Efficacy of Variational Autoencoders and U-Net for Generating Audio Signals

Jacob Sturges

May 2023

1 Introduction

This paper represents a comprehensive synopsis of my final project for ANNE Spring 2022 that I have been dedicating to for the past several months. The final project attempts to deconstruct and summarize the steps involved with generating audio using two AI architectures: Variational Autoencoders (VAEs) and the U-Net. These two architectures were chosen because of their robust utility in the task of generative audio compared to architectures in other ML model classes. This paper will give a brief overview of digital audio signaling and the techniques surrounding them, the preprocessing and postprocessing of the data, the choice of design for the models, and the mathematical theory behind the whole process.

2 Why VAEs and U-Nets?

Both VAEs and U-Nets are extensions of the Autoencoder ML model class. Autoencoders possess unique paradigms that are suitable for audio synthesis.

2.1 Autoencoder Brief Overview

The Autoencoder is an unsupervised learning model designed to recreate the same data in its output that it ingests for its input. The autoencoder consists of two building blocks: the *encoder* and the *decoder*. The encoder is responsible for ingesting and transforming the data by performing dimensionality reduction and mapping that lower-dimensional representation onto a latent space, where it is represented as a point on the hyperplane in the n-dimensional latent space. Thus, a point belonging to the latent space is represented as a vector, where each value i in the vector represents the coordinate value for the point in the i^{th} dimension/axis, this is known as the *latent vector*. For example, if a 2 dimensional latent space contains a point (x_1, x_2), then the latent that represents a lower dimensional encoding of the original input data would be $[x_1, x_2]^T$. This lower-dimensional encoding of the sampled input data represents the output of the encoder. The layer in the model where this latent space resides is called the *bottleneck*, aptly named because one can visualize the data being "squeezed" to its lowest dimensional representation that the model allows.

The decoder is responsible for taking that latent vector and reconstructing an output that matches the input data. The latent vector is considered the input to the decoder, and an error between the original data input and the output is measured and backpropogated as a means to fine-tune the model. This forces the autoencoder to learn a compact, efficient representation of the input data.

In addition to dimensionality reduction, another useful utility of autoencoders is feature extraction, which is actually a direct consequence of dimensionality reduction. This makes sense, because if the goal of a model is to simplify data through compactification, the most important features of that data need to be preserved in order to make the lower-dimensional representation more accurate. Therefore, the encoder is inclined to dispose of unimportant features of the input data.

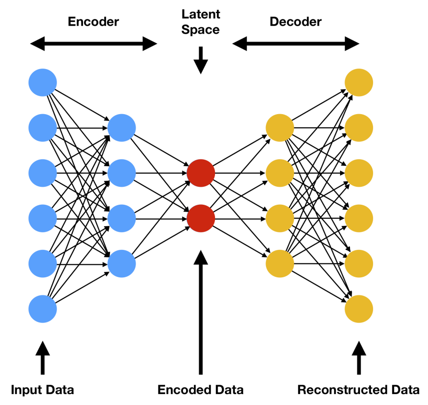

Figure 1: Representation of a basic Autoencoder network.
Source:
`https://www.compthree.com/blog/autoencoder/`

2.2 Variational Autoencoders

Despite the massive utility of vanilla autoencoders, there still exist some large, unremedied deficiencies that prevent a vanilla autoencoder from being able to perform certain kinds of tasks efficiently, such as generating music. One of the main disadvantages of vanilla autoencoders is that it produces a non-continuous latent space. Recall from before that autoencoders map to a single point on the latent space. After training the autoencoder, it should be able to recreate samples that are represented by points its already encoded in the latent vector very well. But, what if it were to try to recreate some data from a point it hasn't seen before (i.e., generating/constructing new music not present in the input data)? This results in a jumbled mess of an output, likely one that doesn't resemble anything that the input data offers. How do we overcome this? This is fixed using an enhanced autoencoder model called the Variational Autoencoder (VAE).

