Music Generator

Abstract

The MAESTRO dataset, a vast collection of classical piano performances in audio and MIDI formats, presents a promising opportunity for advancing artificial intelligence and machine learning in music generation and transcription. This dataset encompasses various compositions by diverse classical composers, including detailed metadata and symbolic annotations. Leveraging the MAESTRO dataset, this study aims to develop a music generator model capable of generating classical piano music that faithfully captures the nuances of the original performances. Additionally, the model aims to explore the synchronisation of audio and symbolic data, contributing to a deeper understanding of the intricacies of classical piano music.

Problem Statement

The MAESTRO dataset provides a unique opportunity to delve into the development of a music generation model targeting classical piano compositions. However, several challenges need to be addressed. First, generating expressive and faithful renditions of classical piano music requires converting symbolic data (MIDI) into audio and the reverse process of transcribing audio into symbolic notation. This bidirectional conversion demands the model's ability to capture the intricate details of timing, dynamics, and phrasing inherent in classical piano performances.

MIDI Files

MIDI, Musical Instrument Digital Interface, is a widely used protocol and file format for representing and communicating musical information between computers, synthesisers, and other electronic musical instruments. MIDI files are a specific type of computer file that contains musical data, and they are used for various purposes in the world of music and audio production.

Here are some critical aspects of MIDI files:

1. Data Format: MIDI files are not audio recordings. Instead, they contain digital instructions or messages describing musical events in the form of sequences. These instructions include information about notes, pitch, duration, velocity, tempo, and more.
2. Polyphonic: MIDI files can represent multiple musical voices or parts simultaneously. This means they can describe complex compositions with various instruments and musical elements.
3. Compact: MIDI files are relatively small compared to audio files (e.g., MP3 or WAV files) because they do not store sound waveforms. Instead, they store instructions on generating the sound, making them efficient for storage and transmission.
4. Editable: MIDI files are highly editable. one can easily change the notes, adjust timing, and modify various musical aspects without losing quality, making them a valuable tool for music production and editing.

5. Platform-Agnostic: MIDI is a standard protocol, so MIDI files are generally platform-agnostic and can be used on various software and hardware instruments.
6. Notation: MIDI files can be converted into sheet music or musical notation using appropriate software, making them helpful in creating and sharing sheet music.

MIDI files can be used for a variety of purposes, such as:

1. Playing music on MIDI-compatible instruments and synthesisers.
2. Sequencing and recording music in digital audio workstations (DAWs).
3. Creating and editing music compositions.
4. Generating background tracks for live performances.
5. Controlling lighting and visual effects in live shows.

fluidsynth
FluidSynth is a software synthesiser that generates and manipulates digital audio, often used for playing MIDI files and generating sound from MIDI input. In Python, the "fluidsynth" library is a Python wrapper for the FluidSynth library, allowing one to interact with FluidSynth's functionality from within a Python program.

With the FluidSynth library in Python, one can perform tasks like:

1. Playing MIDI files: One can use FluidSynth to load MIDI files and play them, which can be helpful in music playback or software that involves MIDI music.
2. Generating sound from MIDI input: One can use FluidSynth to generate sound in real-time from MIDI input, which can be beneficial for creating music applications, synthesisers, and more.
3. Soundfont management: FluidSynth typically uses soundfonts to determine the timbre and quality of the generated sound. The Python wrapper allows one to manage soundfonts, load them, and apply them to the MIDI playback.

glob
The glob library in Python is a module that provides a convenient way to search for files and directories that match a specified pattern using unknown characters, similar to how file searching works in many command-line shells. The primary function of the glob library is to return a list of files and directory pathnames that match specific patterns.

pretty_midi
The pretty_midi library in Python is a powerful tool for working with MIDI (Musical Instrument Digital Interface) data. It allows one to create, manipulate, and analyse MIDI files, making it valuable for various musical and audio applications. Some of the primary features and functionalities of the pretty_midi library include:

1. Parsing MIDI Files: pretty_midi can parse existing MIDI files, extracting information about the notes, instruments, tempo, time signatures, key signatures, and various MIDI events within the file. This makes it easy to inspect and analyse existing MIDI compositions.
2. Creating MIDI Files: One can use pretty_midi to create new MIDI files from scratch, making it suitable for generating music programmatically. One can specify the compositions' notes, instruments, and musical elements.

