AN AUTOMATIC SHEET MUSIC GENERATING ALGORITHM BASED ON MACHINE LEARNING AND ARTIFICIAL INTELLIGENCE

Ruize Yu[1] and Yu Sun[2]

[1]Santa Margarita Catholic High School,
22062 Antonio Pkwy, Rancho Santa Margarita, CA 92688
[2]California State Polytechnic University, Pomona,
CA, 91768, Irvine, CA 92620

ABSTRACT

Due to the ever growing popularity of music as a part of everyday life, and with the continuous advances in AI technology, it is now possible for computers to listen to and recognize music [1]. However, there still exist limitations on machines' ability to recognize audio. This paper proposes an application to simplify the process of music transcription and reduce its runtime [2]. This application was tested in a different range of settings and evaluated. The results show what can be further improved on this application.

KEYWORDS

MIDI, Pitch Recognition, Magenta.

1. INTRODUCTION

The human ear is able to detect a range of frequencies, allowing it to "register" different sounds upon hearing it. Due to ever-advancing technology, it is no surprise that researchers will eventually create pitch detection algorithms that capture sounds and send the information to the algorithm [4]. Ever since pitch detection algorithms have been introduced, people have found creative ways to utilize such algorithms to solve various problems that may present themselves [5]. One of such problems is recognizing the various pitches of a music file, and writing them in a sheet music format. This has always been done with human hearing, thus increasing the room for error if the said person does not have a perfect pitch. Thankfully, researchers have looked into this problem and have created pitch detection algorithms described earlier to deal with problems like such.

Similar programs have already been created to help the user convert a raw music file into MIDI, many of which exist as online conversion tools [6]. An implementation of such programs is Ableton, which comes with many additional settings on top of converting music to MIDI. However, if a user wishes to write the said file into sheet music, they have to go through various processes until they can reach the final result. Another problem that arises with the process is that the MIDI file might take extra space, making it very inefficient if the user only wants to convert the raw audio into a piece of sheet music. Most of the existing programs lack a comprehensive workflow that allows the user to complete the entire process without interruptions.

David C. Wyld et al. (Eds): AI, AIMLNET, BIOS, BINLP, CSTY, MaVaS, SIGI - 2022
pp. 251-258, 2022. CS & IT - CSCP 2022 DOI: 10.5121/csit.2022.121822

In this paper, the method used is very similar to other music-to-MIDI programs. However, our method includes converting the MIDI file to sheet music and writing it as an image for the user. The goal of this method is to attempt to create an algorithm that writes an audio file into sheet music and reduces the number of steps needed to do so. Some of the features included in this algorithm are connected together to provide a satisfactory result. First, the audio is converted from a .wav file into a .midi file, the MIDI file is manipulated by Mido [7]. Magenta and TensorFlow AI provides the conversion needed from raw audio to MIDI. Second, the MIDI file is converted into sheet music, this step requires the use of Music21, which provides all the components needed for a piece of sheet music, such as the tempo, time signature, measures, and such. The third feature of this algorithm is the storage of the converted sheet music, which uses Google Firebase to store the said sheet music, which is an image at the current stage. This image is then sent to the website, which is created using both ngrok and flask.

To demonstrate how the above techniques reduce steps taken to convert audio to sheet music, and how the program is accurate, there will be two different approaches taken. The first approach tests each individual process using google colab [8]. The program is separated into three distinct steps. The first step transcribes a raw audio file into a MIDI file, the second step converts the MIDI file into sheet music, and the last step opens up a webpage that displays the converted sheet music. As shown in a later experiment, the three steps require building up off of each stage to provide the final result. The second approach is through repl.it, and tests the program from end-to-end. This is done to show how the program functions without the need to go through individual steps.

The rest of the paper is organized as follows: Section 2 provides the details on the challenges faced during the experiment and designing stages; Section 3 focuses on the details of the solutions corresponding to the challenges in Section 2; Section 4 presents the details about the experiments done, following by presenting the related work in Section 5; Section 6 will show the concluding remarks, as well as the future work of this project.

2. CHALLENGES

In order to build the project, a few challenges have been identified as follows.

