Digitizing sheet music

Tal Stramer
Department of Computer Science
Stanford University
Email: tstramer@stanford.edu

***Abstract*—Sheet music is used by millions of people to play music every day. While it is a great tool for learning how to play an instrument and/or song, it lacks many of the benefits of a digital format. Being able to organize, share, and edit a piece of music are powerful tools that a digitized format introduces. A digitized format also introduces the possibility of providing a cheaper, and in some cases better, alternative for students to learn an instrument or song through an automated teacher. The reality is, however, that sheet music will be around for a long time to come, so I built a system capable of digitizing an image of a piece of sheet music, which in turn can be used as the basic building block for a better experience for musicians and aspiring musicians to learn, play, organize, and enjoy music.**

Keywords—music; piano; optical music recognition; sheet music;

I. INTRODUCTION

My system takes an image of a piece of sheet music and converts it into a custom digital format. This format can be passed to a separate system I built capable of playing the song through a computer speaker. The system breaks down into 3 parts: 1) low-level image processing, 2) symbol classification, and 3) semantic understanding. Each part builds on the previous one, and allows for a clean separation of concerns. The first stage breaks the image into individual symbols, which include things like note heads, beam lines, breaks, dot sets, accidentals, staff lines, clef, and many more. The second stage takes the symbols detected in the first stage and classifies them using a supervised machine learning algorithm. The third stage takes the symbols, classifications, and spatial relationships between symbols and creates the final digitized format, which can be used in many different applications, including one that I built for the project that can play the song through the computer speaker.

II. LOW-LEVEL IMAGE PROCESSING

The goal of low-level image processing is to convert the raw image of the piece of sheet music into its constituent symbols. This problem breaks down into several sub-parts.

A. Image binarization

The first step is to convert the image into a binary form. To do this, we use a simple approach of using a the mean of the image as a global threshold for determining whether a pixel is 1 or 0. This works in a contained setting. In a more real-word setting where there are different lighting conditions and noise in the image, locally adaptive thresholding would be preferred. Below is an example section from a piece of sheet music after being converted to binary.

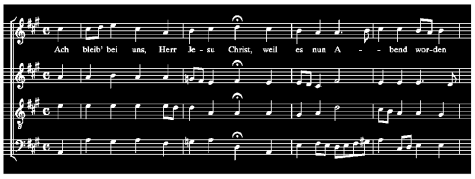

Fig. 1: Binarized sheet music

B. Image orientation detection

The second step is to detect the orientation of the image in order to be able to rotate the image so that it is up-right. This becomes key for later phases of the algorithm, for example detecting staff lines. In order to detect the orientation of the image, we use the hough algorithm to identify the orientation of the dominant lines in the image (in this case the staff lines). The hough algorithm works by converting each active point (x, y) in the image to Hesse normal form, which is a form following the equation:

$$r = x \cos(\Theta) + y \sin(\Theta)$$

The angle Θ corresponds to the orientation of the line and r corresponds to the distance from the origin. Using these values, the hough algorithm finds the pair (r, Θ) that occurs in the most curves defined by the set of input pixels (x, y) in the image. Figure 2 below shows a histogram of orientations and their peaks for a sample input image.

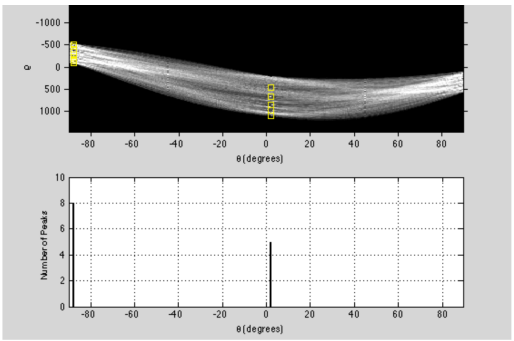

Fig. 2. Hough orientation histogram and peaks

Identify applicable sponsor/s here. If no sponsors, delete this text box (*sponsors*).

Because the staff lines are horizontal in an up-right image, we can then use the angle detected by the hough algorithm to rotate the image so that it is up-right.

C. *Staff-line detection + removal*

The next step is to detect staff lines in the image. To do this, we start by performing a morphological close operation using a 2x2 structuring element in order to fill in any holes in the staff lines. After that, we perform erosion on the staff lines using a 1x49 structuring element. Once the staff lines are detected, we detect the distance between staff lines, which is used in later stages of the image processing algorithm. We then remove the staff lines from the original image. After removing the staff lines, we are left with holes in symbols from the original image that the staff lines intersected with. In order to fill these, we perform a morphological close operation using a structuring element of the form [1; 0; 1]. This detects holes in the image in which the pixels above and below are filled, which are likely to be places where the staff line was.

D. Symbol detection

The final step in the low-level image processing stage is symbol detection. In this stage we detect all the basic symbols in the staff-line removed image. In order to achieve this, we first split the image up into connected components. All non-note connected components then become base symbols. Figure 3 shows a color-coded and numbered representation of the connected components for a sample section from a sheet of music.

