrogram in a significantly lower dimensional space. Then, we feed these encodings into a clustering algorithm, which groups these encodings based on their Euclidean distance from one another. Our goal is to find a more nuanced representation of songs based on subtler features of music that a human cannot necessarily put words to.

2 Related Work

There is a significant body of literature that reports various methods of music classification, and the majority of them use "traditional" machine learning methods. Two reports, "Audio Music Mood Classification Using Support Vector Machine", by Cyril Laurier and Perfecto Herrera [2], and "Music Genre Classification via Machine Learning", by Li Guo, Zhiwei Gu, and Tianchi Liu [3] use support vector machine techniques to classify music into genres based on other feature vectors. This is all

CS230: Deep Learning, Winter 2018, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

contingent upon the music being labeled "correctly" by a human and does not take into account any of the actual audio data.

A slightly more sophisticated method of classifying music is presented in "Hidden Markov Classification of Musical Genres" by Igor Karpov [4], which uses the raw audio data to generate features like the Fourier transform-based Mel cepstral coefficients, Mel cepstra with delta and acceleration information, and linear predictive coding that are then fed into a hidden Markov model which is used to classify songs by genre. Finally, Matthew Prockup of Drexel University, in his Ph.D research, presented in the thesis "A Data-Driven Exploration of Rhythmic Attributes and Style in Music" [5], dives into a deep analysis of how to generate features of music from raw audio, focusing on rhythm and style, and discusses how to use these features to predict genre.

There is also a body of literature reporting the use of convolutional autoencoders, including "Deep Clustering with Convolutional Autoencoders" by Guo, et. al. [6] and "A Deep Convolutional Auto-Encoder with Pooling-Unpooling Layers in Caffe" by Turchenko, et. al. [7], neither of which have specific applications to music. Finally, the paper titled "Convolutional Recurrent Neural Networks for Music Classification" by Choi, et. al. [8] proposes a convolutional recurrent neural network architecture for 'tagging' music. In other words, these authors are simply using a highly sophisticated neural network to classify music into genres.

The present work takes a different approach to music classification, instead allowing the model to generate an encoding of a song, which can subsequently be used to cluster songs together based on feature similarity.

3 Dataset and Features

Our data comes from the UC Irvine Machine Learning Repository's Free Music Archive [9]. It consists of 106,574 thirty-second samples of songs. We pruned the dataset to only include songs that were sampled at 44.1 kHz, and then split each thirty-second sample into ten-second clips before computing the spectrogram of each. We trimmed each spectrogram to produce a 256 * 256 pixel 'image' - in reality just a two dimensional array of different frequency intensities for the time interval over which the spectrogram was calculated. Note that although we might refer to our dataset as 'images', we still only need one channel to represent them. The color we have reproduced our spectrograms with here is a one dimensional function of the input data, and is not intended to communicate any actual RGB values.

After computing the spectrogram, we normalized by the total energy of the song sample, which can simply be computed by summing the squares of all of the pixel values. This resulted in a spectrogram where over 99% of the pixel values were concentrated in the range $0 - 10^{-4}$ while the full range of pixel values was $[0, 1]$. This meant that the features in the spectrogram were indistinguishable, both by humans and by the model. In order to remedy this, we applied a large nonlinearity to the spectrogram to spread out the pixel values that were concentrated near zero but preserve the full range:

$$y_{out} = \tanh\left(cy_{in}\right)$$

where $c = 10^6$. This nonlinearity accentuated the important features of the spectrogram without sacrificing any information (see Figure 1 below).

This resulted in a dataset of approximately 270,000 samples. In the interest of training time and speed with which we could acquire meaningful results, we used a training set of 60,000 examples and an evaluation/test set consisting of 2,000 examples. Because our inputs were inputs to an unsupervised learning algorithm, we did not have any particular features that we were interested in predicting. Instead, we allowed the convolutional autoencoder to generate a set of features at the latent representation it generated for each spectrogram.

4 Methods

As previously stated, it is infeasible to rely on labeled data for this particular task. The 'feel' of a song is relatively ill-defined, and not available in any dataset to the best of our knowledge. For this reason, we extract features from songs by way of a convolutional autoencoder. This is a relatively well researched area of unsupervised learning which has shown promise in producing adequate features

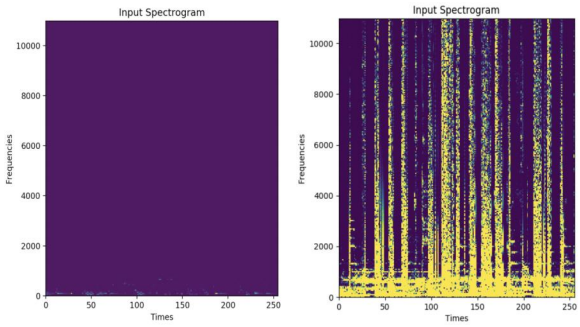

Figure 1: (left) Spectrogram before preprocessing, (right) Spectrogram after being passed through large nonlinearity

upon which to enact clustering algorithms. The exact nature of the features that will be extracted by the autoencoder is initially unknown. This can be attributed to the fact that this particular method of music analysis has not previously been implemented.

