Turning Youtube Videos into Piano Sheet Music

André Correia, Charles Wang, Harman Sihota, Juliana Choi
CPEN291 201, The University of British Columbia
May 12, 2021

Summary and Introduction

This document presents the final project developed for the course CPEN291 during Term 2 of the second year of the Computer Engineering degree at the University of British Columbia. The team is composed by four members, André Correia, Charles Wang, Harman Sihota, and Juliana Choi.

The main goal of the final project is to apply the knowledge attained during the classes and assignments of the course by creating a model that will perform a useful and complex function. In our case, the goal of the project is turning youtube videos of piano compositions into sheet music. The original idea came from wanting to find a free way to learn new piano songs through Youtube, as often buying sheet music can be very expensive. Juliana, one of the team members, had experience learning compositions by slowing down videos and looking at the player's fingers. However, the problem of not having sheet music arose as she dove into harder pieces in which the performer didn't show their hands.

Having that in mind, the group decided to implement a website that asks for a Youtube link, extracts the audio file from the video, uses Machine Learning to create the sheet music for the song, and downloads the file on the computer. In order to do that, the team had to build a website from scratch, create a ML model that would find the relationship between the sounds in the audio recording and the notes in the sheet music, generate and process a dataset of songs for the model, and also process the outputs so they would be displayed correctly.

Dataset

The initial proposal of the project done back in March 2021, said data acquisition would be done through Youtube videos in order to train the model. However, upon further research, our team realized that there are other resources online that make data available in the format we

need. An example is The Mutopia Project, which has over 780 music sheets and audio available for download. We implemented a simple scrapping mechanism using Selenium and downloaded the files that are currently in our M1 folder. Here follows a screenshot of the code:

```
# Code to scrape samples using Selenium. Requires user to install chromedriver.

from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
import time

PATH_TO_CHROMEDRIVER = ''

driver = webdriver.Chrome(PATH_TO_CHROMEDRIVER)

driver.get('https://www.mutopiaproject.org/cgibin/make-table.cgi?Instrument=Piano')

for i in range(77):
    try:
        tables_mid = []
        tables_ps = []

        for j in range(1,11):
            tables_mid.append(driver.find_element_by_xpath(f"//table/tbody//tr[{j}]/td//table//tbody/tr[4]/td[2]"))
            tables_ps.append(driver.find_element_by_xpath(f"//table/tbody//tr[{j}]/td//table//tbody/tr[5]/t'd"))

        for table in tables_mid:
            time.sleep(0.5)
            table.find_element_by_partial_link_text('.mid').click()

        for table in tables_ps:
            time.sleep(1)
            driver.execute_script("arguments[0].click();", table.find_element_by_partial_link_text('.ps'))

        link = driver.find_element_by_link_text('Next 10')
        link.click()

    except:
        continue
```

Picture 1: Code used to scrape audio from The Mutopia Project

Research was one of the main helpers during this project. We found several resources that mentioned Machine Learning for similar purposes. One of the most helpful articles we discovered was Note Recognition in Python, a medium publication by Ian VonSeggern that guided us through the first steps of processing our audio files. It included important insight that confirmed our hypothesis that the audio files would have to be modified (i.e. identify where notes are so that we get short data samples that aren't very populated). Furthermore, it showed code snippets and plots of sample data that helped us know what we were looking for. Lastly, it provided insight on how to classify the notes by using their frequency spectrum.

As a result, we decided to pre-process our audio samples so that the data in them was divided into very small pieces. That way we could avoid multiple notes contained in the same interval, a scenario that would make learning more complicated and predictions less accurate. In

order to do that, we multiplied the number of measurements taken in a second by *sample_length_2*. Initially, we thought the ideal measurement would be ¼, since it is one of the shortest notes described in sheet music. However, upon trial, we discovered that they wouldn't accept a number with so many decimal cases (0.25 didn't work while 0.2 did). Taking that into consideration, we decided to go for 0.5, as it would most likely be able to catch the notes and not take too long to prepare our data. A picture of the code with the sample length is included below.

```
sample_length_2 = 0.5

[ ] PATH_SPEC_2 = 'drive/MyDrive/Spectrograms/Spectrograms_sub_2'
    PATH_MID_2 = 'drive/MyDrive/audio_2'

[ ] # from https://github.com/jsleep/wav2mid/blob/master/examples/one_hot.py

    def create_label_2(pm, time, fs=1):
        for instrument in pm.instruments:
            for note in instrument.notes:
              if note.start >= (time - sample_length_2) and note.start <= time:
                return torch.as_tensor(note.pitch, dtype=torch.long)
              elif note.end >= (time - sample_length_2) and note.end <= time:
               return torch.as_tensor(note.pitch + 128, dtype=torch.long)

        return torch.as_tensor(257, dtype=torch.long)

[ ] def create_sample_2(signalData, fn, j):
        signalData_float = signalData.astype(float)
        f = librosa.cqt(signalData_float, fmin=46.25)
        librosa.display.specshow(librosa.amplitude_to_db(f, ref=np.max), y_axis='log', x_axis='time', sr=22050)
        plt.savefig(PATH_SPEC + '/' + fn.replace('.mid', f'_{j}.jpg'))
        return PIL.Image.open(PATH_SPEC + '/' + fn.replace('.mid', f'_{j}.jpg'))
```

