Music Generation Using ML

Yashayah Tirupati Boram, Shital Mantayya Durgam, Sumit Nandanwar, Naresh Jadhav, Prof. Minakshi Getkar
Department of Computer Science and Engineering, Rajiv Gandhi College of Engineering Research and Technology, Chandrapur, India
Guide, Department of Computer Science and Engineering, Rajiv Gandhi College of Engineering Research and Technology, Chandrapur, India

Abstract-The intersection of music and artificial intelligence (AI) has sparked innovative approaches to music generation, leveraging the power of machine learning (ML) algorithms. This abstract delves into the burgeoning field of music generation using ML techniques, presenting an overview of methodologies, challenges, and future directions.

Music generation with ML involves the utilization of algorithms to analyze patterns, structures, and styles inherent in musical compositions. From classical to contemporary genres, ML models can learn from existing musical datasets to generate novel compositions that mimic the characteristics of the training data. Techniques such as recurrent neural networks (RNNs), generative adversarial networks (GANs), and transformer models have been prominent in this domain.

INTRODUCTION

Recent developments in integrating Deep Neural Networks (DNNs) with audio generators have renewed interest in using the unaltered audio of a musical instrument as a control source for a synthesizer. One such example is the DDSP architecture and its derivatives, that allows for real-time control of a synthesizer using a set of features extracted from an input audio signal. It has been used to develop various creative timbre transformation applications, which we collectively refer to as Tone Transfer applications.

We situate the scope of our work on audio-based synthesis control for real-time performances, looking at sonic diversity and synthesizer phrasing and articulation. These essential components of musical expression have been thoroughly studied for composition with MIDI for decades but we argue that they open new challenges and possibilities when considering an audio-based control approach. Transients at the beginnings of notes and the transitions between notes play a vital role in defining the continuity and flow of musical phrasing. We argue that a continuous control approach such as Tone Transfer could potentially learn mappings that capture beginnings, endings, and the links between notes during performance, generating musically articulated synthetic sounds.

REQUIREMENTS

Project focused on music generation using machine learning (ML) techniques, there are several requirements that are instrumental in ensuring the success and effectiveness of the project. Here's an explanation of some essential requirements:

1. Programming Languages:

- Python: Python is widely used in the machine learning community due to its rich ecosystem of libraries and tools for data analysis, machine learning, and signal processing.
- JavaScript: If the project involves web-based applications or interactive user interfaces, knowledge of JavaScript would be beneficial.
- HTML: HTML forms the structure of web applications, providing the framework for delivering content and interacting with users.

2. Machine Learning Libraries:

- TensorFlow or PyTorch: These are two of the most popular deep learning frameworks used for building and training neural networks, including those for music generation.
- Keras: Keras is a high-level neural networks API, which can run on top of TensorFlow or other frameworks. It simplifies the process of building and training neural networks.
- Scikit-learn: Scikit-learn provides simple and efficient tools for data mining and data analysis, including various machine learning algorithms for

classification, regression, clustering, and dimensionality reduction.

3. Signal Processing Libraries:

- Librosa: Librosa is a Python library for music and audio analysis. It provides functions for extracting features from audio signals, such as mel-frequency cepstral coefficients (MFCCs), spectrograms, and chroma features.
- PyDub: PyDub is a simple and easy-to-use Python library for audio manipulation. It can be used for tasks such as reading and writing audio files, converting between different audio formats, and applying various effects to audio.

4. Data Collection and Preprocessing:

- Dataset: Access to a high-quality dataset of musical compositions across different genres and styles is essential for training machine learning models for music generation.
- Data Preprocessing: Preprocessing steps may include audio file conversion, feature extraction, normalization, and augmentation to prepare the data for training.

5. Model Development:

- Neural Network Architectures: Designing and implementing neural network architectures suitable for music generation tasks, such as recurrent neural networks (RNNs), convolutional neural networks (CNNs), or transformer models.
- Hyperparameter Tuning: Tuning the hyperparameters of the models to optimize performance and generalization.

6. Evaluation Metrics:

- Audio Quality Metrics: Metrics such as signal-to-noise ratio (SNR), perceptual evaluation of audio quality (PEAQ), or melodic similarity can be used to evaluate the quality and fidelity of generated music compared to the original compositions.
- Human Evaluation: Conducting subjective evaluations with human listeners to assess the musicality and creativity of the generated music.

7. Deployment and Integration:

- Web Development: If the project involves deploying the model in a web-based application, knowledge of web development frameworks like Flask or Django for backend development, and HTML/CSS/JavaScript for frontend development would be useful.
- API Development: Creating RESTful APIs or web services to expose the model's functionality for integration with other applications or platforms.

