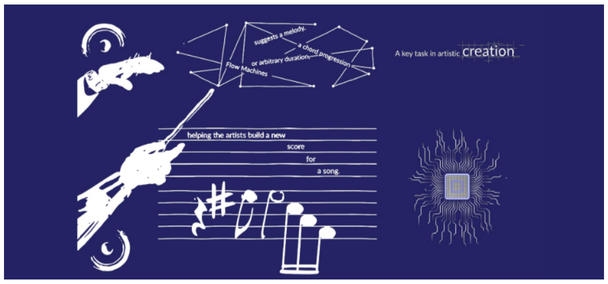
Neural Notes

By John Ahn, Houlin He, Ying Qi Wen, Jefferson Zhai



CPEN 291: An Introduction to Machine Learning
Department of Electrical and Computer Engineering

University of British Columbia

May 16, 2021

Neural Notes

A Music Playback App of Monophonic Scores

A Formal Report

Submitted to
CPEN 291 Teaching Team

CPEN 291:
An Introduction to Machine Learning

Submitted by
By John Ahn, Houlin He, Ying Qi Wen, Jefferson Zhai.

Faculty of Applied Science
University of British Columbia

May 16, 2021

1.0 INTRODUCTION

Music, as a form of artistic expression and emotional communication, is central to the development of human culture. The bloom of technology in the last century has caused a global increase in the production of digital devices and consumption of digital information, which has also greatly influenced the music industry. Digital tools have sparked new genres of music, allowing musicians to express themselves using synthesizers and voice-distortion technology. While much of the current technology works at innovating the sound, there have been limited innovations in understanding the language of music—a set of symbols that capture space, time, and dynamics. There exists no effective tool for the perception of musical notation, inhibiting the improvement of playback and visual music manipulation. This field of research is what is known as Optical Music Recognition (OMR). This report describes our implementation and engineering process of playing back music notes scanned from a musical score. We use machine learning techniques derived from current OMR literature to classify the notes, and we also developed an Android-based mobile application for our user interface, so that users can upload or take images of music-sheets they would like played back. We believe professional or amateur musicians would greatly benefit from hearing the auditory representation prior to parsing its notation. The beginning of this report provides an overview of our note-classification process and goes into detail on the datasets and techniques we utilized for our machine learning model. Next, we describe our front-end implementation, namely our Android-based mobile application and its basic interface. Then, we describe our back-end server implementation, which runs our machine learning model and interacts with the front-end mobile application.

2.0 NOTE CLASSIFICATION

MACHINE LEARNING

The techniques used to extract features of musical symbols in current OMR literature can be categorized into two main ideas: consider the musical document as a single unit and predict its sequence of notes or extract its isolated elements and classify feature each separately. The second idea is high in complexity and requires us to measure the coordinate of every single note and staff-line on the image. This has extremely high overhead and a significant lack of substantial datasets available. Therefore, for our implementation, we chose the first approach, which has a wide variety of datasets, higher performance, and less overhead.

DATASETS

Our dataset consists of two distinct parts for every input: a music sheet and its corresponding semantic representation. The semantic representation contains a sequence of symbols describing the musical meaning of the score. For example, as shown in figures 1a and 1b, the first element in the image, namely the treble clef, is denoted as “clef-G2.” This unique representation is the output of our model, which is then translated into a MIDI file to be played back. In total, the semantic representation contains 1781 elements to describe all types of musical notation, more information can be found in [1]. Our entire dataset consists of over 100,000 distinct images by augmenting existing datasets and mixing new images from different sources to create more variety and increase our model’s accuracy [5].

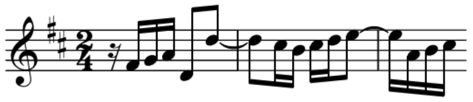


Figure 1a: Sample Musical Score [1]

```
clef-G2, keySignature-DM, timeSignature-2/4, rest-sixteenth, note-F#4_sixteenth,
note-G4_sixteenth, note-A4_sixteenth, note-D4_eighth, note-D5_eighth, tie, barline,
note-D5_eighth, note-C#5_sixteenth, note-B4_sixteenth, note-C#5_sixteenth,
note-D5_sixteenth, note-E5_eighth, tie, barline, note-E5_sixteenth,
note-A4_sixteenth, note-B4_sixteenth, note-C#5_sixteenth
```

Figure 1b: Sample Transcript of Sequences [1]

Dataset Augmentation

For our dataset augmentation, we designed a splitting algorithm that horizontally and vertically splits our inputs.

