Machine Learning Project
Symbolic Music Generation

Cavasinni Matteo
296609

Febrary 8, 2025

1 Introduction

1.1 Project Objectives

The main objective of this project is to compare the various architectures and approaches for the generation of symbolic music.

1.2 Musical Theory

Music theory is the study of how music works. It explores the building blocks of music, such as pitch, rhythm, harmony, and form, and how these elements are combined to create a meaningful musical experience. **Fundamental elements of music theory are:**

- **Pitch:** The frequency of a sound, measured in Hertz (Hz). Pitch determines how high or low a sound is.
- **Rhythm:** The organization of sounds and silences in time. Rhythm includes elements such as tempo, note duration, and the subdivision of time.
- **Harmony:** The combination of multiple notes that are sounded simultaneously. Harmony creates a sense of depth and richness in music.
- **Melody:**A sequence of notes that create a coherent musical line. Melody is often the most memorable part of a song.
- **Tonality:** The system of relationships between the notes in a scale or piece of music. Tonality provides a tonal center and a sense of musical direction.
- **Form:** The overall structure of a piece of music. Form can include elements such as verses, choruses, bridges, and solos.

1.2.1 Intervalls

Musical intervals are the ”distance” between two notes, measured in semitones or based on the ratio between their frequencies. Each interval has a name (second, third, fourth, etc.) and a quality (major, minor, perfect, augmented, diminished) that depend on the number of semitones it contains. Intervals can be ascending (the second note is higher) or descending (the second note is lower). They are fundamental for building scales and chords, analyzing music, and composing. For example, a major third has 4 semitones, while a minor third has 3.

1.2.2 Conclusion

Music theory is a powerful tool that helps us understand, analyze, and create music. It's like a language that describes the inner workings of music, allowing us to delve deeper into its structure and meaning.

1.3 Tensorflow

TensorFlow is an open-source library developed by Google for machine learning. It is widely used to create and train deep learning models, such as neural networks, which can be employed in a wide range of applications, from speech recognition to image processing.

1.4 Pretty MIDI

: Pretty MIDI is a Python library that simplifies the manipulation of MIDI files. It allows you to read and write MIDI files, modify notes, instruments, and other musical events. It facilitates the extraction of useful musical information, such as notes, duration, pitch, and instruments. It can also convert MIDI files into audio formats. It is easy to use, versatile, and integrates with other Python libraries for audio processing

1.5 Long Short-Term Memory (LSTM)

LSTMs are a special kind of RNN designed to address the ”vanishing gradient” problem that plagues traditional RNNs. This problem makes it difficult for RNNs to learn long-term dependencies in data. LSTMs solve this problem by using a more complex internal structure consisting of:

- **Cells**: Each LSTM cell has a long-term memory that can store information for extended periods.
- **Gates**: Gates control the flow of information into and out of the cell, allowing the LSTM to decide which information is important to remember and which can be forgotten.

Thanks to this structure, LSTMs are able to learn long-term dependencies in data, making them more effective than traditional RNNs in many tasks.

1.6 MIDI

A MIDI (Musical Instrument Digital Interface) file is a file that contains digital instructions for playing music. Unlike audio files like MP3 or WAV, which contain the actual sound recording, a MIDI file contains information about which notes to play, with which instrument, at what volume, and for how long. Main features of a MIDI files are:

1. **Does not contain audio:**A MIDI file does not contain the actual sound of the music, but only the instructions to play it.
2. **Small file size:**MIDI files are much smaller than audio files because they only contain data and not the sound recording.