ALGORITHM DESCRIPTION

DDSP

Differential Digital Signal Processing builds on the field of Digital Signal Processing by making its components differentiable. For example, additive filters (which add the two input sources) and reverb filters (which add reverb to the input source) can then be part of our signal processing pipeline, and we can use backpropagation to train these components. Moreover, the fact that the DSP components are differentiable allows us to backpropagate the gradient through these components, meaning we can train differentiable components that come earlier in the pipeline (e.g., neural nets).This means it's easier to use neural networks to adaptively learn signal processing techniques. In what follows, we design a pipeline which adaptively learns latent representations of the "style" of input audio, in such a way that this style can be transferred to a new sequence of tones.

Pipeline Design

The pipeline we use, originally developed by the creators of DDSP, is depicted below. The yellow components are DDSP components which add inductive bias, helping our model better represent the encoded audio. f0f_0f0, ZZZ, and LLL are latent space variables:f0f_0f0 is the fundamental frequency of the input, inversely related to the length of time we wait for the signal to repeat itself. This captures the pitch of the input.LLL is the loudness of the input, which captures the volume of the input.ZZZ is a latent space variable whose representation is learned by end-to-end training. Hypothesis: since the autoencoder reconstructs the original audio given pitch, volume, and ZZZ, we hypothesize that the learned representation of ZZZ will contain information relevant to the style of the original audio

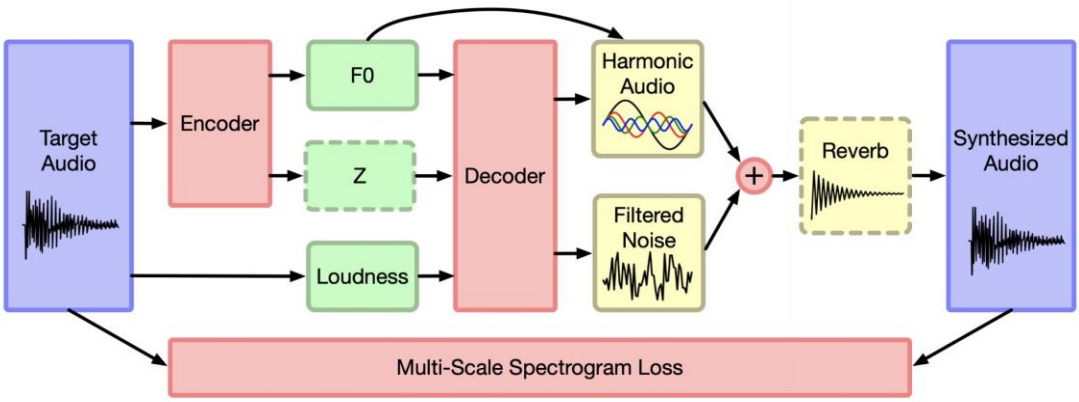

Fig 3: DDSP Pipeline Design

Now let's take a look at the universal function approximators that make everything tick. We need to design architectures for the encoder and decoder. Luckily, the creators of DDSP have already devised working encoders and decoders, so we can simply use their architecture. Both the encoder and decoder rely primarily on the use of a Gated Recurrent Unit (GRU), which is a type of Recurrent Neural Network frequently used in sequence processing tasks. If you want to learn more about how GRUs work, this blog post does an excellent job explaining and visualizing them.

Decoder Design

The creators of DDSP advocate for a decoder design which individually processes the input latent space variables with an MLP (whose design we will see shortly), then processes the output of the ZZZ MLP with a GRU. Next, we concatenate the separate ZZZ, LLL, and f0f_0f0 channels before passing the result through one last MLP (identical deisgn) to assist the interpolation of information from each channel. Finally, two dense layers are tasked with mapping the MLP outputs to the inputs of the harmonic audio and filtered noise components. The creators of DDSP provide the image below as an illustration of the decoder design. Note that this image is misleading; lines from the LLL MLP and f0f_0f0 MLP should be drawn to the second "concatenate" block, after the GRU.

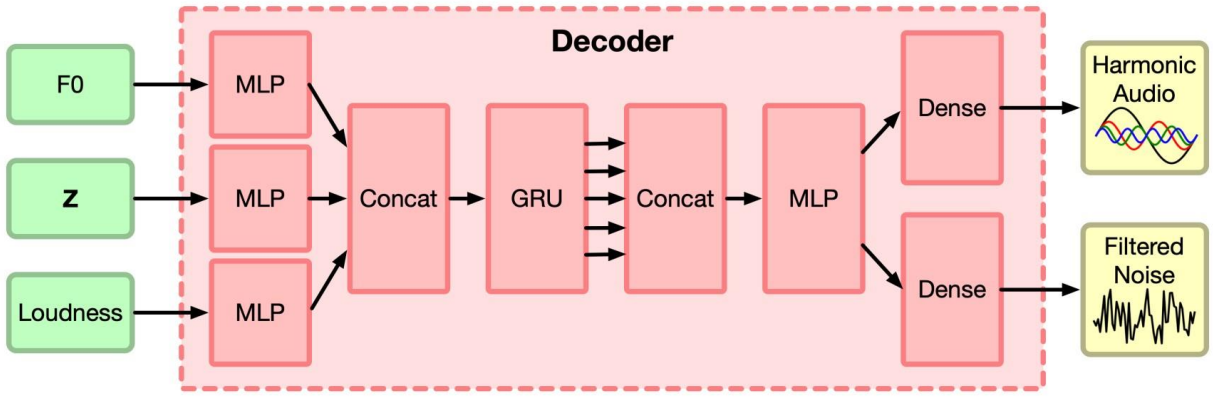

Fig 4: DDSP Decoder Design

Notice the pipeline relies several Multi-Layer Perceptron (MLP) layers, which are essentially just three 512-unit densely connected neural network layers. Here's a visualization of the MLP pipeline used repeatedly in the above decoder design:

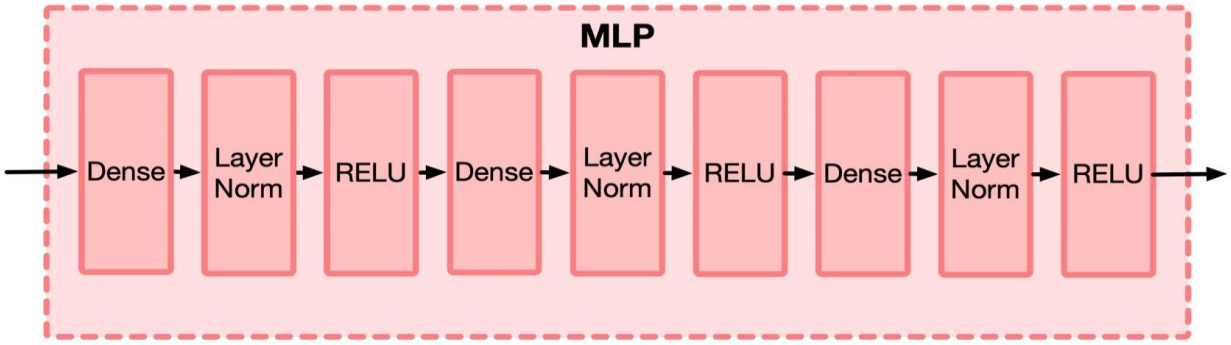

Fig 5: MLP

This decoder, which has over a million parameters, should contain enough flexibility to allow efficient style representation. Now let's take a look at the encoder we'll use.

Encoder Design

The encoder designed by the DDSP authors is relatively simple in comparison to the decoder. To encode the f0f_0f0 latent space variables, we use a pretrained fundamental frequency extraction model; we used the CREPE model. To encode the ZZZ latent space variables, we design our own pipeline as described by the image below. We first calculate the Mel Frequency Cepstral Coeffcients (MFCCs) since these featurize audio data in a way that better represents the periodic nature of audio data. Then we normalize the MFCCs and use a GRU to process them, followed by a final Dense layer to map the GRU outputs to the correct output size.

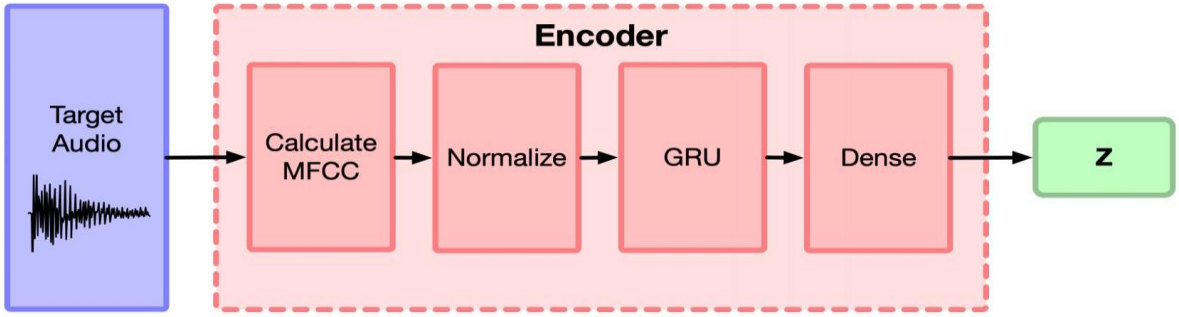

Fig 6: DDSP Encoder Design

With the help of the CREPE model, we have created an encoder capable of capturing as much information about the style of audio data necessary to regenerate the original audio data. Now let's see our design in action.

CONCLUSION

Transforming the audio of an instrument to a synthetic sound is a challenging task, as it involves a one-to-many relationship. Each instrument has its unique timbral palette, dynamic contour, and articulation possibilities, which can vary significantly even among instruments of the same type. On the other hand, the sound produced by a synthesizer can be highly versatile; and only a subset of the source instrument's characteristics may be desired in the output.

We can argue that there is no definitive "gold standard" that can provide a baseline mapping between an instrument's audio and a synthetic sound: tradeoffs are necessary to find viable solutions. In this work, we first analyzed current Tone Transfer architectures and identified a tradeoff in their rendering capabilities: these models learn new timbres from audio corpora and can project the input loudness to the output, at the expense of a good resolution of note beginnings and endings which are essential for musical articulation and phrasing