3. Manipulating MIDI Data: pretty_midi provides functions to modify existing MIDI data, such as changing the pitch, duration, or timing of notes. This is useful for making alterations to existing MIDI compositions.
4. Visualisation: It allows one to visualise MIDI data, including the note positions and durations, using various plotting and visualisation tools.
5. Playback: pretty_midi supports MIDI playback, allowing one to directly listen to the MIDI compositions one creates or modifies within the Python environment.
6. Integration with NumPy: The library integrates well with NumPy, a fundamental library for numerical computing in Python. This makes it easy to work with MIDI data in a structured and efficient manner.

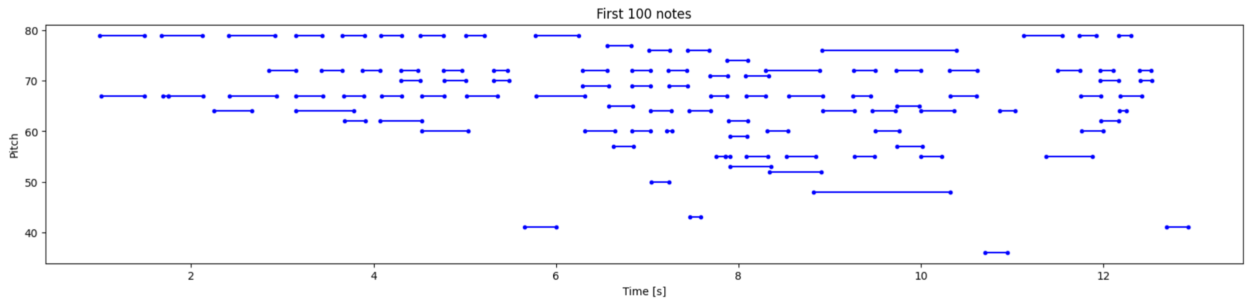
Maestro Dataset
The "MAESTRO" dataset is the "Music Audio and Symbolic Data for Synchronization and Retrieval" dataset. It is a comprehensive collection of classical piano performances in audio and symbolic (MIDI) formats. It is widely used in music information retrieval, machine learning, and artificial intelligence research, particularly for tasks like music transcription, generation, and synchronisation of audio and symbolic representations.

Key characteristics and components of the MAESTRO dataset include:

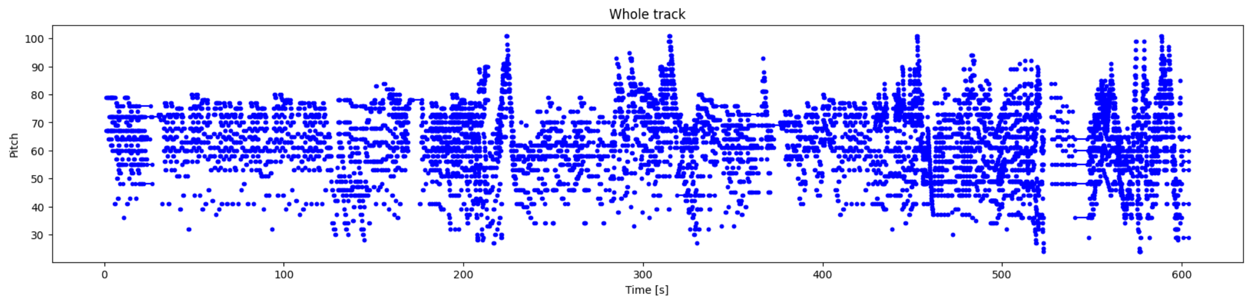
1. Audio Recordings: It contains high-quality audio recordings of classical piano performances. The audio is typically provided in WAV format.
2. MIDI Files: Alongside the audio, the dataset includes corresponding MIDI files. MIDI files represent the symbolic notation of the music, including information about notes, timing, pitch, and dynamics.
3. Metadata: The dataset is enriched with metadata, providing information about the musical pieces, performers, and other relevant details.
4. Annotations: Some dataset versions include beat and bar annotations, facilitating rhythm and structure analysis research.
5. Size: The MAESTRO dataset contains thousands of hours of piano performances and several MIDI files.
6. Variety: The dataset covers a wide range of classical music compositions, offering a diverse set of pieces by various composers, eras, and styles.