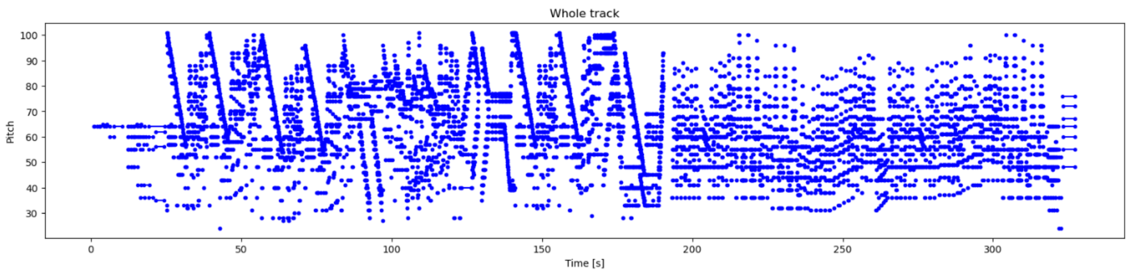


Figure 1: Classic song 1

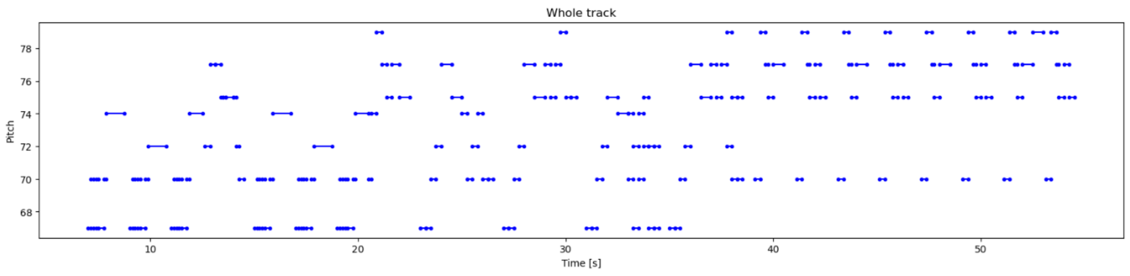


Figure 2: POP song 1

3. **Editable:**The information in a MIDI file can be easily modified, such as changing the instrument, tempo, or notes.
4. **Compatible with various instruments:**MIDI files can be used to control different electronic musical instruments, synthesizers, and music software.

1.7 Data

As a dataset, I used two famous MIDI datasets:

1. **MAESTRO (MIDI and Audio Edited for Synchronous TRacks and Organization):** is a dataset composed of about 200 hours of virtuosic piano performances captured with fine alignment (3 ms) between note labels and audio waveforms.Specifically, I used the smaller version, the 81 MB one.
2. **Pop music MIDI dataset:** 50 pop MIDI songs

2 Data Preprocessing

2.1 Explorative Data Analisis

In the data preprocessing phase, I tried to extract the musical structures of the songs contained within them. First, I extracted one song from each dataset at random and then listed all the notes played (regardless of the instruments they came from), representing them by their pitch, the note name, and their duration.Afterwards, I created piano-rolls of the songs using the plt library in Python. Here are three examples of songs for each genre:

As can be seen from the images, classical songs have a significantly more complex structure than those of the POP genre. It is important to be aware of this characteristic because if the structure is simpler, the model will converge sooner. For this reason, it is advisable to take less data from

