AUDIO COMPRESSION VIA NONLINEAR TRANSFORM CODING AND STOCHASTIC BINARY ACTIVATION

BY

YUANHENG YAN

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2019

Urbana, Illinois

Adviser:

Associate Professor Paris Smaragdis

Abstract

Engineers have pushed the boundaries of audio compression and designed numerous lossy audio compression codecs, such as ACC, WNA, and others, that have surpassed the longstanding MP3 coding format. However most of the methods are laboriously engineered using psychoacoustic modeling, and some of them are proprietary and only see limited use. This thesis, inspired by recent major breakthroughs in lossy image compression via machine learning methods, explores the possibilities of a neural network trained for lossy audio compression. Currently there are few if any audio compression methods that utilize machine learning.

This thesis presents a brief introduction to lossy transform compression and compares it to similar machine learning concepts, then systematically presents a convolutional autoencoder network with a stochastic binary activation for a sparse representation of the code space to achieve compression. A similar network is employed for encoding the residual of the main network.

Our network achieves average compression rates of roughly 5 to 2 and introduces few if any audible artifacts, presenting a promising opening to audio compression using machine learning.

Acknowledgments

I would like to express my sincerest gratitude to my adviser, Professor Paris Smaragdis. He inspired me to pursue the field of digital signal processing, and even influenced my career choices. I could not have finished this thesis without his patience and guidance.

I also want to thank the University of Illinois at Urbana-Champaign for providing a receptive and forward-looking academic palace where students and scholars from all over the world collide and collaborate. The university helped me meet many brilliant minds that inspire and stimulate me.

And of course, I am forever in debt to my parents for their loving trust and unwavering support.

Table of Contents

Chapter 1

Introduction

In the late 1980s, storage and processing of digital media became common, but the storage limit and data transfer speeds had yet to catch up, thus spurring the demand for data compression to facilitate faster transfer and more efficient storage. The demand led to the iconic codecs such as JPEG and MP3, both of which are undoubtedly still the most popular forms of image and audio compression today. These codecs not only helped us appreciate the economy of lossy compression but also made us recognize the power of transform coding. But these codecs are laboriously engineered, and the development process is expensive and time-consuming. Many of the compression codecs do not see advances or much use.

One big advantage of lossy data compression via machine learning over the traditional methods is the adaptability of a neural network. For example, one can train a neural network suited to only classical music which may outperform other compression schemes but not necessarily work well with rock music. The neural network learns underlying particulars depending on the data it was trained on. Neural networks are also relatively cheap and easy to develop compared to traditional methods.

While neural networks provide a novel way to perform transform coding, many parallels can be drawn between the traditional transform coding meth-

ods and the machine learning concepts. This chapter will discuss lossy audio compression and the use of transform coding. By surveying recent related work in the field, this chapter will connect the concepts of traditional transform coding to machine learning.

1.1 Problem Statement

1.1.1 Lossy Compression

We denote the process of lossy compression by a non-invertible function:

$$\hat{y} = Compress(x)$$

The compressed data $\hat{y}$ contains less information than the original data x; therefore, $\hat{y}$ has a smaller data size and can be stored or transmitted efficiently. The *Decompress* function attempts to reconstruct x:

$$\hat{x} = Decompress(\hat{y})$$

Rate-distortion theory address the trade-off between the two key elements of lossy data compression: rate and distortion. The goal is to send the least possible amount of data while maintaining a low distortion that is suited to a particular use.

Rate is the measure of how much data is left after compression. Without delving too much into information theory, we will naively express it as a ratio

of the size of the original data to that of the compressed in a percentage:

$$R = \frac{\text{size}(\hat{y})}{\text{size}(x)} \times 100\%$$

Size is usually expressed as the number of bits needed to store/transmit the data. Smaller rate means the compressed data is smaller than the original.

- If $R < 1$, $\text{size}(\hat{y}) < \text{size}(x)$, the data is compressed and takes up less storage.
- If $R > 1$, $\text{size}(\hat{y}) > \text{size}(x)$, there is more data than before and it takes up more storage. When this happens, the compression function is usually a generative model and has its uses which will be discussed in Section 1.3.
- Unlike lossless compression which is bound by information entropy, lossy compression has no theoretical bound on the $\text{size}(\hat{y})$. By throwing away enough information, it is possible to attain $R \ll 1$.

Distortion is the measure the difference between the decompressed and the original data. For now we will denote distortion as

$$D = d(x, \hat{x})$$

There are various methods of measuring distortion. Often in the "audiophile" world, listeners loosely use the term "audio fidelity" to imply the measure of distortion. "Fidelity" encompasses noise, total harmonic distortion, frequency response of the system, and other characteristics; these measurements can all reflect human-perceived differences between systems in audio listening. In the case of audio compression, the ideal D is an all-encompassing

psychoacoustical measure that perfectly simulates an individual's auditory system.

A trade-off between rate and distortion exists for lossy compression problems. Typically data compressed at a low rate retains less information and the distortion is higher. The goal of designing a good audio compression technique is to throw away most data in a way that is least perceptible to the listener. See Figure 1.1 for an example rate-distortion trade-off plot. For

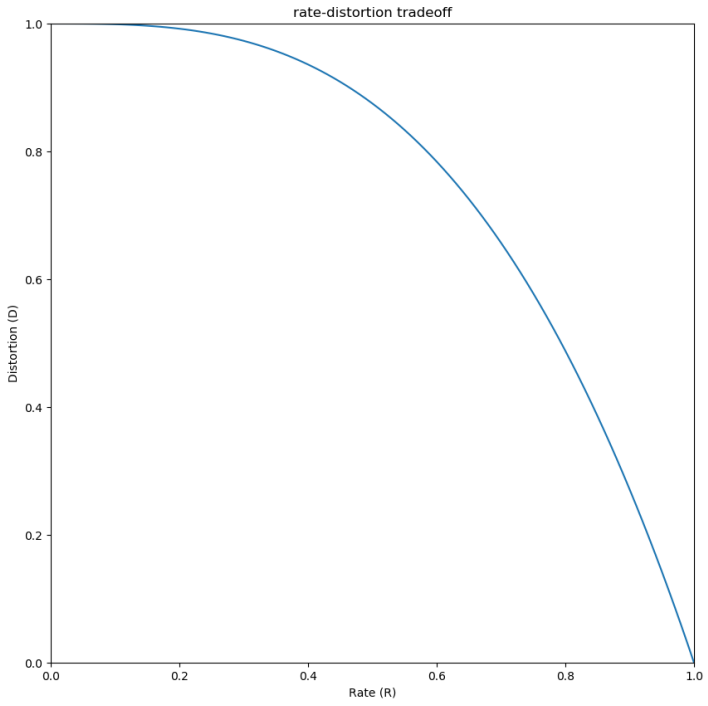


Figure 1.1: The trade-off between rate and distortion.

some measure $D(x, \hat{x})$, $D = 0$ when there is no distortion and increases to $D = 1$ for high distortion. The blue line is the theoretical best compression rate achievable limited by information entropy. All non-ideal compression schemes reside above the blue line.

1.1.2 Transform Coding

Transform coding is essential to the efficiency of MP3 and JPEG encoding. An invertible and linear transform/mapping T, usually one akin to the Fourier transform, maps the raw data to the transform domain in which the compression function operates. For example, the following is a transform that maps data $x(t)$ from the t domain to the τ domain using a basis function $b(t, \tau)$:

$$F(x(\tau)) = \int f(x(t))b(t, \tau) \cdot dt$$

The transform is reversible for a perfect reconstruction:

$$f(x(t)) = \int F(x(t))b_i(t, \tau) \cdot d\tau$$

For the case of discrete data, we can have a similar transform from the n domain to the k domain:

$$F(x[k]) = \sum_{n=-\infty}^{\infty} f(x[n])b[n, k]$$

$$f(x[n]) = \sum_{k=-\infty}^{\infty} F(x[k])b[n, k]$$

There are countless different mappings; an exemplary transform domain will have the following two characteristics:

- Energy compaction due to de-correlation: the basis functions are decorrelated; correlated $x[n]$ gets decorrelated in the transform domain, and energy of $X[k]$ is concentrated in only a few k's. Therefore most of the data resides in the least possible space in the transform domain and more data can be thrown away during compression.

- Ease of data analysis in the transform domain: this enables targeted discarding of information according to the application.

For example, MP3 uses discrete cosine transform type VI (DCT-IV). The DCT has a high energy compaction, close to that of the Karhunen–Loève transform (KLT). Most of energy is concentrated in low frequencies. Engineers have a good understanding of audio in the frequency domain and can easily analyze the data using psychoacoustic models that represent human sensitivity to parts of audio data. The DCT allows the MP3 to target data that humans cannot hear during compression.

Figure 1.2 is a diagram for the flow of a typical transform compression. Note that with a neural network implementation of the transform coding, we are not restricted to linear transformations.

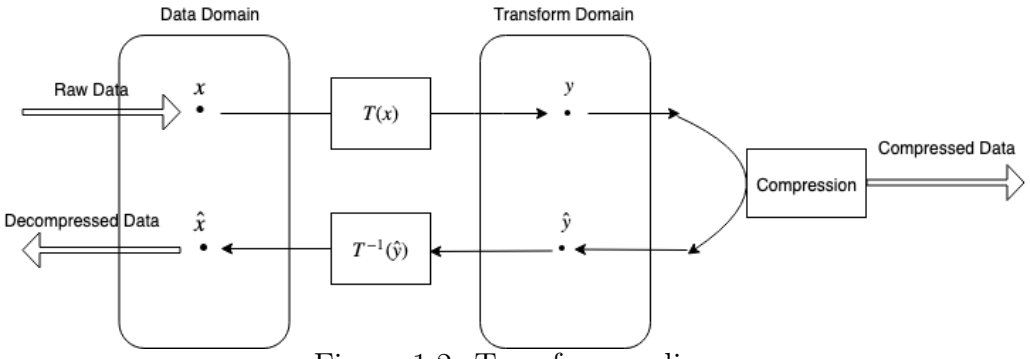


Figure 1.2: Transform coding.

1.2 Audio Compression and Quantization

Before we discuss the compression of audio, we need to understand how uncompressed audio is stored and how the size of audio data is calculated.

1.2.1 Digital Audio storage

Sound is produced by vibrations and travels through air via pressure waves; the pressure waves vibrate our ear drums once they reach our ears. To record audio, microphones capture the change of amplitude of the vibrations over time. This produces an analog waveform $x(t)$. The range of $x(t)$ is usually $[-1, 1]$ representing the maximum amplitude of the vibration available to the microphone. To reproduce the sound, speakers are driven to vibrate with an amplitude according to $x(t)$. To store audio digitally, $x(t)$ is digitized by an analog-to-digital converter (ADC). The typical ADC uses pulse code modulation (PCM) to encode $x(t)$ into $x[n]$.

PCM has two properties: sample rate and bit depth. The 16 bit PCM, commonly used for audio CDs, samples at 44100 Hz with a bit depth of 16 bits. Every 1/44100 seconds, the ADC registers the amplitude of $x(t)$ with an accuracy of 16 bits, producing one sample of $x[n]$ that can be stored digitally. Figure 1.3 is an example of the PCM encoding process. The limit of human hearing is approximately 20 kHz. According to Nyquist sampling frequency, as long as the ADC samples $x(t)$ at more than twice 20 kHz, we can perfectly recover all the audible parts of $x(t)$ without losing audible information. However, we cannot represent real numbers with infinite precision using only 16 bits, so there is an error introduced by the ADC called quantization error. We will discuss this error in detail in Section 1.2.2.

It is natural to measure time series digital data such as audio using bit rate, the number of bits needed every second to store the audio data. The standard rate of uncompressed "original" studio recordings stored in CDs

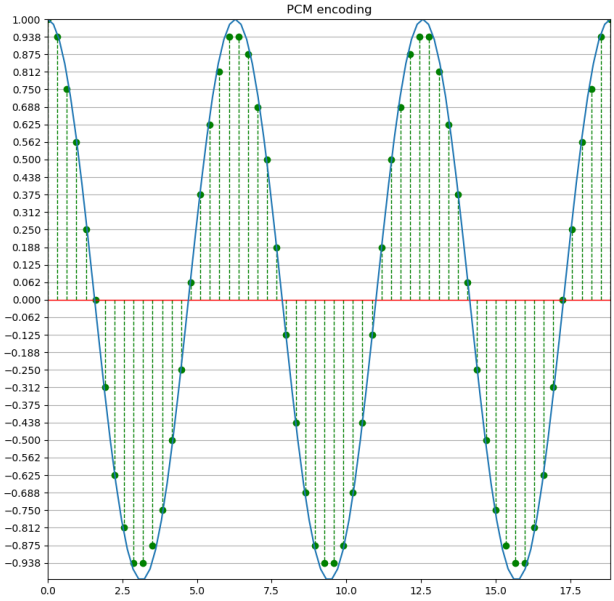


Figure 1.3: 16 bit PCM.

sampled using 16 bit PCM is 1,411,200 bits per second or 1,411 kilobits per second (kbps): two channels corresponding to the left and right, each having a sampling rate of 44,100 Hz, and each sample having a bit-depth of 16.

$$2(\text{ch}) \times \frac{1}{44100}\text{samples/sec} \times 16\text{bit} = 1411200(\text{bits/sec})$$

The actual bit rate for different encoding methods varies. MP3 has several different bit rates depending on the compression scheme used, ranging from high quality 320 kbps, to low quality 64 kbps, to the typical 160 kbps used in the most popular streaming services.

1.2.2 Quantization

As shown in the PCM encoding process, quantization is the irreversible process of mapping multiple values to one singular value. Figure 1.4 is an example of the quantization function that performs decimal rounding.

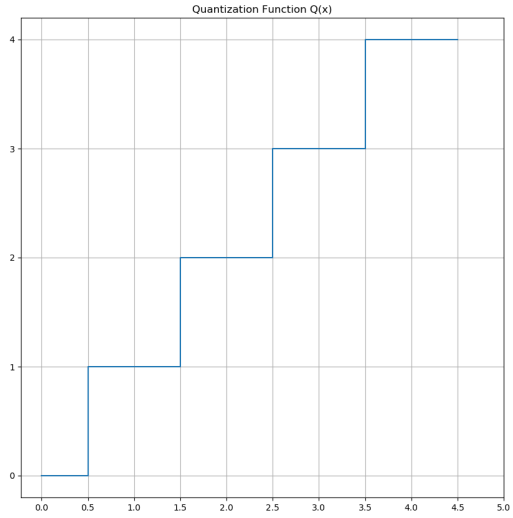
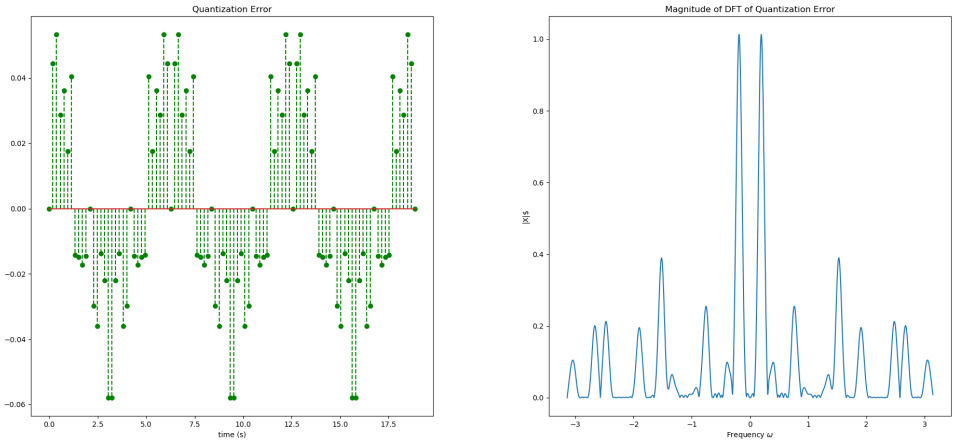


Figure 1.4: Quantization function.

