A NOVEL APPROACH TO EXTENDING MUSIC USING LATENT DIFFUSION

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

Keon Roohparvar

June 2023

COMMITTEE MEMBERSHIP

TITLE:	A Novel Approach to Extending Music Using Latent Diffusion
AUTHOR:	Keon Roohparvar
DATE SUBMITTED:	June 2023
COMMITTEE CHAIR:	Franz Kurfess, Ph.D. Professor of Computer Science
COMMITTEE MEMBER:	Maria Pantoja, Ph.D. Professor of Computer Science
COMMITTEE MEMBER:	Paul Anderson, Ph.D. Professor of Computer Science

ABSTRACT

A Novel Approach to Extending Music Using Latent Diffusion

Keon Roohparvar

Using deep learning to synthetically generate music is a research domain that has gained more attention from the public in the past few years. A subproblem of music generation is music extension, or the task of taking existing music and extending it. This work proposes the Continuer Pipeline, a novel technique that uses deep learning to take music and extend it in 5 second increments. It does this by treating the musical generation process as an image generation problem; we utilize latent diffusion models (LDMs) to generate spectrograms, which are image representations of music. The Continuer Pipeline is able to receive a waveform as an input, and its output will be what the pipeline predicts the next five seconds might sound like. We trained the Continuer Pipeline using the expansive diffusion model functionality provided by the HuggingFace platform, and our dataset consisted of 256x256 spectrogram images representing 5-second snippets of various hip-hop songs from Spotify. The musical waveforms generated by the Continuer Pipeline are currently at a much lower quality compared to human-generated music, but we affirm that the Continuer Pipeline still has many uses in its current state, and we describe many avenues for future improvement to this technology.

ACKNOWLEDGMENTS

Thanks to:

- Dr. Franz Kurfess - His excellent methods for overseeing my work really allowed me to pursue this project in my own way, and it immensely fostered my passion for both this thesis work and Deep Learning. This work would not be possible without Dr. Kurfess, so I am extremely grateful for his advisement.
- My parents - I would not be the person I am today if it wasn't for the love, support, and guidance that they have provided me. I will be forever thankful to them, and I am so proud to call them my parents.
- Jackie Driscoll - I cannot stress how thankful I am for her support through this thesis process; she really enabled me to become the best version of myself, and I am very lucky to have her in my life.
- Sigma Phi Delta - I am so grateful that I joined the professional Engineering fraternity as it has both shaped my professional career and given me an excellent social network during my tenure here at Cal Poly.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

Chapter 1

INTRODUCTION

Recent innovation in the industry of artificial intelligence in image generation has yielded great success, and it has generated a lot of attention from the general public. Tools like OpenAI's famous image generator DALL-E are able to generate an image from only a text description [14]. An area that is not as explored as much by academia is music generation - there are many techniques that have been explored, but the results are not up to the same quality as the results we see today with image generation techniques.

Synthetic music generation is the task of creating music using some statistical technique. When tackling this problem, researchers often focus on the musical waveforms themselves, or they discretize the problem by only performing synthesis on the musical notes rather than the raw frequencies. What we propose in this paper is to treat this problem like an image generation problem. Due to recent advancements in the space of image synthesis, this research was proposed due to the authors' belief that there is merit in using these image synthesis techniques in the context of music synthesis.

To treat music generation as an image generation problem, we need techniques to extend musical waveforms to and from the image domain. Fortunately, techniques in signal processing have allowed us to do this; Fourier transformations allow us to convert musical waveforms to a frequency representation, and we can utilize a method called the Griffin-Lim Algorithm to approximate the transformation back from the image domain to the time domain. When music is in this frequency domain, we can simply treat songs as images; these 2-D representations of music are known as Spectrograms [6].

A subdomain of the problem of generating music is extending music. Many of the existing algorithms that aim to synthetically generate music simply create a short snippet of a song; to a consumer, this is not feasible as a user would very likely want to listen to audio that is longer than a few seconds. Because of this, many implementations utilize different techniques to attempt to extend songs, as this makes their products much more feasible for consumer usage.

In this paper, we propose a new technique to extend music, and we have named it the Continuer Pipeline. We detail our novel pipeline that uses latent diffusion techniques to take an audio file and extend it by an additional 5 seconds; successive calls to the Continuer Pipeline can be made so that the music is extended for periods longer than 5 seconds. We also detail the current results of our novel pipeline, and we discuss future areas of research to improve the results of this work.

Chapter 2

BACKGROUND

This chapter will give background information regarding concepts needed to understand the subsequent chapters.

2.1 Spectrograms

Spectrograms are based off of successive Fourier transforms of data using a sliding window [4]. They allow us to take data in the time domain, like audio files representing music, and convert them to the frequency domain. Where this fits in the context of our problem is that we can use spectrograms to treat musical pieces as images. If we apply Fast Fourier Transformations (FFTs) to a musical waveform, we can convert our musical waveforms to spectrograms; these 2-D arrays are now image representations of the musical waveforms, and we can now use deep learning applications that receive images as their inputs to tackle this problem.

Mel-spectrograms are a subset of spectrograms that are limited to information only within the Mel-frequencies, which are the frequencies that human ears are tuned to focusing on [13]. This allows us to reduce the information of the frequencies that are not important to humans, which simplifies the information being fed to our deep learning methods. This ultimately allows the models to focus their learning on information regarding the most important frequencies.

A raw spectrogram is complex-valued, and thus cannot be an input to our deep learning models. To alleviate this issue, we take the square of the raw spectrograms to

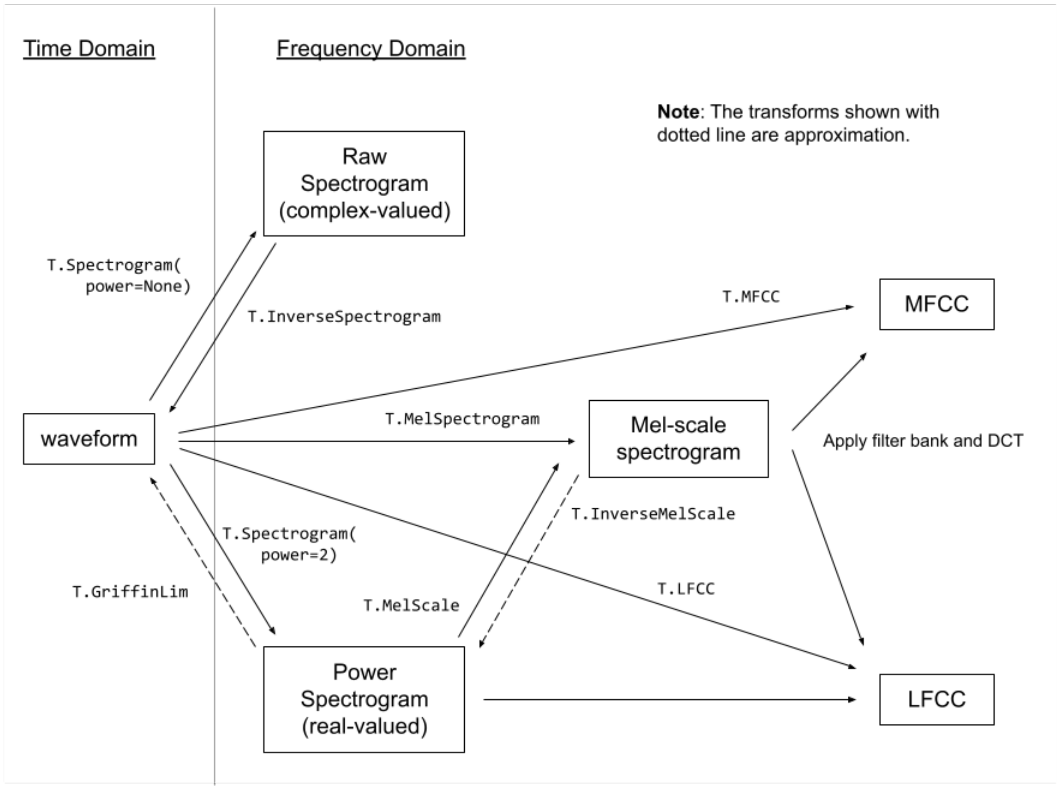

Time Domain
Frequency Domain
Note: The transforms shown with dotted line are approximation.
Raw Spectrogram (complex-valued)
T.Spectrogram(power=None)
T.InverseSpectrogram
T.MFCC
MFCC
waveform
T.MelSpectrogram
Mel-scale spectrogram
Apply filter bank and DCT
T.InverseMelScale
T.Spectrogram(power=2)
T.GriffinLim
T.MelScale
T.LFCC
Power Spectrogram (real-valued)
LFCC

Figure 2.1: Waveform Transformations Possible in Torchaudio [21].

remove the complex values; this is called a power spectrogram. This is a necessary step as our model cannot take complex-valued inputs, but this creates a new problem: the data is now not perfectly invertible, as we do not know which values in the spectrogram were complex when we try to convert the spectrogram back to a waveform. Fortunately, the famous Griffin-Lim Algorithm is an approximation that lets us estimate a waveform from a power spectrogram [6]. Figure 2.1 details how the transformations occur between waveforms and their associated spectrograms using the *Transforms* library from *Torchaudio* [21].

2.2 Generative Models

Deep generative models are a subset of the deep learning space that focus on generating novel pieces of data. The main idea that led to the creation of these models is that data of a certain class has an innate probability distribution, and training a model on this probability distribution will allow the model to create novel, but similar, data examples [11].