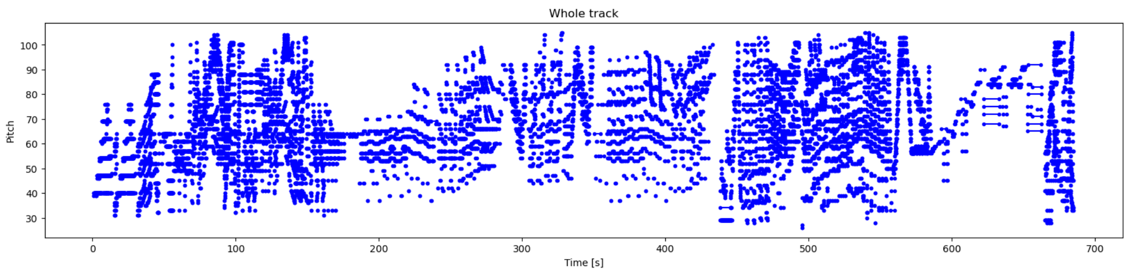


Figure 3: Classic song 2

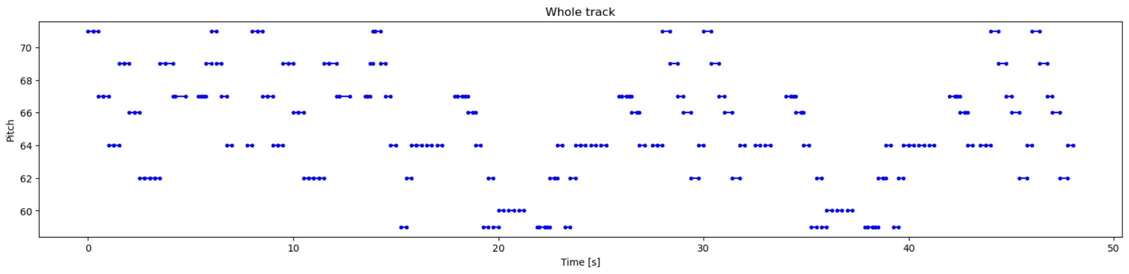


Figure 4: POP song 2

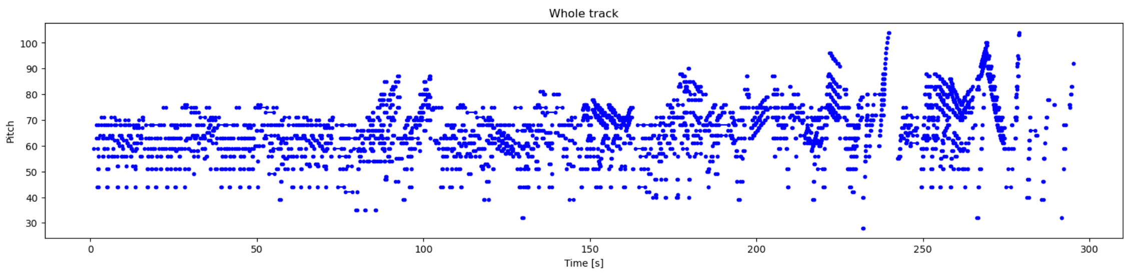


Figure 5: Classic song 3

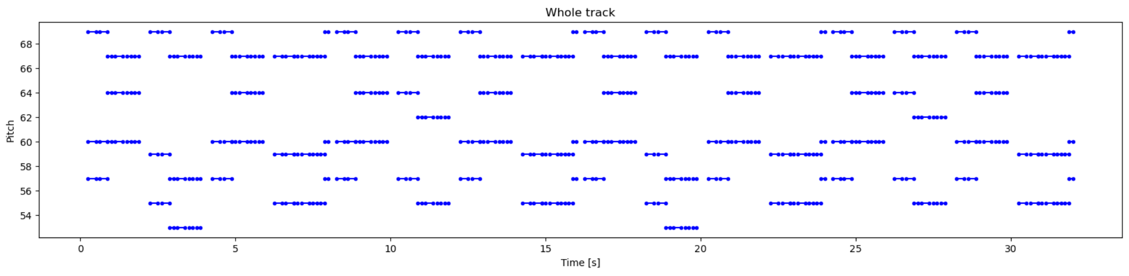


Figure 6: POP song 3

the POP song dataset to prevent the model from overfitting for the POP genre or underfitting for classical songs.

2.2 Encoding

For music generation, I used two different types of encoding for the notes of the songs. The **first** is simply [pitch, step, duration] with the pitch expressed in integers as in the MIDI format, the step is the time difference from the previous note and the duration is how long the note lasts.

- **Pitch:** MIDI format uses integers to represent the pitch of a note. Typically the pitch is between [0,128].
- **Step:** This refers to the time elapsed between the start of one note and the start of the next note. It's essentially the rhythmic distance between notes.
- **Duration:** This indicates how long a note is held. It could be measured in ticks, beats, or some other time unit.

