CSCI-B657
Assignment 1: Image Processing and Recognition Basics

Tejashree Khot
Vijay Hareesh Yadav Avula
Suhas Jagadish

February 9, 2016

1. Implement a function that convolves a greyscale image I with an arbitrary two-dimensional kernel H. Make sure your code handles image boundaries in some reasonable way

   - We have implemented the convolution function by Brute force method, with regular 4 for-loops.
   - Image boundaries have been handled by padding 0's to the input image matrix.
   - The run-time of this implementation is $O(n^2k^2)$ where n is the size of the image matrix and r is the size of the kernel matrix.

2. Implement a function that convolves a greyscale image I with a separable kernel H. Implement efficient convolution by using two convolution passes (with h_x and then h_y), as we discussed in class. Make sure your code handles image boundaries in some reasonable way

   - The convolution separable was obtained using two vectors, In our case, we have implemented it using one row matrix and one column matrix.
   - First, we have convolved the input matrix with the column matrix for which the edges were handled by taking half the rows of the template image above and half the rows of template image below the input image.
   - Similarly, the output of this is taken as input for convolving with the row matrix and here the edges were handled by adding half of template columns on either side of the input image. This reduces the computation steps significantly.
   - The run-time of this implementation is $O(a + b)$ where a is the complexity for convolution of the image with a row matrix ($O(n^3)$) and b is the complexity for the convolution of the image with the column matrix($O(n^3)$).

3. Implement a routine to detect a given template by doing the convolution(Hamming distance). When a note is detected, it should also estimate the pitch of the note (i.e. letter between A and G).

   - We started by translating the input image and template image to binary using our "to_binary_image" function.
   - We defined a similar function which does 1-binary(inverse function) using our "to_binary_image_minus1" function. This is required for calculating second part of the hamming distance formula.
   - We then created a *transpose* of our template images for both the above functions. Lets say we named them "temp_bin_transpose" and "temp_bin_inv_transpose".
   - Now we convolve our input binary image with "temp_bin_transpose". This essentially gives us the first part of the output. We then convolve inverse input binary image with "temp_bin_inv_transpose", which gives us the second part of the output.

- The output, or in other words, the hamming score is the summation of the first and second part.
- However this resulted is too much wastage of memory and we ran into memory dump error.
- So we decided to change the functions for simplicity where a pixel in the input image and its corresponding pixel in the template image are either greater than 128 or both less than 128, then they match. Hamming score is incremented when there is a match. This is a much better approach and very efficient in terms of memory utilization.
- Once we have the similarity score, we check to see if the point exceeds a defined threshold value, only then we select that point. However this leads to a problem of overlapping detection.
- Hence we select only the best match within the overlaps. That is, we check the surrounding pixels of every pixel above the threshold and if this pixel has a neighbor pixel with a better score than itself, then we disregard the pixel. We will only select the pixel with best score among all neighbors. This process is called "*non-maximal suppression*". We employed the same approach in the next problem as well.
- The result of convolution after this step is split across 3 image files - "scores41.png", "scores42.png" and "scores43.png" for the notes, eight and the quarter rests.