The horizontal splitting is an algorithm based on processing pixels of images. It transforms images into np arrays, and processes them with CV2 functions. It is able to observe the pattern behind the music staves in a music score, and detect the white space between the staff lines. The white spaces are then cut off. As the input images can be blurry, the algorithm sets a special variable to tolerate a certain extent of the blurriness. With this algorithm, we are able to play a music score that has multiple staff lines by going through one music staff after another.

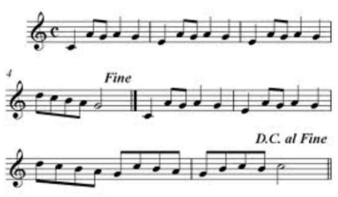


Figure 2a: Before Splitting

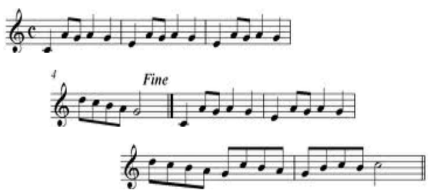


Figure 2b: After Splitting

The basic of the vertical splitting is similar to that of the horizontal splitting. This function is also able to detect and remove the space between notes, splitting the music sheet from left to right into normalized images which contain one symbol each. This is shown in Figure 3. This algorithm aims to crop notes on the music stave, which is then fed into the model for training purposes.

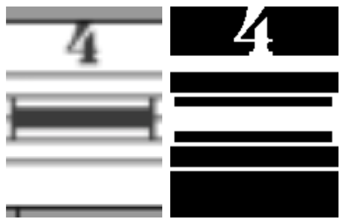


Figure 3: Before Splitting (Left) After Splitting (Right)

Challenges

In Vertical Splitting, there were some special cases that needed to be taken into account. One of the majority is the Clef-C and Clef-F notes. There is a tiny space within the symbol. As notes can be placed tightly near each other, we are able to treat the space in Clef-C and Clef-F notes as special cases, and ignore it.

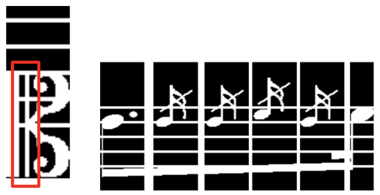


Figure 4: Space In Clef-C (Left) Note Beaming (Right)

The note beaming is another major special case being considered (Figure 4). These notes are connected, however, they should be treated individually; besides, it is possible to place notes within the beaming notes. Our algorithm is able to separate those by detecting the tail of each beaming note, and extract the notes hide within them by observing specific patterns of the beam.

Dataset Construction
The note images are fed into the dataset with their corresponding names. Additionally, the semantic representation of each input is converted into indexed labels using a hash table. When the training machine learning model accesses the dataset, the dataset returns note images in the form of tensors and indexed-labels. The label can be translated back using the hash table when needed.

MODEL ARCHITECTURE

Our model architecture was inspired by J. Calvo-Zaragoza et al. [1]. The model utilizes a Convolutional Recurrent Neural Network (CRNN) and a Connectionist Temporal Classification (CTC) loss function. This model addresses the musical document as a single unit rather than as a sequence of isolated elements that have to be classified separately. This means that for our model, we are not required to provide information about the composition or location of the symbols, but only the musical score and its respective transcript of music symbol sequences as its inputs. Figures 1 and 2 show an example of the music score and its corresponding transcript of sequences.

The Convolutional Neural Network is responsible for extracting the input images features and learning how to process it; the Recurrent Neural Network is in charge of producing the sequence of musical symbols. By combining these two structural components together, each musical symbol in the input image is treated as an individual frame and is predicted. However, the mechanism of the CRNN model alone is restrictive as it requires us to provide the total number of individual frames in the input image. This can be nicely solved by the CTC loss function, which allows a variable input width when training.

