Treble in the Sheets: Optical Music Recognition

Christian Foreman
University of Michigan
cjforema@umich.edu

Ashwin Saxena
University of Michigan
ashwinsa@umich.edu

Andrew Scheffer
University of Michigan
drewskis@umich.edu

1. Introduction

Optical Music Recognition (OMR) is an often overlooked field of research that investigates how to systematically read music notation in documents using computer vision. The goal of OMR systems is to accurately and efficiently produce machine-readable versions of sheet music by creating MIDI files, MusicXML files, etc. Optical musical recognition is often underestimated given the fairly small number of characters used in standard music notation; however, the difficulty of recognizing variations in notation (handwritten, unconventional style, etc), the challenge of reconstructing complicated music semantics, and the nearly limitless ways of arranging many notes on a page make this problem an active area of current research [1].

In recent years, the demand for OMR systems has increased dramatically. Recently, initiatives have been launched worldwide to digitize and distribute musical works in an effort to make music and musical education more accessible to everyone. The International Music Score Library Project (IMSLP), for example, is currently the world's primary provider of sheet music worldwide aiming to do just that [2]. Developing a way to not only distribute image scans of sheet music but also to make their structured representations available to the public would yield extraordinary benefits for the musical community. For example, the structured machine representations of musical scores would allow for content-based search techniques, giving conductors and scholars alike a way to find versions of, identify, or determine similarities in pieces of music. Additionally, the digitization of music would also have various applications in educating students about musical notation. These applications (and plenty more) have been envisioned for some time. The development of OMR technologies would decrease the cost and tedium of the mundane tasks of music transcription done by humans today and would also provide everyone with better access to sheet music.

In an attempt to fulfill the need for this high demand, this report presents a simplified optical musical recognition system, modeled off the state-of-the-art OMR pipeline from [1]. Our system takes images of sheet music as input and outputs both an image of the annotated sheet music and a MIDI file containing the machine-readable representation of the sheet music. Our team used the DeepScores V2 dataset [3] in conjunction with an encoder-decoder (U-net) model to perform semantic segmentation. This segmented image was then further processed to reconstruct basic music semantics. This process was evaluated using both qualitative and quantitative test metrics.

2. Approach

2.1 Semantic Segmentation

The first stage in our OMR approach was to implement semantic segmentation to identify and localize the various musical elements in an input image of sheet music. The U-Net is a machine-learning model that is capable of performing this semantic segmentation by assigning each pixel in the input image to a certain class. Because of this, our team decided to use a basic U-Net model for our segmentation task. The specific details of the model we designed are shown in Figure 1. In this model, a 3-channel RGB image is inputted to the encoder, which encodes a feature vector that is then upscaled by the decoder to produce a 13-channel probability map that determines each pixel's "probability" of being in each class (i.e. quarter note, treble clef, etc).

For training, our team preprocessed segmented musical scores from DeepScores [3] to have 13 classes to use as a ground truth. However, we found that the musical notation was so small compared to the size of the image that our model would often predict the entire image as "background." To combat this, our team instead sampled an image window of size 256 x 512 uniformly at random for every training image and used the image window as an input to the U-net instead. This approach not only drastically decreased training time but also improved the accuracy of the model.

Similar to the training scheme outlined by Pacha et. al [1], our team chose the cross-entropy loss function throughout the training process with an adaptive learning rate: starting at 0.001 and multiplied by 0.2 every five epochs for a total of 50 epochs.