There are many techniques that have been used for years in the space of generative models, including autoencoders, generative adversarial networks, Boltzmann machines, deep belief networks, and more. There are also more recent techniques that have emerged, which include diffusion probabilistic approaches. Depending on the nature of one's problem, there are often better techniques to use, as each approach has different potential use cases.

In the domain of generative models, there are two subdomains that are addressed in this paper: unconditional generation, and conditional generation.

2.2.1 Unconditional Generation

Unconditional generation is the problem of having a model attempt to learn the probability distribution from a set of data, and exploit this to create new pieces of data from this estimated probability distribution. This is more specific than generative models as the model receives no other information in its generative process; once trained, the model will generate its estimation of a new sample from the target data distribution, and the user is unable to affect this generative process.

2.2.2 Conditional Generation

Conditional generation is the act of generating new pieces of data from a target distribution, but its generation process can have influence from user input. The work done by Van den Oord et al. highlights how an unconditional generator can be converted to a conditional variation [19]. They took a previously constructed architecture, PixelRNN, and created a conditional generator that could generate outputs from three different classes specified by a user. This conversion from unconditional to conditional generation occurred because the new architecture was able to take information from user inputs on what class to generate to, which differs from its unconditional generator counterpart.

2.3 Diffusion Models

Diffusion probabilistic models, abbreviated as diffusion models for brevity, are a generative deep learning technique that emerged in 2020, and a recent innovation in the image synthesis domain [7]. The technique has gained a lot of popularity as it has demonstrated excellent success in various image generation tasks. While the industry has devoted a lot of resources to exploring diffusion models in the context of image generation, less effort has been invested in the study of its use in musical generation.

The inspiration of diffusion models is derived from the diffusion laws found in thermodynamics. Ho et al. believed that having a deep learning model generate an image from pure noise is too challenging, but a Markov chain model that incrementally denoises a sample of Gaussian noise to a realistic image is a lot more feasible [7]. Thus, a diffusion model is characterized as a parameterized Markov chain that is trained to incrementally denoise an image through each transition of the chain. After training is

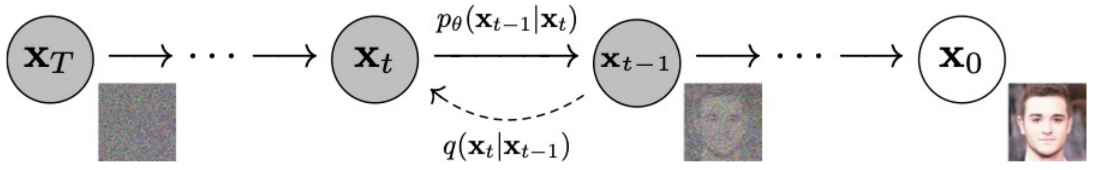

$\mathbf{x}_T$
$\mathbf{x}_t$
$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$
$q(\mathbf{x}_t|\mathbf{x}_{t-1})$
$\mathbf{x}_{t-1}$
$\mathbf{x}_0$

Figure 2.2: The Diffusion Model Markov Chain [7].

done, a sample of Gaussian noise that is incrementally denoised by our trained model through the Markov chain will generate a novel piece of synthetic data that appears to be from the original data distribution.

The training for diffusion models can be broken down into two related processes: the forward process q and the learned reverse process p_θ. The main goal is for our model to learn an approximation of the reverse process p_θ, as this will allow us to apply the model to incrementally denoise a sample of Gaussian noise to generate new pieces of synthetic data. Figure 2.2 illustrates the process of q and p_θ being applied to images in the Markov chain. We will now discuss the forward process q and the reverse process p_θ in more detail.

2.3.1 The Forward Process

For the forward process, we first take a training image x, and apply noise via the posterior function q. What makes diffusion models different than other latent variable models is that the noise is added via a fixed Gaussian noise schedule in a Markov chain according to a variance schedule $\beta_1, ...\beta_T$:

$$q(x_{1:T}|x_0) = \prod_{t=1}^{T} q(x_t|x_{t-1})$$

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I)$$

This is done for a fixed number of timesteps T, and the noise added to the image x_t in the chain is in accordance with the respective value from the scheduler, namely B_t.

2.3.2 The Reverse Process.

The reverse process is done by removing Gaussian noise from images through learned Gaussian transitions in the Markov chain. The diffusion model is tasked to learn this reverse process by approximating the transitions. These transitions are modeled by the function $p(x_T) = \mathcal{N}(x_T; \mathbf{0}; \mathbf{I})$:

$$p_\theta(x_{0:T}) = p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1}|x_t)$$
$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t)$$

2.3.3 Training and Sampling the Diffusion Model

Training is done by optimizing the usual variational bound on negative log-likelihood, but it has been reparameterized by Ho et al. to improve training stability and efficiency. The algorithm for training a diffusion model is seen in Algorithm 2.1; $\mathcal{N}$ is the Gaussian noise function, ϵ_θ is our denoising model that predicts the noise at a given timestep, T is the number of timesteps in the diffusion process, $\alpha_t = 1 - \beta_t$, and $\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$.

Algorithm 2.1: Training Diffusion Models

```
begin
    repeat
        $\mathbf{x}_0 \longleftarrow q(\mathbf{x}_0)$
        $t \longleftarrow \text{Uniform}(1, 2, ..., T)$
        $\epsilon \longleftarrow \mathcal{N}(\mathbf{0}, \mathbf{I})$
        /* Perform gradient descent of $\epsilon$ and $\epsilon_\theta$'s prediction     */
        $\nabla_\theta ||\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0, \sqrt{1-\bar{\alpha}_t}\epsilon, t)||^2$
    until converged
```

If training is done correctly, and the model is able to approximate the reverse process p_θ, we can sample the model for generated pieces of data. We do this by feeding the model samples of Gaussian noise, and it will extrapolate new, synthetic pieces of data. Because the function p_θ maps samples of pure noise to the domain of the original input data, a diffusion model that has learned this mapping will be able to convert a sample of Gaussian noise to new, synthetic data examples that resemble what the model saw from the training data. Algorithm 2.2 details the sampling process for a diffusion model; $\mathcal{N}$ is the Gaussian noise function, ϵ_θ is our denoising model that predicts the noise at a given timestep, T is the number of timesteps in the diffusion process, $\alpha_t = 1 - \beta_t$, and $\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$.

Algorithm 2.2: Sampling Diffusion Models

```
begin
    $\mathbf{x}_T \longleftarrow \mathcal{N}(0, I)$
    for $t = T, ..., 1$ do
        if $t > 1$ then
            $\mathbf{z} \longleftarrow \mathcal{N}(\mathbf{0}, \mathbf{I})$
        else
            $\mathbf{z} = 0$
        $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha_t}}}\epsilon_\theta(\mathbf{x}_t, t)) + \sigma_t \mathbf{z}$
    return $x_0$
```

2.3.4 Embedding Time

One important piece is to incorporate time into the diffusion process; our model needs to know where it is in the Markov chain when learning the reverse approximation p_θ. The method for incorporating this temporal information is transformer sinusoidal time embeddings, which is a technique used in transformer architectures [20]. Transformer sinusoidal time embeddings are a technique to use sinusoidal functions to transform an integer representation of time into something easier for our model to understand. Denoted as positional embeddings in the original paper, the functions for determining the position *pos* with a dimension i and output model dimension of d_{model} can be seen below:

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}})$$

This allows us to embed a position *pos* into a vector of dimension d_{model} with ease, and this new information is vital to our learned diffusion process as this will allow our model to know what step it is in the diffusion Markov chain for each training example.

2.4 Latent Diffusion Models

Latent diffusion models (LDMs) are a subset of diffusion models developed by Rombach et al., as described in the paper *High-Resolution Image Synthesis with Latent Diffusion Models* [15]. LDMs differ from common diffusion models as they split up the training process into two parts: the autoencoder training phase, and the diffusion model training phase.

2.4.1 Training the Variational Autoencoder

For the first step of the training process, Rombach et al. propose training a variational autoencoder (VAE) that can provide a lower-dimensional representational space for the data. VAEs are a deep learning architecture with an encoder-decoder structure, and in the middle, they contains a compressed, latent representation of the data [18]; this latent representation is a much more condensed representation of the original data space, which makes these latent vectors computationally simpler to interact with than images.