Instead, the **second** encoding used is the one that makes use of musical intervals and is represented in this way.I encoded the notes using an integer interval from 0 to 12 which indicates the interval, another integer diff which indicates the difference in octaves between one note and another, and step and duration two integers that indicate the time in sixteenths of a second. Specifically, the Python code used to encode the sequence is as follows:

```python
def calc_intervallo(pitch1:int,pitch2:int):
    semitoni = (pitch2 - pitch1) % 12

    return int(abs(semitoni))

def calc_diff(pitch1:int,pitch2:int)->int:
    ottave = (pitch2 - pitch1) / 12
    return int(ottave)

def create_intervall_data(df):
    notes = []
    rate = 16
    prev_note = df[0:1]
    for _, row in df.iterrows():
        intervall = calc_intervallo(prev_note['pitch'],row['pitch'])
        difference = calc_diff(prev_note['pitch'],row['pitch'])
        duration = int(row['duration']*rate)
        step = int(row['step']*rate)
        notes.append([intervall,difference,step,duration])
    return pd.DataFrame(notes,
                        columns=['intervallo','diff','step','duration'])
df = create_intervall_data(train_notes)

seq_length = 10

dur_max = df['duration'].max()
int_max = df['intervallo'].max()
```

```
diff_max = df['diff'].max()
diff_mini = df['diff'].min()
step_max = df['step'].max()

#Dataset with pitches
def create_sequences(dataset, seq_length: int):
    seq_length = seq_length+1

    seq= []
    for i in range(len(df)-seq_length):
        seq.append(df.iloc[i:i+seq_length].values.tolist())

    return seq
data = create_sequences(df,seq_length)

df = pd.DataFrame(data)
```

Then I created the dataset files in sequences of 10 notes with only one note as the target in order to achieve good accuracy in note predictions.

2.2.1 Musical Genre

To experiment with how the models learned, I created three datasets. One with only classical music, the other with only pop music and the last one with both genres. In terms of dataset size, all three datasets contained 50 songs. The third data set was evenly divided between classical and pop music, with 25 songs from each genre

3 Models

3.1 Model with pitchs

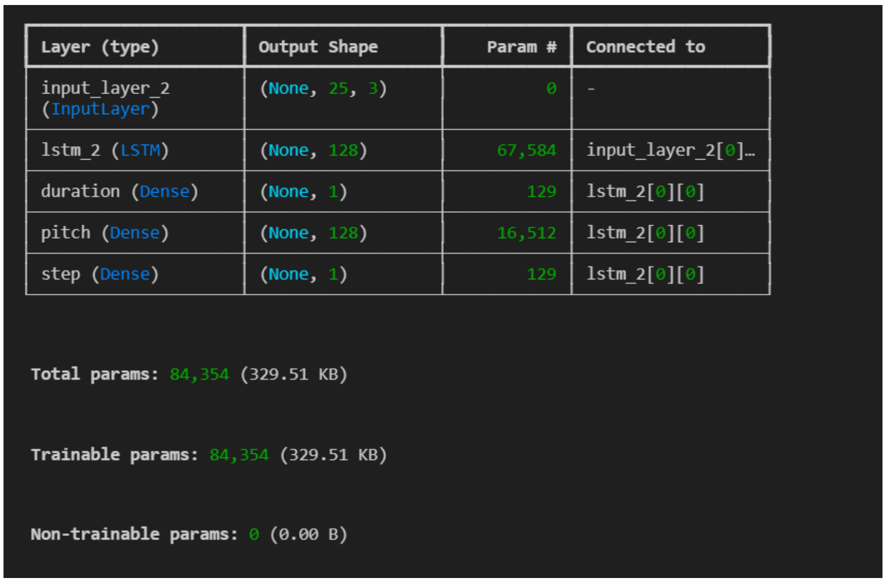
In the model with pitch, I used a classifier for pitch with 128 values, while for step and duration I opted for a regressor (so I did not discretize the musical times).

3.2 Model with intervall

In the model with intervals, I tried various combinations of architectures with both discrete and continuous time intervals. In particular, I found that the model, especially for classical songs, has a minimum interval (without loss of information) of a sixteenth of a second, so I chose to quantize the times in sixteenths of a second. Furthermore, I saw that both the steps and durations of the notes in the entire dataset reach a maximum of 300 sixteenths of a second (about 18 seconds) so I created the model was a multiple classifier on all four outputs.

3.3 Training

Some considerations to make about training are that the first model is much faster than the second, however in that model I only used the loss functions as metrics. The second one requires about 40 minutes to be trained with a totally classical dataset (therefore the richest in notes). I used separate accuracy metrics to train it.