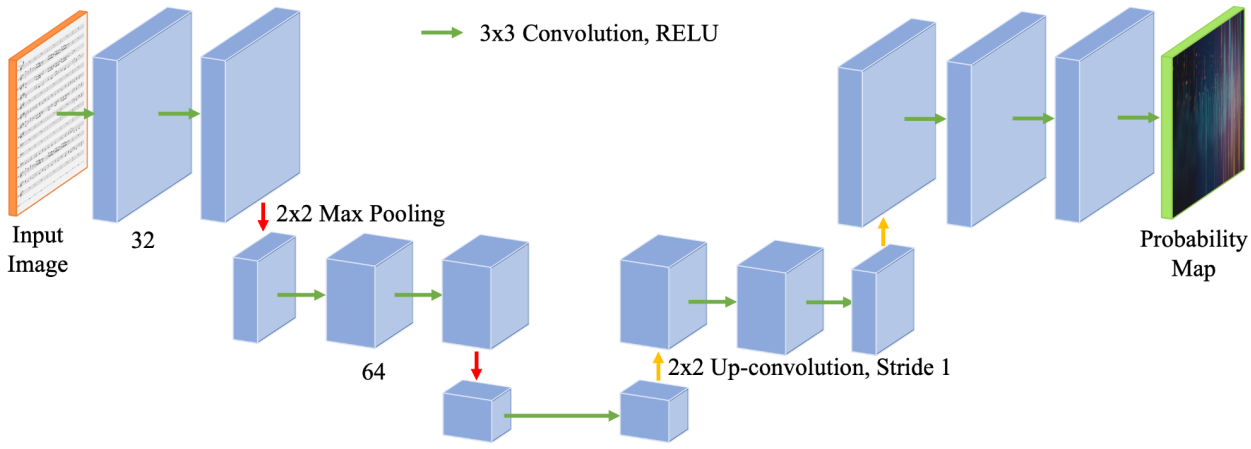

Figure 1: Visual representation of an encoder-decoder U-Net model architecture with computation flowing left to right. Green arrows indicate 2D convolutions with 3 x 3 kernels, red arrows indicate 2 x 2 Max Pooling, and yellow arrows indicate 2 x 2 up-convolution.

2.2. Staff Identification

To determine what note each musical element represented, our computer vision system needed a reliable way to identify the staff lines. To identify the staff lines, we began by doing a vertical scan of the input image and determined the number of black pixels on each row of the image. This process was similar to what Bainbridge and Bell did to get a horizontal projection of an image [4]. Figure 2 shows an example figure obtained after running this vertical scan staff identification on a test image. The rows which had more black pixels than a certain threshold (half the width of the image) were the location of the staff lines. After obtaining the staff lines, the middle line was identified via a simple loop so that a baseline could be established for note placement.

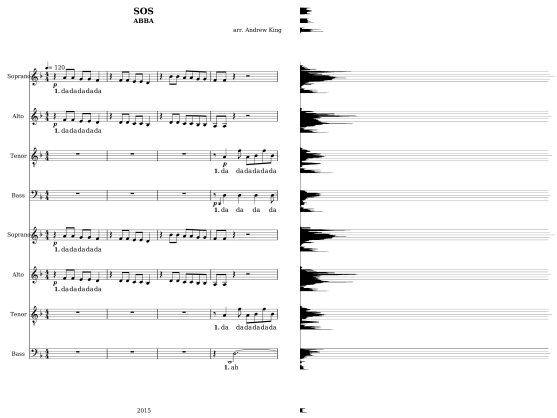

Figure 2: Intermediate result of running vertical scan staff identification to obtain the frequency of black pixels on each row of the image.

The distance between any two lines on the staff was also calculated during the vertical scan to aid us in note placement.

2.3. Note Detection and Placement

After semantic segmentation and staff identification, the next step was to place the recognized notes on the staff to identify their tone. Using the semantic segmentation map as input, our team used basic color filters to separate the solid note blobs from the rest of the image. Once the notes were separated, our team constructed image contours from the note blobs using the "findContours" function from the OpenCV python library. These contours were used to construct bounding boxes (shown in Figure 3) around every solid note blob in the segmented image. Because the segmented image was the same size as the original image, the bounding boxes could easily be applied to the original image.

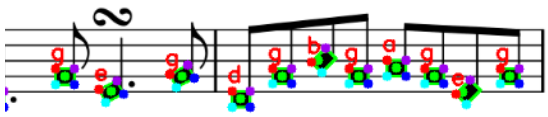

Figure 3: Result of applying note detection and placement to a section of music.

The creation of these bounding boxes for each note was convenient because they allowed for the relatively accurate determination of where the center of each note blob was. These centers are depicted in Figure 3 as the green dots in the image. After the approximate center of each blob was determined, both the section number and

note that it corresponds to could be determined. The section number was found by determining which middle line was closest to the note's row.

To determine the actual tone of each blob (note), the linear distance between the center-point's row and the corresponding section's centerline was calculated. Our team had the realization that every note was a unique integer multiple of ½ of the line spacing from the centerline. Therefore, since the average distance between nearby lines in each section was already calculated previously, all that needed to be done was divide the distance of the blob from the centerline by half of this spacing to get a unique note number.

While this technique works in principle, the issue of pixel precision presented itself through testing. We found that the ½ of the line spacing distance was on the order of a few pixels which made the arithmetic quite inaccurate. To combat this, our team investigated introducing a distinction between "line notes" (notes that go through a line) and "gap notes" (notes that do not go through a line) in the semantic segmentation model. By introducing this distinction, the critical distance between comparable notes was effectively doubled which helped give much more accurate predictions of note names. An example of this pipeline in action can be seen in Figure 3.

