Anime-Style Music Generation
Project Report

Xiong Maihe 1004537
Jin Ziqi 1004531
Dong Jiajie 1004512
Wu Xinyun 1004536
He Haolan 1004530

Introduction

Previously music generation models usually focused on simple classic music, we want to build a model that could learn the variant styles of anime music. We followed the idea of this project as reference and made adjustments upon the model, gathered a set of training data and trained the model. Some result samples are acquired, though some part of the output music sounds unnatural, our generated music does indicate that our model displayed some ability to learn the pattern of anime music.

Input

We've selected 102 pieces of soft and smooth anime music with lengths ranging from 2 minutes to 5 minutes. Then a pretrained model and a python music transcription interface library are used to convert these music files from MP3 to MIDI, generating our training samples in the compatible format. The format of MIDI, which stands for music instrument digital interface that works as a digital signal, enables us to have the quantifiable input as a series of binary digits (0s and 1s).
Note: The music used in our dataset is copyrighted, only valid for non-commercial use.

Setup Process

Package Dependencies

python library (install via pip):

- music21: A toolkit for computer-aided musical analysis
- piano-transcription-inference: Provide piano transcription inference, transcribe piano recordings to MIDI file

system package (install via package manager apt):

- lilypond: for displaying music notations
- fluidSynth: interpret from MIDI signals and synthesis audio
- ffmpeg: convert file format from MIDI to WAV and MP3 to MIDI

Note:

CUDA is not required during training though it is capable of speeding up the training. During our training, CUDA is enabled. The installation and setup process of CUDA is not included in this report.

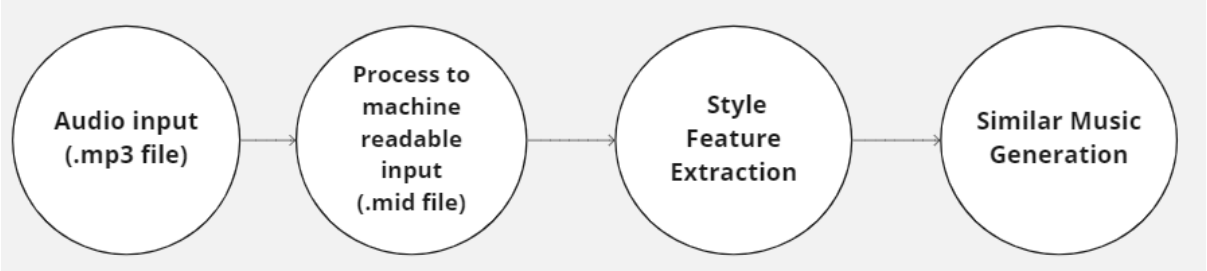
Generating Music

- Clone this repository
- Open music-generation-lstm-comic.ipynb
- Run the last four blocks (loading the model) to generate new music.

Training LSTM Model

- Use piano_transcription_inference.ipynb to convert the audio files from .wav to .mid
- comic.zip was used as our training set
- unzip the music to the directory ./input/comic
- Run music-generation-lstm-comic.ipynb

System flow



Here is how our system works: we first convert the mp3 file to midi format so that the model can take in the files as input. Then the input is passed to the model to learn the pattern of the music, and generate new pieces of music that are of similar style to the input.

Model architecture

Model: "sequential_14"		
Layer (type)	Output Shape	Param #
lstm_33 (LSTM)	(None, 40, 512)	1052672
activation_7 (Activation)	(None, 40, 512)	0
lstm_34 (LSTM)	(None, 512)	2099200
dropout_17 (Dropout)	(None, 512)	0
dense_18 (Dense)	(None, 303)	155439
Total params: 3,307,311		

Trainable params: 3,307,311
Non-trainable params: 0

Areas of improvement

A few improvements over the reference model have been made to better suit our training data.

1. The reference model uses Chopin's music for training, which does not take into account different tunes and styles of the training samples. For our model, slow and smooth music acquired from Anime episodes was used as training samples instead, which objectively improved the consistency of the training set, allowing the trained model to more accurately generate a class of music.
2. In our model, larger LSTM layers were used in the hope that more features can be captured by the model since we are using a significantly larger dataset. Additionally, it enables the model to converge easier.
3. An activation layer was added in between the 2 LSTM layers, replacing the original dropout layer. By doing so we also wanted for feature preservation, out of similar reasons as mentioned.

Result and comparison with the reference framework

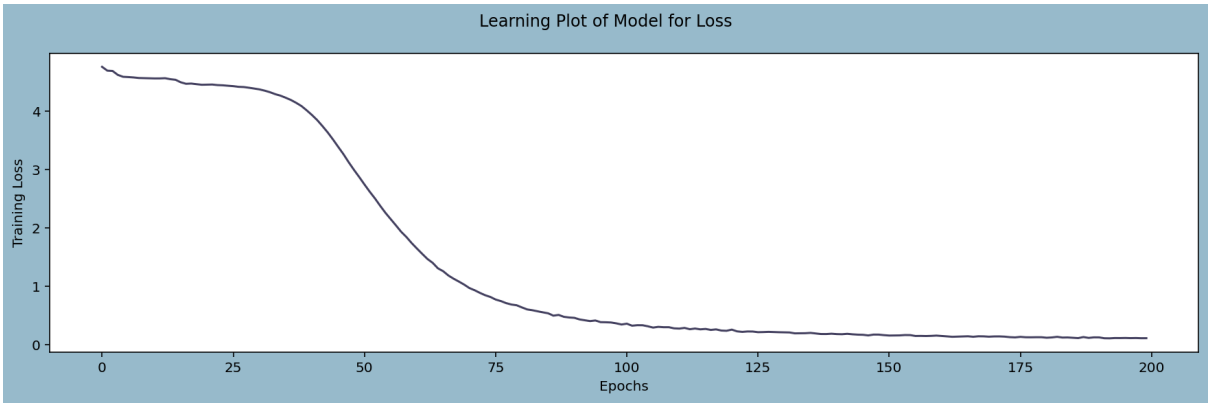


Figure 1: Performance of reference framework (loss curve)

Figure 2: Performance of our framework (loss curve)

The loss function of categorical crossentropy was used, which is often used in multi-class classification tasks. This well suits our case, as each unit of output is a single note out of 8 possible notes.
In comparison with the reference model, our model shows a lower convergence rate due to training on a larger dataset, though in the end the losses of the two models are similar. The reason why the loss curve of our model is not as smooth as the reference model is we divided the whole dataset into several epochs for training due to the large dataset, which means for each training epoch, parts of the dataset are different with the last epoch, causing the loss displayed to fluctuate.

Music generation

soundfont for translating MIDI file to wav
https://musescore.org/en/handbook/3/soundfonts-and-sfz-files#list

Evaluation of output

No validation set was created and used, as there are no distinct ways to directly evaluate how good the output is. Instead, you are encouraged to listen to the generated music, which could be a direction for evaluating our model.

Limitations

1. Every note generated is limited to a quarter-note long in length, as we didn't plan for building a model that generates music of more complex rhythm.
2. Though all of the training samples sound smooth and soft to the human ear and are assumed to be within the same category, from an acoustic standpoint the features within each piece of music might be very different and convoluted across. Our model might not be able to learn well from the dataset for this reason.

Appendix

Reference project: https://www.kaggle.com/code/karnikakapoor/music-generation-lstm
The pretrained model: https://zenodo.org/record/4034264
The music transcription interface library used:
https://github.com/bytedance/piano_transcription