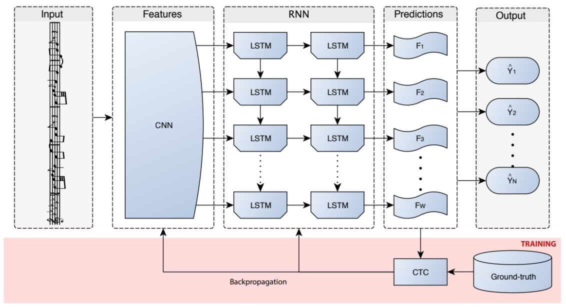


Figure 5: Graphical Scheme of the Model [1]

Implementation Details
Here, we describe our implementation of the model on Pytorch [5]. The image inputs are first grayscale and normalized and then rescaled at a fixed height of 128 pixels without altering its aspect ratio. The details of our configuration of the neural model are given in Table 1. The learning process is carried out by utilizing the Adam optimizer, an extension to the stochastic gradient descent that modifies the CNN parameters through back-propagation to minimize the CTC-loss function. In all cases, Rectified Linear Unit activations are considered.

Input (128 x *W* x 1)
Convolutional Block
Conv(32, 3 x 3), MaxPooling(2 x 2)
Conv(64, 3 x 3), MaxPooling(2 x 2)
Conv(128, 3 x 3), MaxPooling(2 x 2)
Conv(256, 3 x 3), MaxPooling(2 x 2)
Linear(2048, 256), Dropout(0.2)
Recurrent Block
GRU(256, 128)
Linear(256, vocabulary size + 1)

Table 1. The layout model used in this work consists of four convolutional layers and one recurrent layer. There are linear layers in between the CNN and RNN and as the output; they are used as transitional inputs. In all cases, Rectified Linear Unit activations are considered. Note: *W* stands for width.

Limitations
Due to the nature of CRNN and our output sequence, our model can only accept monophonic scores. This means that it cannot classify chords or musical scores that require double staff-lines to be played simultaneously (piano sheet music). To accept homophonic or polyphonic scores requires further research to be conducted in the OMR field.

EXPERIMENTAL ANALYSIS
When trained, our model achieves an accuracy of about 90% when evaluating our validation set. We have trained it on over 70,000 sample images, as described in our dataset section above.

Errors
Some of our errors stem from the model detecting the notes a half-step off; for example, classifying the note as a flat or a sharp when it should be a natural or vice versa. However, we found most of our inaccuracies predominately stem from a misclassification of bar-lines within the input image—our model detects bar-lines with 60% accuracy.

Optimizer
The model originally used the stochastic gradient descent optimizer (SGD), but we found this decreases our training accuracy and gets stuck on a local minimum extremely fast. We found that using AdaGrad (Adam) optimizer immensely increases our training and validation accuracy and minimizes our training loss—it reaches a steeper local-minimum. Figure 4 shows our experimental results with the two optimizers; the SGD optimizer never gets below the 20% loss mark, whereas Adam reaches the 5-10% loss mark.

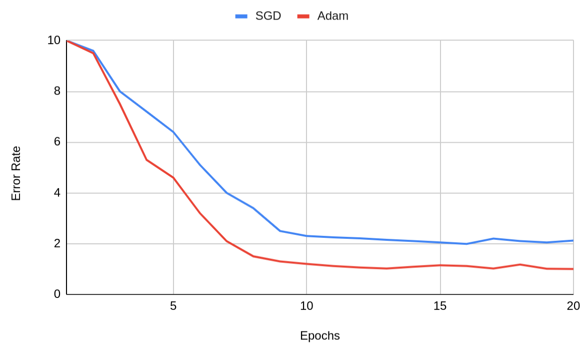


Figure 6: Error rate for our model with SGD and Adam

Batch Size:
We train our model on a standard batch size of 16. We experimented with different batch sizes and concluded this was the optimal size for both RAM usage and accuracy.

3.0 MOBILE APPLICATION

ANDROID VS iOS
Due to the limited amount of time for this project, we chose to only implement the mobile application on the devices using the Android OS. This was done for a couple of reasons. The first reason is that we are familiar with the programming language (Java) predominantly used to create android applications, unlike iOS which uses an unfamiliar language (Swift or Objective-C). The other, more compelling reason was due to hardware barriers in iOS development that prevented us from doing so. The iOS SDK requires XCode which only runs on Mac OS X, and since no team members own a Apple laptop or computer, it means developing for iOS requires either buying one, downloading a VM running OS X, or trying to install a “Hackintosh” OS, which are hurdles or limitations that effectively prevent us from developing for iOS.

FUNCTIONALITY

Selecting a music sheet
Neural Notes offers 2 options to send a sheet of music. One way is taking a picture with a camera, but the preferred way is the second method, which is to send a scanned image from the phone's local image gallery.