Though convolutional networks (and to a lesser extent, convolutional autoencoders) have been used quite a bit for the purposes of image classification and analysis, applying them to musical data is a comparatively unstudied task. This, coupled with the fact that network architecture design seems to be a somewhat offhand process as of the writing of this paper, left us with little in the way of guiding principles for our layer and filter hyperparameters. After many iterations on our initial best guesses, we ended up with the architecture shown in Figure 2:

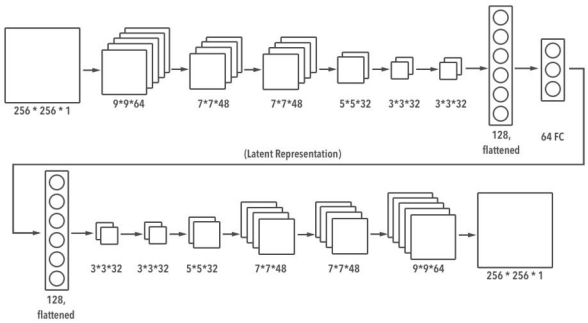

Figure 2: Convolutional autoencoder architecture. Encoder and decoder each have six convolutional layers and one fully-connected layer, leading to a 12-layer network.

A few factors led to some of the specific decisions we made:

- Filter sizes are based loosely on some of the other architectures we found while looking at related work and some of the more prominent CNN's out there. The main goal we kept in mind was to try not to shrink the dimensionality of the input too quick.
- The dimensionality of our latent representation was an ongoing process throughout this project, and we really only settled on 64 at the very end after realizing that the 128-dimensional representation we had been using had quite a few zeros that were consistent across multiple encodings. While we would rather be overcomplete than the opposite, we took the risk in an attempt to increase training speed.
- Initially, the fully connected inputs going into and coming out of the latent representation were around 7000 parameters long. After realizing that this was contributing the overwhelming majority of our trainable parameters and slowing us down dramatically, we downsized this significantly.

- The goal of our autoencoder is to reproduce exactly the input x at the output y. To this end we use a simple mean-squared loss to compare the two. Here N is the number of training examples, and h and w are the dimensions of our inputs/outputs (256, in our case):

$$\mathcal{L}(y, \hat{y}) = \frac{1}{Nhw} \sum_{i=1}^{N} \sum_{j=1}^{h} \sum_{k=1}^{w} \left(y_{jk}^{i} - x_{jk}^{i}\right)^2$$

With the autoencoder properly and adequately trained, we then use the latent representation produced as an encoding for our songs. From these encodings we attempt to cluster the songs in our dataset using the method of Affinity Propagation. This is a particularly suitable method for this problem, largely for the reason that it does require an initially specified number of clusters to use. Rather, it picks certain key elements of the data as exemplars for larger clusters. Given that we cannot readily place a number on the number of different 'feels' of songs in existence, nor how many of those are present in our dataset, this is appropriate for our task.

5 Experiments/Results/Discussion

The overriding feature of our training/hyperparameter selection process turned out to be minimizing computational effort and time necessary. As stated, we used 6 layers of convolutions to get down to our latent representation. This was good because it gave us a great deal of granularity and capacity in our architecture (and, as we'll address, we believe it was effective) but bad because it required another 6 layers post-encoding. The large number of resulting trainable parameters forced us to take optimization strongly into account in the rest of our process.

As we gained confidence in the adequacy of our architecture we chose to undertake a small sub-experiment to validate our intuitions. In order to confirm that our network had the requisite depth to accurately fit to any spectrogram at all, we trained a copy of our network on a set of 10 examples for 100 epochs. The resulting input and output are shown in Figure 3. While this is certainly a case of overfitting it is evident that the network does indeed have sufficient 'memory' to reconstruct the input spectrograms, despite having to compress them from 256*256 floats to only 64.

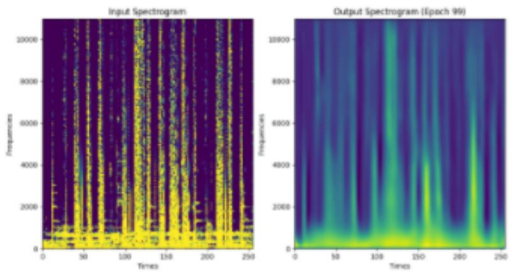

Figure 3: Reconstruction of the input (left) at the output (right)

Having seen this, we trained our model on our full dataset. We used minibatches of size 64 and a learning rate of .001 (we tried higher, but it led to wildly fluctuating losses without decrease). We also elected to train on batches of size 60,000 instead of the full 270,000 samples that are dataset contained; training on the full set took a prohibitively long time and slowed down the speed at which we could iterate. To make use of the entire dataset in spite of this, we randomized which 60,000 examples we used every 2 epochs. We speculate that this may also have had the benefit of preventing overfitting.

