Muzip: Music Compression Using Neural Networks

Divya Saini
Department of Computer Science
Stanford University
sainid@stanford.edu

Michael Arruza
Department of Computer Science
Stanford University
marruza@stanford.edu

David Morales
Department of Computer Science
Stanford University
mrlsdvd@stanford.edu

1 Introduction

In recent years, significant effort has been put into image compression. Several compression frameworks have been proposed in attempt to achieve high-quality image compression at low bit rates and achieve success in consolidating repetition, redundancy, and irrelevancy in images. What has not been as heavily investigated, however, is a similar task on audio. Thus, for this project, we are investigating a similar question of how to build off of the advancements in deep learning for compression, but for music. The aim of music compression is to reduce the redundancy in a song in order to be able to store, transmit, and search for the song at low bit rates.[5]

Music files are traditionally relatively large and difficult to process, so compression methods are often used to reduce the file size. MP3 files are one such compressed representation that results in lost data. Our project seeks to use neural networks to convert audio files into a compressed representation that reduces file size while still allowing reconstruction of the original file. Unlike prevailing literature that tends to center around spectogram representations of an audio file[6], we attempt to investigate whether it is possible to garner higher quality and better compressed output on raw audio signals.

1.1 Applications

The applications of a properly compressed music file could be significant in several contexts. We believe that if the representation is smaller than those one could normally garner from other compression methods, or if the representation is quicker to compute, then it could potentially be used to improve current music storage and reduce the data that must be transferred while streaming. Proper and quick compression might also have application in the analysis of music data, in the case that these compressed representations encode semantic information present in the song that could be used for overall deciphering, searching, and indexing.

2 Dataset

We use the FMA music analysis dataset [1][4], which provides 917 GiB of audio from 106,574 tracks from 16,341 artists and 14,854 albums of 161 genres. Along with this audio, the dataset provides pre-computed features with track- and user-level metadata, tags, and free-form text.

For rapid iteration, we use the smaller version of the dataset containing 8,000 30-second snippets taken from a multitude of songs across 8 balanced genres, in MP3 format. The next step is to try the medium dataset available (containing over 25,000 tracks of 30-seconds) and then the largest dataset available (containing over 106,000 untrimmed tracks).

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

2.1 Pre-processing:

We convert each MP3 track to WAV format, which yields a vector of floats between -1 and 1 representing the audio signal. Because the vectors representing each 30-second track is quite large (1,320,000 values), we split the WAV file into 30, 1-second chunks. Each of these chunks is used as a training example. To improve training, signal values are multiplied by 10.

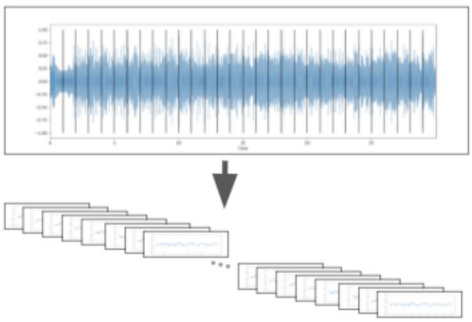


Figure 1: Audio sample is split into shorter 1-second chunks.

3 Approach

Initially, we considered approaches such as encoder decoder RNNs to compress the WAV file down by some factor and then attempt to rebuild the original file. However, because of the large amount of data in even one second of data (the wav files for one-second snippets ultimately form vectors of 44,000 floats), we fear that training RNNs on this data will be too difficult. Instead, we opt to use convolutional neural networks. Taking inspiration from Feng Jiang et al. [5], we use an encoder network as a compressor, and a decoder network for decompression.

The two components are trained in conjunction, with the output of the encoder being fed into the decoder, and the output of the decoder being evaluated. The combined network is trained to minimize the mean squared error between the one-second sample of the original track and the the generated one-second output signal.

$$MSE = \frac{1}{T}\sum_{t=1}^{T}(f(t) - \hat{f}(t))^2$$

Where T is the total number of samples in the signal (4400 per second of audio), $f(t)$ is the value at time t of the original signal, and $\hat{f}(t)$ is the value at time t of the generated, decompressed signal. We train the network on a very small batch of the data (roughly 48000 one-second snippets from sampled songs) to test the network's compression and reconstruction capabilities.