Quantization is often used in lossy compression to throw away information. The distortion it produces is called quantization noise. Using Figure 1.3 as an example from the previous section, at time $t = nT$, corresponding to sample n, the error $e[n]$ caused by the quantization process is:

$$e[n] = x(t) - x[n]$$

What we effectively hear is the original sound mixed with quantization error $x[n] = x(t) - e[n]$, producing noise outside the original frequencies of $x(t)$ as seen in Figure 1.5.



(a) Quantization noise $e[n] = x[n] - y[n]$. (b) The magnitude spectrum of $e[n]$

Figure 1.5: The quantization noise and its magnitude spectrum.

1.2.3 Comparing Traditional Transform Coding and Machine Learning

In this section we will use MP3 as an example of the traditional transform coding method and link it to machine learning concepts. Figure 1.6 gives an overview of the encoding process of MP3.

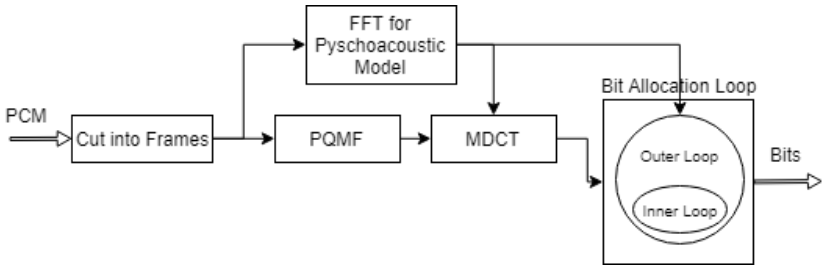


Figure 1.6: MP3 encoding process.

1. The raw PCM audio data is cut into frames for a lapped transform. This is the usual method of processing audio data; see Section 2.1.2 for detailed discussion of lapped transform.

2. Perform fast Fourier transform (FFT) on each frame for frequency domain psychoacoustic modeling. The psychoacoustic model will be used in the MDCT and quantization steps.

3. A two-stage filter bank is applied to the raw PCM audio frame: the 32 band Pseudo Quadrature Mirror filterbank (PQMF) is cascaded with a MDCT filterbank. Based on psychoacoustics, the MDCT uses 18 subbands for steady-state tones for better frequency resolution, or 6 subbands for transients for better time resolution. The outputs of the hybrid filter banks are sorted into scale factor bands which will be quantized.

4. Each of the scale factor bands has its own adjustable gain. Bands that are more "important" are scaled by higher gain, and vice versa. The "importance" of a scale factor band is determined by the psychoacoustic model of the masking tones. Masking tones make the quantization noise in some scale factor bands harder for humans to hear, and these bands use fewer bits for quantization at a cost of increased imperceptible quantization error. Ex: After quantizing the number 1000 with a uniform quantizer with bin size of 256 (8 bits), $floor(1000/256) = 3$, the quantized value has a quantization error of $\frac{1000-256*3}{1000} = 0.232$. If the number 1000 is multiplied by a gain of 2 before quantization, $floor(2 * 1000/256) = 7$, the quantized value will have a smaller quantization error: $\frac{2000-256*7}{2000} = 0.104$.

5. The quantized scale factor bands are further compressed using Huffman coding. The bit allocation loop contains an inner loop and outer loop, and they control the number of bits used for Huffman coding and the gains of the scale factor bands. The inner loop adjusts a global gain

until the Huffman coding produces a small enough bit rate set by the user. The outer loop adjusts the scale factor of each scale factor band. The inner loop is nested in the outer loop to adjust the rate according to the new scale factors. The bit allocation loop is run until the quantization noise is below the masking threshold in each scale factor band.

There are some similarities between the methods used in MP3 encoding and frequent notions in machine learning that are important in designing a neural network implementation of lossy audio compression.

- MP3 uses transform coding to compress and encode data in transform space, using filter banks and discrete cosine transforms. The concept of latent space representation of data is commonly used in encoding and decoding neural networks.

- DCT filter banks are similar to applying different kernels for a convolutional neural network.

- Discrete cosine transforms have high energy compaction close to that of Karhunen–Loève transform (KTL). The KTL is also known as the principle component analysis (PCA) and happens to be the solution of an autoencoder with linear activations.

- The inner and outer loops of the bit control loop in the MP3 encoder adjust the trade-off between rate and distortion. We can model that trade-off as a loss function for the neural network

$$L = R + \lambda D$$

1.3 Survey of Related work

1.3.1 Image Compression via Neural Network

End-to-end optimized image compression was once state-of-the-art image compression in the machine learning world, reporting consistently better performance than JPEG2000 image compression at similar bit rates. Most notably, it gets rid of blocking artifacts of JPEG2000. Balle et al.[1] present a great overview of nonlinear transform coding which heavily influenced this thesis, and they also link generative neural nets to nonlinear transform coding. In a generative model, the learned latent space representation is a probability distribution that models the data. This latent space has a high rate and explains fluctuations and variations of the data very well. Thus samples in the data space can be generated by sampling the latent space. On the other hand, the latent space for data compression needs to model the data at as low a rate as possible while explaining most of the perceptible fluctuations and variations.

Other than [1], there are several similar works on image compression [2]–[6]. The recurrent elements in these works are:

- a nonlinear transform on the image to bring the image into a latent space
- a bottle-neck to reduce the dimension of the latent space during transform
- a quantization in the latent space
- an estimate of the gradient of the quantization process during the gra-

dient descent

- an inverse transform on the compressed data to bring the image back to the data space

This thesis employs similar elements and overall data flow.

One particular paper, Toderici et al. [6], has some novel elements. Namely, the authors employ a binarization in code space and they encode the residual error of the first network to reduce quantization error.

1.3.2 Audio Compression via Neural Network

Kankanahalli [7] is one of the few works, if not the only one, on audio compression using a neural network. Kankanahalli uses an end-to-end deep neural network based encoder for speech compression. The network uses mel-frequency cepstral coefficients (MFCC) for a perceptual loss to combat the degradation of high-frequency content. For the quantization, the network adpots a scalar version of the quantization introduced by Agustsson et al. [8]. Kankanahalli reports PESQ measures on par with AMR-WB at various bit-rates and slightly lower preferences during a subjective test.

Kuleshov et al.[9], while they do not use an audio compression neural net, employ a convolutional autoencoder (CAE) as a generative model for upsampling audio. They report better results than DNN and spline upsampling. As described previously, generative models are closely related to compressive models, thus Kuleshov's work shows promise for CAE as a nonlinear transform for audio.

Venkataramani and Smaragdis [10] use an end-to-end network for supervised single-channel speech separation. The innovation of their work is an adaptive front end implemented by a convolutional neural network (CNN). The adaptive front end is in line with our vision for training transforms based on the input audio. However, the adaptive front end only works well for short strides, which increases the size of the latent space drastically.

Chapter 2

System Overview and Techniques

This chapter first presents a high-level overview and the data path of our compression system using Figure 2.1. The system is set up like a traditional transform coding process but with additional layers of nonlinear transformations realized by a neural network, and a binarizer acting as a quantizer. Then this chapter will review the technical background for all the components used in the network.

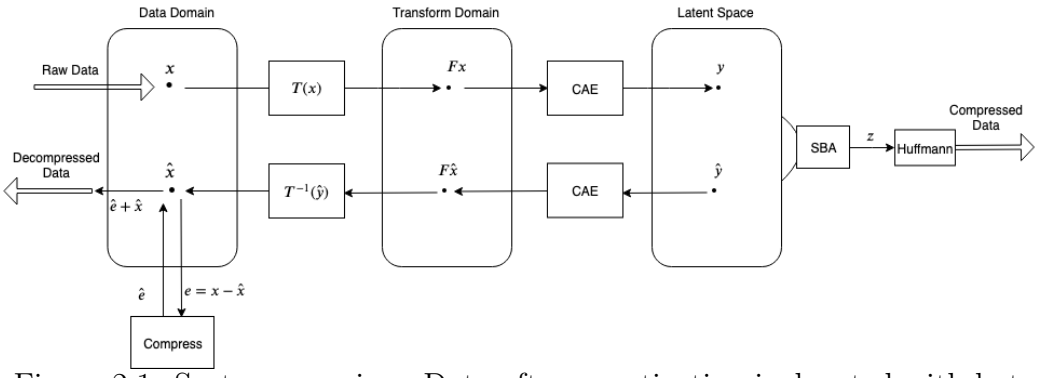


Figure 2.1: System overview. Data after quantization is denoted with hat.

Compression

1. The input raw audio data is framed and windowed to produce audio frames x.

2. An analysis transform is applied $Fx = T(x)$, to take x into the transform domain. The transform decorrelates x, and compacts the energy.

3. Several layers of convolutional autoencoders (CAE) with nonlinear activations are applied to map Fx to the latent spaces and produce y. Some quantization noise is introduced during this process.

4. A stochastic binary activation (SBA) is applied to y for a binary representation of the latent space. This is the quantization process that produces binary bits $\hat{y}$.

5. $\hat{y}$ is grouped into bytes z and encoded by Huffman encoding for further lossless compression.

6. The output is the compressed data stream out of the whole compression system.

Decompression

1. Decompression process begins with recovering $\hat{y}$ from decoding the Huffman codes.

2. $\hat{y}$ is then passed into the decoder part of the CAE to recover the transform domain representation $F\hat{x}$.

3. The synthesis transform is applied $\hat{x} = T^{-1}(F\hat{x})$ to recover data domain signal.

4. Reconstructed audio frames $\hat{x}$ are overlap added to recreate the decompressed audio data for playback.

To reduce the quantization noise, the same compression system is applied, every frame, to the residual noise $e = x - \hat{x}$ as well. The compressed and encoded residual noise is transmitted along with the compressed audio. During

decompression, the decompressed error is added back to the decompressed audio each frame to produce the final output frame: $\hat{x} + \hat{e}$.

The following sections will summarize the background knowledge of all the techniques used for each step of the system. The detailed design and implementation will be designated to Chapter 4.

2.1 Discrete Cosine Transform (DCT)

There are different types of DCTs based on different boundary conditions as derived by Strang [11]. The first DCT discovered in 1974 by Ahmed et al. [12] was a type-II DCT. They established that the DCT has low transform domain variance close to that of the optimal Karhunen–Loève transform (KLT); this means most of the energy of the variance of the transform domain is in the first few coefficients. This characteristic is sometimes called high-energy compaction and is ideal for transform coding because most of the coefficients can be aggressively compressed or even discarded while maintaining low distortion.

There are key characteristics when comparing the DCT with the KLT and the discrete Fourier transform (DFT):

- Although the KLT has the best energy compaction, the transform varies by input. The covariance matrix and eigenvectors need to be calculated on a sample-by-sample basis for any new data. DCT on the other hand is a fixed transform with an explicit transform kernel with no data dependence.
- The DFT is a complex transform; the phase information of audio is

encoded by the phase of the complex number. The DCT is a real transform and encodes the phase information. Thus the DFT needs twice as much storage for the real and imaginary parts and is harder to compress. It would also require a compression network that can deal with complex numbers.

- The DFT assumes periodicity of the signal, thereby introducing discontinuities during truncation and producing artifacts in the frequency domain. The DCT assumes both periodicity and even symmetry.

2.1.1 DCT-IV

DCT is a representation of a finite N point sequence $x[n]$ as a linear combination of cosine functions of different frequencies. We used type-IV DCT for our system, and it is calculated by:

$$X[k] = \sum_{n=0}^{N-1} x[n] \cos\left[\frac{\pi}{N}\left(n+\frac{1}{2}\right)\left(k+\frac{1}{2}\right)\right] \qquad k = 0, \ldots, N-1$$

We denote the basis of the transform as:

$$b[n,k] = \cos\left[\frac{\pi}{N}\left(n+\frac{1}{2}\right)\left(k+\frac{1}{2}\right)\right] \qquad k = 0, \ldots, N-1$$

As with all linear transforms, the DCT can be rewritten as a matrix multiplication: $X = Wx$, where:

$$W(i,j) = \frac{2}{\sqrt{N}} \cos\frac{\pi}{N}(i+\frac{1}{2})(j+\frac{1}{2}) \qquad i,j = 0, \ldots, N-1$$

$W(i,j)$ is i$^{\text{th}}$ row, j$^{\text{th}}$ column of the W matrix.

A key property of the DCT-IV matrix is that, with the added normalizing term $\sqrt{\frac{2}{N}}$, the W matrix is both orthonormal and symmetric. Therefore, the inverse of the W matrix is just itself:

$$W^{-1} = W^T = W$$

The DCT-IV and its inverse can now be calculated using matrix multiplication and easily implemented as a linear layer of the neural network:

$$X = Wx$$

$$x = W^{-1}X = W^T W x$$

2.1.2 Overlap Add Method

Typically, audio cannot be processed in one go. With a sampling rate of 44100 Hz, there would be 441000 samples to process just for 10 seconds of audio. Thus audio processing usually involves block transform, where audio data $x[n]$ is divided into blocks of length L, and processing is done on each block separately. The m^{th} block is:

$$x_m[n] = \begin{cases} x[n + mL], & n = 0, 1, ..., L - 1 \\ 0, & else \end{cases}$$

Therefore

$$x[n] = \sum_m x_m[n - mL]$$

Any linear transforms or filtering we apply to the whole of $x[n]$ now can be instead applied to each block:

$$\begin{aligned} X[k] &= \sum_{n=0}^{N-1} x[n]b[n,k] \\ &= \sum_{n=0}^{N-1} \sum_{m} x_m[n-mL]b[n,k] \\ &= \sum_{m} (\sum_{n=0}^{N-1} x_m[n-mL]b[n,k]) \\ &= \sum_{m} (X_m[k-mL]) \qquad k = 0, \ldots, N-1 \end{aligned}$$

$$x[n] * h[n] = \sum_{m} (x_m[n-mL] * h[n])$$

However, the discontinuities at the block boundaries introduce artifacts during reconstruction when the blocks are pieced back together. Many artifacts are introduced by the DCT since the DCT assumes periodicity and symmetry. A method known as the overlap add method (OLA) builds on the blocked transform and changes it into a lapped transform to reduce these artifacts.

During OLA, an overlap is introduced between blocks, and the next block starts at $H (H < L)$ samples after the start of the previous block. H is called hop size. Each block $x_m[n]$ is also multiplied with a window to reduce the discontinuity at edge:

$$x_m[n] = x[n]w[n-mH]$$

where the window is 0 outside the block:

$$w[n] = 0, \quad n < -\frac{L}{2}, n \geq \frac{L}{2}$$

Finally, take the transform of each new overlapped block to complete the lapped transform. To recover the original signal, take the inverse transform of each block and overlap add them together.