2.4.2 Training the Latent Diffusion Model

Once the VAE is trained, latent diffusion models perform their training on data represented in the *latent space*, as opposed to data in its original image space. Rombach et

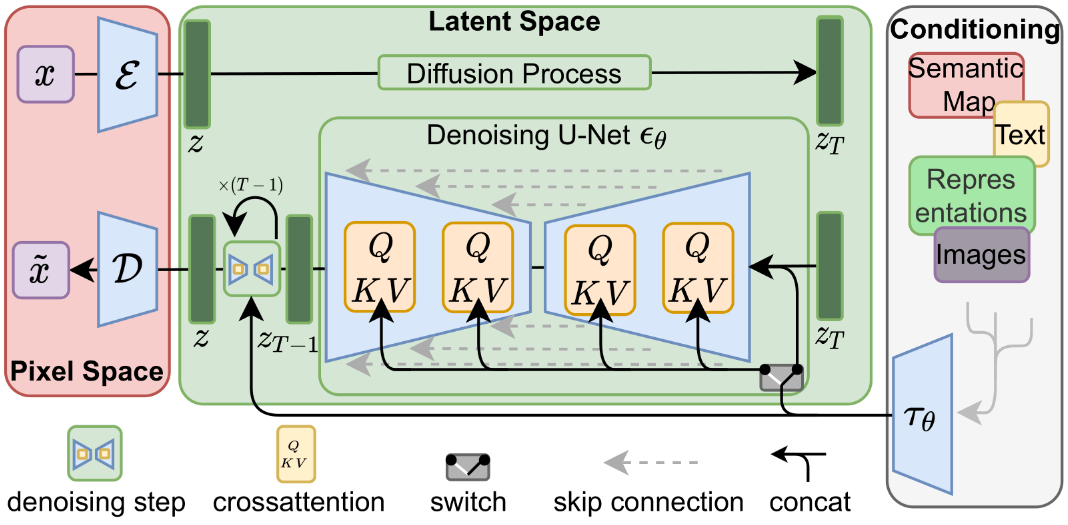


Figure 2.3: The Latent Diffusion Model Architecture [15].

al. depicts how training diffusion models in the latent space allows for reduced complexity in the model, which ultimately leads to efficient image generation from the latent space with one single network pass on the decoder-portion of the autoencoder [15].

As seen in Fig. 2.3, the Latent Diffusion Model encodes the data, x, in the pixel space via the encoder ϵ to the latent space, where it is represented as the vector z. During training, this z vector undergoes a diffusion process to yield z_T, before it goes through a denoising U-Net model ϵ_θ which yields an approximation of the vector z_{T-1}. This vector finally undergoes a denoising step which returns z, and this is finally passed through the decoder D to give us a reconstructed image $\tilde{x}$, which is the approximation of the original input x in the image space.

Please note that Fig. 2.3 contains conditional inputs that are embedded into the z_T vector both during the Denoising U-Net ϵ_θ and the denoising step; our implementation utilizes an unconditional latent diffusion model, so this conditioning is not used in the context of our work.

Chapter 3

RELATED WORK

The domain of synthetic music generation is extremely broad, and researchers are able to take vastly different approaches when attempting to create music. One distinction that we aim to define in the context of this paper is that there is a difference between music generation and music extension; music generation aims at synthetically creating a piece of music, while music extension receives musical information as input and attempts to extend the music by predicting the following portion of music. A majority of implementations are only focused on the generation problem, while we are focused on the extension problem.

As there are many attempts to synthetically generate music, we will narrow the focus of this chapter to only cover the subset of methods that involve spectrograms, as we believe these methods pertain to the subject of our thesis the most.

3.1 Audio Diffusion

HuggingFace is an online platform that provides users with easily accessible models, datasets, and demo apps [1]. A library provided by HuggingFace is the *Diffusers* library, which allows users to interact with state-of-the-art diffusion models with ease. It also allows users to utilize transfer learning with pretrained diffusion models hosted on the platform. One package hosted on HuggingFace that is similar to our work is the *Audio Diffusion* package; Robert Dargavel Smith was the author of this package, and it contains various implementations of diffusion probabilistic models in the context of spectrogram music generation [16]. Our project utilizes this repository to convert audio waveforms to and from Mel spectrograms.

To extend their music, the *Audio Diffusion* package generates variations of a root image by tweaking the starting step of the diffusion process; this allows them to create and concatenate different variations of similar synthetic audio, which ultimately creates longer periods of novel music.

3.2 Riffusion

Another related method is the Riffusion library, which uses stable diffusion in the context of spectrogram music generation [3]. They create music by generating multiple spectrograms of novel music via a text prompt from the user, and they use tools to loop between these spectrograms to extend their songs. The Riffusion team trained their denoising model via a predefined diffusion architecture provided by the HuggingFace platform [1]. Their implementation differs from Audio Diffusion in that it uses stable diffusion, which uses transformers to embed textual information that can influence the diffusion process. This ultimately allows users to specify what kind of music they want to generate via a text prompt.

The Riffusion team's technique for extending music is to interpolate vectors in the latent space, which is possible due to properties of variational autoencoders. They generate various spectrogram images of music from the same text prompt, and then convert each to the latent space using the encoder portion of their trained VAE. If they were to simply concatenate these spectrograms, the combined music would sound very disjoint as the transitions between the music derived from the spectrograms are often abrupt. To improve the transitions between two seed images, they take the linear interpolation of two latent z vectors of these images at fixed intervals in

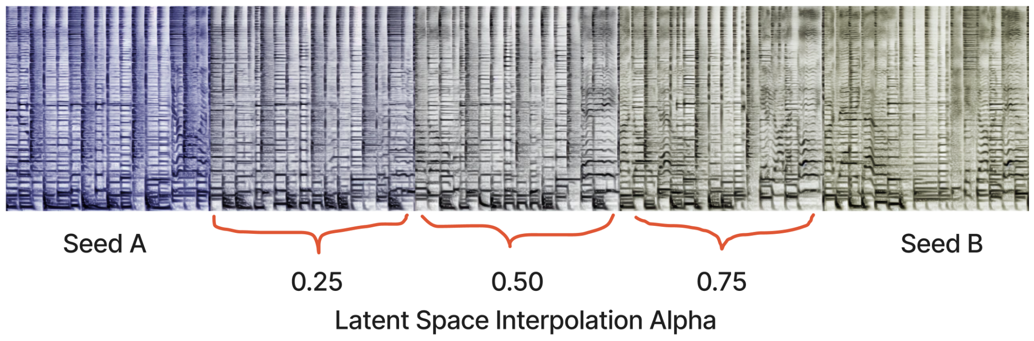


Figure 3.1: Riffusion's Music Extension Technique [3].

the latent space, and each of these interpolations corresponds to a new image once decoded by the VAE. Thus, this allows them to seamlessly transition between two seed images without the harsh jumps. Figure 3.1 shows the images that result from the interpolation done in the latent space.

Chapter 4

IMPLEMENTATION

We have split this implementation chapter into two distinct parts: our attempted methods while navigating the domain of music generation using diffusion, and the implementation details of the Continuer Pipeline, which is our proposed, novel technique to extending music.

4.1 Attempted Methods

At the start of this thesis work, music synthesis techniques using diffusion were nonexistent; thus, the scope of what we were working on was variable, and it depended on where the scientific community was in the process of creating diffusion-driven audio synthesis techniques. At the start of the work, our goal was to demonstrate that diffusion could be done on spectrograms to generate novel music, because this was originally a completely novel idea that we created. As the scientific community was able to implement this idea faster than the authors, we changed the scope of the thesis to the Continuer Pipeline. Below, we detail the work done before creating the Continuer Pipeline in the same sequence that they were developed.

4.1.1 Diffusion from Scratch on Images

At the start of the thesis, we first wanted to create a codebase that was able to generate images using diffusion from scratch, without the help of diffusion libraries. To do this, we followed various open-source tutorials to implement a diffusion model on the

Stanford Cars dataset [8]. Once our implementation matched what was found online, we tweaked the model architecture and hyperparameters to try and improve the quality of the newly generated images. We also wanted to extend the model architecture to support the generation of 256x256 images, as a diffusion model implemented in the tutorial could only generate images with a 64x64 resolution. We deemed 64x64 resolution spectrograms as not adequate for the audio generation problem as the lower resolution would only correspond to a very small increment of music.

The goal of this intermediate implementation was to demonstrate that diffusion could be done from scratch, and we planned on extending the framework to train on spectrograms instead of the cars dataset.

4.1.2 Spectrogram Converter

To handle the conversions between spectrograms and waveforms, we decided to implement a custom class that is able to do these conversions dynamically. Using the Python audio library *Librosa*, we were able to successfully convert audio waveforms to spectrograms with ease [9]. With the nature of this problem, this work is pointless if we are not able to convert Spectrograms back to audio waveforms.

As described in the Background Section, the Griffin-Lim algorithm allows us to reconstruct waveforms from a spectrogram using an approximation technique [6]. To see how this would be used in the context of our problem, we took a segment from a Jazz song, converted it to a Spectrogram, and then approximated the original audio using the Griffin-Lim algorithm.

In this early stage, we initially determined that the lossy nature of the Griffin-Lim algorithm was going to severely hinder the results of our work. Because of this, we decided to experiment into determining if we can derive tangible results by performing diffusion on the raw waveforms themselves.

4.1.3 Diffusion on Waveforms

This work was tangential to the original research focus, and our hope was to create a baseline of results that we could improve when returning to Spectrogram-focused diffusion techniques.

To attempt this, we simply modified the architecture of the model described previously that performed diffusion on the Stanford cars dataset [8]. Because the dimensionality of our data was different, we needed to make various changes to the model architecture. For example, the waveform data contains two dimensions, which differed from the three dimensions that the spectrogram has. Because of this, we adjusted our model to have 1-D convolutional layers instead of 2-D convolutional layers.

We then trained this on a dataset that was obtained via a custom-built script that downloaded songs from a YouTube Jazz playlist and split them into five-second waveforms.

The results of this were poor; the model either produced waveforms that resembled noise, or the output waveform was too quiet to hear. It was around this time that diffusion approaches to spectrogram generation became prevalent in the scientific community, so we decided to switch our focus to analyzing the existing implementations.

4.1.4 Recreating Riffusion

Around this point of the developmental process, the Riffusion team in San Francisco was able to deploy an open-source implementation of a stable diffusion model that utilized spectrograms to create novel music [3]. This implementation was essentially a superset of what we had originally planned to do: our initial goal was to create a diffusion model that generated Jazz segments, while the Riffusion team built a music generator that allowed a user to pass in text specifying a desired genre. Thus, we deemed that the next step in our process was to recreate the Riffusion project in the context of unconditional generation only on Jazz songs.