VAEs possess powerful attributes that aren't present in vanilla autoencoders. The most important of these is the way VAEs encode representations onto the latent space. Instead of plotting a single point as an autoencoder does, the VAE plots a representation to a *distribution* centered around an instance of the data. The encoder is given two extra parameters to learn for each instance of the input data: the mean (μ) and the variance (σ^2) for all dimensions in the hyperplane. For example, an instance of data x_i being plotted to an nD latent space hyperplane would carry the properties $[\mu_{x_{i1}}, \mu_{x_{i2}}, ..., \mu_{x_{in}}]^T$ and $[\sigma_{x_{i1}^2}, \sigma_{x_{i2}^2}, ..., \sigma_{x_{in}^2}]^T$, representing the mean and variance vectors, respectively. You can almost think of the mean as the encoded position of a point from the vanilla autoencoder, while the variance determines the "spread" around this point. The mean and variance vectors are initially randomized at the beginning of training and update as the training continues. The importance of this property is that instead of plotting a single point on a latent space, this "point" is now its own distribution, which is conveniently Gaussian-like. Thus, instead of a simple, static point being sampled and fed to the decoder input, the sample may now take any values around this distribution mean and generally within it's "spread" or variance, and reconstruct data similar to that of previous sample points that are close by. This property of treating sample instances as distributions promotes continuity and small changes to new sample points while preserving important features that reflect "closeness" to the other sampled points clustered around any new sample point. With vanilla autoencoders, the absence of continuity in the latent space presented problems with generating coherent new samples that were not previously encoded directly.

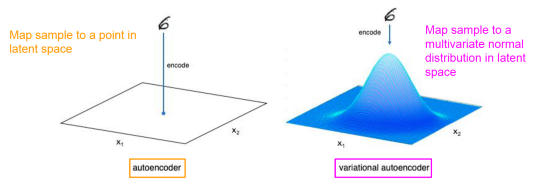
Map sample to a point in latent space
6
encode
x_1
x_2
autoencoder
6
encode
Map sample to a multivariate normal distribution in latent space
x_1
x_2
variational autoencoder

Figure 2: Comparison of latent space mapping between a basic Autoencoder and a Variational Autoencoder. The VAE maps to an entire Gaussian distribution centered around a sampled point.
Source: *Generative Deep Learning* by David Foster

So, how do VAEs find the correct mean and variance vectors for the encoder to calculate? It is crucial to note that, at the high level, there is an overall mean and variance vector of the latent space itself, apart from the individual mean and variance vectors of each sample instance. With latent spaces, symmetry around the origin is far easier and convenient to work with, which is why we want to shape the accumulation of the distribution of sampled points/means to converge to a standard Gaussian distribution $N(0,1)$. However, we also want to preserve the data in such a way where the most important features are intact, which will almost certainly not be a standard Gaussian. How do we make this compromise? The magic lies with a special loss function: We attempt to combine two loss functions into one. The first loss function, called the Reconstruction error, measures the RMSE of the decoder output to the input:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{1}$$

This equation aims to minimize the difference between the original and reconstructed data.

The second loss function used by VAEs is called the Kullback-Leibler Divergence (KL-divergence). This function aims to quantify the difference between any Gaussian distribution (in our case, the distribution belonging to the latent space) and the standard Gaussian distribution:

$$D_{\mathrm{KL}}(N(\mu,\sigma)\|N(0,1)) = \frac{1}{2}\sum_i(1+\log(\sigma^2)-\mu^2-\sigma^2) \tag{2}$$

The KL-Divergence acts as a regularizer by penalizing observations where the mean and log variance vectors differ greatly from those belonging to a standard Gaussian distribution (with mean of 0 and log variance of 0). In essence, this is "pulling" the overall latent space distribution to conform to a standard Gaussian. This accomplishes two things: (1) it promotes symmetry about the origin, and (2) it decreases large gaps between clusters of encoded instances/points.

The overall loss function is a combination of RMSE and KL-Divergence:

$$LOSS = \alpha \cdot RMSE + KL \tag{3}$$

where α is a hyperparameter that represents the reconstruction loss weight. This weight value is necessary because it provides a balance between the RMSE and KL losses. If α is too small, KL becomes the dominant term and thus the data will be poorly reconstructed. If α is too large, RMSE becomes the dominant term and KL cannot regularize the data, so the VAE inherits the same issues as a vanilla autoencoder.

Hopefully it is now clear why the nature of treating sample instance points as distributions make VAEs a prime candidate for audio generation: we simply need to draw any point from a fine-tuned latent space distribution and feed it to the decoder to construct a new sound sample output.

2.3 U-Net

Like the variational autoencoder, the U-Net is another enhancement of the vanilla autoencoder. The U-Net, developed in 2015 by Ronneberger et al., is an ML model architecture that was originally developed to segment images. The model relies on heavy data augmentation to maximize image segmentation while preserving context from parsing through each hidden layer of the network. [1] Data augmentation is the process of manipulating existing data in a way where one can extract a larger quantity of quality data than the original dataset, in order to compensate for a insufficient amount of feature data that may lead to overfitting. The data augmentation technique described in the U-Net is carried out using consecutive convolution, ReLU, and max pooling operations in the first half of the architecture (called the encoder) to capture and preserve attention in the latent space, while the latter half (the decoder) expands the image to its original size. The magic in this architecture lies with its *long skip connections* (see Figure 3) that concatenate the output of every layer in the encoder to the corresponding layer in the decoder. This concatenation preserves important attention data gathered at every level, while simultaneously addressing the vanishing gradient problem by providing a shorter path for the chain rule to operate. Though empirically proven, the mathematical theory behind why long-skip connections work is still a work in progress.