To ensure the overlap add reconstructed data is the same the as original, the window has to satisfy constant overlap add (COLA). The conditions for COLA can be derived:

$$\begin{aligned}
\sum_{m=-\infty}^{\infty} X_m[k] &= \sum_{m=-\infty}^{\infty} \sum_{n=0}^{N-1} x[n]w[n-mH]b[n,k] \\
&= \sum_{n=0}^{N-1} x[n]b[n,k] \sum_{m=-\infty}^{\infty} w(n-mH) \\
&= X[k] \sum_{m=-\infty}^{\infty} w(n-mH)
\end{aligned}$$

Therefore as long as the window satisfies

$$\sum_{m=-\infty}^{\infty} w(n-mH) = 1$$

the OLA will be a COLA and the $x[n]$ can be recovered. A detailed discussion of different window types and their COLA conditions is presented by Heinzel et al.[13]. We tested the most common Hamming window and Hanning window in Chapter 3.

Overall, a higher overlap gives more faithful reconstruction due to the more

redundant information, but higher overlap also increases data size in the transform domain. For $x[n]$ of length N and lapped transformed using hop size H and frame size L, the ratio of the size of the lapped transformed data to the original data is:

$$\frac{\frac{N}{H} \times L}{N} = \frac{L}{H}$$

Therefore a hop size of $L/4$ would increase the data size by 4.

2.2 Convolutional Autoencoder (CAE)

2.2.1 Autoencoder (AE)

Depending on the literature, an autoencoder (AE) is grouped into the sub-branch of either unsupervised or "self-supervised" learning. A typical AE is set up like Figure 2.2, where each line represents a linear function with an activation: $\sigma(Wx + b)$. The goal of an AE is to learn a lossy encoder and decoder pair which are nonlinear transforms that map data from $\mathcal{X}$ (data space) to $\mathcal{L}$ (latent space) and back:

$$E : \mathcal{X} \to \mathcal{L}$$

$$D : \mathcal{L} \to \mathcal{X}$$

The encoder and decoder must minimize the reconstruction error for any given $X \in \mathcal{X}$:

$$E, D = \arg\min_{E,D} \|X - (D \circ E)X\|^2$$

The latent space $\mathcal{L}$ is where the data after the last hidden layer of the encoder and the input of the decoder resides; we do not know what the latent

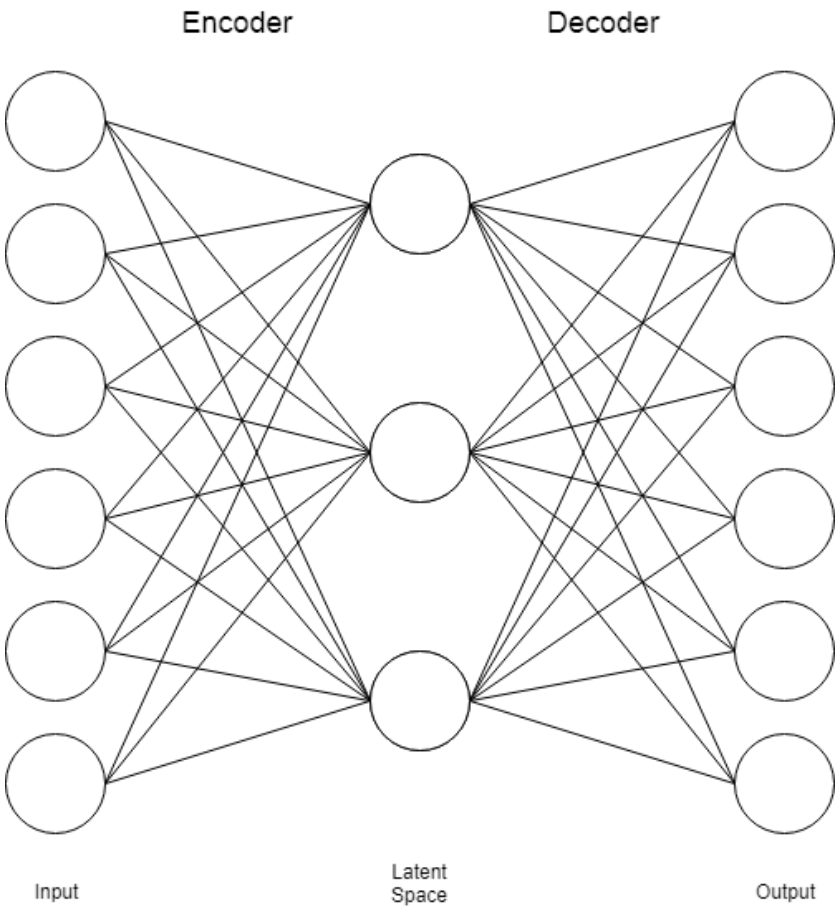


Figure 2.2: Typical autoencoder.

space is without analyzing the weights. Typically, by limiting the number of nodes in the last hidden layer, a bottleneck is imposed, constraining the size of the latent space, and therefore forcing the network to learn a compressed representation of the input data. Think of it as a nonlinear principal component analysis. The challenge of AE is making the network learn meaningful representations rather than just a random unknown representation.

Recall the AE minimizes $\|X - (D \circ E)X\|^2$. We can create a synthetic input $x' \in \mathcal{X}$ and minimize instead $\|X - (D \circ E)X'\|^2$; or we can make a custom loss function $l(x, \hat{x})$. For example:

- Adding random noise at the input $x' = x + n$ creates a denoising AE

where the latent space is resilient to the random perturbations.

- Withholding parts of the input data, by methods such as downsampling, $x' = \text{Downsample}(x)$, creates an upsampling AE.
- Optimizing the network using a loss function $L = l(x, \hat{x})$, where l measures the color difference of an image, creates an AE focused on making sure the outputs have the same color instead of shape.

In the case of audio compression, the target is just the original audio data; the loss function is the interesting part that requires deliberation.

2.2.2 Convolutional Neural Network (CNN)

The AlexNet[14] ushered in the era of convolutional neural networks (CNN). The CNNs are able to capture spatial dependencies of an image through layers of feature extraction via "convolutions" with different filters. We will focus on 1D CNNs with a stride of one to maintain the size. The output $y_q^{(l)}[n]$ of the convolution at layer l at channel q with stride one is:

$$y_q^{(l)}[n] = \sigma(\sum_p \sum_{u=0}^{U-1} K_{p,q}^{(l)}[u] y_p^{(l-1)}[n-u] + b_q^{(l)})$$

U samples around $y_p^{(l-1)}[n]$ from all the input channels p at layer $l-1$ are weight summed using the trainable kernel $K_{p,q}^{(l)}$. This process captures information spatially and is repeated for all output channels q, producing q different $y_q^{(l)}[n]$'s at the output layer l.

For an input layer with length L_{in}, the length of the output layer is $L_{out} = L_{in} + 2 \times \text{padding} - \text{kernel size} + 1$. By having an odd kernel size and a zero padding size of (kernel size $- 1$)/2, we are able to keep the output size

the same as the input. With a bigger stride, the CNN will be downsizing the data substantially; however, we discovered that much of the frequency information is lost with downsizing.

Since we preserved the size of the data, the CNNs can be reversed by either a regular convolution or transposed convolution with the same stride and padding.

2.2.3 Convolutional Autoencoder (CAE)

Convolutional autoencoders are autoencoders that use convolutional layers instead of linear layers for analysis transforms and transposed convolutional layers (or normal convolutional layers) for synthesis transforms. A CAE is able to better capture spatial information compared to an AE because of the convolutional layers; the encoder only extracts features important for reconstructing the input.

The ways to impose a bottleneck for CAE at each layer are:

- Downsampling the output of the convolution at each encoder layer and upsampling at the decoder layers.
- Decreasing the number of output channels at each encoder layer and increasing at the decoder layers.
- Penalizing and zeroing out activations. CAEs using this method are known as sparse autoencoders.

The network in our CAE does not impose a bottleneck, we only take advantage of the nonlinearity of the CAE to map the audio from transform

domain to a latent space better suited for quantization by the stochastic binary activation (SBA).

2.3 Quantization and Compression

2.3.1 Stochastic Binary Activation

A stochastic binary activation (SBA) outputs 1 with probability p and 0 otherwise. We will rely on SBA as our main method of compression because of two major advantages:

- The SBA have an extreme rate of lossy data compression; inputs of 16 bit/32 bit floating numbers will become a 1 bit binary number. Although SBA introduces a lot of quantization noise (see Section 3 for an example of quantization noise produced by SBA), there are methods to reduce it.

- The gradient of a quantization function such as the one in Figure 1.4 is 0 almost everywhere, and would need an estimated gradient based on quantization setup to be used for gradient descent. By putting an activation with a range of $[0, 1]$ in front of the SBA, all data use the same quantization. We do not have to design specific quantization schemes and their gradient estimates for different frequency bins or sub-spaces of the latent space. The CAE will take care of finding a subspace best suited for the SBA.

R2RT [15] is a blog-post comparing different types of gradient estimations for sigmoid/softmax activations with a randomizer; the author calls them Binary Stochastic Neurons. He found that a slope-annealed straight-through

estimation proposed by Gulcehre et al. [16] gave th best results. We will use the slope-annealed straight-through method to estimate the gradient of a scaled hard tanh and a Bernoulli sampler.

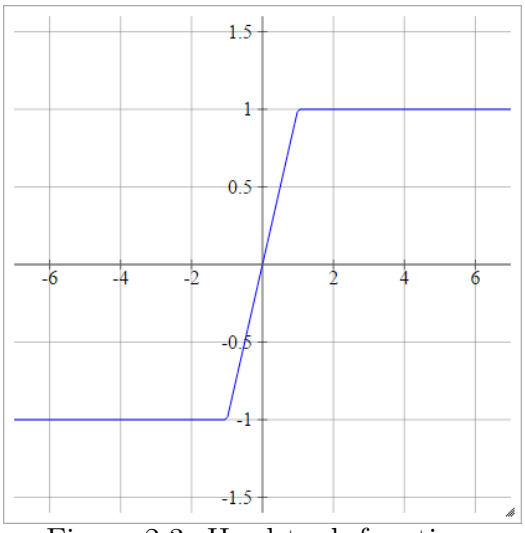


Figure 2.3: Hard tanh function.

The hard tanh function (Figure 2.3) is computationally cheaper to calculate than the tanh, but has issues of gradient saturation:

$$\text{HardTanh}(x) = \max(-1, \min(1, x))$$

$$\frac{\partial \text{HardTanh}(x)}{\partial x} = \begin{cases} 1, & -1 \leq x \leq 1 \\ 0, & \text{else} \end{cases}$$

Our SBA is implemented by scaling the hard tanh to have the output range of $[0, 1]$, and then Bernoulli sampling the output.

$$\text{SBA}(x) = \text{Bernoulli}(\frac{\text{HardTanh}(x) + 1}{2}) = \text{Bernoulli}(\max(0, \min(1, \frac{x+1}{2})))$$

where the Bernoulli sampler produces 1s or 0s depending on the input:

$$\text{Bernoulli}(p) = \begin{cases} 1, & p \text{ percent of the time} \\ 0, & (1-p) \text{ percent of the time} \end{cases}$$

The slope-annealed straight-through gradient estimation consists of two steps. First, proposed by Bengio et al. [17], the gradient of the SBA is estimated by

$$\frac{\partial \text{HardTanh}(x)}{\partial x} = 1$$

Then, the slope of the hard tanh is annealed to increase such that it gradually matches the step function, the deterministic binarization.

2.3.2 Dithering

As we discussed in Section 1.2.2, quantization noise is potentially cyclical, producing noisy artifacts in particular frequencies in audio compression that are very obtrusive to human hearing. Adding random white noise before quantization reduces these audible artifacts and diffuses the quantization error into different frequency bins. This process is called dithering and it minimizes average error.

For example, if we are quantizing the number $x = 2.2$ with a floor function $\lfloor (2.2) \rfloor = 2$, then the quantization error is: $e = \frac{2.2-2}{2} = 0.1$. If we add a uniform noise $U = [0, 1]$ to x before quantization:

- 80% of the time, we are adding $u < 0.8$, $\lfloor (2.2 + u) \rfloor = 2$.
- 20% of the time, we are adding $u > 0.8$, $\lfloor (2.2 + u) \rfloor = 3$.

- Overall, the expected value is $\mathbf{E}[\lfloor(2.2+u)\rfloor] = 0.8 \times 2 + 0.2 \times 3 = 2.2$. So with enough samples, the expected error is $\mathbf{E}[e] = 0$.

Dithering the input to the stochastic binary activation (SBA) reduces the quantization noise.

2.3.3 Huffman Coding

A coding system maps a set of symbols to binary bit strings so that they can be transmitted digitally. Huffman coding is a variable length prefix code. Prefix code is a coding system where no whole code word is a prefix of any other code word, which eliminates any ambiguity when separating the stream of code into code words; therefore, a prefix code is uniquely decodable. Huffman coding exploits the fact that some symbols occur more often than others, assigns fewer bits to these symbols, and thus uses fewer bits overall in data transmission.

In our case, the output bits of the SBA are grouped into 8-bit bytes; each byte is now a symbol. Since the bits are sparse, the most common byte symbol is x00 (x representing hex-decimal). Instead of using the same length of 8 bits for every symbol, the Huffman coding will use only 1 bit to represent x00. We will then proceed to find all the probabilities of all byte symbols and design a byte symbol to code mapping for all the 8-bit symbols. Given that the probability of each symbol i occurring is p_i, the expected length of the Huffman code is:

$$\mathbf{E}[L] = \sum_i p_i l_i$$

where l_i is the number of bits symbol i is coded with.

In our network, since the binary bits are not saved to save computation time, an empirical probability distribution is calculated for each frame for a Huffman coding, and the overall code length is estimated by averaging across multiple frames.

2.3.4 Residual Compression

Inspired by the differential pulse-code modulation (DPCM) which encodes the difference between samples (prediction error) to improve encoding, the same compression network is applied to the reconstruction error of the first network. We call it the residual compression network. It has greatly reduced the noise of the SBA at the cost of roughly doubling the compression rate. The residual from the first network is much smaller in magnitude, thus using a new network to capture the error is ideal.

2.4 Loss Functions

The loss function of a convolutional autoencoder (CAE) controls how the CAE optimizes and what the latent space represents. Recall that lossy audio compression is a trade-off between rate R and distortion D; therefore the loss function L must force the CAE to learn a latent space with a low rate while maintaining a low distortion:

$$L = D + \lambda R$$

where λ is a weighting parameter to adjust the trade-off. The next question to ask is what are good measures for distortion and rate?

Depending on the distortion measure D, the CAE will learn a latent space focused on modelling D, and will lead to the quantization process discarding information based on D. We will divide distortion into 2 parts: structural and perceptual. Structural distortion is a measure of the difference in overall shape of the audio waveform, spectrograms, etc. Perceptual distortion is a measure of the perceived difference in the audio as heard by humans.

The rate measure R should closely represent the actual amount of information in the latent space that will be discarded by the quantization process so that the correct amount quantization is applied to maintain a proper compression rate.