The Riffusion team had left their interface with their trained model open source, but their training script was in a private repository; thus, we took inspiration from how they interacted with their trained model in our implementation.

4.1.5 HuggingFace Audio Diffusion Implementation

Around this time in the thesis work, we discovered the *Audio Diffusion* library within the HuggingFace platform [16]. Using this package, we utilized the *Mel* class, which gives streamlined functionality to convert audio waveforms to Mel spectrograms and back with ease.

After reviewing this library, we wanted to extend their techniques to our dataset. To do this, we trained a latent diffusion model from scratch on our Jazz dataset. We used a pretrained variational autoencoder from the *Audio Diffusion* package that was able to bring spectrograms of 256x256 resolution to the latent space with a resolution of 64x64. This allowed us to focus on the diffusion model's training, rather than spending effort attempting to also fine-tune a VAE.

4.2 The Continuer Pipeline

At this point in the thesis work, we aimed to contribute something new to this domain. Our initial goal was a novel idea at the start of this thesis work, but it was implemented in an excellent fashion by groups such as the Riffusion team and the contributors of the *Audio Diffusion* library.

Thus, we decided to focus on a related problem: extending music. We found that the existing techniques are focused primarily on generating novel music; we determined that once the music is generated, there was room for improvement in how these models are able to extend the existing music. Thus, we propose a new method for extending music that we coined as the *Continuer Pipeline*.

The main idea for this technique revolves around changing how we extend music. Inspiration was drawn from how recurrent neural networks (RNNs) are able to handle temporal data; we have created a custom diffusion architecture that takes in multiple spectrograms of a song stacked on top of each other in the depth axis, and the model is trained to produce the *next* spectrogram. This is described in more detail below.

4.2.1 The Pipeline Implementation

First, we must address the format in which this pipeline was created in. Because of the accessibility and functionality that HuggingFace provides, we decided to implement this pipeline extending HuggingFace's *DiffusionPipeline* interface.

4.2.1.1 The DiffusionPipeline Interface

Because diffusion is becoming such a popular deep learning technique for generative tasks, HuggingFace has devoted a large portion of its platform to hosting various diffusion models; the main interface for developing and using these models is the *DiffusionPipeline* class [1]. A custom class that extends the *DiffusionPipeline* class will have a lot of functionality provided by HuggingFace, including methods that can access many existing models and datasets that are already hosted on the HuggingFace platform. Because of this, we decided to implement ours in this fashion; specifically, the Continuer Pipeline extends the *DiffusionPipeline* class, and it contains all of functionality implemented in the *DiffusionPipeline* class.

4.2.1.2 Pipeline Architecture

This section describes the architecture details of the pipeline. At a high level, our pipeline takes in an input of two spectrograms stacked on each other, denoted as x_0 and x_t, which correspond to both the start and a portion of the same song. The output of the pipeline is the model's prediction of the spectrogram representing the next five seconds of the song, denoted as $\hat{x}_{t+1}$. Figure 4.1 illustrates this process at a high level with example spectrograms for the inputs and generated output.

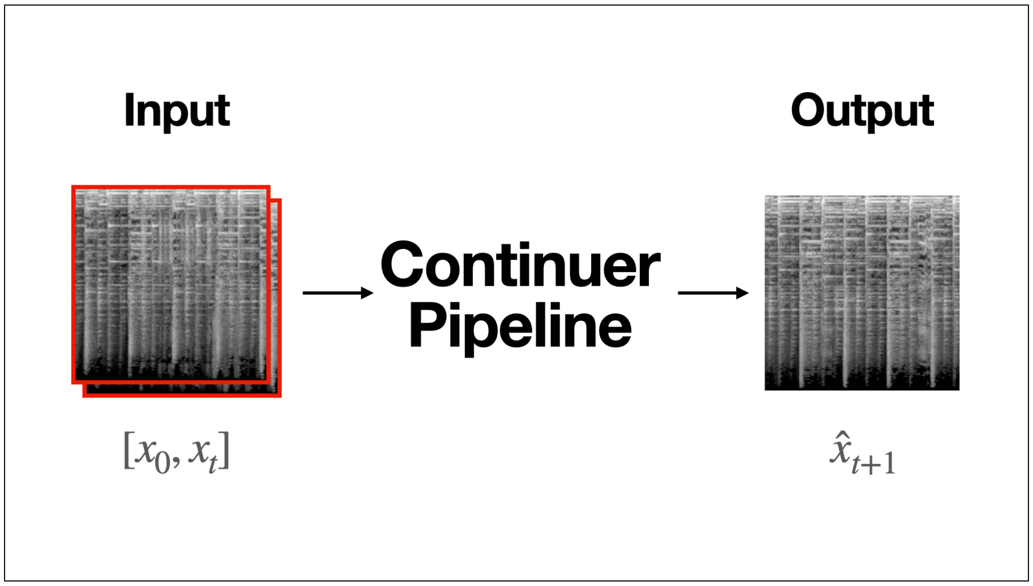


Figure 4.1: Input and Output of the Continuer Pipeline.

To define this in more clarity, imagine a song is split into five-second increments, and each increment is converted to a Mel spectrogram. Thus, we can imagine that a song can be converted to the form $x_0, x_1, x_2, ..., x_t$ where x_0 is the Mel spectrogram representing the first five seconds of the song, and x_t is the spectrogram representing the seconds $5t$ to $5t + 5$ of the song. This can be illustrated in Figure 4.2 below; there is a 15-second waveform, and its partition into into three Mel spectrograms is illustrated.

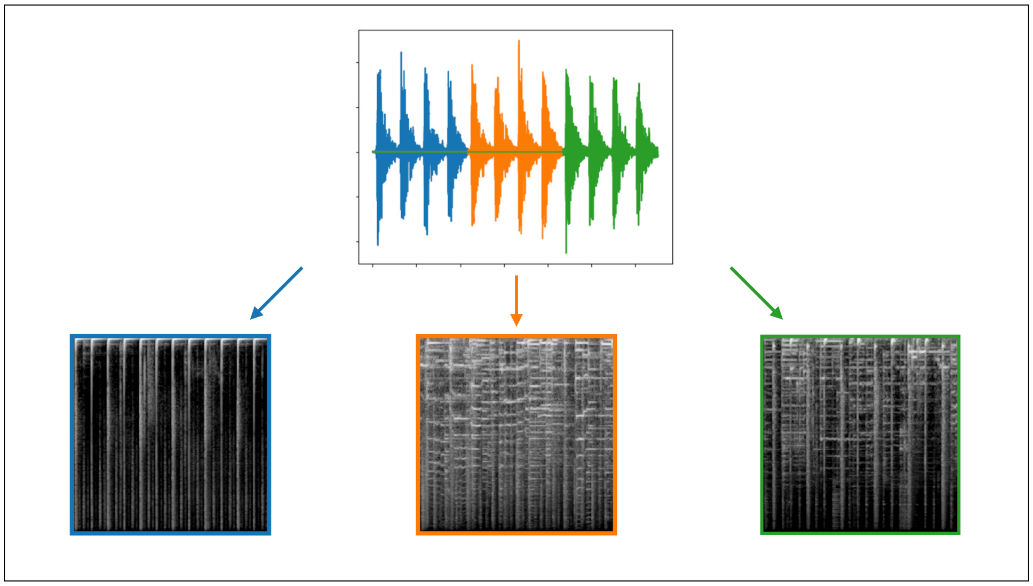


Figure 4.2: The Partitioning of a Waveform into Spectrograms.

Our initial goal was to have our model simply predict the spectrogram that represents the next five seconds of a song. Our first plan to achieve this was to have our model receive any spectrogram x_t, and use latent diffusion to predict what the spectrogram x_{t+1} could look like. If we achieve this, this will allow us to extend music infinitely, as iteratively calling the model will continuously keep extending the song by five seconds. In theory, this could work, but we foresaw a problem with this approach.

Imagine the scenario where we want to create a 30-second synthetic extension to a song. First, we must generate seconds 0 through 5 of the synthetic extension. This process would entail taking the last spectrogram of the original song, namely x_t, and using it as input to the Continuer Pipeline. The output would then be the synthetic spectrogram representing the next five seconds, x_{t+1}. The next step in this process is to generate seconds 5 through 10; this would entail passing in the generated spectrogram x_{t+1} to our model as input, and the output of the pipeline would be x_{t+2}. This process would repeat four more times to generate the remaining 20 seconds, and we can see that the last call to the pipeline would be passing in x_{t+5}

as input, and the resultant output would be x_{t+6}. This could work in theory, but any small imperfections that could occur in any given synthetic spectrogram will very likely be exacerbated in the generation of all future spectrograms.

This concept heavily resembles the game of telephone, which is a game where a message is passed sequentially from one person to the next through whispered communication. One of the main issues with this game, and the reason that it often proves to be a challenge, is that any altercations to the original message will be propagated onward through the chain; for example, if a person mishears the word "chains" as "brains", their error will be propagated onward as the subsequent person will incorrectly tell the next person that the secret message is the word "brains."