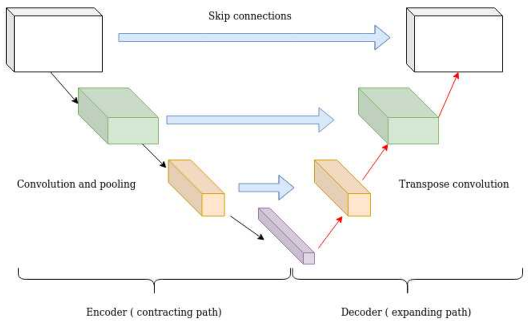

Figure 3: Diagram of long-skip connections. The output of each layer in the decoder is concatenated with the input of each layer in the decoder.
Source:
`https://theaisummer.com/skip-connections/`

The architecture of the U-Net outlined in [1] is as follows: n layers of Convolution-ReLU-Max Pooling/Copy/Crop in the encoding half, followed by n layers of Up-Convolution-Concatenation-Convolution-ReLU in the second half. A final Convolution layer is imposed to make a final prediction.

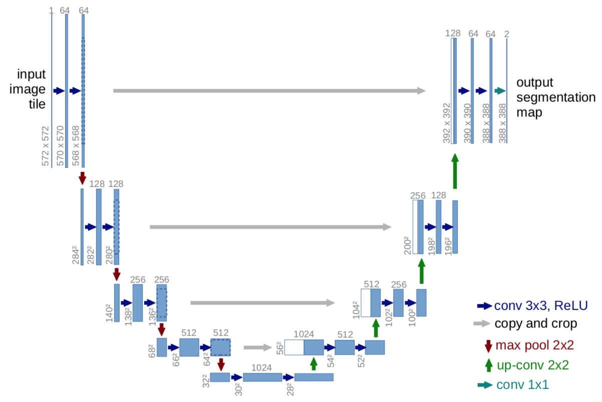

Figure 4: Diagram of U-NET proposed in [1].

How does this relate to generating audio? If you look closely, the U-Net is essentially an autoencoder with skip connections and carefully placed convolutional layers. The bottleneck at the bottom is still present, but with the addition of skip connections, we may treat every layer as its own "bottleneck". Thus, though the U-Net was developed to segment images, we can also leverage these special properties to make it into an audio generator. (*Spoiler: it works very well*).

The main difference between the VAE and U-Net architectures is that the VAE can make use of latent space distributions, while the U-Net has several bottlenecks, but does not leverage these statistical parameters for its latent spaces.

3 Brief Overview of Digital Audio Signaling

The data we will be working with, obviously, will be sound files. How do we process and work with this type of data, and how do we make sense of it all? This section briefly breaks down the complex process.

3.1 Sound and Waveforms

Sound is energy that is represented by an acoustic wave. These waves are produced by the vibration of an object through a medium (solid, liquid, or gas), but generally, most sound we experience is from oscillating air molecules, which are caused by a change in air pressure.

A continuous sound wave can be represented as a *waveform* (see Figure 5). Waveforms carry multifactoral information about the sound, which includes frequency (measured in Hertz (Hz)), intensity (amplitude), among other information.

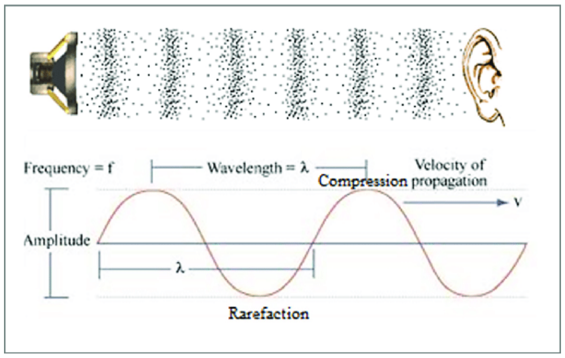

Figure 5: Waveform of a sound.
Source: `https://www.mediacollege.com/audio/01/sound-waves.html`

These continuous waveforms are known as analog signals, and in order to process these signals, we need to convert them in a way that computers can work with, called digital signals. Since analog signals are continuous and computers have finite memory, they need to convert the continuous signal into a sequence of discrete values, which is determined by the sampling rate (see Figure 6). A common sampling rate is 44.1 kHz.