We will use mean squared error (MSE) loss for structural distortion, highpass filter loss and mel-frequency cepstral coefficient (MFCC) loss for perceptual distortion, and sparsity loss for rate measure. The effects of these loss functions are demoed in Chapter 3.

2.4.1 Sparsity Loss

Since the binary bits of the output of the stochastic binary activation (SBA) are grouped into bytes for Huffman encoding, by enforcing sparsity of the binary bits, we generate more 0's which roughly estimates to a more concentrated byte symbol distribution; i.e., a high sparsity after SBA will correspond to more x00 bytes. With a more concentrated distribution of symbols, a better Huffman encoding rate can be achieved. The sparsity loss is a L1 norm:

$$l_{sparse} = \sum |z[n]|$$

where $z[n]$ is the binary outputs of the SBA.

The disadvantage of using the L1 loss for sparsity is the lack of control over the actual compression rate; it measures the actual amount of information the SBA discards but not the rate of the Huffman encoding. Furthermore, the weighting of the loss must controlled; too great of a weight will unduly degrade audio quality or even zero out the outputs of the SBA.

2.4.2 Mean Squared Error (MSE) Loss

To maintain the general structure of the audio after compression and decompression, we will use the mean squared error (MSE) loss of the original audio and the reconstructed audio:

$$l_{MSE} = \mathbf{E}[(x - \hat{x})^2]$$

2.4.3 Highpass Filter (HPF) Loss

Due to the nature of some genres of audio, they lack information in high frequencies. The small magnitudes of the DCT of the high frequencies are hard to quantize, and this introduces a lot of quantization noise. This problem is similar to the lack of high spatial frequency information in 2D CAEs trained on images. A mitigation was proposed by Ichimura [18], called the spatial frequency loss. He applies a Laplacian filterbank on the images and uses the MSE of the features in the subbands as a loss function.

In order to boost the high frequencies of the audio for our CAE, we will apply a highpass filter h to the original and reconstructed audio, and the MSE of

the two is added to the overall loss as a loss called the highpass filter (HPF) loss:

$$l_{HPF} = \text{MSE}(h(x), h(\hat{x}))$$

2.4.4 Mel-frequency Cepstral Coefficients (MFCC) Loss

We will use mel-frequency cepstral coefficients (MFCC) to model the perceptual loss of the compressed audio along with HPF loss. We will use the frame-wise MSE of the MFCC of the compressed vs. original audio.

$$l_{MFCC} = \text{MSE}(\text{MFCC}(x), \text{MFCC}(\hat{x}))$$

To calculate the MFCCs:

1. The audio is segmented into frames and a window is applied.
2. The square of the Fourier transform of each frame is calculated. This is the power spectrum: $|\mathcal{F}\{x\}|^2$.
3. The power spectrum is mapped onto the mel scale using triangular filter banks.
4. The energy in each filter is summed to get a coefficient for each mel-frequency bin.
5. The bins are highly correlated. To reduce the number of bins needed, a DCT of the log of the bins is taken to decorrelate them. See Section 2.1 for details of decorrelation.
6. Keep a few of the coefficients after the DCT.

The mel scale used in MFCC puts more emphasis on lower frequencies to match human audio perception. MFCC models the envelope of the time

power spectrum which is representative of the vocal tract; thus MFCC has been very popular and successful in feature extraction for speech. Much progress has been made in understanding the use of MFCC for music features. Logan [19] discovered that it is successful for classifying music vs. speech, and DCT is still comparable to KTL in decorrelating the spectra. Li et al. [20] report that music genre classification accuracy using MFCC as feature extraction is comparable to that using FFTs.

2.5 Normalization

The output of the DCT, also the input of the CAE, may have various different magnitudes. Passing data with different magnitudes through the CAE to be convolved with the same filters of a particular magnitude contributes to noise and potential reconstruction errors during inverse DCT. We have taken various precautions to avoid this issue and to reduce the quantization noise. A combination of normalizing the DCT output and batch normalization in the CAE is used after some experimenting. See Chapter 3 for our experiment results on normalization schemes.

2.5.1 Normalization of Transform Domain

Each frame of the output of the DCT is divided by its norm, which is then cast into a 16-bit float and also multiplied back at the end of the decoder of the CAE. The norm of each frame will also be added to the calculation of the compression rate as it is needed during decompression.

We also experimented with subtracting the mean of each frame, but it brings no visible/audible improvement as seen in Chapter 3.

2.5.2 Batch Normalization

Originally introduced by Ioffe and Szegedy [21] as a method to speed up training, batch normalization whitens each mini-batch $x_{(b)}$ using its mean and variance:

$$\hat{x}_{(b)} = \frac{x_{(b)} - \mathbf{E}[x_{(b)}]}{\sqrt{\text{Var}(\hat{x}_{(b)})}}$$

We take advantage of mini-batch whitening to normalize the first few input layers of the CAE. During inference, a running mean and variance are used instead for the batch norm to apply the same transform.

2.5.3 Weight Normalization

Salimans and Kingma [22] invented weight normalization to reparameterize the weight vector in order to speed up training:

$$w = \frac{g}{||v||}v$$

This reparameterization decouples the Euclidean norm of the weight vector $||w|| = g$ from the direction vector v such that the step sizes are much more controlled during gradient descent:

$$\nabla_g L = \frac{\nabla_w L}{||v||}v \qquad \nabla_v = \frac{g}{||v||}\nabla_w L - \frac{g\nabla_g L}{||v||^2}v$$

We attempted to use weight normalization to control the norm of the weights along with l2 regularization. However, as we found out the main cause of artifacts is the un-normalized DCT coefficients, we left it in to speed up training.

Chapter 3

Experiments

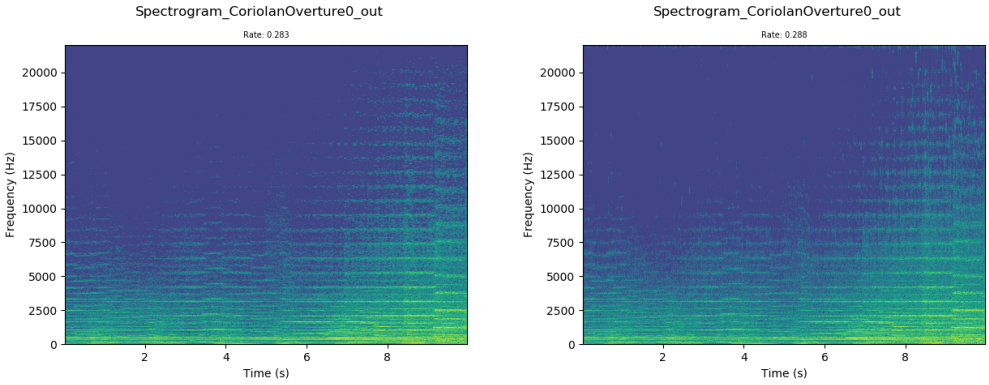
This chapter provides side-by-side comparisons of the spectrograms of the audio compressed with different methods and techniques we experimented with when designing the network, and these experiments dictate how the final network is designed in Chapter 4. This chapter does not report any detailed results of the actual performance of the network; the reports are delegated to Chapter 5.

3.1 Constant Overlap-add (COLA)

We will experiment with different setups for COLA and find one best suited for our task of audio compression.

3.1.1 Frame Size and Psychoacoustics

We tested our network using various frame sizes and report the results of the comparison of frame size 2048 vs. frame size 512 in Figure 3.1. The noise distribution in the spectrogram of the figure is a manifestation of the classic time-frequency trade-off of based on the entropic uncertainty principle. The uncertainty principle states that a function cannot be both time limited and band limited. In a lapped Fourier transform, a wide window yields good frequency resolution but poor time resolution; a narrower window yields good time resolution but poor frequency resolution.



(a) Frame size 2048. (b) Frame size 512.

Figure 3.1: Comparison of frame sizes.

Let us closely examine the distribution of the quantization noise of our neural network. The spectrogram of the smaller frame size has better onsets of notes because there is less noise diffusion across time frames. The spectrogram of the bigger frame size has better steady state tones because there is less noise diffusion across frequency bins.

Our decision to use the bigger frame is based on the two auditory masking phenomena of psychoacoustics well studied by Egan and Hake [23]: frequency masking and temporal masking.

- If two tones of different frequencies f_0 and f_1 that are close to each other are played at the same time, the louder tone f_0 will have a masking effect that will cover the softer tone f_1, rendering the softer tone inaudible. The shape of the mask varies based on the loudness and frequency of the tone; usually it spans the range of only a few hertz before and after f_0.

- If two tones of same frequencies f_0 are played one after another at t_0 and t_1 in close succession, the louder tone at t_0 will have a masking effect that will cover the softer tone at t_1, rendering the softer tone

inaudible. The shape of the mask varies, usually occurring 20 ms in front of t_0 and 100 ms after t_0.

With the masking effect in mind, we examine the spectrograms of the frame size 512 and 2048:

- The frequency difference between DCT bins with frame size of 512 is 44100/2/512 = 43 Hz, and no frequency masking will occur between the bins at such a large difference. We can see the quantization noise smeared across 43 Hz in the spectrogram.

- The time difference between frames of size 512 and hop size of 256 is $256/44100 \times 1000 = 5.8$ ms, well within the range of 20 ms for temporal masking to occur.

- The frequency difference between DCT bins with frame size of 2048 is 44100/2/2048 = 10.8 Hz, and some frequency masking will occur between the bins depending on the frequency center.

- The time difference between frames of size 2048 and hop size of 1024 is $1024/44100 \times 1000 = 23.2$ ms, still within the range of 20 ms for temporal masking to occur.

In conclusion, the quantization noise is much less audible in the 2048 frame size implementation due to both temporal and frequency masking while the 512 frame size implementation will only have effective temporal masking.

3.1.2 Hops and Windows

We have established in Section 2.1.2 that during OLA, the ratio of the size of the lapped transformed data to the original data is $\frac{L}{H}$, where H is the hop

size and L is the frame size. Therefore, in order to prevent encoding extra data, the hop size must be kept as large as possible (overlap as small as possible). A detailed comparison of different windows applied during OLA is provided by Heinzel et al. [13]; we will focus on the two concepts they introduced: "amplitude flatness" and "correlation of frame".

- During OLA, each point of the data is assigned a weight created by multiplying the window before being added back together. We want the weight to be the same for all data points at all samples to achieve COLA. With COLA, we will have a perfectly flat amplitude. "Amplitude flatness" is the measure of how close we are to COLA.

- Because there is much overlap of the frames, each frame is correlated after applying the windows. The greater the overlap, the more redundant data there is in each frame and the more correlated each frame is.

Based on the plots of percent overlap vs. correlation of frames and amplitude flatness provided by Heinzel et al., the best two windows with least needed overlap to achieve amplitude flatness of one and least correlation between frames are the Hamming and Hanning windows. The Hamming window of length N is:

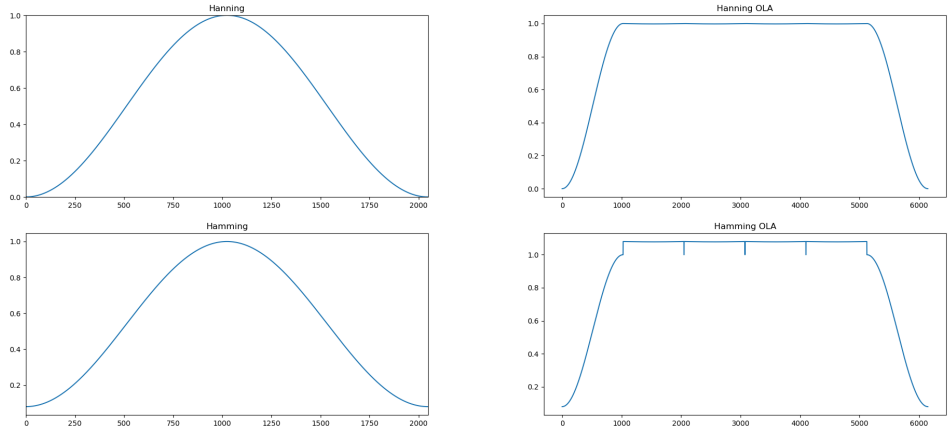
$$w(n) = 0.54 - 0.46\cos\left(\frac{2\pi n}{N-1}\right) \qquad 0 \leq n \leq N-1$$

And the Hanning of length N is:

$$w(n) = 0.5 - 0.5\cos\left(\frac{2\pi n}{N-1}\right) \qquad 0 \leq n \leq N-1$$

Notice the Hamming window does not attenuate all the way to 0 at edges;

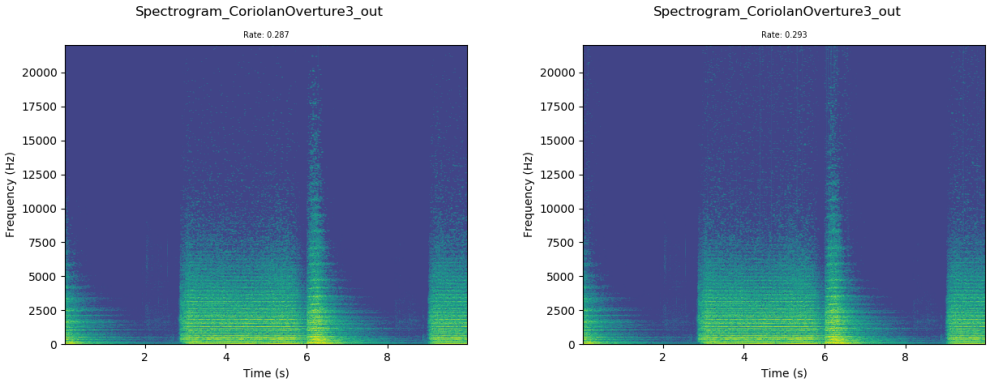
this makes the Hamming window prone to artifacts during reconstruction using OLA as demonstrated in Figure 3.2. We compare the results of overlap adding 4 frames using frame size of 2048 and hop size of 1025, which is only one off from the COLA condition for both windows. Notice the artifacts created by frame discontinuities in the Hamming window reconstruction.



(a) Hamming vs Hanning window. (b) Hamming vs Hanning window after OLA.

Figure 3.2: Comparison of Hamming and Hanning window.

Since the frames of audio are processed separately in our neural network, there will be discontinuities between frames even if we satisfy the COLA condition. These discontinuities will be amplified by the Hamming window and attenuated by the Hanning window. Results of experimenting with different windows, shown in Figure 3.3, support our hypothesis. Notice the long vertical lines in the highest frequencies of the spectrogram from using Hamming window; these lines are the artifacts created from discontinuities and can be heard distinctively as a loud "pop".



(a) Hanning window. (b) Hamming window.