Fig. 3: Connected components of sheet music sample

Connected components are further processed in order to detect connected notes, which are then split up into more basic components, like note heads and beams. To do this, we attempt to detect stem lines in each connected component, which is achieved by performing erosion using a vertical structuring element half the size of a group of staff lines (e.g. a row of the sheet of music). This threshold allows us to detect note stems while avoiding false positives from other symbols containing vertical lines, like a sharp. For each connected group of notes found, we further split the component up into sub-components by splitting along each stem line. Then, for each of these sub-components we split the sub-component vertically into smaller sub-components, which isolates stem lines from the note head, and accidentals from beam lines. Figure 4 illustrates this segmentation process:

Figure 4: Horizontal segmentation (left) and vertical segmentation (right)

III. SYMBOL CLASSIFICATION

The goal of symbol classification is to take the symbols generated in phase 1 and label them with their symbol type. To do this, we use supervised machine learning to build a n-class SVM classifier. Specifically, we start with a training set [4] of around 5-50 examples per symbol. Figure 5 shows some example symbols from this training set.

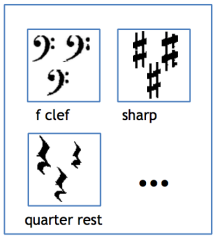

Figure 5: Example training set symbols

For each example, we first resize the image so that it is 20x20. Then, we project the resized image into a lower-dimensional feature space consisting of a histogram of orientation gradients (HoG), which detects gradient orientations in the image. HoG features are extracted from each 4x4 window. We chose 4x4 because it provided a good trade-off between finding granular features in the image and not making the feature space too large. Figure 6 below depicts HoG features for a sample image:

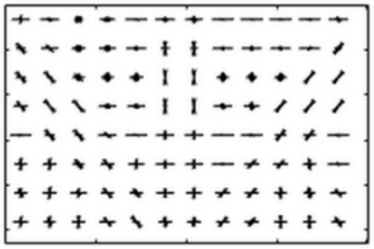

Figure 6: HoG features for a sample image

After we have converted each example symbol into a feature vector, we train a n-class SVM classifier using these feature vectors. This classifier is then used to label the symbols extracted from phase 1.

IV. Semantic Understanding

The last phase of the image processing algorithm, semantic understanding, combines the symbols, classifications, and spatial relationships between symbols into the final digitized format. This stage breaks down into several smaller parts.

A. Section segmentation

Section segmentation splits the piece of sheet music into distinct sections, which consist of a row of the song containing a key signature and clef. In order to split the song up into sections, we match each symbol to its closest staff line. Then for each group of 5 staff lines, we consider all symbols closest to it part of the same section. After grouping symbols by section, we sort all the symbol in a section by x coordinate which is necessary for later stages of semantic understanding.

B. Clef detection

Clef detection is concerned with detecting the clef, either base or trebel, of a section. Clef detection is achieved for each section by looking for a symbol classified in stage 2 as a treble or base clef.

C. Key signature detection

Key signature detection is achieved for each section by counting the number of sharps and flats to the left of the notes section. This is done based on the symbol classifications in stage 2.

D. Note position + length detection

Using the size of the gap between staff lines and the position of the note head, we can determine the note value. For length detection, we use a combination of symbol classifications and information about connected components extracted from 1. Specifically, if a note is connected to a stem that is classified as an 8th note stem, we classify it as an 8th note. If a note is connected to other notes by an 8th note beam, we classify it as a 8th note. Similarly, if a note is connected to other notes by a 16th note beam, we classify it as a 16th note. To distinguish between quarter notes and half notes, we use a hit-miss filter to detect whether the note head as a hole in it. To detect a dot set, which extends the length of the note by 3/2, we look at the surrounding context of the note and try to find a dot on the same level and to the right of the note.

E. Note + accidental grouping

In order to detect sharps, flats, and naturals, we look at the surrounding context of each note for symbols labeled with these classifications. In addition, we require that the symbol be on the same level and to the left of the note.

F. Break detection

Break detection works by going through the symbols in a section and finding symbols classified in section 2 as a break, which can be either an 8th note break or quarter note break.

V. Experiment Results

Using 5 songs as a test set [5], we found that 95% of notes were classified correctly, 92% of note lengths were classified correctly, 100% of clefs and key signatures were classified correctly, and 95% of breaks were classified correctly.

References

1. *Bellini, Pierfrancesco, Ivan Bruno, and Paolo Nesi. "An OffLine Optical Music Sheet. Recognition." Visual Perception of Music Notation: On-Line and OffLine Recognition. Hershey, Pennsylvania: Idea Group (2004): 4077.*
2. *Bellini, Pierfrancesco, Ivan Bruno, and Paolo Nesi. "Assessing optical music recognition. Computer Music Journal 31.1 (2007): 6893.*
3. *Bainbridge, David, and Tim Bell. "The challenge of optical music recognition."Computers and Computers and the Humanities 35.2 (2001): 95121.*
4. *Retrieved from https://github.com/acieroid/overscore/tree/master/training-set*
5. *Retrieved from http://scores.ccarh.org/bach/chorale/chorales.pdf*