Sending to the server
Once a photo is selected, the "Create Music" button will send the music sheet to the server in the form of a byte stream. The application uses the Volley library from the official android developer guides through a multipart request. Since Volley does not support Multipart requests by default, we used a program from the internet, which integrates them together. We modified this program by removing the parts where they send the metadata and header of the file alongside the body to sending only the body of the file itself so that the server is able to reconstruct the byte stream into a proper PNG file.

Receiving a Response
After the server finishes running the machine learning model on our input, we will receive a string which represents the music that will be played. During the project, we decided that it will be the responsibility of the Neural Notes application to parse the string output of the machine learning model and generate the appropriate audio file, since the PRIMUS library we used that converts a string to audio file is written in Java—which makes it simpler to integrate into the application as it is also written in Java. Infrequently, the machine learning model from the server might return a string that may contain certain duplicate tokens that do not affect the music and would ruin an otherwise accurate output and thus a "fail-safe" function, that uses regex to slightly adjust inputs that might be invalid, before it is passed into the appropriate functions the PRIMUS library.

Generating the music
Continuing from the above, while integrating the library into the android application was most likely easier than doing it on the python-based server, it still came with its own host of challenges. Although Android Studio (the IDE we use to develop) was able to decompile the jar file containing the PRIMUS library, being able to decompile the Java Bytecode into readable Java code, there was a lack of documentation while costed us lots of time as we were required to study to code and figure out what functions in the library we will need to use.

Additionally, one obstacle that almost ruined the project was that the PRIMUS library itself was dependent on the javax.sound.midi library, which is a library that android SDK did not support. Fortunately, someone has ported the javax.sound.midi library to Android, which prevented the days of work analyzing the PRIMUS library from going to waste. Finally, one of the functions in the PRIMUS library also needed to be rewritten to suit the application's needs - since the program was designed to write to a file on core Java so necessary changes were made so that the library can write to a file to the internal storage of the android application.

GUI OF THE APPLICATION

Figure 8: Camera icon

Figure 9: Gallery icon

Figure 9: Music icon

Figure 10: Audio icon

4.0 BACKEND SERVER

Choice of Platform
The place on which the server is hosted is provided by Google Cloud Services. The platform provides resources to create Linux virtual machines (VMs) on which servers can be hosted. This provides three major advantages. It is possible to create backup VMs (machine images) during testing that allows the user to quickly revert to an old snapshot. This makes testing the server much faster and less risky. In addition, the deployment of a server on a VM allows the server to be deployed and available at all times without needing to keep the local machine up indefinitely. Lastly, the VM provided by Google Cloud Services provides greater stability than local virtual machines created using Virtualbox and many problems can be avoided.

Deployment of Server
The connection between the Android application and a remote server can be established in various ways depending on the type of server. The server of deployment is chosen to be a web server. There are two reasons why a web server is preferred, the first reason being the difficulty of setting up a web server is the smallest compared to other types of servers. This does not only apply to the setting up of the server, but also to adding functions and troubleshooting in later stages. The second reason is there exists a method known to the team of sending and obtaining data directly from web services in Android applications.

The actual deployment and creation of a web server is done on a Linux (virtual) machine and therefore the server of choice must be compatible with the operating system. The deployment server is an Apache server because of a tutorial that is easily obtainable and doable [6].

Deployment of Machine Learning Model
In order to work with the web server one must work with and alter the web application. Without a framework the programming will be done in PHP. However with the use of an appropriate framework it is possible to edit the web application using python. The web development framework of choice is Flask, this is due to a tutorial that is readily available. In integrating the Flask application, which is written in Python, into the web server, the Web Server Gateway Interface (WSGI) is used. This is done by simply configuring and adding a .wsgi file [6]. This makes the deployment of the machine learning application much easier, as the code can be imported as a class and executed as long as an image, a path to the ML model, and a path to the vocabulary semantics are given. This also made the troubleshooting of the web application much easier as it is in a programming language the entire team is familiar with.

Data Processing and Communication with App
The server receives an image as a string of bytes from a POST request. Therefore the server parses the string of bytes as a PNG image. A python script is then used to split the image into lines of music bar lines and feed them individually into the ML model. As an output a string is generated that consists of notes and music instructions and this string is sent back to the Android application.