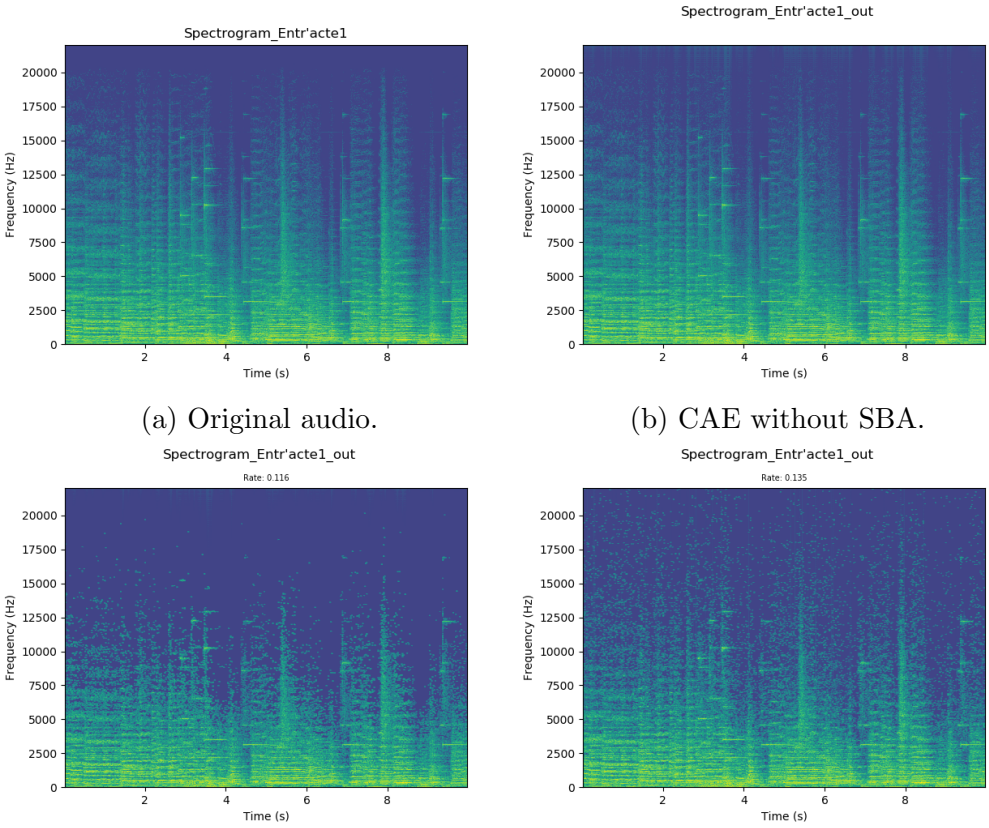
Figure 3.3: Comparison of audio compression using Hanning and Hamming window.

3.2 Convolutional Autoencoder (CAE) and Stochastic Binary Activation (SBA)

We found that we can get a good reconstruction using our CAE by having a latent space with only four channels, but as we implemented the SBA, the quality degraded and lots of noise was introduced despite tuning the network for less compression.

Three types of artifacts can be seen in Figure 3.4, and they all correspond to distinctly audible noises. The vertical bars are edge discontinuities of frames, heard as a "pop". The grid-shaped noise is due to incorrect scaling of the input to the inverse DCT, and can be heard as a sustained "bzzt" sound. The huge amount of musical noise and white noise is introduced when the quantizer is not fine enough to properly quantize small values.

Most of the high-frequency information is lost during quantization due to the dynamic range difference between the lower and higher frequencies. To make SBA work for audio compression, we tested a few different techniques.



(a) Original audio.

(b) CAE without SBA.

(c) CAE with SBA.

(d) CAE with SBA at higher rate.

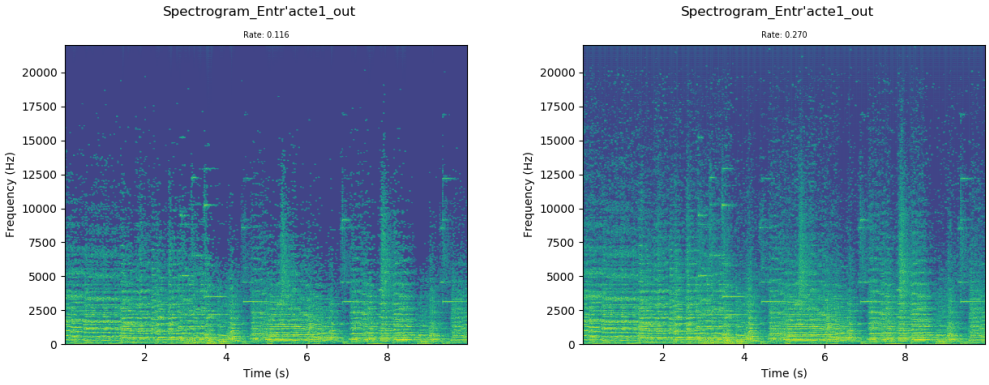
Figure 3.4: Comparison of the CAE and SBA.

3.2.1 Noise Reduction

This section will report the noise removal abilities of the following methods:

1. Encoding the residual
2. Dithering the input to SBA
3. Batch normalization
4. Normalizing the DCT before the CAE

First and foremost, encoding the residual greatly improves audio quality at the cost of a much lower compression rate as seen in Figure 3.5. Much more of the higher frequency information with small magnitude values is being reconstructed rather than thrown away because the residual audio would



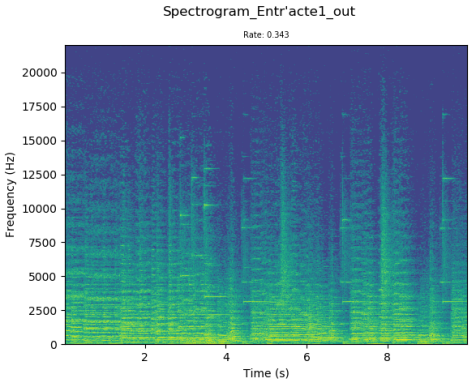
(a) With SBA. (b) With SBA and residual encoding.

Figure 3.5: Experimenting with effects of encoding the residual.

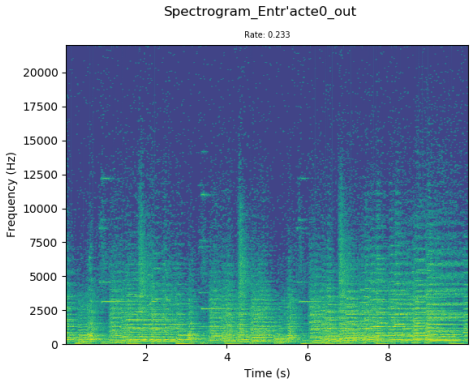
have a much more uniform dynamic range in all the frequencies. Encoding the residual also leads to a bigger data size than the original because the quantization is finer and discards much less data. However, the three types of distinct noises are still in the residual encoding network. To remove these types of noise, we experimented with improving our compression network so that both the main and residual encoding network perform much better.

The results of our experimentation for the dithering, batch normalization, and normalizing DCT are shown in Figure 3.6. Figure 3.6a is the reference with all the methods applied. Neither of the methods adds a lot of size to the compressed data, so the rate has almost no difference. The three methods all remove most of the quantization noise in the sparse high frequencies. And the parts with low energy are falsely boosted if no batch normalization is applied.

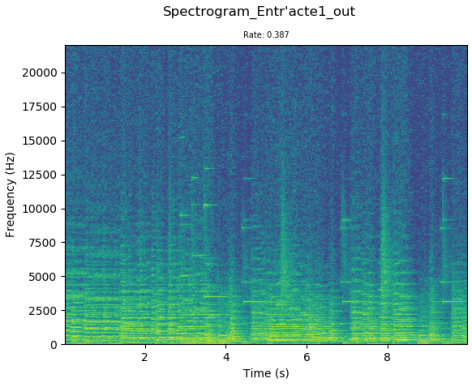
Last, we attempted to subtract the mean of the DCT before feeding it into the CAE. However there were no perceptible improvements as shown in Figure 3.7



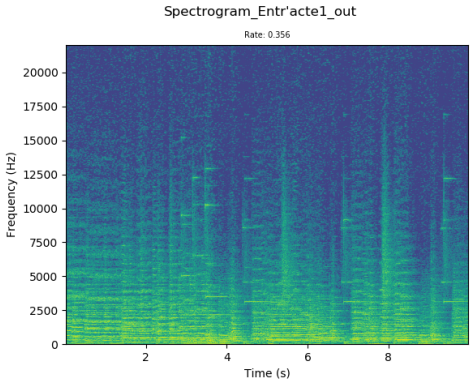
(a) Our Setup.



(b) Without DCT normalization.

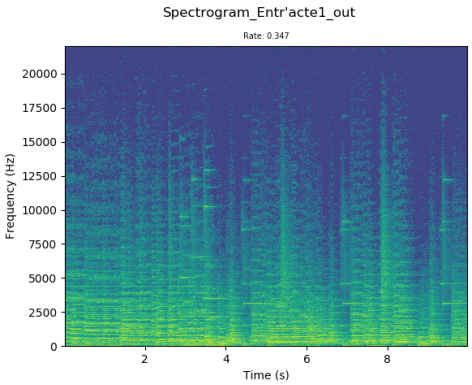


(c) Without batch normalization.

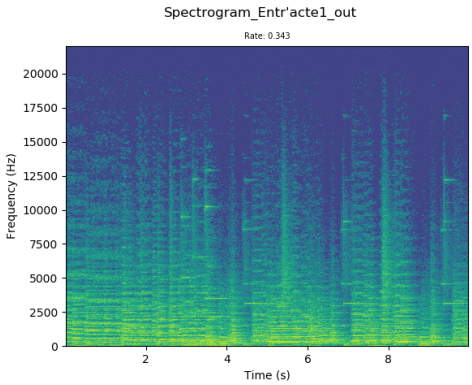


(d) Without dithering

Figure 3.6: Without various noise removal.



(a) With subtracting mean of DCT.



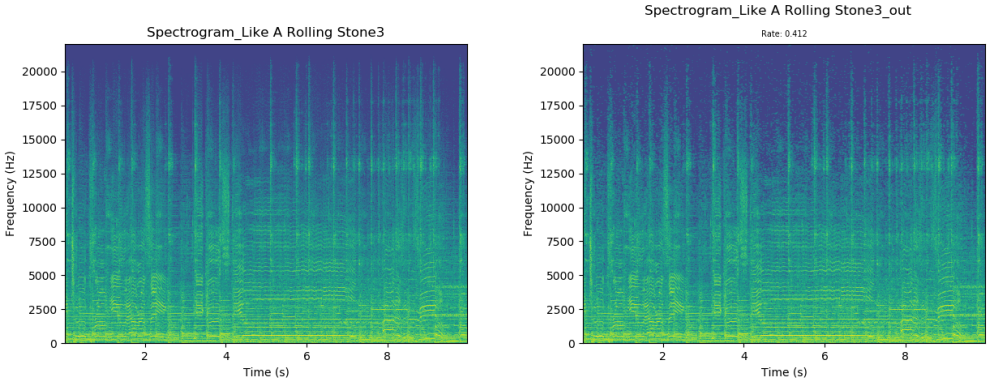
(b) Without subtracting mean of DCT.

Figure 3.7: Comparison of subtracting the mean of DCT.

3.3 Loss Functions

Recall from Section 4.3 that our loss includes: mean squared error (MSE) loss for structural distortion, highpass filter loss and mel-frequency cepstral

coefficient (MFCC) loss for perceptual distortion, and sparsity loss for rate measure. In this section we will experiment with the effects of the different loss functions. Figure 3.8 is the spectrogram of the decompressed audio when all the losses are implemented and will serve as a baseline for comparing the effects of not using each loss.

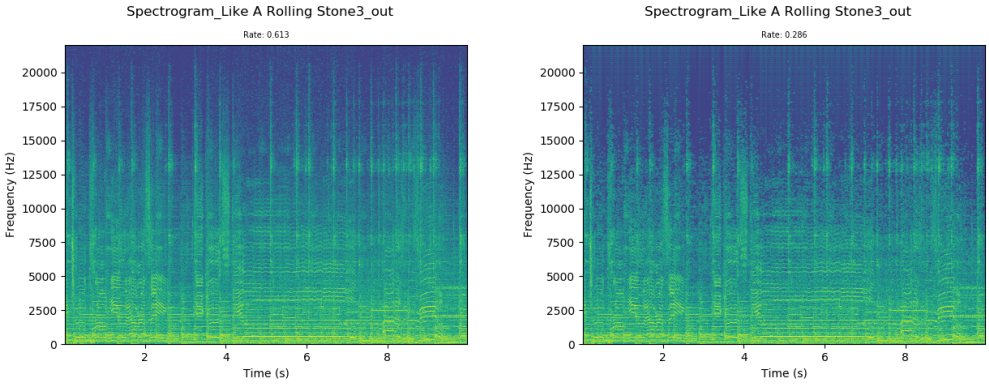


(a) Original. (b) Our setup.

Figure 3.8: Comparison of original to our setup.

We first experiment with different sparsity losses in Figure 3.9. Although sparsity loss controls the final compression rate of our network, we experienced the disadvantage of the sparsity loss. We can see in Figure 3.9b that if we went overboard with the weight on the sparsity loss, we will have too many 0's in the subspace for low magnitude components and will introduce noise for the inverse DCT.

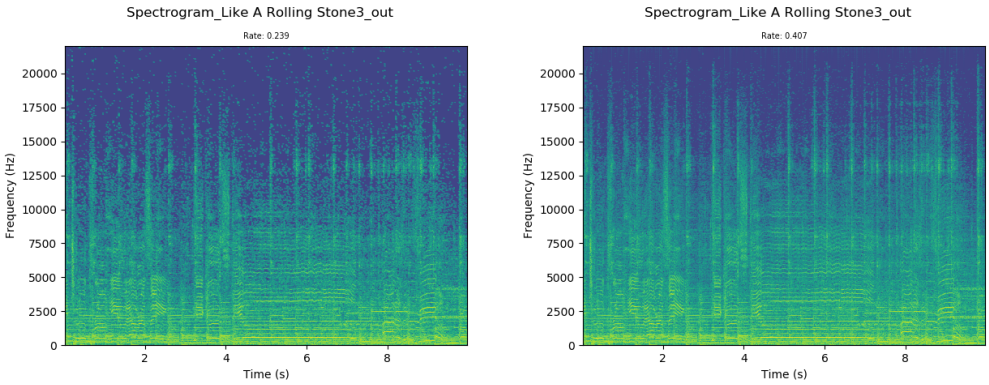
We now experiment with dropping one of the perceptual losses in Figure 3.10. We can see that highpass filter loss is the most important loss in boosting high-frequency audio quality at the cost of a higher compression rate. Because the high frequencies have lower dynamic range than the low frequencies, the quantization will become finer and will throw away less data



(a) No sparsity enforcement. (b) High sparsity loss weight.

Figure 3.9: Comparison of normalizing.

in the high frequencies.



(a) No HPF loss. (b) No MFCC loss.

Figure 3.10: Without using perceptual loss.

Chapter 4

System Design and Implementation

This chapter presents the detailed design and implementation of the network by following the overall data flow as seen in Figure 4.1: from the compression network on the top half to the decompression network on the bottom half.