Thus, we hypothesized that only including x_t in the input would be poor, and some information of the original, non-synthetic song should always be included as part of the input to the model. Our current approach is the following: the input to the Continuer Pipeline will include information from both the first five seconds of the song and the last five seconds of the song. To define this more concretely, the input to our Continuer Pipeline would be the spectrograms $[x_0, x_t]$ stacked on top of each other, and the output would be the predicted spectrogram x_{t+1} that represents the following five seconds.

For the internal neural networks within the Continuer Pipeline, we have a variational autoencoder (VAE) that can bidirectionally convert the spectrograms to the latent representation and back, and we have our denoising U-Net that is responsible for learning the diffusion process of these latent vectors.

As described previously, the Variational Autoencoder is a pretrained VAE from the *Audio Diffusion* library [17]. The model was trained on the *audio-diffusion-instrumental-hiphop-256* dataset, which is described in more detail below. The VAE has an encoder-decoder structure; the encoder has four down blocks with each block containing two convolutional layers. In each block, the two convolutional layers are the same size, and the sizes of the blocks are 128, 256, 512, and 512, respectively. The decoder is essentially a mirrored architecture of the encoder, but it has deconvolutional blocks, with sizes 512, 512, 256, and 128 over its four blocks.

The core model that learns the diffusion process is a U-Net model that extends the *UNet2DModel* class from the *Diffusers* package [1]. The model architecture was inspired by the architecture of the U-Net present in the *Audio Diffusion* package, and it resembles a classic U-Net shape; it has 6 blocks in the first half, each with two layers. The block sizes in the first half of the U-Net are 128, 128, 256, 256, 512, and 512. The second half is a mirrored representation of the first half; there are 6 blocks in this portion, each with two deconvolutional layers, and they have sizes 512, 512, 256, 256, 128, and 128. Also, there are residual connections that connect the blocks in the first half to their corresponding block in the other half; for example, there is a residual connection that connects the first block with the last block of the U-Net, a connection that connects the second block with the second-to-last block, etc. Figure 4.3 depicts graphically the architecture of the denoising U-Net model.

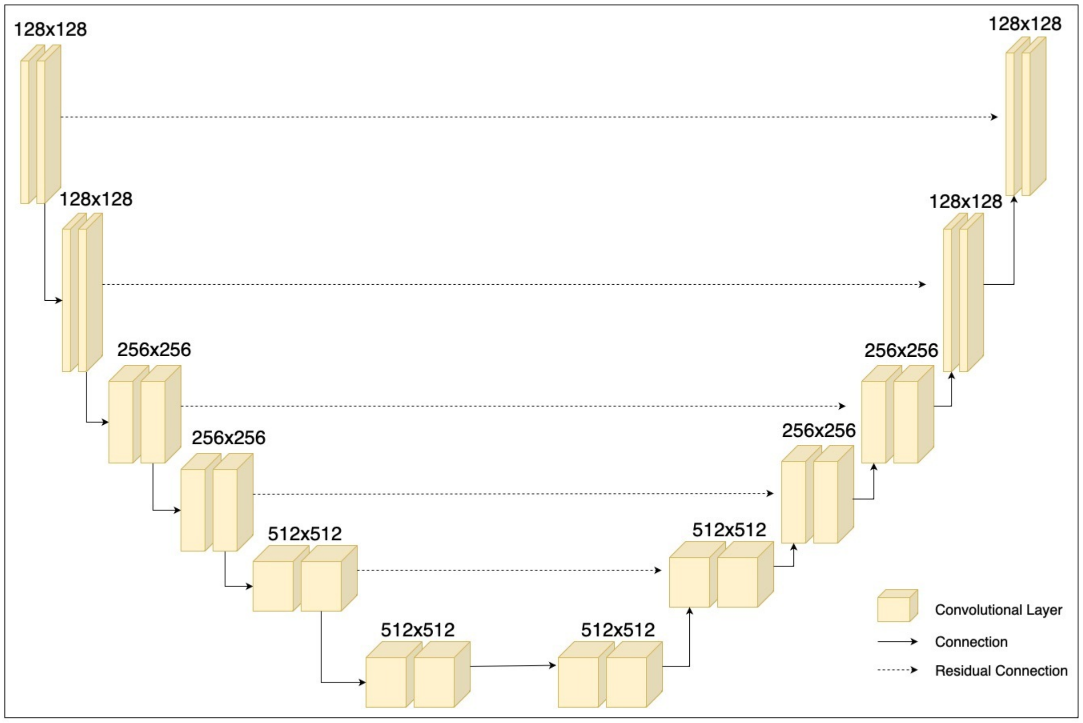
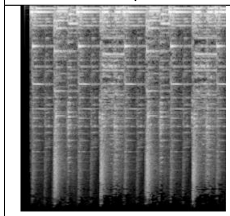
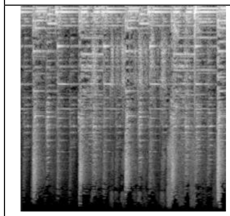
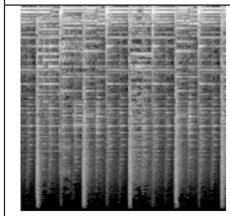
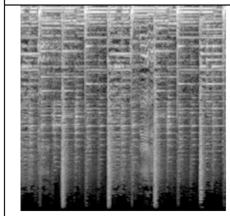


Figure 4.3: The Continuer Pipeline's U-Net Architecture.

4.2.2 The Data

The data used to train the Continuer Pipeline is the *audio-diffusion-instrumental-hiphop-256* dataset [17]. This is a publicly hosted dataset on the HuggingFace hub, and it was created by the same author of the *Audio Diffusion* library. The dataset consists of 256x256 spectrograms of hip-hop songs, where each spectrogram corresponds to five seconds of audio. The dataset contains three columns: *image*, *audio_filename*, and *slice*. The *image* column contains the raw image data of the spectrograms, the *audio_filename* column contains the name of the song that spectrogram was created from, and the *slice* column is an integer that states what slice of a song a specific spectrogram corresponds to. For example, if a song had a slice value of zero, we would know that this corresponds to seconds 0-5 of the song. Table 4.1 contains the first few entries of the dataset.

Table 4.1: The audio-diffusion-instrumental-hiphop-256 Dataset from HuggingFace [17].

image (image)	**audio_file (string)**	**slice (int16)**
	"./Onra and Quetzal - Gotta Have It.mp3"	0
	"./Onra and Quetzal - Gotta Have It.mp3"	1
	"./Onra and Quetzal - Gotta Have It.mp3"	2
	"./Onra and Quetzal - Gotta Have It.mp3"	3

4.2.3 Custom Data Loading

The HuggingFace *Diffusers* library is built upon the popular deep learning library *PyTorch*, which is a Python library that implements various deep learning architectures and methods [12]. Because of the unique way that the Continuer Pipeline handles the training data, we had to develop an algorithm to dynamically load in data from the *audio-diffusion-instrumental-hiphop-256* dataset during the training of our model; this algorithm is described in more detail below.

This algorithm can be broken into two portions; the first one is done statically, and it is counting the number of slices in each song in the data set, and storing the individual lengths of each song in a hashmap. The second part is creating a function that allows us to dynamically retrieve the correct three spectrograms for any index given by the *PyTorch* library.

4.2.4 Counting the Number of Slices

The reason for counting the number of slices in each song is because this is necessary data for our indexing algorithm. It is not necessary information if you are randomly retrieving a chosen spectrogram at any time, but the nature of how we train our Continuer Pipeline makes the total number of slices in each song a necessary piece of information. To give a concrete example, we will look at the song "Gotta Have It" by Onra and Quetzal. The song has 20 total slices, which indicates that it is a length of $5 * 20 = 100$ seconds long. Thus, the resultant hashmap entry for "Gotta Have It" will have a value of 20.

4.2.5 Developing a One-to-One Function for Dynamic Indexing

The second part is creating a one-to-one function that is able to dynamically convert any specified index within a certain range to a data example that we will use to train our model. The reason this needed to be created is because of how the *PyTorch* library handles loading data for training a neural network; to create a custom class for data loading that the *PyTorch* API can interact with, you need to specify two items: the length of the dataset, and what piece of data a certain index corresponds to.

To describe this in more detail, we will first concretely define a single training example for a Continuer Pipeline; the diffusion model ultimately needs three spectrograms to train: x_0, x_t, and x_{t+1}. x_0 is the spectrogram pertaining to the first five seconds of a song, x_t is the spectrogram of the seconds $5t$ to $5t + 5$ of the same song, and x_{t+1} is the spectrogram corresponding to seconds $5t + 5$ to $5t + 10$. With supervised learning tasks like this, we need a feature vector $\mathbf{X}$, and a corresponding label vector $\mathbf{y}$ to perform train of a model. In this context, we defined our feature vector $\mathbf{X}$ and label vector $\mathbf{y}$ as seen in Algorithm 4.1.

Algorithm 4.1: Preparing the Continuer Pipeline's Training Data

```
Data: $S$ is a collection of songs where each $s_i \in S$ can be represented with
      the form $s_i = (x_0, x_1, x_2, ..., x_{l(i)-1})$, where each $x_i$ is a five-second
      spectrogram, and the function $l(i)$ returns the number of
      spectrograms inside of the song $s_i$.
begin
    $\mathbf{X}, \mathbf{y} \longleftarrow \emptyset$
    for $s_i \in S$ do
        for $j \in (0, 1, 2, ..., (l(i) - 2))$ do
            $\mathbf{X} \longleftarrow \mathbf{X} \cup \{(x_0, x_j)\}$
            $\mathbf{y} \longleftarrow \mathbf{y} \cup \{x_{j+1}\}$
```

As we can see, our algorithm takes our dataset of individually sliced spectrograms and aggregates them in a way that we can easily get tuples of three spectrograms corresponding to (x_0, x_t, x_{t+1}), which is what we ultimately need to train our model. We can see that the our training data will for each example take in the first spectrogram of a song and a spectrogram x_t from $\mathbf{X}$, and the corresponding element in $\mathbf{y}$ will be the spectrogram x_{t+1}. Figure 4.4 details this song partitioning in further detail; we can see that we have two songs, a and b, with lengths of four and three, respectively. The song with a length of four will yield three examples for training, and the song with a length of three will yield two examples to train on.