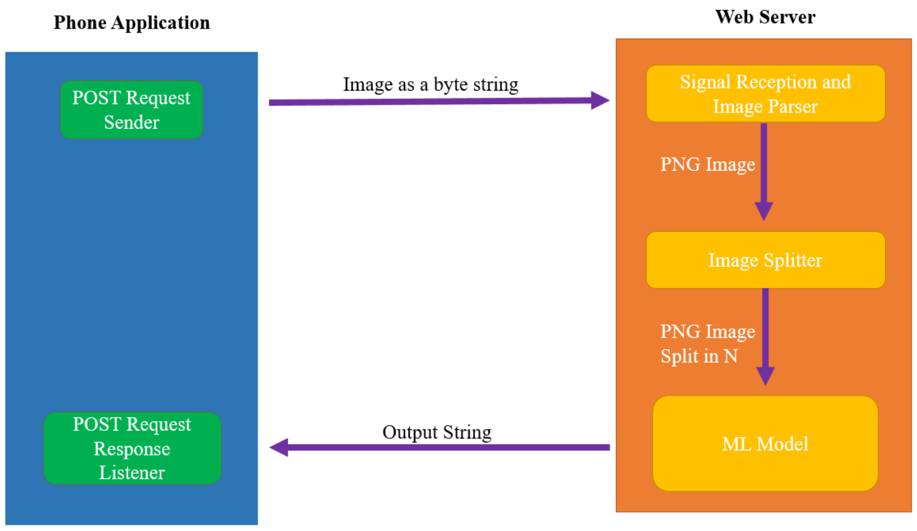


Figure 11: High Level Description of Server Functionality

5.0 CONCLUSION

In the end, we were able to convert basic music scores into their corresponding sound files with reasonable accuracy in both music notes and tempo. Collectively, we have gained great knowledge of android development and backend server management, while sharpening our machine learning hacking abilities. However, the greatest learning experience was the coordination between the 3 key components (the android application, the web server, and the ML model) of the project and the team's ability to adapt to changes with the overall project. Key parts that required coordination were the communications between the application and the server, the integration of the ML model into the server, and how the application handles output of the ML model which is received from the server. Overall, the process of developing separate components and then combining every component of the project was a rewarding experience.

Team Contributions:

- Houlin He: Developing the image splitting algorithms and constructing the dataset.
- John Ahn: Developed the classification machine learning model.
- Jefferson Zhai: Worked on the android application.
- Ying Qi Wen: Worked on the server.

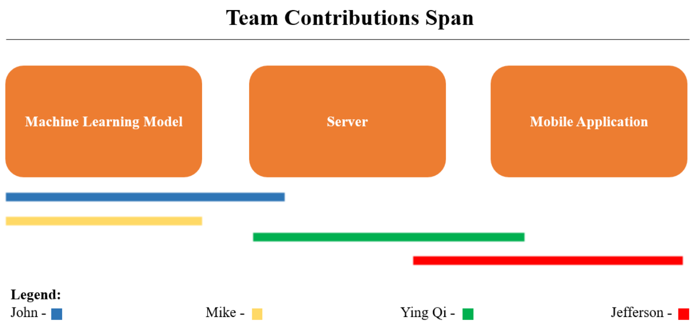


Figure 12: Contribution Span of Team Members: **A longer bar signifies a more diverse work and a shorter bar signifies a more focused work.**

REFERENCE

[1] J. Calvo-Zaragoza and D. Rizo, “End-to-End Neural Optical Music Recognition of Monophonic Scores,” *MDPI*, 11-Apr-2018. [Online]. Available: https://www.mdpi.com/2076-3417/8/4/606.

[2] “The Printed Images of Music Staves (PrIMuS) dataset,” *PrIMuS dataset*. [Online]. Available: https://grfia.dlsi.ua.es/primus/.

[3] B. Khan, “Android Upload Image to Server using Volley Tutorial,” *Simplified Coding*, 25-Oct-2017. [Online]. Available: https://www.simplifiedcoding.net/upload-image-to-server/#Android-Upload-Image-to-Server-using-Volley.

[4] ag5ur, “ag5ur/javax.sound.midi-android,” *GitHub*. [Online]. Available: https://github.com/ag5ur/javax.sound.midi-android

[5] Anonymous. (2021) Supplementary Materials; Project source. Anonymized for the Review. https://github.com/UBC-CPEN291/project-team-leptoceratops

[6] K. Singh, “How To Deploy a Flask Application on an Ubuntu VPS,” *DigitalOcean*, 03-Jul-2013. [Online]. Available: https://www.digitalocean.com/community/tutorials/how-to-deploy-a-flask-application-on-an-ubuntu-vps. [Accessed: 16-May-2021].