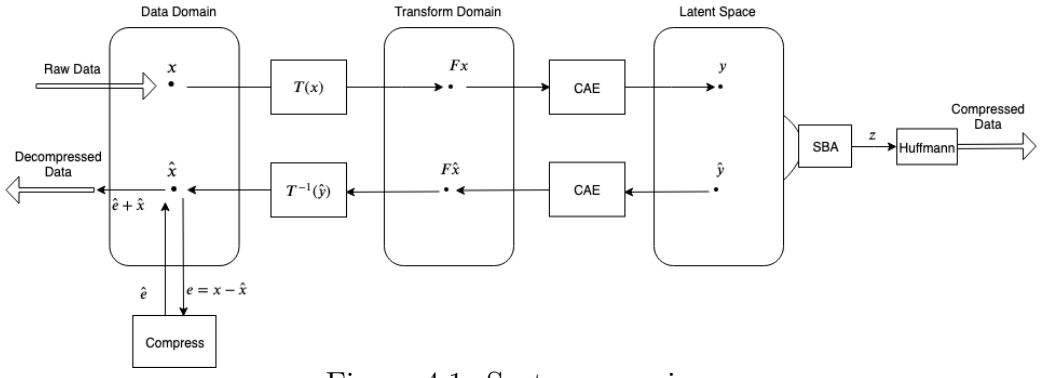


Figure 4.1: System overview.

4.1 Compression

Our whole neural network is a frame-wise process where each audio frame is used as a single data point. We found that block-wise implementation is harder to normalize. When a block of audio contains many frames that have high dynamic range, the mean, variance, and norm do not statistically model all the frames in that block very well.

4.1.1 Preparing Raw Audio for Frame-wise Processing

The first stage of overlap add is to break the raw audio $x[n]$ into overlapping frames and apply the window function. The windowing for the m^{th} block starts at mH, where H is the hop size:

$$x_m[n] = x[n]w[n - mH]$$

In Chapter 3, we established that 2048 frame size is better than a shorter frame size so as not to smear the quantization noise across frequencies and still take advantage of psychoacoustic temporal maskings. And a hop size of H/2 is the biggest available while maintaining COLA. We also found that Hanning window outperforms Hamming window for our compression network as it better attenuates the discontinuities between frames caused by different processing. The Hanning window function of length $N = 2048$ is:

$$w[n] = 0.5 - 0.5\cos(\frac{2\pi n}{N-1}) \qquad 0 \leq n \leq N-1$$

Therefore each $x_m[n]$ block has the size $(1, 2048)$.

4.1.2 Analysis Transform $T(x)$

After the audio data $x[n]$ have been sliced into overlapping blocks of $x_m[n]$ and the window function has been applied, the frames are ready for the DCT analysis transform $T(x)$. As established in Section 2.1.1, the N point DCT of the frame is calculated by:

$$X_m[k] = \sum_{n=0}^{N-1} x[n]w[n - mH]b[n, k]$$

And this process can be replaced by a matrix multiplication:

$$X_m[k] = Wx$$

where

$$W(i,j) = \frac{2}{\sqrt{N}} \cos \frac{\pi}{N}(i + \frac{1}{2})(j + \frac{1}{2}) \qquad i,j = 0, \ldots, N-1$$

We thus implement the DCT as a simple linear layer in the neural network. Before feeding the result of the DCT into the CAE, we make sure the output is of unit norm:

$$X_m[k] = \frac{X_m[k]}{\|X_m[k]\|}$$

The normalizing coefficient $\|X_m[k]\|$ is cast to 16 bit float and passed into the compressed data stream to be used later for inverting the normalization during decompression.

4.1.3 Convolutional Autoencoder (CAE)

Before the first three CAE layers, we apply batch normalization to the input. For a mini-batch $x_{(b)}$ at batch b:

$$\hat{x}_{(b)} = \frac{x_{(b)} - \mathbf{E}[x_{(b)}]}{\sqrt{\mathrm{Var}(\hat{x}_{(b)})}}$$

During inference the running mean and variance is used instead.

The CAE consists of 4 1D convolutional layers, each layer having a filter size of 3, stride size of 1 and zero padding size of 1. Each CAE layer is calculated

using:

$$y_q^{(l)}[n] = \sigma(\sum_p \sum_{u=0}^{U-1} K_{p,q}^{(l)}[u] y_p^{(l-1)}[n-u] + b_q^{(l)})$$

For the first three layers the nonlinear activation $\sigma(x)$ is a leaky rectified linear unit (leakly ReLU):

$$\sigma(x) = \text{LeakyReLU}(x) = \begin{cases} x, & x \geq 0 \\ 0.01 \times x, & x < 0 \end{cases}$$

For the fourth layer, the activation is a tanh: $\sigma(x) = \tanh(x)$.

The length of the output of the 1D convolutional layer with stride size 1 at each channel is:

$$L_{out} = L_{in} + 2 \times \text{padding size} - \text{kernel size} + 1$$

By padding the input with (kernel size $-1)/2 = (3-1)/2 = 1$ zeros, we are able to maintain the same length for the output as the input. The size of the output at each layer is listed in Table 4.1.

Table 4.1: Encoder data sizes.

Layer	Filter	Size
in	-	N_batch x 1 x 2048
$y^{(0)}$	3	N_batch x 128 x 2048
$y^{(1)}$	3	N_batch x 128 x 2048
$y^{(2)}$	3	N_batch x 64 x 2048
$y^{(3)}$	3	N_batch x 4 x 2048

Note our CAE encoder does not perform dimension reduction. In the end we have four output channels and the data size is expanded by 4 times. We rely on our SBA for actual data compression, and having a bigger latent space

gives more leeway to learn a good representation for the SBA.

4.1.4 Stochastic Binary Activation (SBA)

Before quantizing our latent space of the CAE with SBA, we first dither each channel and each batch with uniform noise with a magnitude 0.99 of the mean of the batch and channel.

$$y_b[c, n] = y_b[c, n] + 0.99 \times \mu$$

where $\mu = \text{mean}(y_b[c, n])$. And then the SBA is calculated by:

$$\begin{aligned}
\hat{y} &= \text{SBA}(y) \\
&= \text{Bernoulli}(\frac{\text{HardTanh}(y) + 1}{2}) \\
&= \text{Bernoulli}(\max(0, \min(1, \frac{y+1}{2})))
\end{aligned}$$

where the Bernoulli sampler produces 1s or 0s depending on the input:

$$\text{Bernoulli}(p) = \begin{cases} 1, & p \text{ percent of the time} \\ 0, & (1-p) \text{ percent of the time} \end{cases}$$

During training, the slope of the hard tanh is annealed to gradually increase according to the epoch the training process is on:

$$x = (1.005^{(epoch-1)}) \times x$$

During inference, no annealing occurs, and the epoch is set to one.

The gradient of the SBA is estimated with straight-through gradient estimation:

$$\frac{\partial \text{HardTanh}(x)}{\partial x} = 1$$

y_m, the output of the SBA at frame m, is 2048 binary bits. We group the bits into bytes and encode them using Huffman encoding.

4.1.5 Huffman Encoding

Each byte from a frame is represented as a symbol for the Huffman encoding. Since we do not have access to all of the frames during training, an empirical probability distribution is calculated for each frame and is to be used for Huffman encoding on the frames separately. We use the average number of bits used across all frames as an estimate of the true compression rate of using Huffman encoding on all frames.

4.2 Decompression

First the Huffman code is decoded into 2048 binary bits. These binary bits $\hat{y}$ are fed into the inverse CAE.

4.2.1 Inverse CAE

The inverse CAE is implemented by 4 layers of transpose convolution with kernel size 3, stride size 1 and padding size 1 to maintain the same data length in each of the channels. The first 3 layers of the CAE have a leaky ReLU activation. The last layer has no activation as we do not want to suppress the range of the input into the inverse DCT. The size of the data at each layer is listed in Table 4.2.

Table 4.2: Decoder data sizes.

Layer	Filter	Size
in	-	N_batch x 4 x 2048
$\hat{y}^{(0)}$	3	N_batch x 64 x 2048
$\hat{y}^{(1)}$	3	N_batch x 128 x 2048
$\hat{y}^{(2)}$	3	N_batch x 128 x 2048
$\hat{y}^{(3)}$	3	N_batch x 1 x 2048

The norm from the compression process must be multiplied back to the output of the CAE $\hat{y}_m[k]$ to prepare it for inverse DCT.

$$\hat{X}_m[k] = \|X_m[k]\| \cdot \hat{y}_m[k]$$

4.2.2 Synthesis Transform ($T^{-1}(\hat{y})$)

Since the W matrix for the DCT is symmetric and orthonormal, the inverse DCT can be simply calculated by a simple linear layer of matrix multiplication:

$$x_m[n] = WX_m[k]$$

4.2.3 Residual Training

At this stage, a frame of the original audio has been compressed and decompressed, and the residual (error) of the reconstruction for this frame can be calculated:

$$e_m[n] = x_m[n] - \hat{x}_m[n]$$

The error $e_m[n]$ of this frame is now passed into the same compression and decompression network for both training and inference; the reconstructed error for the frame $\hat{e}_m[n]$ is added with $\hat{x}_m[n]$ to reduce the noise from compressing $x_m[n]$.

4.2.4 Overlap Add Reconstruction

The final step of decompression is overlap adding all the blocks of decompressed audio:

$$x[n] = \sum_{m=-\infty}^{\infty} (\hat{x}_m[n] + \hat{e}_m[n])\delta[n - mH]$$

The extra samples resulting from the OLA at the tail end are discarded.

4.3 Loss Functions

The loss functions of our neural network are:

$$l_{sparse} = \sum |y[n]|$$

$$l_{MSE} = \mathbf{E}[(x - \hat{x})^2]$$

$$l_{HPF} = \text{MSE}(h(x), h(\hat{x}))$$

$$l_{MFCC} = \text{MSE}(\text{MFCC}(x), \text{MFCC}(\hat{x}))$$

Our highpass filter h is a simple FIR designed using the window method with a cutoff of at $0.4f_s$ and a transition bandwidth of $0.8f_s$.

The MFCC calculates 40 coefficients. The overall loss function for the main network is:

$$l_{main} = l_{MSE} + \lambda_0 l_{sparse} + \lambda_1 l_{HPF} + \lambda_2 l_{MFCC}$$

The overall loss function for the residual network is:

$$l_{residual} = l_{MSE} + \lambda_0 l_{sparse} + \lambda_1 l_{HPF}$$

We trained the network for two setups: a lower compression, high-data-rate network for sparser audio and a high compression, low-data-rate network for denser audio. The λ weightings used for the loss functions are listed in Table 4.3.

Table 4.3: Decoder data sizes.

-	Sparsity	HPF	MFCC
High-rate main	0.001	20	0.00002
Low-rate main	0.08	15	0.00001
High-rate residual	0.001	10	-
Low-rate residual	0.0025	0.05	-

4.4 Data and Training

We use various genres of music to focus on different characteristics for training and testing our network:

- Orchestral for high dynamic range and low energy in high-frequency bins (sparse).
- Instrumental and jazz for smaller arrangements.
- Pop and rock for low dynamic range, high amount of information in all frequency bins (dense).
- A capella and new age for vocal.
- Dub step and electronic for interestingly shaped spectrograms.

All of our data is re-sampled to 44100 Hz with a bit-depth of 16 bits, thus conforming to the standard CD quality audio. The two channels of all the audio data are added together into a mono track. For training, we randomly

select frames of 2048 samples (46.4 ms) from all of the audio files as a training set. We use the Adam optimization [24] for gradient descent. For testing, we randomly selected 5 different 10 second excerpts from all of the audio files and performed compression and decompression on them.

The original data is 10 seconds audio with sampling rate of 44100 Hz and a bit depth of 16 bits. The data rate is: 44100 Hz $\times$ 10 s $\times$ 16 bits = 7056000 bits/s. The size of the compressed data depends on the Huffman encoding. However, since different Huffman encoding schemes are used for different frames of the audio compression network, we will calculate an estimate: for each frame with P symbols, suppose the length of each encoded symbol is l_i, and each symbol occurs p_i times. The number of bits used by Huffman encoding in this frame is $p_i \cdot l_i$ bits. The sum of the number of bits across all frames is the estimate. The rate of compression is calculated as the ratio of the size of the original data to that of the compressed.

However, some kinds of music such as orchestral, instrumental, etc., are "sparser" than others, there is not as much information in all the frequency bins compared to that of pop, rock, etc. We have no direct control over the actual bit rate of the compressed data; we can only adjust the loss functions for it to have more or less compression. The actual rate changes when compressing "sparse" versus "dense" audio. Therefore we trained our network with two different setups, a high-data-rate low compression network and a low-data-rate high compression network.

4.4.1 Training Parameters

The number of data points used, the batch size, and the learning rate are different for the high-data-rate network and low-data-rate network. They are listed in Table 4.4.

Table 4.4: Decoder data sizes.

-	Samples	Learning Rate	Epochs	Batch
High-rate main	2000	5×10^{-4}	70	20
Low-rate main	1000	5×10^{-4}	70	20
High-rate residual	2000	1×10^{-4}	70	20
Low-rate residual	1000	1×10^{-4}	70	20

For testing, we select 5 random 10 s excerpts from every audio file in our test suite and pass them through the compression and decompression.

4.4.2 Worst Case Compression Rate

Supposing the original 16 bit PCM audio data has N samples, the size of that data is $16N$ bits. We established in Section 2.1.2 that OLA increases data size by the ratio of the frame size to the hop size; therefore the data size is increased by 2 in our setup. The latent space of the CAE has four channels, and each channel maintains the same size as the input, thus the data size is further increased by 4. During the SBA, the data is binarized and the size of the data is reduced to $N \cdot 2 \cdot 8 \cdot 16/16 = 8N$ bits. Since we encode the residual using the same setup, the size of the residual encoding is also $8N$ bits. Therefore the SBA output has a total of $16N$ bits, the same size as original audio data. In the worst case scenario, if the Huffman encoding does not decrease data size at all, we will have a compression ratio of 1 to 1.

Chapter 5

Results

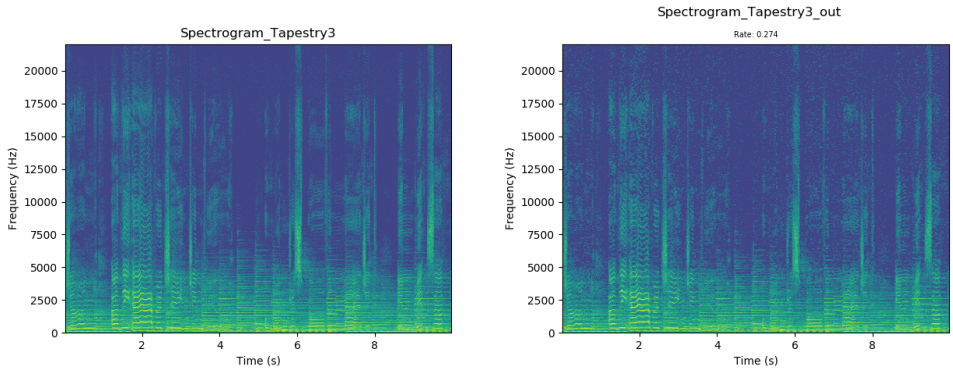
This chapter reports the results of our audio compression network and examines the noise and artifacts introduced by the network. The spectrograms will use a frame size of 2048 (bins) and a hop size of 2048/4. We use the same frame size as our network setup so that we can preserve the exact location of the quantization noise in the spectrograms and not smear it across frequency bins. We will report two fidelity measures: frame-wise signal-to-noise ratio (SNR) and frame-wise MSE. The frame-wise SNR at frame m is calculated by:

$$\begin{aligned} SNR &= 10\log_{10}(\frac{P_{\text{signal}}}{P_{\text{noise}}}) \\ &= 10\log_{10}(\frac{\frac{1}{N}\sum_{n=0}^{N-1}|x_m[n]|^2}{\frac{1}{N}\sum_{n=0}^{N-1}|x_m[n]-\hat{x}_m[n]|^2}) \end{aligned}$$

As we have trained a high-rate network and a low-rate one, we report results for both of these networks. We also present a side-by-side comparison of the results of our network against the fixed rate 320 kbps MP3 compression.