Generator Stats

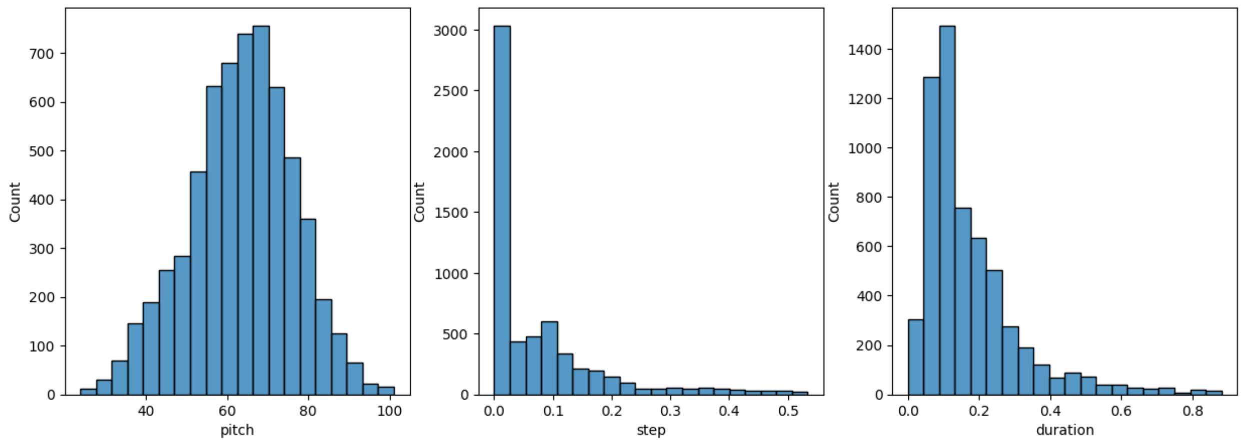
Number of MIDI files in MAESTRO Dataset: *1282*
Sample file used for the generator:
data/maestro-v2.0.0/2006/MIDI-Unprocessed_04_R1_2006_01-04_ORIG_MID--AUDIO_04_R1_2006_01_Track01_wav.midi
Number of instruments: *1*
Instrument name: *Acoustic Grand Piano*