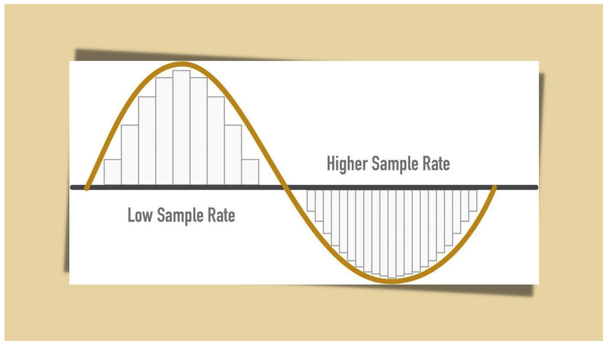

Figure 6: Illustration of sampling rate.
Source: `https://create.routenote.com/blog/what-is-an-audio-sample-rate/`

Thus, these converted digital signals can now be used to process data. The digital signal resembles power (amplitude) over time (s), called the *time domain.* One can think of this sound being represented by an overall waveform, where this waveform is a sum of sine waves of different frequencies, amplitudes, and phases put together. These individual waves carry important information about the overall audio signal, so it's of great interest to understand these components. To extract the individual frequencies, a Fourier Transform is used to decompose this overall signal into each sinusoidal frequency (I go over the details of this in the Appendix section). The result of this is the *frequency domain.*

In order to convert back from the frequency domain to the time domain, we simply use an Inverse Fourier Transform to reconstruct our signal.

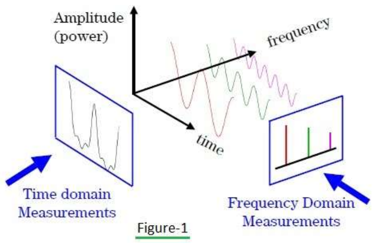

Figure 7: Illustration of converting digital audio signal from time to frequency domain.
Source: `https://www.test-and-measurement-world.com/Measurements/Time-domain-measurements-vs-Frequency-domain-measurements.html`

Lastly, the human ear is not receptive to linear changes in the amplitude of sound, but rather, it's been empirically determined that log representations of sound is far more accurate. Thus, we convert amplitude measured in power to decibels (dB). In order to gather and represent these frequencies and decibels in a convenient way, the use of spectrograms is encouraged. Spectrograms plot every frequency's intensity (dB) or contribution to the overall audio piece over time (see Figure 7). Audio spectrograms can be though of as a 2D matrix of decibel values, where each row is a frequency bin and every column is a time bin/frame.

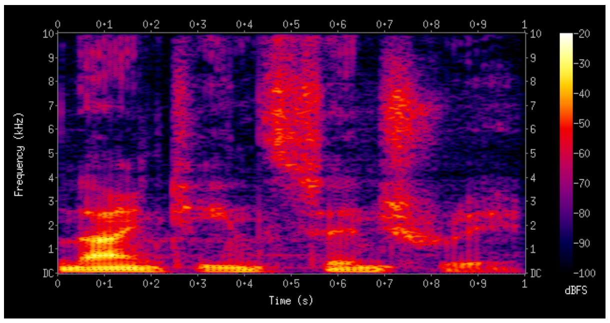

Figure 8: Audio spectrogram.
Source:
`https://en.wikipedia.org/wiki/Spectrogram`

4 Methods

I wanted to test the efficacy of how well the Variational Autoencoder and U-Net processes and recreates audio signals. The audio data is called the free spoken digit dataset and is an extensive collection of .wav sound files that represents various individuals speaking utterances of every digit, e.g., ”one,” ”two,” and so on. The was retrieved from here. The goal of this experiment is to determine how quickly the models converge to a coherent audio signal. Python, Tensorflow, and Librosa contributed significantly to the overall implementation of the experiment. Additionally, a portion of the Python implementation was sourced from [2], including the pre and postprocessing and VAE model. I did a large extension to the code by writing and adding the U-Net model and modifying the pre and postprocessing, as well as adding files for analysis for the purposes of this experiment.

The data was preprocessed by taking every audio file and converting it to a 2D matrix representing a log-spectrogram. In total, there were 3,000 samples. Each of these log-spectrograms were then normalized between [0, 1] in order to ease training. After the preprocessing, the input shape had dimensions of (256, 64, 1), which represents the number of frequency bins (range of frequencies), the number of time frames covering the duration of the signal, and an extra axis to make Tensorflow's convolution layers happy by treating the data like an RGB image.

I decided to run the Variational autoencoder on 150 epochs, and an alpha value of 1,000,000 for the loss function.