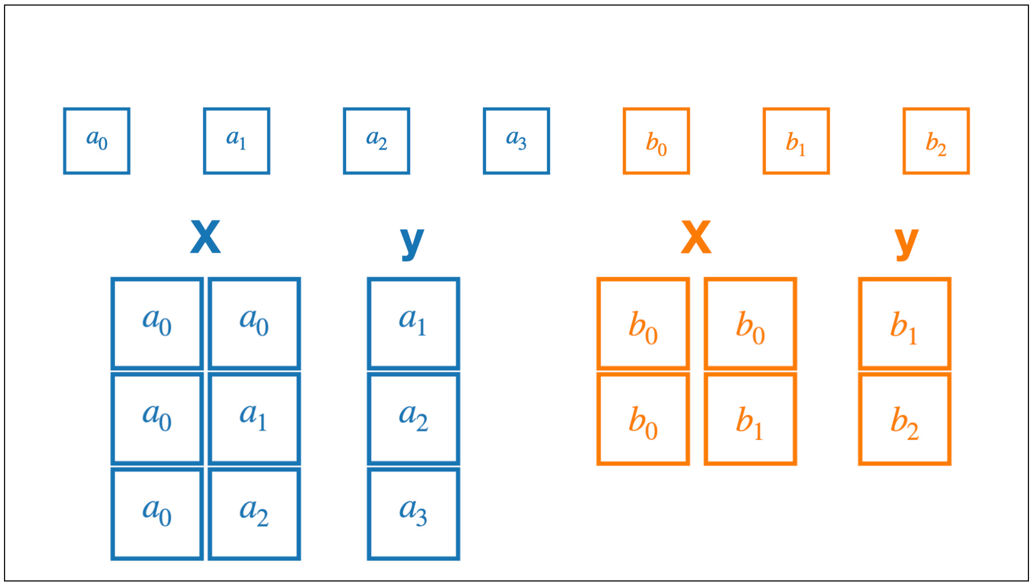


Figure 4.4: The Partitioning of Songs for Training.

4.2.6 Training the Continuer Pipeline

To train the denoising U-Net at the core of the Continuer Pipeline, we converted a training script provided by HuggingFace to train our model using the custom data loading class that implements the Algorithm 4.1 above. We wanted to focus our attention on training the U-Net model, so we utilized a pretrained Variational Autoencoder from the *Audio Diffusion* library.

During training, the VAE used in the pipeline is pretrained, so our training suite only focuses on updating the weights of the denoising U-Net. The training process is similar to what is present in literature for training diffusion models, but it differs in that the model takes in two spectrogram latents, namely z_0 and z_t, in addition to Gaussian noise. The algorithm for training our denoising U-Net model ϵ_θ can be found below in Algorithm 4.2. The intuition behind this algorithm is described in more detail in the Background chapter, but $\mathcal{N}$ is the Gaussian noise function, ϵ_θ is our denoising model that predicts the noise at a given timestep, T is the number of timesteps in the diffusion process, $\alpha_t = 1 - \beta_t$, and $\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$.

Algorithm 4.2: Training the Continuer Pipeline's U-Net ϵ_θ

```
Data: $X, y$ are our datasets from Algorithm 4.1, where each element of $X$
      can be represented in the form $(x_0, x_t)$, and the corresponding
      element in $y$ is $x_{t+1}$.
begin
    for epoch = $1, 2, ...,$ Number of Epochs do
        for $(x_0, x_t), x_{t+1} \in X, y$ do
            /* Convert spectrograms to latent space                */
            $z_0, z_t, z_{t+1} \longleftarrow \text{VAE}(x_0), \text{VAE}(x_t), \text{VAE}(x_{t+1})$
            $x' \longleftarrow [z_0, z_t, z_{t+1}]$
            $t \longleftarrow \text{Uniform}(1, 2, ..., T)$
            $\epsilon \longleftarrow \mathcal{N}(0, I)$
            /* Perform gradient descent of $\epsilon$ and $\epsilon_\theta$'s prediction  */
            $\nabla_\theta ||\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t} x' + \sqrt{1 - \bar{\alpha}_t}\epsilon, t)||^2$
```

As we can see in Algorithm 4.2, we choose pairs (x_0, x_t) and x_{t+1}, transform them to latent vectors z_0, z_t, and z_{t+1}, and we perform training by having our model, ϵ_θ predict what the noise is given the stacked input $[z_0, z_t, z_{t+1}]$ and timestep t. Once our model learns this approximation of the Gaussian noise at a certain timestep, the sampling process for creating new examples is identical to what is found in diffusion model literature. This ultimately trains the U-Net to predict the noise that was added to the previous spectrogram in the Markov chain, which is what allows us to use this model to predict a spectrogram x_{t+1} given spectrograms x_0 and x_t.

For the training process, our model trained for a total of 5 epochs over the entire dataset, and our batch size was set to 1. As described previously, the dataset was the *audio-diffusion-instrumental-hiphop-256* dataset, which were 256x256 spectrograms that correspond to five-second increments of various hip hop songs.

The environment that this pipeline was trained in is the Massively Parallel Accelerated Computing (MPAC) laboratory located at California Polytechnic State University, San Luis Obispo. The system that this model was trained on contains two AMD Ryzen Threadripper 3990X 64-Core Processor CPUs, and one NVIDIA RTX A6000 GPU with around a total of 48GB of RAM.

Chapter 5

DISCUSSION OF RESULTS

The nature of the problem that we are attempting to solve is to have our model essentially *guess* what the next five seconds of a song will sound like. Because of this, challenges arise when attempting to evaluate the results of our Continuer Pipeline. In this chapter, we will first discuss the challenges of creating an objective evaluation metric for the quality of our pipeline. We will then provide a subjective evaluation, detailing both the existing limitations of the Continuer Pipeline, and what its intended use should be in its current state.

5.1 Challenges of Defining Objective Criteria

There exist means to evaluate music in an objective manner, but we will show that they are not applicable in the context of our problem. Some of the objective metrics that exist, such as FAD (Frechet Audio Distance), Scale Consistency (SC), and Pitch Entropy (PE), are metrics that can measure the *closeness* between two pieces of music, but this is not applicable towards our problem [23].

In the other implementations of synthetically creating music, utilizing these objective evaluation metrics is possible, as the focus of these other implementations is different than that of the Continuer Pipeline. To give an example of where that these techniques are applicable, we will analyze inpainting in the context of music synthesis.

Inpainting is a technique that allows a deep learning model to generate only portions of a larger image. It allows a user to pass in a mask detailing what specific pixels the

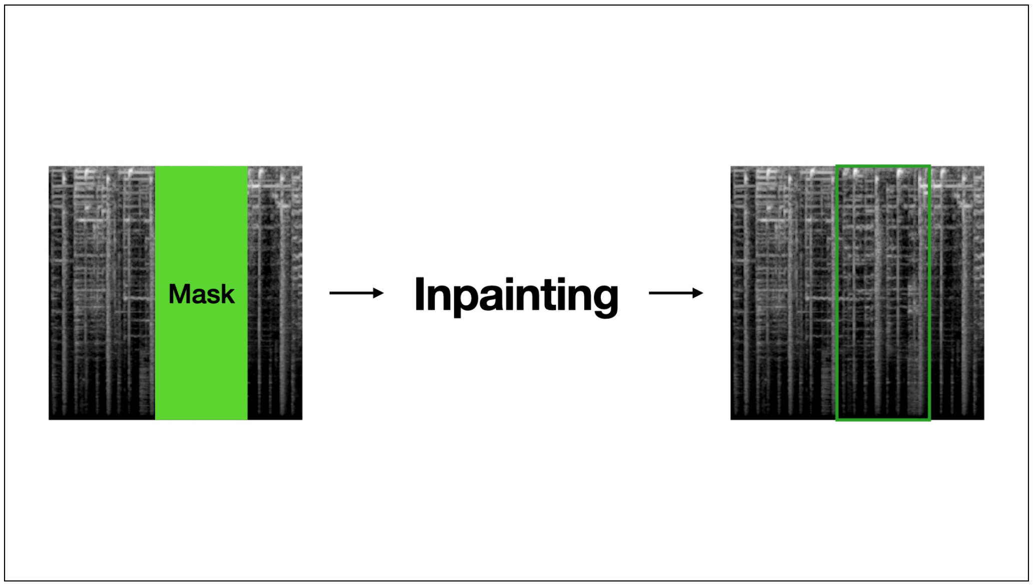


Figure 5.1: Spectrogram Inpainting Illustrated.

deep learning model should synthetically fill, and the model will replace that mask with the pixel values it believes are in the image [2]. This is used specifically in the context of music generation by a user masking a portion of a spectrogram, and the model filling in the masked portion with what it thinks the spectrogram should look like; Figure 5.1 illustrates this process. The inpainted spectrogram is finally converted back to audio by the same techniques done in our work, and this is how inpainting is able to create music.