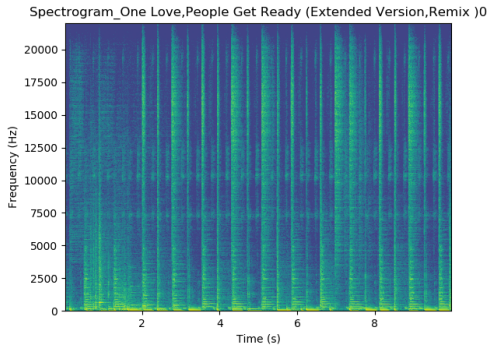
5.1 High-Rate Network versus Low-Rate Network

As we have discussed in Section 4.4, the same network will not perform uniformly across all audio genres as seen in Figure 5.1.

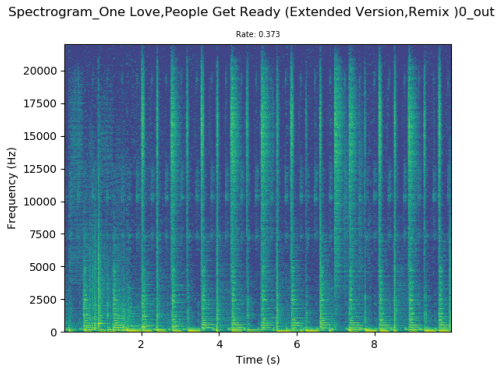


(a) Audio with little high-frequency content.

(b) After compression, quantization noise is focused in high frequencies.



(c) Audio with lots of high-frequency content.



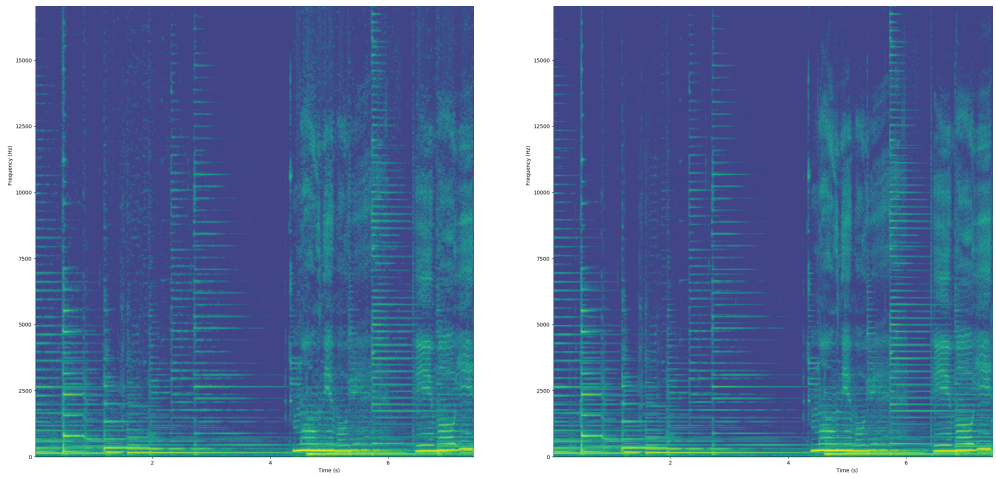
(d) Much less quantization noise after compression.

Figure 5.1: "Sparse" and "dense" audio before and after compression.

Some instrumentation, singing, etc., lack high-frequency content and have low energy in the high-frequency bands. For a low-data-rate network, the high-frequency information may get dropped out. However for denser audio with lots of high-frequency information, the same network will not quantize the higher frequencies as much and will have a much higher data rate.

5.2 Artifacts and Quantization Noise

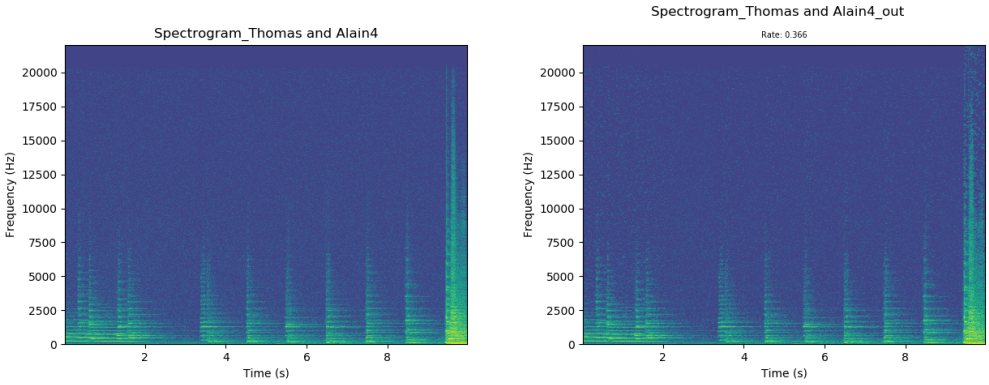
A close-up of the quantization noise in our network can be seen in Figure 5.2.



(a) Compressed audio. (b) Original audio.

Figure 5.2: Close-up image of the quantization noise.

Unfortunately, despite various normalization methods, quantization noise for extremely low energy values cannot be ignored; it is especially visible for recordings with a noise floor and the noise is amplified by our network as seen in Figure 5.3.



(a) Noise floor as in the original record-ing.

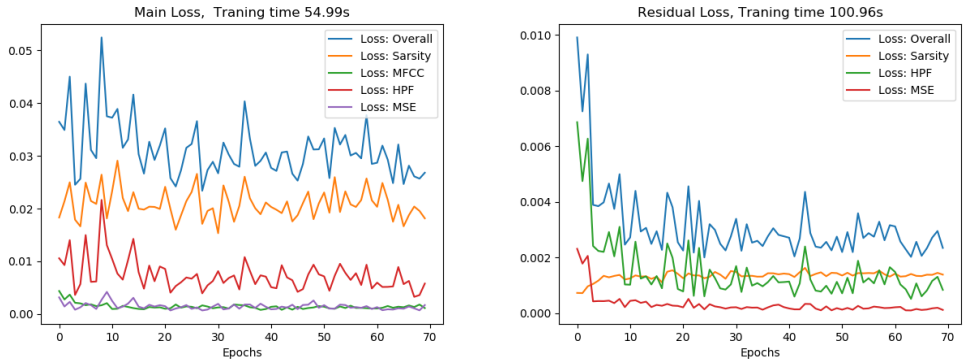
(b) The quantization error boosted the noise.

Figure 5.3: Noise floor issue.

At times, artifacts in extreme high-frequency ranges are introduced by the network if the parameters of the loss functions are not set properly, as we have seen in Chapter 3.

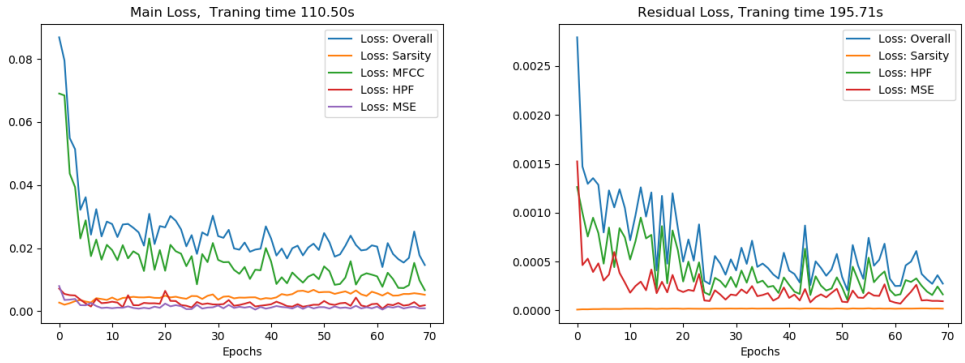
5.3 Reports

Figure 5.4 and Figure 5.5 are typical epoch loss plots of our network. As we are using only small data sets, it only takes roughly 150 seconds to train the low-rate network and 300 seconds to train the high-rate network on a GTX1080 GPU. The training time for the low-data-rate network is roughly half the time of the high-data-rate one since we are using half the amount of data. Moving the data onto GPU is much more time-consuming than the training.



(a) Main compression low-rate. (b) Residual compression high rate.

Figure 5.4: Epoch vs. loss plot of low data rate.

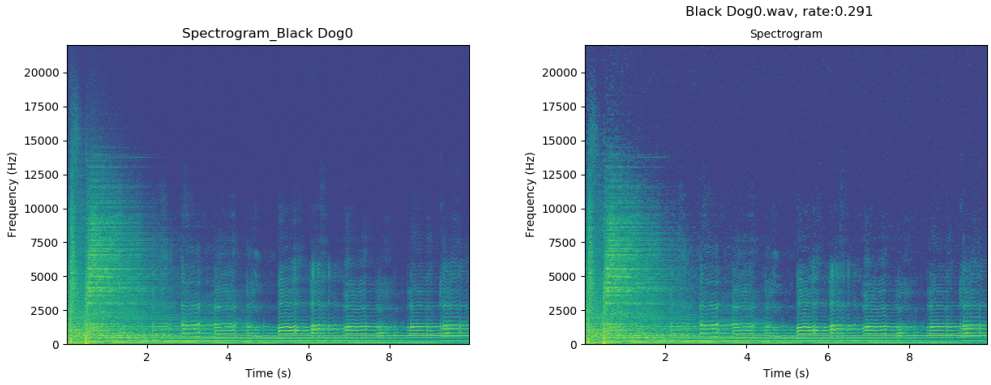


(a) Main compression of high rate. (b) Residual compression.

Figure 5.5: Epoch vs. loss plot of high rate.

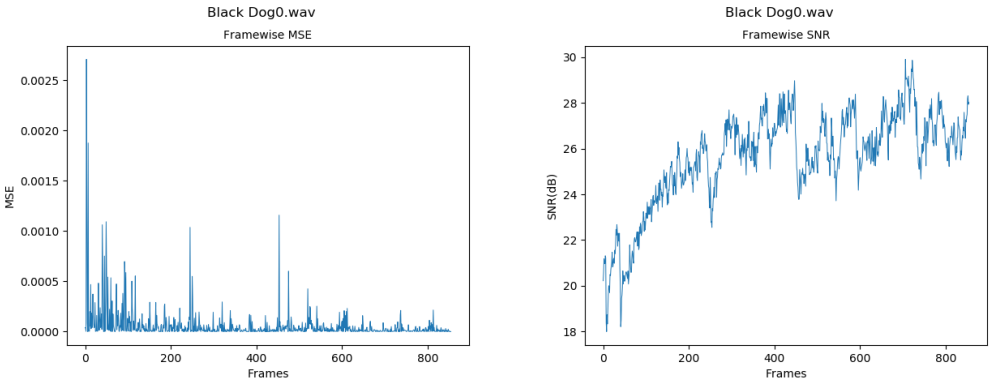
5.3.1 Results of Low- and High-Data-Rate Networks

Figure 5.6 shows the spectrogram of a "sparse" audio compressed using a low-rate network. Figure 5.7 shows the corresponding MSE and SNR plots.



(a) Original "sparse" audio. (b) Compressed "sparse" audio.

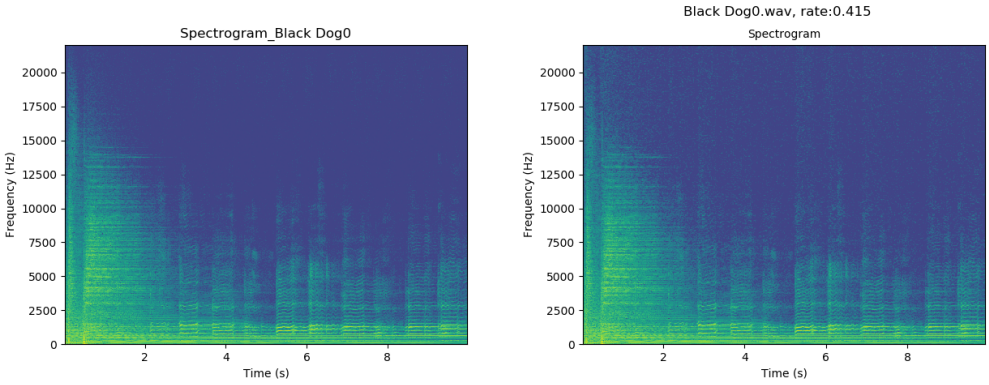
Figure 5.6: Spectrogram of compressed "sparse" audio using a low-rate network.



(a) Frame-wise MSE. (b) Frame-wise SNR.

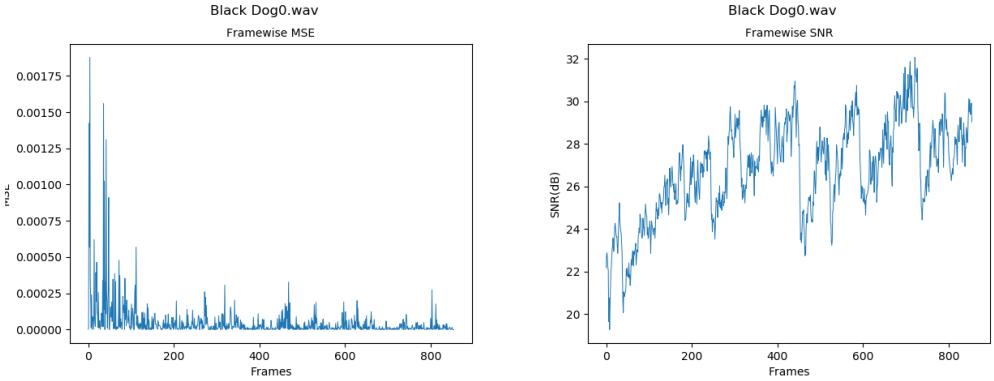
Figure 5.7: "Sparse" audio compressed using low-rate network.

Figure 5.8 shows the spectrogram of a "sparse" audio compressed using a high-rate network. Figure 5.9 shows the corresponding MSE and SNR plots.



(a) Original "sparse" audio. (b) Compressed "sparse" audio.