Picture 2: Code used to create the dataset and the labels of the model, with each sample having length 0.5

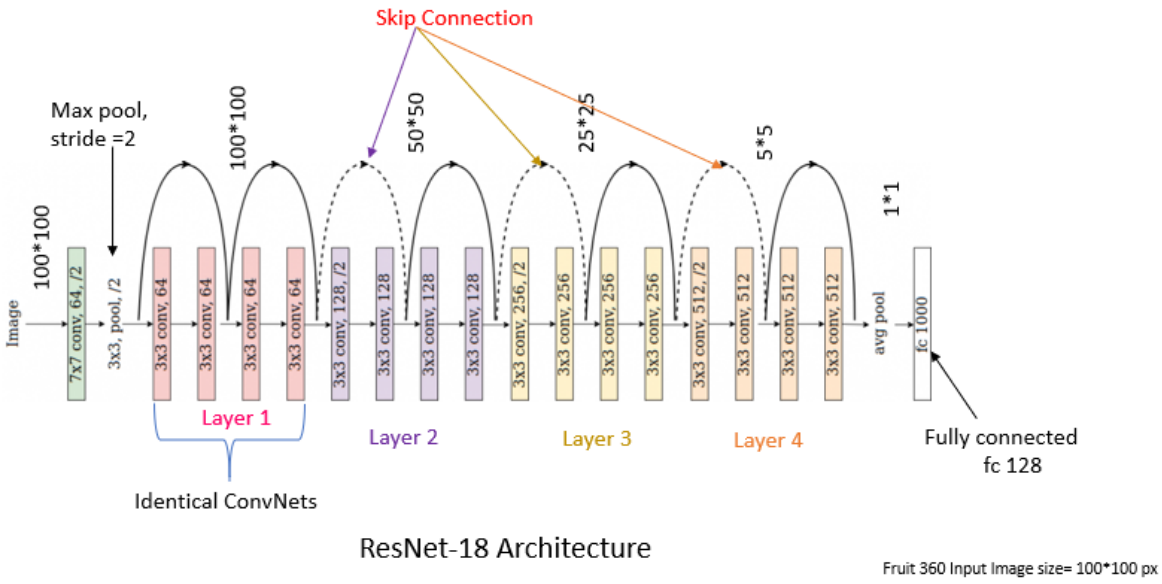
It's important to note that we divided the data that was contained inside the audio recording before actually turning it into something that could be used by the model. That was another one of the technical issues we faced, as we found out through the research article that frequency had to do with the model but we didn't know how to get a spectrogram from the data in the audio files. Luckily, we ran across Automatic Music Transcription Using Neural Networks, a research paper that lead to very important insights, such as that we should use Constant-Q Transform to transform the audio data to the frequency domain in a logarithmic scale and that we could use one-hot encoding vectors to represent the output data (the music notes).

After that, we searched for libraries that could be used to turn the audio file into spectrograms. We found Librosa, which we used to create spectrograms using Constant-Q Transform. Then, we were able to create the dataset that was filled with notes in form of spectrograms, as we wanted.

Model

Similar to the dataset, the team did a lot of research in order to create the best model possible for our project. An interesting resource found was Play Sheet Music with Python, OpenCV, and an Optical Music Recognition Model, an article that describes research done to decode and play sheet music. Despite doing the opposite of what the project aims to do (having sheet music as the outputs, not inputs), it brings great insight on how the model can be built, as it also used Machine Learning to solve the problem. It used a convolutional neural network combined with a recurrent neural network and trained the model with stochastic gradient descent (SGD), which aimed at minimizing the connectionist temporal classification (CTC) loss function.