The evaluation process of this is defined via the nature of the problem: you can simply compare the inpainted prediction with the true image. For example, assume a spectrogram corresponds to 10 seconds of audio. The user specifies the mask over seconds 5 through 8, and the inpainting technique attempts to fill in this three-second mask by looking at the context of the surrounding pixels. The evaluation for this is clear and can be done quantitatively; you can simply utilize the existing tools that measure the distance between audio, such as FAD, SC, and PE, and measure the distance between the audio of the original spectrogram and its inpainted counterpart. This will yield an objective evaluation for the performance of the inpainting technique, as the nature of the inpainting problem is to correctly inpaint an image with the correct pixels.

The Continuer Pipeline's goal is different than this, as it aims to extend music by adding new, unique spectrograms based off of previous portions of a song. To train the Continuer Pipeline, we implemented a diffusion-model training suite which ultimately used an objective loss function. Although this was a way to quantify how close the Continuer Pipeline's predicted music was to the original audio, we believe that this does not accurately reflect the true intentions of the Continuer Pipeline.

We want the Continuer Pipeline to maintain some creativity in its generation process when extending songs, and using an objective metrics to measure audio distances fail at quantifying the pipeline's goal. Because the Continuer Pipeline is aiming to extend work with its own creative motifs, we believe that quantifying the work in an objective manner is nearly impossible. If one were to use the objective measures described previously on the results of the Continuer Pipeline and the true music samples, we believe that creativity from the Continuer Pipeline would be punished. Thus, we affirm that objective metrics in this problem's context are inapplicable, and the best way to evaluate this work is to provide a subjective evaluation.

To further evaluate this work, we believe that utilizing musical professionals or listeners of our target musical domain should be consulted as they will provide the best subjective evaluations. Unfortunately, this was not practical due to time constraints of the thesis, so we will instead provide a subjective evaluation from the authors below; it will reflect the authors' opinions on the current state of the work.

5.2 Subjective Evaluation

This section is the authors' attempt at providing an evaluation for the current status of the Continuer Pipeline; as detailed previously, it is very difficult to provide an objective evaluation, but we will describe the results qualitatively while attempting to be as objective as possible.

The current state of the Continuer Pipeline is that it is extremely early in development, and the music quality is definitely not at the quality of the original samples. A listener could definitely tell that this music was synthetically created. The synthetic extensions of music from our pipeline generally sound like they could be in the same song as the original samples, but they would never pass as being human-generated. We believe that future developments that yield improvements to the quality of the sound are extremely possible, but the technology in its current state produces synthetic samples that are inadequate compared to human-generated music.

Because of the inability to create music at the same quality of the samples it was trained on, we believe that the motivation for using this technology should be changed until the quality of generated music increases. We propose that this technology could be used as a means for aiding musicians as a supplemental tool when creating music.

In the generated waveforms from the Continuer Pipeline, there are elements of a beat with some melodic properties, so we believe that musicians can use the Continuer Pipeline on their music to essentially act as a tool for inspiration. A musician could produce a song, extend it with the Continuer Pipeline, and replicate elements of the generated music that they find interesting. The artists can essentially use the Continuer Pipeline to inspire them by providing new motifs that they can reproduce in their music, and we believe that using the pipeline in this manner would be best until the quality of the generated audio becomes comparable with human-generated music.

It should be noted that the output of the Continuer Pipeline will ultimately be waveforms, and it is not likely that musicians would directly include these in their songs because the generated samples' quality is currently pretty poor. Rather, we affirm that the musicians can first listen to the synthetic music extensions, and then draw creative ideas from the synthetic extensions that they can reproduce later.

In the Future Work chapter, we will detail improvements we believe will aid in the generation process, but it should be said in this evaluation that the current technology is not capable of producing synthetic waveforms that are comparable with human-generated music.

Chapter 6

CONCLUSION

Although the scope of this thesis fluctuated through the development process, we affirm that tangible results were retrieved and valuable contributions to the scientific community were made. Below, we detail what we achieved during the development of this project; we will discuss our various approaches during the first portion of the work, and then we will describe what was done to implement the Continuer Pipeline.

6.1 Various Approaches to Diffusion on Audio Attempted

While arriving at our final implementation, we traveled down a lot of different avenues of development. At the start of the project, we were able to implement a successful diffusion model that generated fully synthetic car images found in the Stanford Cars dataset [8]. The point of doing this was to create a sufficient diffusion suite that could be converted to generating spectrograms, but this ultimately was not usable as the scale of the generated images was too small; the model architecture was sufficient for generating images with a 64x64 resolution, but this is not feasible for the music extension problem as 64x64 resolution spectrograms have too little musical information to produce good results.

After completing this first step, we had a functional suite for training diffusion models in PyTorch from scratch, but the models were limited to only producing 64x64 car images. Our next step was to attempt to extend the codebase towards training diffusion models on spectrogram images instead of the cars dataset. This entailed analyzing if we could simply increase the model complexity to be able to generate

256x256 spectrograms, and running various training iterations to see if this would work. Unfortunately, this was not suitable, as we believe the basic network architecture of the U-Net portion of the diffusion model was not sufficient, so we started to look into changing our approach to this problem.

Our next step was a quick exploration into focusing the training of diffusion models on the raw waveforms themselves rather than the spectrogram representations. This was an explorative portion of the project, where we wanted to see if approaching the problem in this manner could yield any tangible results. Unfortunately, our trained diffusion model was not able to reach convergence on the raw waveforms; the outputs were either noise or empty waveforms. We hypothesize that the model could not learn the diffusion process on musical waveforms directly as the phase information is extremely variable between different songs, and diffusion on images is a much more feasible task.

After analyzing diffusion on waveforms themselves, we began to explore the existing implementations to see if there were contributions to be made. We simultaneously analyzed both the Riffusion library and the Audio Diffusion work from HuggingFace; it was after this that we performed training of an existing diffusion model architecture on a custom dataset generated by the Audio Diffusion authors. This work was done to increase our understanding of transfer learning in the diffusion model space, and it was during this work that we thought of the idea for our novel approach to extending music.

We believe that all of the intermediate steps that were done on the path to constructing the Continuer Pipeline were extremely valuable; although no tangible results were derived from the work, we believe that the exploration done during these varying branches of development is a very beneficial contribution to the scientific community.

6.2 Novel Technique Proposed

Our main achievement in this work is the proposed, novel technique for extending music, coined as the Continuer Pipeline. This technique was inspired by how Recurrent Neural Network nodes handle temporal data; we believe that techniques for extending music should include the most recent increment of music as input, *and* they should always include additional information regarding a portion of the original, non-synthetic song. This was implemented by including two spectrograms for generation in our Continuer Pipeline, namely the first and most recent spectrograms of a song, and the output will be a new spectrogram that corresponds to what the model thinks the next five seconds of the song will be.

This was implemented as a class that extends the HuggingFace *DiffusionPipeline* interface. This was done so that once the code is published online, users can extend music with ease. All they need to do is configure their HuggingFace credentials, and then the process for downloading and using the pipeline will be trivial.

Chapter 7

FUTURE WORK

As mentioned previously, there is much room for improvement in this work; we proposed the basis for a novel technique for extending music, but we are confident that the quality of the music generated by the Continuer Pipeline can increase with future development. We describe some of these avenues for improvement below.

7.1 Changing the Influence of the Root Music

One of the core concepts behind the Continuer Pipeline is that the extension of music should have influence from the most recent portion of the song *and* a portion of the song that was not synthetically generated. To implement this, we handle music generation by taking the spectrogram of the last five seconds of the song, which is denoted as x_t, and the first five seconds of the song, which is denoted as x_0. Our input is then the stack of these two spectrograms, $[x_0, x_t]$, and the output is the model attempting to predict the spectrogram of the next five seconds, x_{t+1}. This is done so that the generation has an influence of the most recent portion of the song, x_t, and it contains information from another, non-synthetic portion of the song, x_0.

We believe it is very important to include influence in the generation process from non-synthetic data, and this is why we chose x_0; the first spectrogram is always going to be part of the original music, so including it as an input is a static approach to including original music. Although we believe it is vital to the generation process to include information that is not synthetic, we believe that this initial approach is naive and can be vastly improved.

A potential means to improve generation is to include additional spectrograms. Currently, only spectrograms x_0 and x_t are used for generation, resulting in an input dimension of 256x256x2. If we were to increase the depth by adding more spectrograms, say x_0, $x_{t/2}$, and x_t, our resultant input dimension would be 256x256x3, and the Continuer Pipeline would have access to more information when performing inference. Extending this, one could even implement a sliding window to include a range of spectrograms before x_t while including x_0. These are some of the many ways that the input dimensions could be tweaked to change the influence of the root music, and these are viable options that could be explored further to see if they result in an improvement in the quality of the extended music.

7.2 Implementing Time Embeddings

In the current implementation, there is no information passed to the model that details the distance between the root image and where we are in the generation process. This is something that can be improved, and it may be vital information for the generation process.

To analyze this more in-depth, we will look at two examples. For the first example, let us assume that we have a 20-second audio clip that we want to extend. Thus, our generation process would entail taking two spectrograms, namely x_0 and x_t, where spectrogram x_0 pertains to seconds 0 through 5 and spectrogram x_t would be the last five seconds, or seconds 15-20. Our generated spectrogram, x_{t+1}, would be what the model would predict seconds 20-25 of the song would be.