2.1. Identifying the Pitches

The first challenge that presented itself is identifying the different pitches within a raw audio file. Since the computer reads in pitches and processes it in a different way than the human ear, it needs to first know which pitch matches which music note. Additionally, it needs to be able to recognize the length of each pitch and record it inside the MIDI file. The second part of this challenge is writing into a MIDI file, which requires an algorithm and tuning [9]. The program cannot proceed without an accurate MIDI file, which will then be converted into a music sheet.

2.2. Recognizable Sheet Music

Another problem that came up was ensuring that the MIDI conversion writes an accurate and readable sheet music for the reader. Because music sheets have different clefs and such, the program needs to convert notes to their respective clef and will need to transcribe the notes depending on the clef. Another problem within this situation is writing in a different tempo and music notes, such as eighth notes, or the common quarter notes. The program needs to differentiate between when to use different notes and when to just change the tempo to make the sheet easier to read.

2.3. Creating the Website

After finishing the algorithm for the conversion, a website needs to be made. While a website is not very hard to create, the problem resides in connecting the algorithm, which uses google firebase, with the website itself.

3. SOLUTION

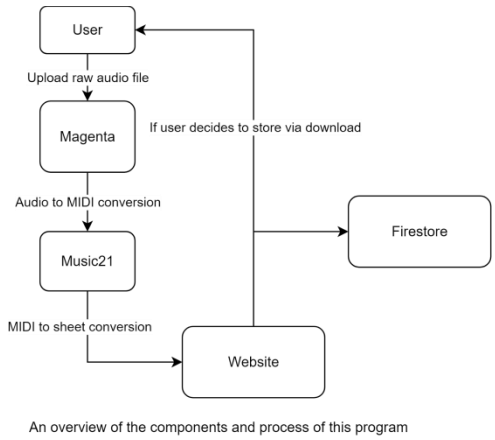

Figure 1. An overview of the components and process of this program

First, this process requires the user to upload a raw audio file in the form of .wav. This audio file is received by Magenta, a machine learning library focused and trained on music provided by Google, as an input to convert to MIDI [3]. Magenta will then use its Onset and Frames feature to convert the file to MIDI and will return it as an output. Then, the output will be received by Music21, a python-based toolkit for musicology. Music21 will primarily be used to write the MIDI file to sheet music in this process [15]. After Music21 finishes running, the output will be returned as a png image of the sheet music. This file will then be returned to the website, hosted by flask and ngrok, which will display the finished image to the user. The file will also be uploaded to Firestore, a cloud storage location.

Figure 2. Select file screen

The website first instructs the user to upload a file and press submit.
Using python, the music to sheet conversion is done with the help of Magenta, specifically, the use of its onset and frames function.

```python
import os
#return the midi file path
def generateMidi(filename:str):
  #generate Midi file and saved as song.wav.midi
    command_line = f'onsets_frames_transcription_transcribe \
    --model_dir="models/maestro/train" \
    --load_audio_with_librosa=True \
    --transcribed_file_suffix="" \
    {filename}'
    print(filename)
    os.popen(command_line).read()
    path = f'{filename}.midi'
    return path
```

Figure 3. The Screenshot of code 1

The command_line provides the necessary program configuration, with the audio file as the target. This method returns the transcribed audio file as a MIDI file, which is then used to convert to sheet music. To convert MIDI to sheet music, the program uses Music21 to parse the MIDI file (shown below as midi_path) to parse it into a music21 object.

```python
def midi2sheet_music(midi_path:str, song_path):
  #print("hi")
  filename = pathlib.Path(song_path).stem
  create_directory(filename)
  print(filename)
  print(f"###############{midi_path}")
  parsed = music21.converter.parse(midi_path)
  print (type(parsed))
  #parsed.show()
  #parsed.write(f'musicxml.png',f'{filename}/{filename}.png')
  return parsed
```

Figure 4. Screenshot of code 2

The annotated object is then used as a parameter for the generate_sheet_music function, which will use it to return a png file of the sheet music.

```python
def generate_sheet_music(song_path, folder_path = '/content/drive/MyDrive/Music_Sheet_Thing/'):
  midi_path = generateMidi(filename = song_path)
  print(f"*******{midi_path}*******")
  sheet_music_dir = midi2sheet_music(midi_path = midi_path, song_path= song_path)
  print(f"SHEET MUSIC DIR: {sheet_music_dir}")
  print(f"{folder_path}{midi_path}.png")
  #find why midi path adds a 1
  sheet_music_dir.write("musicxml.png", f"{folder_path}{midi_path}.png")
  url = firebase_storage.upload_file(local_path=f"{folder_path}{midi_path}.png",file_name= f'music/temp')
  return url
```

Figure 5. Screenshot of code 3

The program uses music21's .write function to write the object into a png file, which is then returned after being stored in the firestore.