From all of this image analysis, the end product of a list of notes can be obtained by looping over every note blob in the image, assigning it a section and note name, then finally sorting the notes in each section based on their center's column number. Currently, this process only works for note blobs which are all treated as the same length; however, with more time, other elements such as rests, eighth notes, etc could easily be added to the music reconstruction algorithm.

2.4. Audio Generation

Once the list of notes is obtained, the notes are individually converted to frequency values based on the corresponding note through a simple for loop and lookup table. Then, using the MIDIUtil python library, the list of frequencies is converted to a .mid file with a constant tempo, duration, and volume [5]. The resulting .mid file can now be played and displayed on commercial audio software like GarageBand.

3. Experiments

3.1. Data

Our team used the DeepScores V2 [2] dataset of electronic sheet music to train and test our OMR pipeline. The DeepScores dataset consisted of 1716 semantically segmented images of sheet music for various instruments with 135 individual classes of segmented musical elements. To use this dataset for our image segmentation task, the number of classes was distilled by recreating the ground truth images using only 13 classes. Focusing on the 13 most important music symbols made training our model faster and allowed us to focus on the symbols that appear in almost every music score.

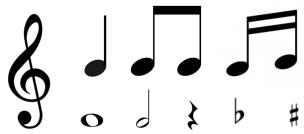

Figure 4: An image of the main symbols we attempted to detect in our model.

3.2. Metrics

Our team used several metrics, both qualitative and quantitative, to determine the effectiveness of our pipeline at performing optical music recognition. Qualitatively, the output of the algorithm (an audio file) was compared to recordings of the music to determine if there were a reasonable amount of similarities. This is reasonable because our model makes many simplifications in transcribing music that make many quantitative methods of analysis infeasible. We also use qualitative analysis as a way of gauging the quality of the segmentation model.

Quantitatively, our team primarily used cross-entropy loss and average precision to evaluate our segmentation model. We decided to use cross-entropy loss because it combines the negative log-likelihood loss and the softmax loss which is very valuable for many-class segmentation tasks. Additionally, the cross-entropy loss function is a widely used loss function for skewed datasets in semantic segmentation [6].

3.3. Qualitative Results

Since our project was done in several different stages, there were several intermediate qualitative results. After running semantic segmentation, we obtained an image where each symbol was colored differently as shown in Figure 5.

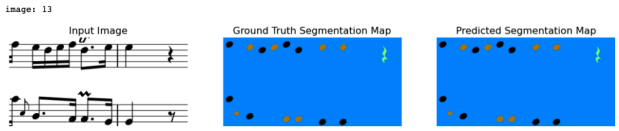

Figure 5: Result of running semantic segmentation on a section of the score

After running through the steps of our model, we obtained a MIDI file which was processed in Garageband so we could hear our results, see Figure 6. We compared the audio we generated for a score against the actual recordings from the composers and found that the melodies we recreated were pretty similar to the actual recording. We did not get to recreate rhythms in this

project, however the sequence of tones matched the original recording almost perfectly.

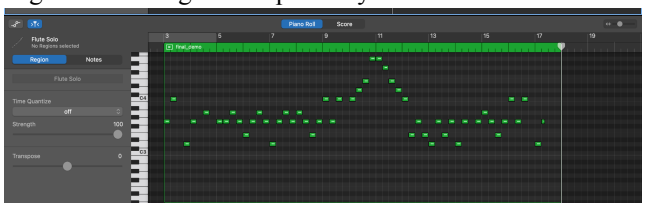

Figure 6: The MIDI file after converting the list of notes to frequencies.

3.4. Quantitative Results

The semantic segmentation created the majority of the quantifiable results for us, which include the average precision for the classes in Figure 6, and the training plot for the U-Net as shown in Figure 7. The average precision across all classes was 0.81 which is almost 10 times better than that of random chance (0.08).

```
AP = 0.9999704959394963 void
AP = 0.9901417477413665 lineQuarterNote
AP = 0.9280797589250609 quarterRest
AP = 0.9118321386950468 trebleClef
AP = 0.9853271008869076 gapQuarterNote
AP = 0.95828370111876   stem
AP = 0.9870761622442161 sixteenthEighthBars
AP = 0.8715664873186116 gapWholeNotes
AP = 0.3165580270091363 lineWholeNotes
AP = 0.801586204948396  gapHalfNotes
AP = 0.8218623941173531 lineHalfNotes
AP = 0.2930063873684463 sharps
AP = 0.6724167775832581 flats
Average Precision (all classes) = 0.81059287568
```

Figure 7: The average precision for all thirteen classes, and the average precision for all classes.