For the second example, we will do the same generation process on a 2-minute song. In this case, our inputs to the diffusion model are the same two spectrograms, x_0 and

x_t. Similar to before, x_0 is the spectrogram corresponding to the first five seconds of the song, but now the x_t spectrogram corresponds to seconds 115-120.

The difference we wanted to illustrate is that the gap between the two input spectrograms can drastically change; in the second example, the distance between x_0 and x_t is 110 seconds, which differs from the 10-second gap in the first example. This difference is not accounted for in the input to the model, so we hypothesize that this will negatively impact the diffusion model's performance; to counter this, we believe that including information about the gap between the two spectrograms will improve the results of the generation process.

We believe that the best way to do this is transformer sinusoidal time embeddings; this is a technique that is already apparent in the diffusion process, so extending it to the input of the model should be trivial. Low-level details of these time embeddings are described in the Background chapter, but we believe that including these as explicit inputs to the model will give the generator more information about the distance between the x_0 and x_t spectrograms, and this will hopefully improve the results of the generated music.

7.3 Diffusion in the Image Space

Another potential method to explore is performing diffusion in the image space rather than the latent space. The main benefit of performing latent diffusion rather than pure diffusion is that the complexity of performing diffusion on the latent vectors is considerably less than diffusion on the images themselves, as the latent vectors are condensed versions of the images. The drawback of doing this is that information is lost during the encoding-decoding processes of the VAE; thus, latent diffusion is ultimately sacrificing quality for lessened computational complexity.

If one were to train the Continuer Pipeline using an architecture that performed the diffusion process in the image space, one would need considerably more computing resources, but it is almost certain that this would yield improved results. The *Audio Diffusion* team found that diffusion done in the latent space yielded worse results than diffusion done in the image space, so it is likely that this same idea will hold for the Continuer Pipeline [16].

7.4 Converting Problem to Stable Diffusion

Another way to improve the results of this research is to convert this work to stable diffusion, which would allow users to pass in a text description that will influence the generation of the Continuer Pipeline.

We do note that this is a very broad statement, and converting the approach to stable diffusion is not a trivial task. Fortunately, one could look at the Riffusion team's implementation for guidance when retraining the Continuer Pipeline with a stable diffusion architecture, as the Riffusion Team's model architecture allows text embeddings to influence the model's inference process [3].

It should be noted that this will not necessarily increase the quality of the audio generated, but rather it will allow the user to influence the music generation process with textual data. Simply converting the architecture to take text embeddings will not necessarily *improve* the quality of music generated with all other variables held constant; it will only expand the capabilities of the model, as a Continuer Pipeline trained with stable diffusion will be able to extend music in the context of different genres specified by textual information from the user.

7.5 Changing the Generative Approach

The core idea that inspired the work behind the Continuer Pipeline was the fact that we can treat the music generation problem as an image generation problem; the ability to dynamically convert musical waveforms back and forth to spectrograms allows us to essentially use all of the research and tools in the image synthesis domain, and we can extend them to the domain of musical generation. The choice for explicitly using diffusion models was due to the fact that the diffusion techniques have shown a lot of promise in generating high-quality images, but the choice of this specific generative architecture was not grounded in more logic beyond that.

An area to explore that might improve the results of the generation process is analyzing how different generative architectures are able to complete this task. As mentioned previously in this work, there are numerous deep learning generative techniques that one could use; for example, Generative Adversarial Networks (GANs) have been shown to yield excellent results in high-quality image synthesis, so it is very feasible that they would yield excellent results in this domain. It should be noted that GANs do occasionally have issues with reaching convergence, as the training process is a min-max approach with two combatant Neural Networks training simultaneously.

Also, the challenge of incorporating textual embeddings has been done and researched thoroughly with Stable Diffusion, but GANs are generally used for unconditional generation; this would mean that future researchers would have to find a novel way to incorporate textual information in the music synthesis process if they were to change the generative model in this work to a GAN.

In the context of specifically the Continuer Pipeline, GANs may yield excellent results; it is an unconditional generation problem, as the only inputs to the generative algorithm are the x_0 and x_t spectrograms. Thus, the common use case for a generative adversarial network would align with the scope of what the Continuer Pipeline is trying to achieve, but it should be noted that the task of adding textual information to influence the generation process will be more challenging than if a diffusion probabilistic model approach was taken.

7.6 Increasing the Scale

As per the context of this research, we did not have ample access to the best computing resources that exist in the industry; because of this, we took a few approaches to sacrifice quality of the generated audio in exchange for less required computational complexity.

One example of this is that we only employed the use of one GPU while training. We believe that parallel computing should be done, as this will allow future researchers to perform more exhaustive training and improve the results of the Continuer Pipeline. This is not a unique approach to this project, as many approaches in the industry are improved by simply increasing the scale at which they are tackled.

Beyond the training scale, future researchers can expand the model architecture to potentially yield better results. We settled on a model architecture that would allow us to train on a single GPU, so we are assuming that we sacrificed a bit of quality by this constraint architecture choice. As described earlier, the denoising U-Net is a typical U-Net structure, so adding more layers and blocks will likely allow the model to learn more abstract relationships regarding the data; we hypothesize that this will ultimately improve the model's performance.

When running training on 512x512 images, we were running out of RAM as PyTorch was unable to put all of the operations on the GPU. Because of this, we resorted to using images with a resolution of 256x256. This sacrifice meant that the input to our pipeline now had only one-quarter of the information as originally intended; thus, we believe that performing training on 512x512 images will yield better results as the model will be able to see four times more information on a snippet of the song than what it would see in a 256x256 image. It should be noted that this will increase the computational demand by much more than four times, but modern systems should be able to do this with ease.

BIBLIOGRAPHY

[1] C. Delangue, J. Chaumond, and T. Wolf. Hugging face hub. https://huggingface.co. Accessed: 2023-03-18.

[2] O. Elharrouss, N. Almaadeed, S. Al-Maadeed, and Y. Akbari. Image inpainting: A review. *Neural Processing Letters*, 51:2007–2028, 2020.

[3] S. Forsgren and H. Martiros. Riffusion-stable diffusion for real-time music generation. https://riffusion.com/about, 2022.

[4] M. French and R. Handy. Spectrograms: Turning signals into pictures. *Journal of Engineering Technology*, 24(1):32, 2007.

[5] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[6] D. Griffin and J. Lim. Signal estimation from modified short-time fourier transform. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 32(2):236–243, 1984.

[7] J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020.

[8] J. Krause, M. Stark, J. Deng, and L. Fei-Fei. 3d object representations for fine-grained categorization. In *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*, pages 55–60, June 2013.

[9] B. McFee, C. Raffel, D. Liang, D. P. Ellis, M. McVicar, E. Battenberg, and O. Nieto. librosa: Audio and music signal analysis in python. In *Proceedings of the 14th Python in Science Conference*, volume 8, pages 18–25, 2015.

[10] G. Mittal, J. Engel, C. Hawthorne, and I. Simon. Symbolic music generation with diffusion models. *arXiv preprint arXiv:2103.16091*, 2021.

[11] A. Oussidi and A. Elhassouny. Deep generative models: Survey. In *2018 International Conference on Intelligent Systems and Computer Vision (ISCV)*, pages 1–8. IEEE, 2018.

[12] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[13] K. Prahallad. Spectrogram, cepstrum and mel-frequency analysis. https://www.cs.brandeis.edu/~cs136a/CS136a_docs/KishorePrahallad_CMU_mfcc.pdf.

[14] A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, and I. Sutskever. Zero-shot text-to-image generation. In *International Conference on Machine Learning*, pages 8821–8831. PMLR, 2021.

[15] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10684–10695, 2022.

[16] R. D. Smith. Audio diffusion. https://github.com/teticio/audio-diffusion, 2022.

[17] R. D. Smith. Audio diffusion 256 dataset. https://huggingface.co/datasets/teticio/audio-diffusion-256, 2022.

[18] A. Vahdat and J. Kautz. Nvae: A deep hierarchical variational autoencoder. In *Advances in Neural Information Processing Systems*, volume 33, pages 19667–19679, 2020.

[19] A. Van den Oord, N. Kalchbrenner, L. Espeholt, O. Vinyals, A. Graves, et al. Conditional image generation with pixelcnn decoders. *Advances in neural information processing systems*, 29, 2016.

[20] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 30, 2017.

[21] Y. Yang, M. Hira, Z. Ni, A. Chourdia, A. Astafurov, C. Chen, C.-F. Yeh, C. Puhrsch, D. Pollack, D. Genzel, D. Greenberg, E. Z. Yang, J. Lian, J. Mahadeokar, J. Hwang, J. Chen, P. Goldsborough, P. Roy, S. Narenthiran, S. Watanabe, S. Chintala, V. Quenneville-Bélair, and Y. Shi. Torchaudio: Building blocks for audio and speech processing. *arXiv preprint arXiv:2110.15018*, 2021.

[22] C. Zhang, Y. Ren, K. Zhang, and S. Yan. Sdmuse: Stochastic differential music editing and generation via hybrid representation. *arXiv preprint arXiv:2211.00222*, 2022.

[23] P. Zhu, C. Pang, S. Wang, Y. Chai, Y. Sun, H. Tian, and H. Wu. Ernie-music: Text-to-waveform music generation with diffusion models. *arXiv preprint arXiv:2302.04456*, 2023.