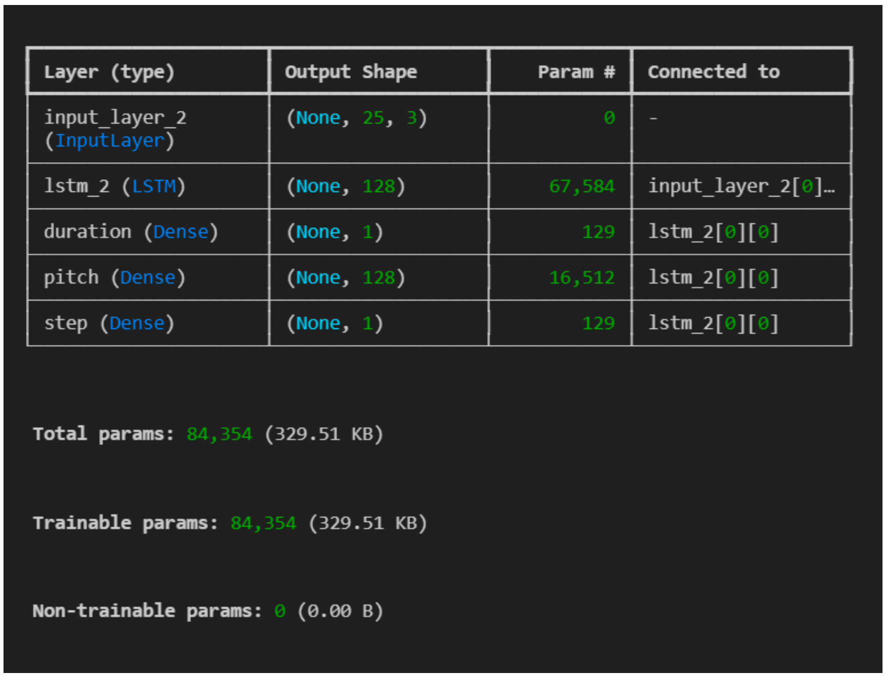
```
┏━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━┓
┃ Layer (type)        ┃ Output Shape      ┃    Param # ┃ Connected to      ┃
┡━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━┩
│ input_layer_2       │ (None, 25, 3)     │          0 │ -                 │
│ (InputLayer)        │                   │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ lstm_2 (LSTM)       │ (None, 128)       │     67,584 │ input_layer_2[0]… │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ duration (Dense)    │ (None, 1)         │        129 │ lstm_2[0][0]      │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ pitch (Dense)       │ (None, 128)       │     16,512 │ lstm_2[0][0]      │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ step (Dense)        │ (None, 1)         │        129 │ lstm_2[0][0]      │
└─────────────────────┴───────────────────┴────────────┴───────────────────┘

Total params: 84,354 (329.51 KB)

Trainable params: 84,354 (329.51 KB)

Non-trainable params: 0 (0.00 B)
```

Figure 7: Summary of model pitchs



```
┏━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━┓
┃ Layer (type)        ┃ Output Shape      ┃    Param # ┃ Connected to      ┃
┡━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━┩
│ input_layer_2       │ (None, 25, 3)     │          0 │ -                 │
│ (InputLayer)        │                   │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ lstm_2 (LSTM)       │ (None, 128)       │     67,584 │ input_layer_2[0]… │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ duration (Dense)    │ (None, 1)         │        129 │ lstm_2[0][0]      │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ pitch (Dense)       │ (None, 128)       │     16,512 │ lstm_2[0][0]      │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ step (Dense)        │ (None, 1)         │        129 │ lstm_2[0][0]      │
└─────────────────────┴───────────────────┴────────────┴───────────────────┘

Total params: 84,354 (329.51 KB)

Trainable params: 84,354 (329.51 KB)

Non-trainable params: 0 (0.00 B)
```

Figure 8: Summary of model intervall

4 Results

4.1 Pitch Model

The model with pitches seems to perform well in both generation cases. However, some songs may sound very similar to those in the dataset, with differences in pitch or other aspects.In particular, the generated song called ”pitchPOP” is very similar to a famous song titled ”Forget about it”.

4.2 Intervall Model

The model with interval encoding has many problems, especially in learning classical patterns. In fact, the octave difference encoding probably should have been expressed in a way that keeps the pitch within the range of a specific instrument (such as the piano). It can be seen that if the song starts from a high pitch (or vice versa), the model can easily go out of the range of allowed pitches. As a solution, I scaled the positive and negative differences by 2 so that I can generate more reliably.

4.3 notes

In the generated notebook you will find all the methods to generate and play the files in question.