The cross-entropy loss of the model after 50 epochs is almost 0.01 for both the training and the validation sets, signifying the minimized errors in our prediction. We were initially aiming at achieving a goal loss of about 0.05, but the model exceeded our expectations.

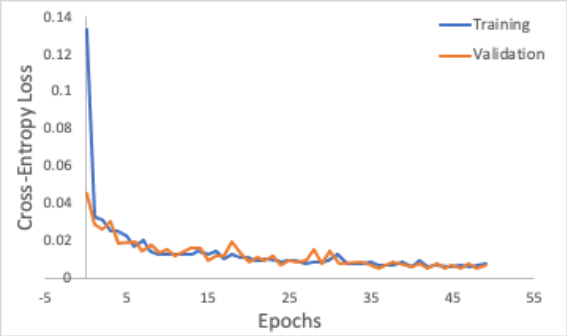

Figure 8: The training plot for the U-Net for 50 epochs depicting an initial sharp decrease in loss followed by a steady decrease of loss to a value of about 0.01

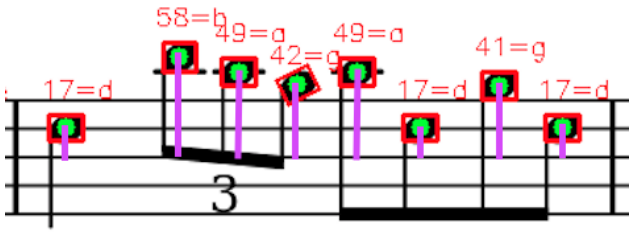

Figure 9: The numbers above represent the number of pixels the symbol is away from the baseline (centerline) of the staff and the corresponding note assignment based on the distance.

4. Implementation

Our team used PyTorch, an open-source deep learning framework, in the implementation of this project. We adapted and referenced the semantic segmentation code provided in HW5 of EECS 442 (Computer Vision) to build the U-Net responsible for classifying music notation. Our idea to use a U-net was inspired by Patch et. al. [1] because they compared various models for semantic segmentation, and the idea to do a vertical scan for staff identification was inspired by Bainbridge and Bell [4]. We used the python OpenCV library as a way to generate filters and contours in images. Finally, our team utilizes the MIDIUtil python library to produce midi files in python. Any additional tasks, including but not limited to image preprocessing, splitting, recombination, staff line identification, note placement, audio generation, and figure creation were work done by ourselves.

5. Conclusion

Our team was able to perform optical music recognition for certain symbols at a high degree of accuracy, however, more work needs to be done to include the limitless ways music can be represented. For example, being able to distinguish between the different lengths of notes (whole, half, eighth, sixteenth, etc.) would allow us to recreate rhythms and have more interesting results to listen to. Moreover, currently, the model only works well with scores formatted in the same way as the DeepScores V2. With a more diverse dataset that synthesizes different sources, and includes handwritten scores, we could add in the ability to convert and recognize virtually any score.

References

[1] A. Pacha, J. Hajič, and J. Calvo-Zaragoza, "A Baseline for General Music Object Detection with Deep Learning," *Applied Sciences*, vol. 8, no. 9, p. 1488, Aug. 2018, doi: 10.3390/app8091488.

[2] "International Music Score Library Project Petrucci Music Library," *IMSLP*. [Online]. Available: https://imslp.org/wiki/Main_Page. [Accessed: 26-Apr-2022].

[3] L. Tuggener, Y. P. Satyawan, A. Pacha, J. Schmidhuber, and T. Stadelmann, "The DeepScoresV2 dataset and benchmark for Music Object Detection," *2020 25th International Conference on Pattern Recognition (ICPR)*, 2021.

[4] Bainbridge, David & Bell, Timothy. (2001). The Challenge of Optical Music Recognition. Computers and the Humanities. 35. 95-121. 10.1023/A:1002485918032.

[5] M. C. Wirt, "Markcwirt/MIDIUtil: A pure python library for creating multi-track MIDI files," *GitHub*. [Online]. Available: https://github.com/MarkCWirt/MIDIUtil. [Accessed: 25-Apr-2022].

[6] S. Jadon, "A survey of loss functions for semantic segmentation," *2020 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB)*, 2020.