We iterate over several models, each with different encoder and decoder architectures as follows:

3.1 Model 1

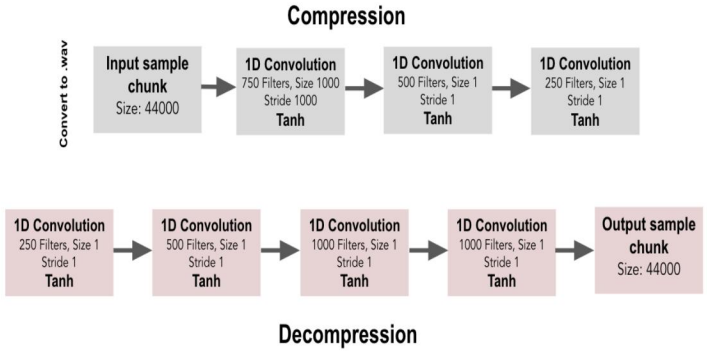


Figure 2: Grey boxes show encoder layers and pink show decoder layers

3.2 Model 2

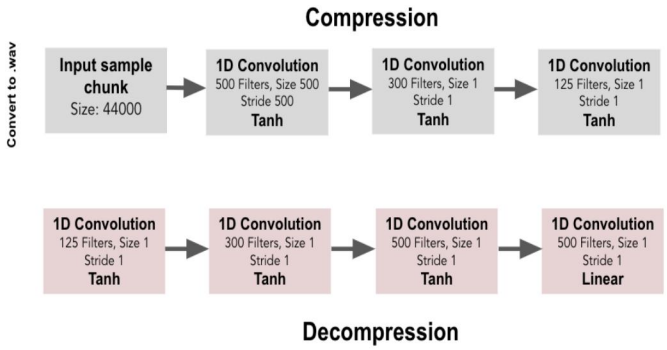


Figure 3: Grey boxes show encoder layers and pink show decoder layers

3.3 Model 3

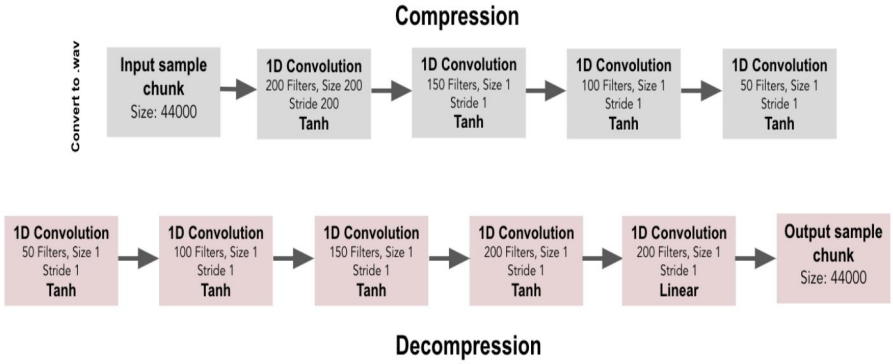
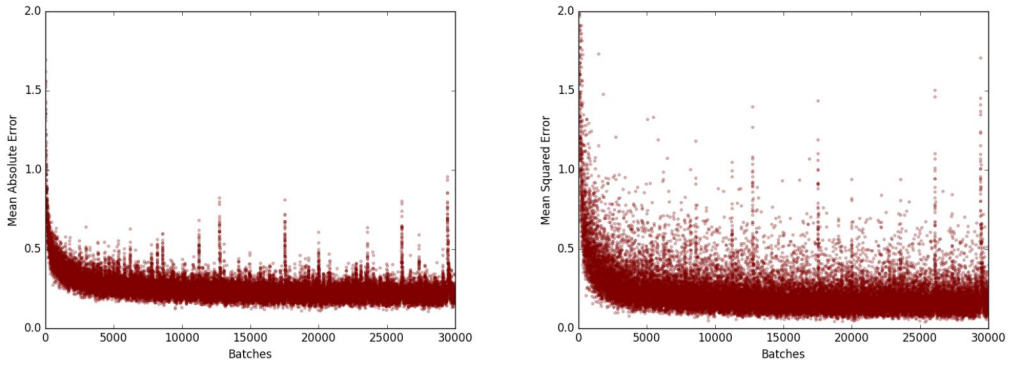


Figure 4: Grey boxes show encoder layers and pink show decoder layers.