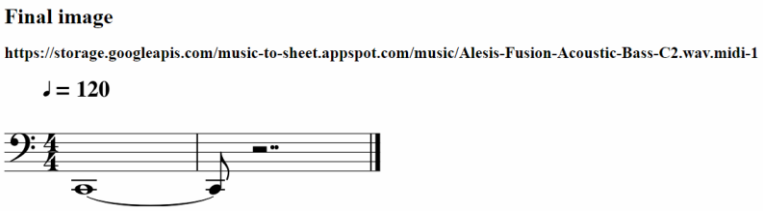

Figure 6. Final image

The png file is then displayed on the website that originally prompted the user to input a .wav file.

4. EXPERIMENT

4.1. Experiment 1: Individual Steps via Google Colab

In this experiment, we tested and timed each function of the program bit-by-bit. This is to find where the algorithm takes the most time to execute, and where the bottleneck occurs.

To proceed with the experiment, each code block is run individually using Google Colab, and their runtime and memory usage are all recorded with %%timeit, %%prun, and %%memit.

4.2. Experiment 2: Complete Process Using the Flask Website

Have multiple testing data be uploaded to the website and be converted to sheet music check for errors, inaccuracies, and see what can be improved. The result will be collected and will provide feedback on the accuracy of the algorithm. This experiment will provide us with data on how to improve the algorithm, and where the errors are occurring.

To test the accuracy of the converter, three different wav files of C major notes were created using MuseScore. The three different data were uploaded to the website, where it is then converted back to sheet music. The first wav file is a C Major scale played on a piano in common time. The second file is a scale of C Major triads, played on a piano in common time and in ascending scale. The last file is a variation of the first file but split up with a quarter rest between the notes.

After creating the sample data and running them through the algorithm, it appears that the general shape and notes of the audio file are preserved. However, there are cases where the converter showed inaccuracies. For example, in the test case where chords are uploaded to the website (Figure 2), the only inaccuracy shown is in the first chord, where the algorithm missed concert C4 (Figure 2b). Another example is Figure 1b, where the C Major ascending & descending scales showed that some notes are carrying on the next beat. When separated (Figure 3), the algorithm works fine again.

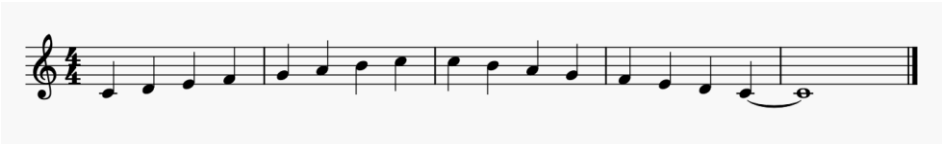

Figure 7. Original Version (Figure 1a)

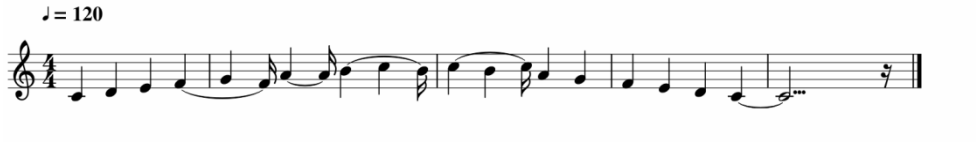

Figure 8. Magenta Version (Figure 1b)

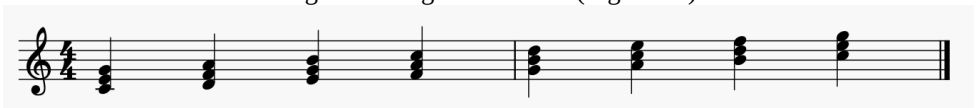

Figure 9. Original Version (Figure 2a)