Our model was based on ResNet18 and used transfer learning. The main change was that the fully connected layer was replaced with a linear layer with 257 outputs instead of 1000. Initially the team tried to make a convolutional model with a non-linear last layer, in order to have both the timing and the note in that interval. However, the model ran into a few issues and with the professor's guidance, the team decided it would be best to have only one dimension on the last layer.



Picture 3: Picture of ResNet-18 Architecture found online

Hence, since there are 128 midi notes and each note has an onset and an offset, the model has 128 + 128 + 1 classes (the extra one is if there is neither a note onset or end in that time frame). The onset represents the beginning of the note (when the note started to be played) and the offset the end of the note (when it stopped being played). With that information, we are able to determine not only what note is being played, but also its length.

Since our project is in its biggest part a classification task, the group decided to use cross entropy loss to improve the model. After testing it a few times, overfitting was found. To counteract it, the team implemented AdamW as the optimizer and found that a weight decay of 2 was optimal. Moreover, StepLR was used to further better the model. At last, a learning rate of 0.001 was found to produce the best results.

Another insight taken from the model was that adding more layers to the fully connected layer was actually detrimental to performance, so the fully connected layer was left as a single linear layer. With approximately 10000 samples, and the parameters as described above, the model achieved a test accuracy of 47% in 15 epochs.

Despite seeming a bit low compared to previous projects done in class, the result was actually very successful in comparison to others that tried to do the same. For instance, Automatic Music Transcription Using Neural Networks, the research paper we based ourselves on, had an average 37.25% accuracy (despite being able to recognize different instruments playing at the same time).

Finally, since our model was built around MIDI notes to begin with, creating a file with that format wasn't too hard. The biggest issue we faced after converting the output to a MIDI file was finding a way to convert it into sheet music. We found resources such as MIDIToSheetMusic, but we didn't want to risk copyright issues with using their service. As a solution, we decided to make our website download the MIDI file to the computer, and direct the user to their website so they could easily get access to the sheet music.

Lastly, it is important to recognize that the model developed in the project was made in a short period of time, which constrained how many problems could be tackled at once. If the team were given more time, it would focus on recognizing multiple notes in the same sample (Neural networks for musical chords recognition is an interesting source), and much more, as was mentioned in the Youtube video made for the project.

Website

The website was built in order to have a platform that connects the user to the Machine Learning model. It is based on two files of code, an HTML file (front-end) and a python file (back-end). The HTML gives the user an area to input a valid url which will be then processed by the python script by downloading the video's audio file in .m4a format. Finally, this type of file is already a valid input to our model.

Currently, the website is being hosted on the members' computer's local host. This process is very simple and is detailed in the project's video so that anyone can easily use the platform by having the code downloaded.

It was challenging for our team to develop a website, but it was a great experience as we learned the basics of web development, new programming languages and networked in order to get the platform working at our best potential. The whole process can be divided into 3 major parts.

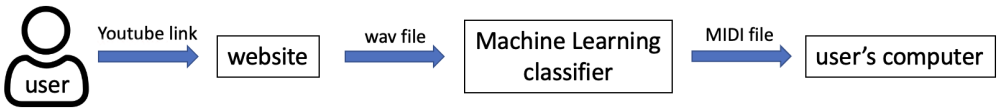
First, we learned HTML and CSS so that we could have a layout that is simple, user-friendly and visually pleasant. Second, in order to develop the back-end of the website so that the data inputted could be processed, we used python and flask. Finally, to connect to youtube's platform and extract audio from any video on it, we used youtube-dl's audio_downloader, in which we can specify the format of the audio file we want.

The website folder will have input (called audios) and output folders. The first one will contain the .m4a audio file and after some processing the second one will be filled with the .mid files that can be converted to sheet music using the hyperlink that is also contained in the website.