I then extended this implementation to the U-Net. I decided to let the U-Net utilize the RMSE loss function only, as it does not utilize KL-divergence.

I then decided to run the U-Net given the same specifications. However, the U-Net did not need more than 5 epochs to converge to its desired value, as it only worked with the RMSE.

Ultimately, I want to determine two things: (1) how well both models did recreating the input signals, and (2) determine the robustness of the models by measuring how they perform when fed a slightly modified version of input data they just generated, and compare them.

I want to make a quick note that determining the quality of sound can be just as artistic as it is empirical at times–and this is one of them. I will give results that make some empirical sense of the data, but I also want you to determine the effectiveness of the models subjectively through listening to their outputs.

5 Results

The results of the experiment was as follows: The run time to train the VAE was approximately 2 hours, 30 minutes to train on 150 epochs, while the U-Net took approximately 5 minutes to train. The RMSE + KL-Divergence plot of the two models is shown below:

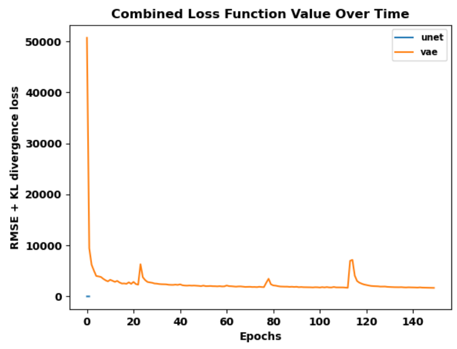

Figure 9: Value of loss function over time during training phase.

The comparison is misleading due to the U-Net only utilizing RMSE and avoiding the KL-divergence penalty term altogether. However, as it turns out, this is not an issue for the U-Net as it is for the VAE. Also, it appears that the VAE found its minima around epoch 20, with little improvement over the next 130 epochs. The U-Net ended up stopping its own training due to hitting the minimum delta threshold provided by the EarlyStopping Keras class.

As it turns out, the U-Net bested the VAE in terms of sound recreation. Included in a file 'samples/original' are examples of a target sound that the models aim to recreate. 'samples/generated' contains the corresponding sample sounds that were generated from each model, respectively. (I wish I could embed audio into LaTex, but I spent 2 hours toiling with this to no avail. I'm sorry).

To me, it appears that the generated sounds from the VAE are slightly off in pitch compared to the U-Net. In addition, I noticed that the U-Net also captures minute changes in sound much more efficiently than the VAE. This is most noticeable in audio samples 2 and 3. Overall, the VAE's reconstruction output is much more dull compared to its competitor's.

For the second part of the analysis, I wanted to test the robustness of the models when presented with a small change in input. These reconstructed outputs are located in the 'samples/modified' file. I decided to take the original samples, and added or subtracted their values by a random value that did not exceed 10. I chose 10 due to it being one order of magnitude on the decibel scale, which correspond to the units for the preprocessed spectrograms. I then passed these modified spectrograms to be processed by the two models.

Shockingly, the U-Net is far more robust than the VAE in maintaining a quality of "closeness" to its generated sample. In other words, the audio it generated from the modified data sounds very similar to the audio it generated from the original sample. As for the VAE, it seems to have a lot of trouble constructing a new signal from a modification of one order of magnitude in dB. The audio is far more noisy and incoherent than the U-Net model.

6 Discussion and Conclusion

In my opinion, the main reason why the U-Net outperformed its VAE counterpart: the presence of skip connections means that there are more channels for the decoder to reference against and work with, while the VAE's decoder can only rely on the latent representation in the bottleneck. Even with a utility advantage leveraging statistical properties, this pales in comparison to the U-Net's overwhelming, constant access to multiple latent spaces of the same data, representing by many dimensions coming from multiple latent spaces, vs. just one, small dimensional latent space for the VAE.

It's apparent that the VAE is far more sensitive to changes in data than the U-Net. This makes sense, as the VAE was conditioned by KL-divergence to constrain itself into a distribution that followed a standard normal distribution. However, the U-Net did not have this chore, and thus it would take far more of a nudge in data to see a noticeable difference in output from its generation on the original sample. I think that, for future work, it will be important to investigate what a good balance between the two may be.

7 References

[1] Ronneberger, O., Fischer, P., Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. In: Navab, N., Hornegger, J., Wells, W., Frangi, A. (eds) Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015. MICCAI 2015. Lecture Notes in Computer Science(), vol 9351. Springer, Cham. https://doi.org/10.1007/978-3-319-24574-4_28

[2] Valerio Velardo (2021). https://github.com/musikalkemist/generating-sound-with-neural-networks