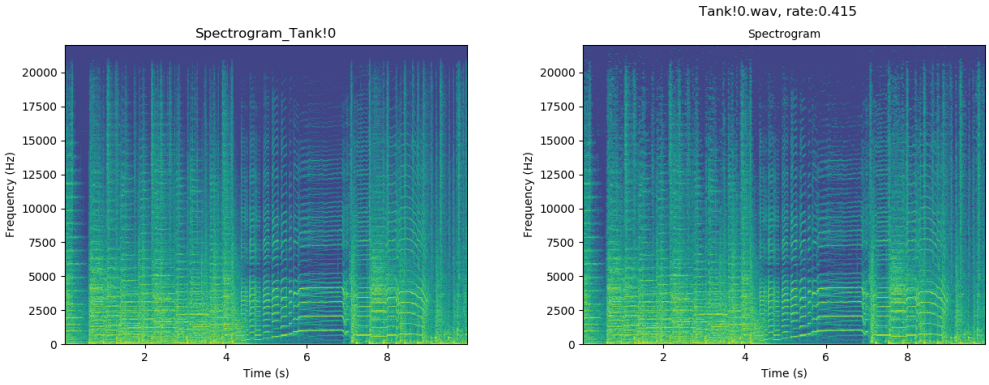
Figure 5.8: Spectrogram of compressed "sparse" audio using a high-rate network.



(a) Frame-wise MSE. (b) Frame-wise SNR.

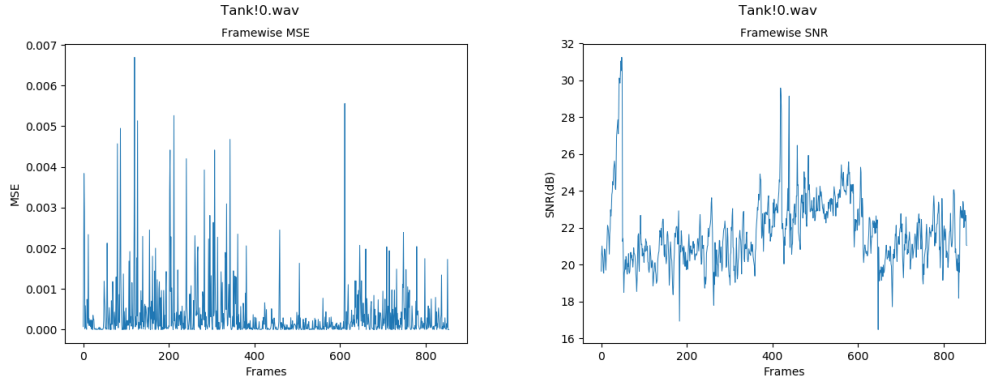
Figure 5.9: "Sparse" audio compressed using high-rate network.

Figure 5.10 shows the spectrogram of a "dense" audio compressed using a low-rate network. Figure 5.11 shows the corresponding MSE and SNR plots.



(a) Original "sparse" audio. (b) Compressed "sparse" audio.

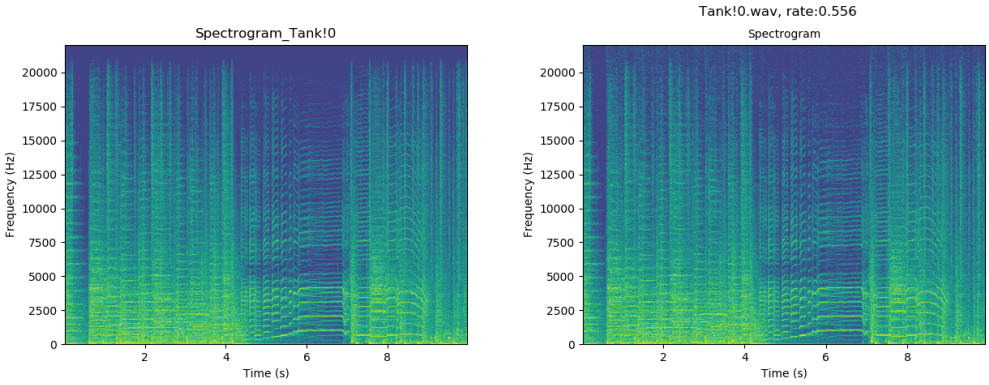
Figure 5.10: Spectrogram of compressed "sparse" audio using a low-rate network.



(a) Frame-wise MSE. (b) Frame-wise SNR.

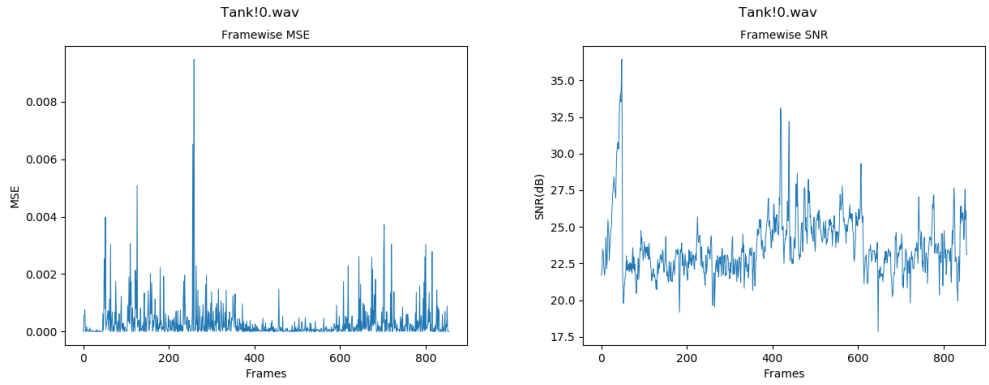
Figure 5.11: "Sparse" audio compressed using high-rate network.

Figure 5.12 shows the spectrogram of a "dense" audio compressed using a high-rate network. Figure 5.13 shows the corresponding MSE and SNR plots.



(a) Original "dense" audio. (b) Compressed "dense" audio.

Figure 5.12: Spectrogram of compressed "dense" audio using a high-rate network.



(a) Frame-wise MSE. (b) Frame-wise SNR.

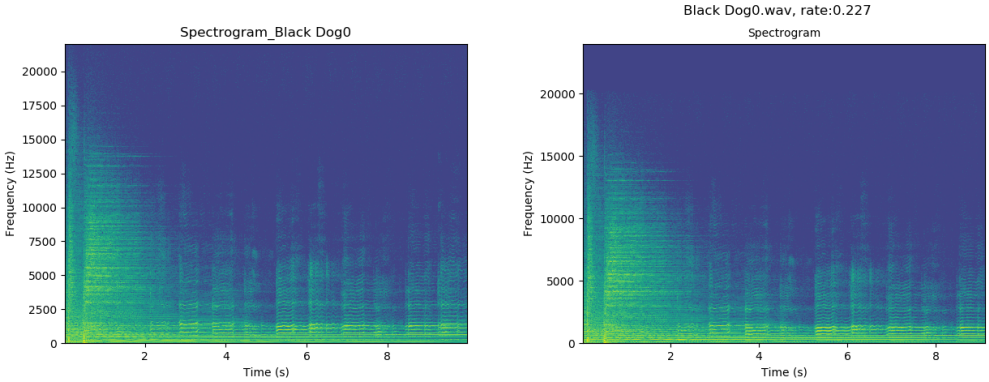
Figure 5.13: "Dense" audio compressed using high-rate network.

5.3.2 Discussion

We can see a slight boost of around 2 dB in the frame-wise SNR when using a higher data rate network and an overall increase in the frame-wise MSE. However, only "sparse" audio have a perceptible increase in quality. The "dense" audio data have a lot of psychoacoustic masking throughout the frequency bins; so it is not worth compressing "dense" audio using a high-data-rate network.

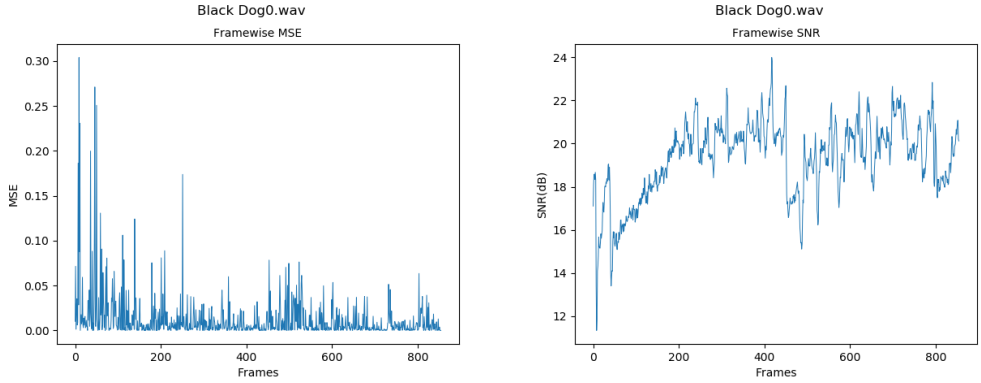
5.4 Compared against MP3

We encoded a 16 bit PCM .wav file using 320 kbps MP3 compression; we can see in Figure 5.14 that the MP3 compression cuts off all information beyond 20 kHz as it is the limit of human hearing. MP3 also creates a minor offset in time which causes alignment issues. The frame-wise SNR and MSE are calculated by manually adjusting the off-set of the MP3 compressed audio.



(a) Original "sparse" audio. (b) Compressed "sparse" audio.

Figure 5.14: Spectrogram of audio compressed using MP3.



(a) Frame-wise MSE. (b) Frame-wise SNR.

Figure 5.15: "Sparse" audio compressed using 320 kbps MP3 compression.

Even though MP3 exhibits a slightly lower SNR in Figure 5.15, the quantization noise is much more evenly distributed and is barely perceptible. Our network is no match yet for the MP3 encoder.

Chapter 6

Conclusion and Future Work

We proposed a convolutional autoencoder network for audio compression. We are able to learn a nonlinear representation of the DCT, and used a stochastic binary activation (SBA) to binarize the latent space as a method of quantization. The neural network is trained in an end-to-end framework for ease of optimizing the rate-distortion trade-off. We are able to reduce the quantization noise of the SBA and provide good audio reconstruction with acceptable SNR. The network demonstrates the powerful capabilities of a nonlinear transform in finding a subspace that allows data to be binarized. This is a good starting point for audio compression using neural networks. Yet we are far from the carefully engineered audio codecs like MP3 and AAC.

One major constraint is the limit of DCT transform without psychoacoustics to take advantage of the different maskings in the frequency bands. In our network, the less prominent information of the DCT is thrown away, but that is not necessarily the less audible information. One possible improvement is to use nonlinear frequency bins such as the mel scale which have finer frequency resolution at frequencies where our human hearing is more sensitive. With such a scaling, the frequency masking will better mask out the quantization noise in these bins.

The disadvantage of using the end-to-end frame work is that it puts much of

the burden of learning a useful transform on designing a proper loss function.

In conclusion, some of the future steps to take are:

- Design a different network for training the residual. The residual is very different from the original audio data and it should be compressed differently.
- Use different networks to compress different lengths of frames based on the energy distribution of the audio in different subbands. A bandpass filter loss could be used instead of the highpass filter loss.
- Test different entropy encoding methods better suited to the problem of encoding sparse binary bits.
- Incorporate the model output variables (MOVs) of the perceptual evaluation of audio quality (PEAQ) as loss function. PEAQ is designed to measure the perceived audio quality using psychoacoustics.
- Step out of the confines of frame-wise processing and autoencoders. Our current network does not capture temporal information; the audio frames in the past and future are not utilized during training.

Bibliography

[1] J. Ballé, V. Laparra, and E. P. Simoncelli. (2016). End-to-end optimized image compression. arXiv: 1611.01704.

[2] L. Theis, W. Shi, A. Cunningham, and F. Huszár. (2017). Lossy image compression with compressive autoencoders. arXiv: 1703.00395.

[3] Z. Cheng, H. Sun, M. Takeuchi, and J. Katto. (2018). Deep convolutional autoencoder-based lossy image compression. arXiv: 1804.09535.

[4] M. Li, W. Zuo, S. Gu, D. Zhao, and D. Zhang. (2017). Learning convolutional networks for content-weighted image compression. arXiv: 1703.10553.

[5] G. Toderici, D. Vincent, N. Johnston, S. J. Hwang, D. Minnen, J. Shor, and M. Covell. (2016). Full resolution image compression with recurrent neural networks. arXiv: 1608.05148.

[6] G. Toderici, S. M. O'Malley, S. J. Hwang, D. Vincent, D. Minnen, S. Baluja, M. Covell, and R. Sukthankar. (2015). Variable rate image compression with recurrent neural networks. arXiv: 1511.06085.

[7] S. Kankanahalli. (2017). End-to-end optimized speech coding with deep neural networks. arXiv: 1710.09064.

[8] E. Agustsson, F. Mentzer, M. Tschannen, L. Cavigelli, R. Timofte, L. Benini, and L. V. Gool. (2017). Soft-to-hard vector quantization for end-to-end learning compressible representations. arXiv: 1704.00648.

[9] V. Kuleshov, S. Z. Enam, and S. Ermon. (2017). Audio super resolution using neural networks. arXiv: 1708.00853.

[10] S. Venkataramani and P. Smaragdis. (2018). End-to-end networks for supervised single-channel speech separation. arXiv: 1810.02568.

[11] G. Strang, "The discrete cosine transform," *SIAM Review*, vol. 41, no. 1, pp. 135–147, Jan. 1999. DOI: 10.1137/s0036144598336745. [Online]. Available: https://doi.org/10.1137/s0036144598336745.

[12] N. Ahmed, T. Natarajan, and K. Rao, "Discrete cosine transform," *IEEE Transactions on Computers*, vol. C-23, no. 1, pp. 90–93, Jan. 1974. DOI: 10.1109/t-c.1974.223784. [Online]. Available: https://doi.org/10.1109/t-c.1974.223784.

[13] G. Heinzel, A. Ruediger, and R. Schilling, "Spectrum and spectral density estimation by the discrete Fourier transform (DFT), including a comprehensive list of window functions and some new at-top windows," Max Planck Inst, 2002.

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, p. 2012.

[15] r2rt. (2016). Binary stochastic neurons in Tensorflow, [Online]. Available: https://r2rt.com/binary-stochastic-neurons-in-tensorflow.html (visited on 01/24/2019).

[16] C. Gulcehre, M. Moczulski, M. Denil, and Y. Bengio. (2016). Noisy activation functions. arXiv: 1603.00391.

[17] Y. Bengio, N. Léonard, and A. Courville. (2013). Estimating or propagating gradients through stochastic neurons for conditional computation. arXiv: 1308.3432.

[18] N. Ichimura. (2018). Spatial frequency loss for learning convolutional autoencoders. arXiv: 1806.02336.

[19] B. Logan, "Mel frequency cepstral coefficients for music modeling," in *ISMIR*, 2000.

[20] T. Li, M. Ogihara, and Q. Li, "A comparative study on content-based music genre classification," in *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Informaion Retrieval*, ser. SIGIR '03, Toronto, Canada: ACM, 2003, pp. 282–289. DOI: 10.1145/860435.860487. [Online]. Available: http://doi.acm.org/10.1145/860435.860487.

[21] S. Ioffe and C. Szegedy. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv: 1502.03167.

[22] T. Salimans and D. P. Kingma. (2016). Weight normalization: A simple reparameterization to accelerate training of deep neural networks. arXiv: 1602.07868.

[23] J. P. Egan and H. W. Hake, "On the masking pattern of a simple auditory stimulus," *The Journal of the Acoustical Society of America*, vol. 22, no. 5, pp. 622–630, Sep. 1950. DOI: 10.1121/1.1906661. [Online]. Available: https://doi.org/10.1121/1.1906661.

[24] D. P. Kingma and J. Ba. (2014). Adam: A method for stochastic optimization. arXiv: 1412.6980.