Picture 4: The website designed for our project

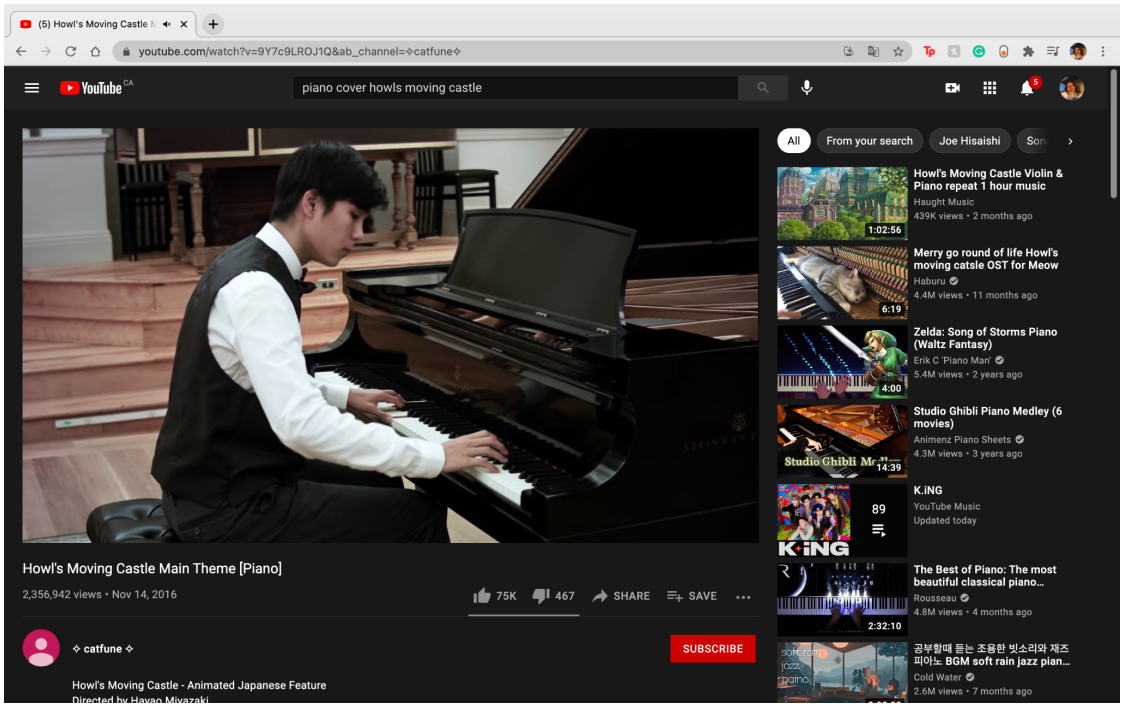
Walkthrough of the Project

A well-defined diagram of the system and its data flow follows below:



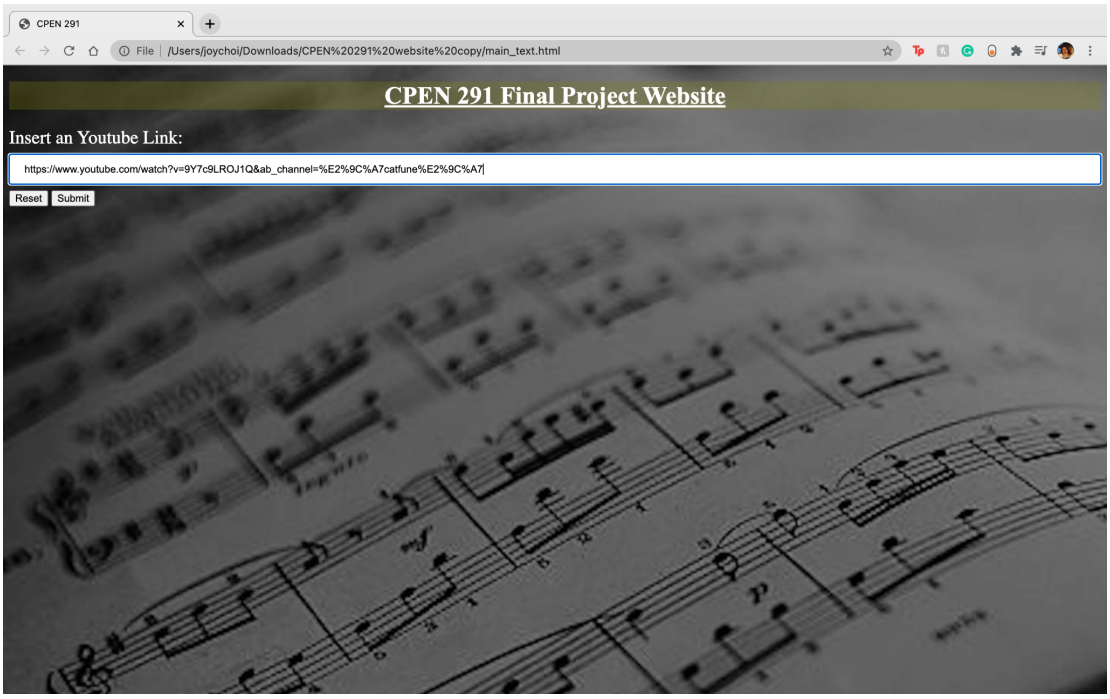
Picture 5: Diagram showing every component present in the system