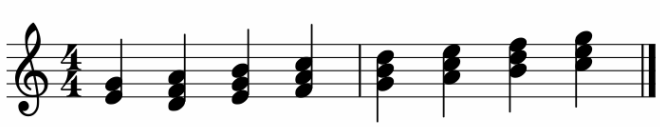

Figure 10. Magenta Version (Figure 2b)

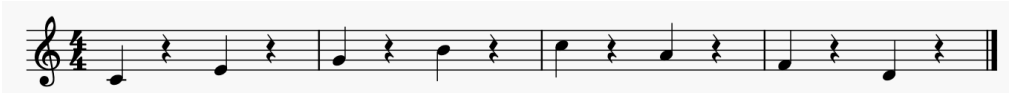

Figure 11. Original Version (Figure 3a)

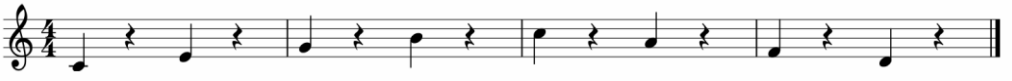

Figure 12. Magenta Version (Figure 3b)

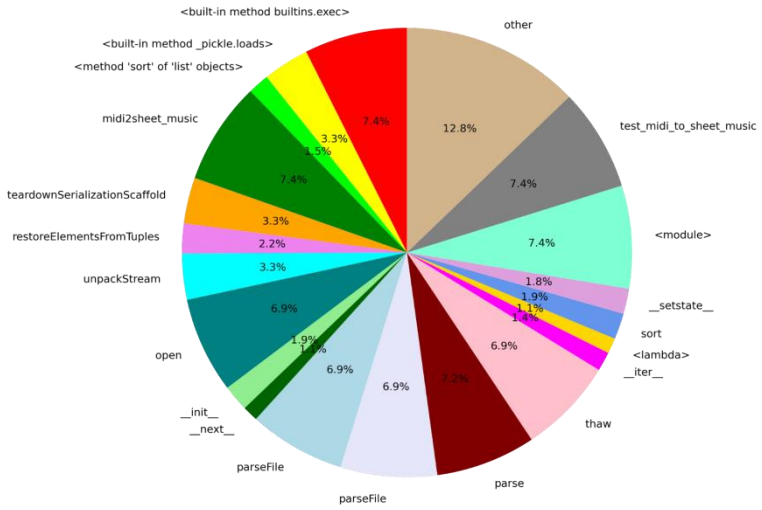

Figure 13. Chart graph

For the generateMidi function, a lot of time was taken up because the function was executing the return string from the command line. However, it was unnecessary for the program as it did not read the output. By removing the .read function, the runtime was reduced from around 25 seconds to 0.019 seconds.

For the midi to sheet music conversion, there is no need for optimization, as the entire method only took 0.449 seconds to execute around 68k function calls, with the most time taken being around 0.11 seconds for a single call.

5. RELATED WORK

Aitan Gossman and Josh Gossman, in their research Automatic Music Transcription: Generating MIDI From Audio, propose an approach that utilizes deep learning's superior learning ability to learn structures within images, and uses this ability to convert audio into MIDI [11]. As the two authors stated, they hope to achieve a similar network as Magenta, and ideally provide similar results. The result of their experiment proved it to be better than Google's Project Magenta,

reaching a F1 score of 0.95 compared to Magenta's F1 score of 0.9. Their AMT design proved to be better than Magenta, achieved through their neural network design and training of data. This AMT tool will have an edge over other music transcription algorithms due to it using a different deep learning API trained by themselves.

Mingheng Liang, in his research Music Score Recognition and Composition Application Based on Deep Learning, presents a deep-learning based music score recognition model [12]. The said model employs a deep network, accepts the entire score image as input, and outputs the note's time value and pitch directly. The work in question is almost the opposite of the algorithm described in our work; instead of accepting and reading an audio file as an input, Liang's work processes a musical score image and transcribes it into notes to be played.