Piano Roll for First 100 Notes



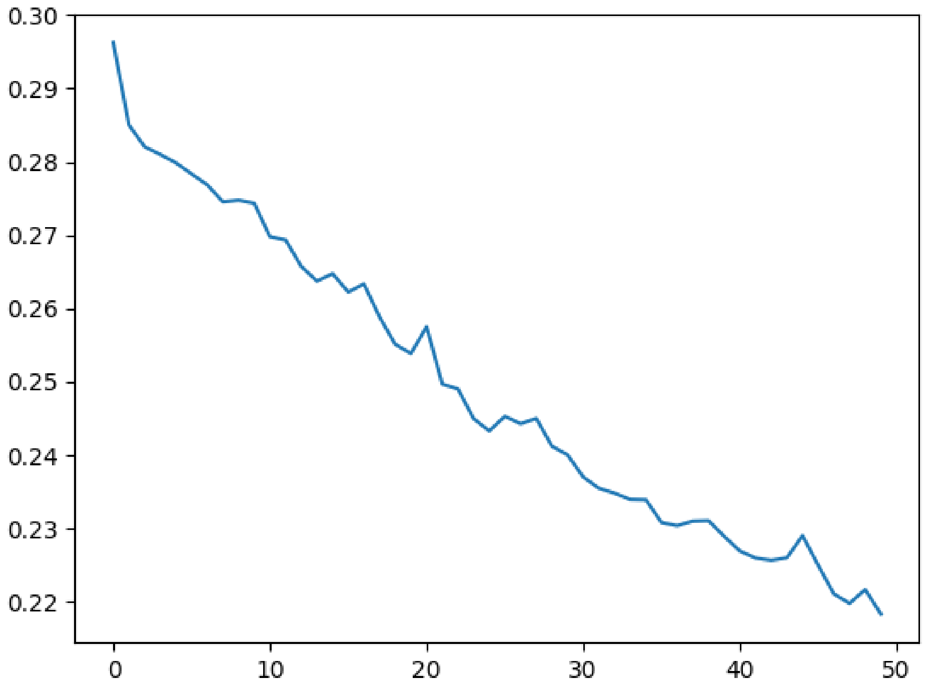
Piano Roll for Whole Track



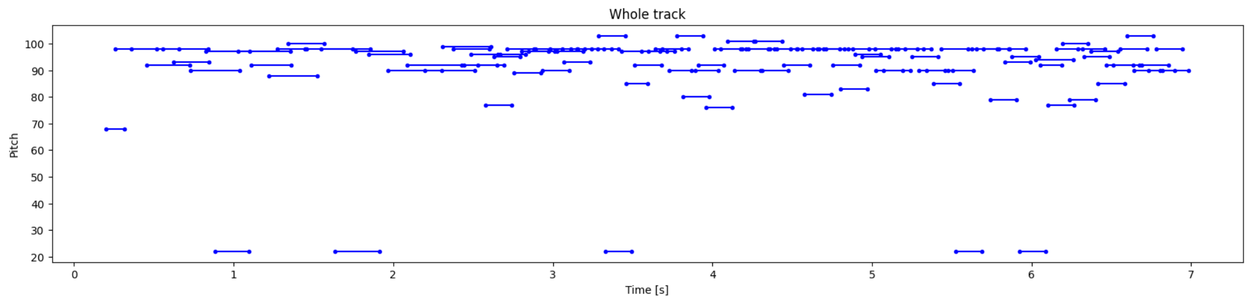
Distribution of Pitch, Step and Duration for Raw Notes

Training Dataset

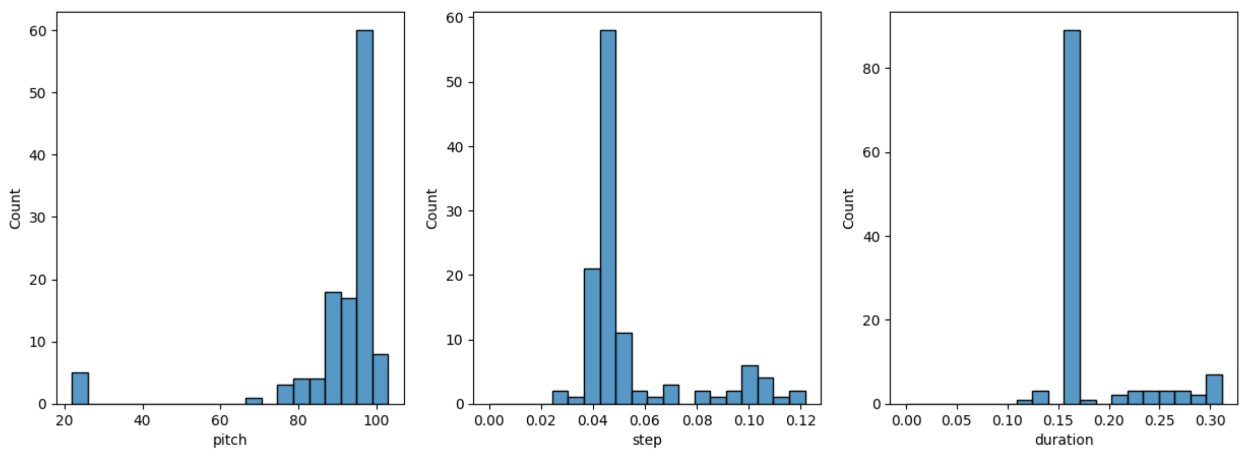
Number of notes parsed: *58032*
Sequence Length *= 25 (Hyperparameter)*
Vocab Size *= 128 (The vocabulary size is set to 128, representing all the pitches supported by pretty_midi)*
batch_size *= 64*
Model Trained with *50* ***Epochs***



Loss vs Epoch Graph



Piano Roll for Generated Notes



Distribution of Pitch, Step, and Duration for Generated Notes