The first step of the project would be finding a piano cover on Youtube that the user would be interested in learning, but has a hard time finding free sheet music for. We show an example below:



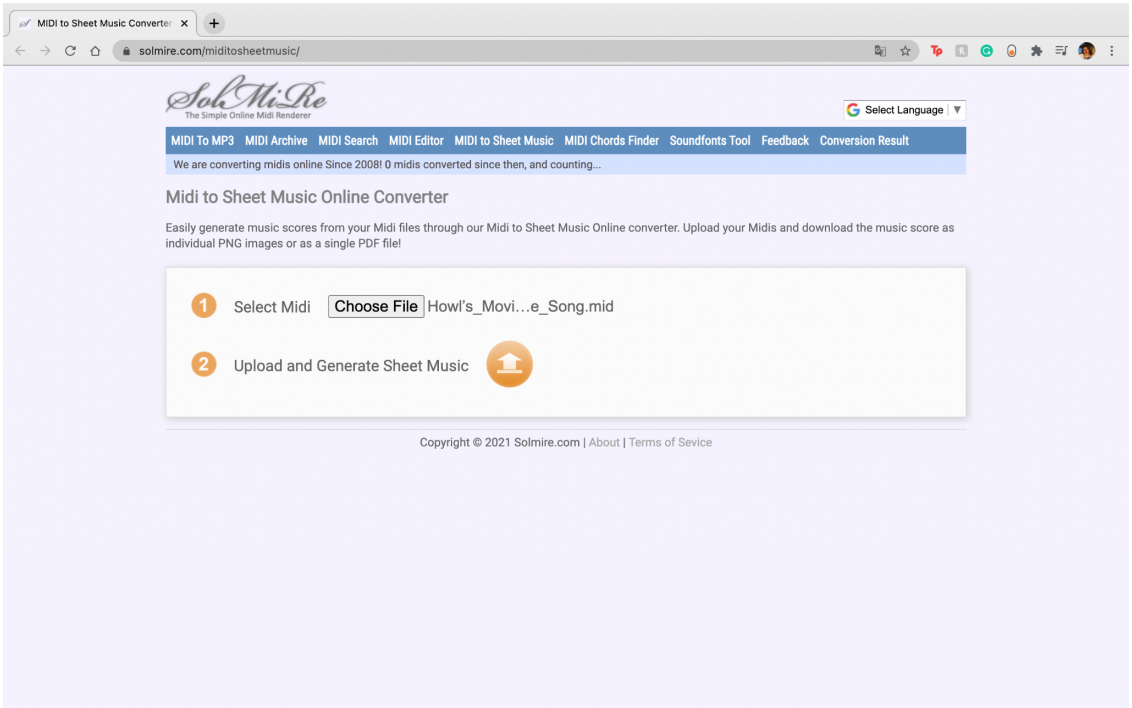
Picture 6: Screenshot of [Youtube video] with pianist playing Howl's Moving Castle Main Theme

Then, the next step would be going to the website of the project and inputting the Youtube address of the video, as shown in Picture 6.



Picture 7: Screenshot of website with link to video referred to in Picture 5

The final step is uploading the MIDI file in [MIDIToSheetMusic](), which is linked in the original website.



Picture 8: Screenshot of [MIDIToSheetMusic]() with MIDI file uploaded

Contributions

Andre Santiago coded the first steps of the model, but later on focused on working on the website. He built the front-end of the website and connected the back-end to our model by learning with resources online. He also reviewed the reports before they were submitted and filmed a few shots for the final video.

Harman Sihota was the biggest contributor to the Machine Learning code for our model and dataset collection and representation. He researched the best possible models for our project and tested out the ones that worked. He also was the one who programmed the training and testing code and found combinations of hyperparameters that made the accuracy go as high as 47%. He was also responsible for the preprocessing and postprocessing of the data.

Juliana Choi was the one responsible for most of the research made, learning from resources spread through the internet. She wrote the Milestone Reports and the Final Report, plus filmed and edited the video for the project. Lastly, she was the one who had the most experience with music and guided decisions in the model and dataset that would make most sense from her musical background.

Charles Wang conducted some preliminary research for the model, such as usage of Youtube to scrape data. He later also helped with working on the back end of the website for an initial file upload version of the website, researching for and working on connecting the trained model to the website to make predictions.