Due to time and money limitations, we were only able to train for 22 epochs. Notably, our losses continued to drop and our training loss did not diverge from our test loss (Figure 4), indicating that we could have continued to train.

After this training, we ended up with a reconstruction like that in Figure 4. While it is apparent that we have begun to pick up on the key features of the input, we have not yet come to the level of granularity in our proof of concept. This makes sense, since we're short about 80 epochs.

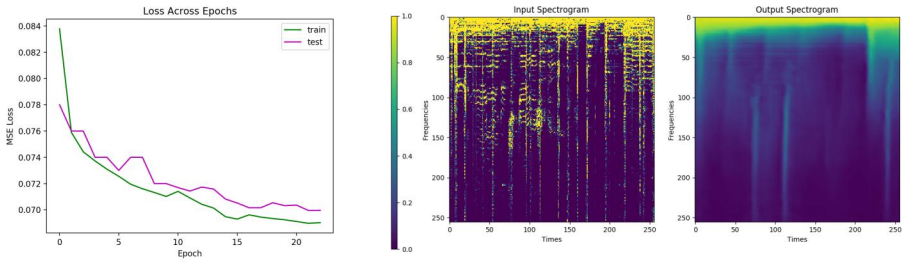

Figure 4: (left) Train and test loss over 21 epochs of training (right) Input and reconstructed spectrograms after training on large dataset

With our model (relatively) trained, we implemented Affinity Propagation on the data. Unfortunately, there does not exist at this time any efficient way to test the clustering of music based on feel short of manually listening to the outputs and checking for perceived accuracy. After doing this, we can report two main results:

Firstly, we believe that training with a different dataset (ideally for longer) would benefit us greatly. While the Free Music Archive is fantastic for legally experimenting on over the course of a few weeks, it is notably deficient in a number of ways, most notably the inclusion of mainstream or recognizable music. Many tracks were no more than ambient sounds without a clear tempo or instrumentation (one was just the sound of footsteps with a low bass frequency) and some were even audiobooks. We believe that training on these examples most likely did not help our autoencoder learn the structure of the actual sort of music that we would like to categorize.

And secondly, despite this, our model was in many cases able to recognize some of the more broadly defined characteristics of the music it trained on. We found one output cluster, for example, that contained a number of tracks with high, airy instrumentation and a lethargic, ambient sort of feel. In this regard, we consider ourselves to have found some success with our implementation.

6 Conclusion/Future Work

In conclusion, we have presented a novel model architecture for automatically generating encodings of songs and clustering them based on the similarity of these encodings using a convolutional autoencoder.

For future work, we could certainly train the model for significantly longer than we had the time for. The loss curves for the training and evaluation/test sets had not diverged at the time this paper was written, and thus we could have trained the model for a longer period of time in order to generate better encodings.

In addition, we would perform a more sophisticated statistical analysis of the input data to determine trends and features that would allow us to a) generate spectrograms of songs while giving up as little information as possible and b) come up with a model architecture that is general enough to be able to pick up on and reconstruct a spectrogram from any audio sample. Furthermore, in order to balance training time with the accuracy of the autoencoder's reconstruction, we would need to look further into lossless compression rates for audio *and* image to determine a better dimensionality for the latent space.

7 Contributions

As a pair, each member contributed equally. Jordan performed more of the literature review and background research, constructed part of the model, and did a significant amount of pre- and post-processing of the data. Alex generalized the convolutional neural network model, implemented the clustering algorithm, and took the lead on training the model. Both members contributed equally to this paper.

References

[1] Music Genome Project. https://www.pandora.com/about/mgp.

[2] Laurier, C. and Herrero, P., "Audio music mood classification using support vector machine," in *International Society for Music Information Research Conference, 2007*.

[3] Guo, L., Gu, Z., and Liu, T., "Music Genre Classification via Machine Learning," CS 229 Final Project, 2017.

[4] Karpov, I., "Hidden Markov Classification of Musical Genres," COMP 540 Final Project, Rice University, 2002.

[5] Prockup, M. "A Data-Driven Exploration of Rhythmic Attributes and Style in Music," Ph.D. dissertation, Department of Electrical Engineering, Drexel University, Philadelphia, PA, 2016.

[6] Guo, X., et. al., "Deep Clustering with Convolutional Autoencoders," in *International Conference on Neural Information Processing, October 2017*, pp 373-382.

[7] Turchenko, V., Chalmers, E., & Luczak, A., "A Deep Convolutional Auto-Encoder with Pooling-Unpooling Layers in Caffe," arXiv preprint arXiv:1701.04949 (2017).

[8] Choi, K., et. al., "Convolutional Recurrent Neural Networks for Music Classification," arXiv preprint arXiv:1609.04243 (2016).

[9] UC Irvine Machine Learning Repository Free Music Archive. https://archive.ics.uci.edu/ml/datasets.html.

Link to Code Repository

https://github.com/jbenf3/autogroove Note that this github repo is private. We've shared it with Zahra Koochak and the CS230 account, but if we need to share it with anyone else please let us know! It's a live link, we promise.