4 Evaluation & Results

Model	MSE	MAE	R^2	Duration
Model 1 3 Compression Layers 4 Decompression Layers	0.0040	0.0388	0.8561	112.87 ms
Model 2 3 Compression Layers 4 Decompression Layers	0.0021	0.0278	0.9206	77.29 ms
Model 3 4 Compression Layers 5 Decompression Layers	0.0010	0.0182	0.9639	61.27 ms

Figure 5: Model performance. Performance metrics of our two best music compression and decompression models on 131 examples. Both provide 4x compression.



(a) Mean Absolute Error

(b) Mean Squared Error

Figure 6: Mean absolute error and mean squared errors over batches for Model 3.

We evaluate the model on 131 complete, 30-second tracks, by processing one second at a time and then combining the one-second, decompressed samples for each track. For each model we look at mean squared error, mean absolute error, correlation, and compression/decompression duration. The network is able to learn most of the underlying pattern, but the loss graphs are very jagged, suggesting that maybe a smaller learning rate should be used. The training was done on a very small batch of data, so it is possible that more data will smooth out the curve and allow the network to learn better. We also notice that the errors have a sharp drop within the first 5000 batches and then seem to stabilize. Perhaps this is an indication that too much compression is happening in the first few layers.

We notice that the best model is that with smaller filter sizes and more convolutional layers in the decoder component (Model 3). This suggests that a more gradual approach to compression and decompression, over a deeper architecture may be more effective in avoiding a stabilizing error and instead allowing for one that compresses more cleverly over layers that eventually result in an ever lower error.

5 Conclusion and Next Steps

Through our iterations we see that convolutional neural networks are able to fairly successfully compress music samples to one fourth of their original size. This is a significant result given that MP3 compression typically achieves between 75 and 95 percent reduction in size of original file[3]. Our methodology allows for significantly more reduction. Furthermore, while the decompressed samples are somewhat noisy, they preserve most of the underlying music signal and performance scales well across genres. Voice also comes through well throughout this process. There are techniques for removing sound through conventional filters that we can attempt to use in ridding some of this noise fairly easily. Another thing we can look into is incorporating an attention mechanism into our networks [7].

We believe that taking genre into account as an input may result in better decompression. Perhaps the network will learn to account for similarities in genre. We also believe that evaluating semantic information captured in compressed representations may reveal potential use for compressed samples. If, for example, we're able to compress to a level greater than 4x in future models and not achieve sound quality that is comparable to MP3 quality, we might consider using the compressed representations for quicker search and compare amongst a database of song representations. This would be useful in creating much more efficient searches that use fewer computing resources and time due to the representations of music that contain enough content to encode the underlying meaning and representation of the song yet without enough encoded content to be able to decompress the song to a quality that consumers would find indistinguishable from an MP3 quality version of the song.

6 Contributions

We have mostly worked on the code concurrently, though much of the division of labor has involved coding up and trying different models at the same time. Divya and David focused a little more on getting the data and model evaluation pipeline working so we can iterate on models more quickly, whereas Michael focused on refining the models once we settled on the convolutional framework. Our work can be viewed on our github page [2].

References

[1] `https://github.com/mdeff/fma`.

[2] `https://github.com/michaelArruza/CS230-Compression`.

[3] Brandenburg and Karlheinz. Mp3 and aac explained. audio engineering society, audio engineering society.

[4] Michaël Defferrard, Kirell Benzi, Pierre Vandergheynst, and Xavier Bresson. Fma: A dataset for music analysis, 2016.

[5] Feng Jiang, Wen Tao, Shaohui Liu, Jie Ren, Xun Guo, and Debin Zhao. An end-to-end compression framework based on convolutional neural networks, 2017.

[6] Laurent Girin Jonathan Pinel. Informed source separation through spectrogram coding and data embedding, 2012.

[7] Heliang Zheng, Jianlong Fu, Tao Mei, and Jiebo Luo. Learning multi-attention convolutional neural network for fine-grained image recognition.