In the work Magenta Studio: Augmenting Creativity with Deep Learning in Ableton Live by researchers from Google and the University of California, Berkeley, a design of a music generation application is provided, as well as its implementation as a plugin for the audio workstation Ableton Live [13]. The authors provided a flowchart of their application, which shows that the API references Magenta; the same library that the algorithm in this paper uses. However, the major difference between the two works is that Magenta Studio features multiple different plug-ins in addition to music generation, while this algorithm only focuses on the conversion of audio files to sheet music.

6. CONCLUSION

In this paper, we proposed a method that combines the steps of converting audio to MIDI, converting MIDI to sheet music, and uploading the sheet music to a website, into one algorithm. To test this algorithm's efficiency and accuracy, two different experiments were conducted, one tests the runtime and memory usage, while the other one tests the accuracy of the application. In the first experiment, each stage of the music transcription algorithm was timed during its runtime and recorded. In the second experiment, three different piano audio files were prepared using MuseScore 3, and the accuracy of this algorithm is tested by running it on all three audio files and comparing the result with the original music sheet [14]. The first experiment showed that the average run time is prolonged due to an unnecessary line during testing, which was eventually removed to reduce the run time. The second experiment showed Magenta's limitations on music transcription.

Currently, this system's accuracy is limited by using Google's Project Magenta [10]. Although accurate, it only achieves an F1 score of 0.9, which shows that it is still prone to external noises such as lingering notes and audio. This will only worsen when this application is used to transcribe any complex composition. During testing, it is also shown that there are no current ways to make adjustments midway, as the transcription is done from end-to-end.

These limitations will be solved if the future implementation of this program allows for access to the algorithm in between steps to adjust for error, as well as possibly training a deep learning network similar to Gossman's AMT system in work 1.

REFERENCES

[1] Extance, Andy. "How AI technology can tame the scientific literature." Nature 561.7722 (2018): 273-275.

[2] Benetos, Emmanouil, et al. "Automatic music transcription: An overview." IEEE Signal Processing Magazine 36.1 (2018): 20-30.

[3] Benetos, Emmanouil, et al. "Automatic music transcription: challenges and future directions." Journal of Intelligent Information Systems 41.3 (2013): 407-434.
[4] Rabiner, Lawrence, et al. "A comparative performance study of several pitch detection algorithms." IEEE Transactions on Acoustics, Speech, and Signal Processing 24.5 (1976): 399-418.
[5] Luengo, Iker, et al. "Evaluation of pitch detection algorithms under real conditions." 2007 IEEE International Conference on Acoustics, Speech and Signal Processing-ICASSP'07. Vol. 4. IEEE, 2007.
[6] Loy, Gareth. "Musicians make a standard: the MIDI phenomenon." Computer Music Journal 9.4 (1985): 8-26.
[7] Snyder, David, Guoguo Chen, and Daniel Povey. "Musan: A music, speech, and noise corpus." arXiv preprint arXiv:1510.08484 (2015).
[8] Alves, Francisco Regis Vieira, and Renata Passos Machado Vieira. "The Newton fractal's Leonardo sequence study with the Google Colab." International Electronic Journal of Mathematics Education 15.2 (2019): em0575.
[9] Yamamoto, Kotaro, and Munetoshi Iwakiri. "A standard MIDI file steganography based on fluctuation of duration." 2009 International conference on availability, reliability and security. IEEE, 2009.
[10] Roberts, Adam, et al. "Magenta studio: Augmenting creativity with deep learning in ableton live." (2019).
[11] Grossman, Aitan, and Josh Grossman. "Automatic Music Transcription: Generating MIDI From Audio." (2020).
[12] Byrski, Aleksander, and Marek Kisiel-Dorohinicki. Evolutionary Multi-Agent Systems: from inspirations to applications. Vol. 680. Springer, 2016.
[13] Roberts, Adam, et al. "Magenta studio: Augmenting creativity with deep learning in ableton live." (2019).
[14] Todea, Diana. "The Use of the MuseScore Software in Musical E-Learning." Virtual Learn (2015): 88.
[15] Cuthbert, Michael Scott, and Christopher Ariza. "music21: A toolkit for computer-aided musicology and symbolic music data." (2010).