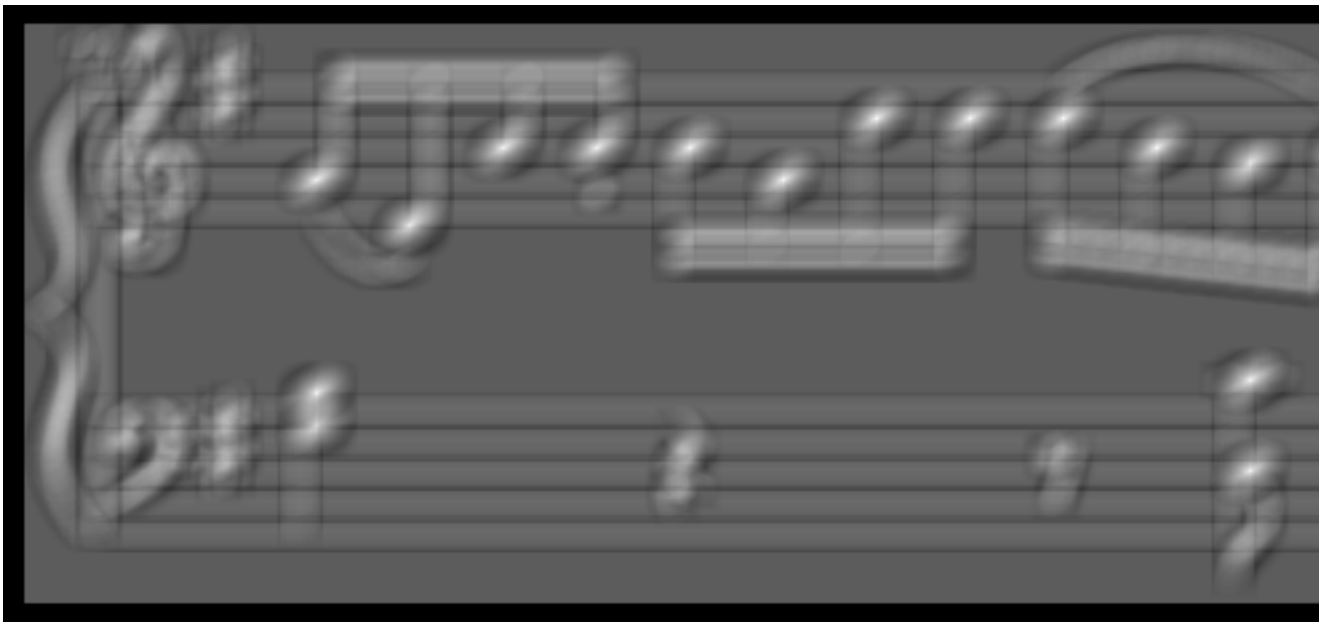


Figure a. Notes

Figure b. Eight rest

Figure c. Quarter rest

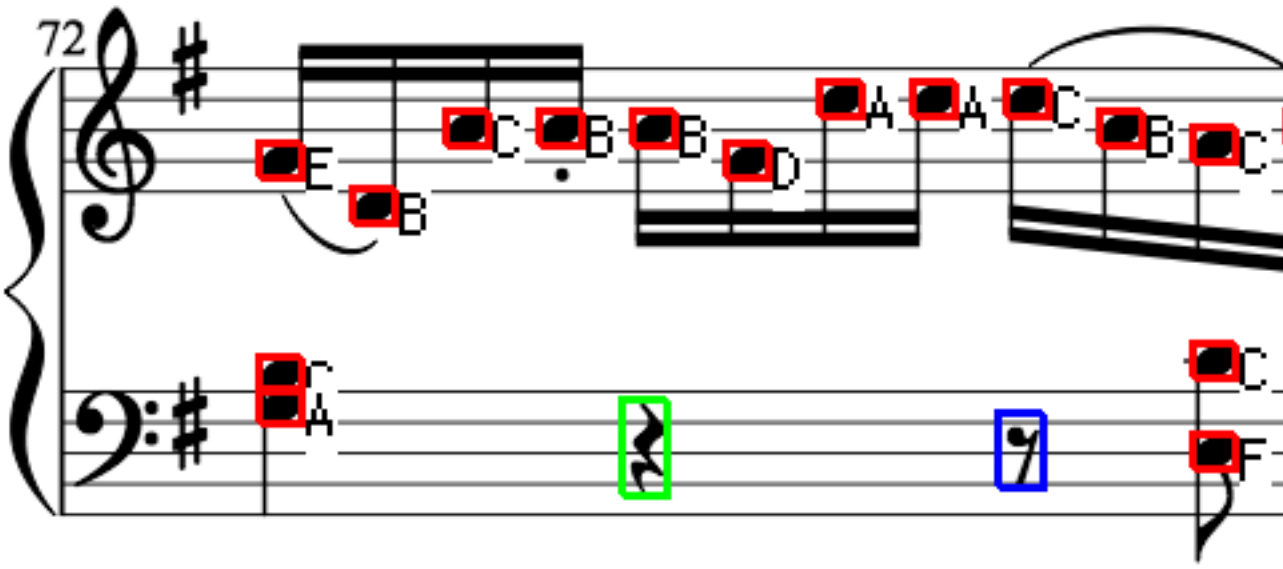


Figure d. Detected symbols after this step

4. Run an edge detector on the template and the input image. Then, implement a version of template matching that uses the scoring function.

   - We use Sobel operator for detecting edges in our input image. We first pick the horizontal and vertical sobel gradient filters and convolve them with our input image.
   - When we combine the convolution results obtained above(horizontal and vertical edges), we will get an edge detected image. We have computed the sum of squares of convolution results, to eliminate negative values, if any.
   - The resulting edge detected image is taken and made binary within this function.
   - The dynamic programming approach to reduce the time complexity was handled using two for loops with chess board distance implemented (idea taken from Cornell Vision CS664 class of Distance Transformation while code was written through understanding).
   - The Distance matrix thus obtained is multiplied with the template according to the description given to form a matrix 'match'.
   - However, this matrix is normalized to find out the individual peaks to label as the music notes.
   - The normalization technique was such as taking the maximum of the obtained match matrix and individually defining the threshold to detect the notes.
   - Various thresholds were tested like with a minimum bound, maximum bound, range of the boundary i.e., lying in between boundaries.
   - As per the instructions, in $\gamma(.)$, the pixel should be initiated to ∞ for a non-edged pixel. So we initialized to a double maximum value.
   - We then realized we have to add some value to the function later on so we couldn't use the double maximum value(math had to be performed).
   - Hence we reduced the $\gamma(.)$ to arbitrarily large value of 10000, which gave us better distance from the edge function.
   - On further reducing this value, we kept on getting better results and we achived our best result when $\gamma(.)$ was set to 1. Hence we inverted the original edge map and it gave us the best results.

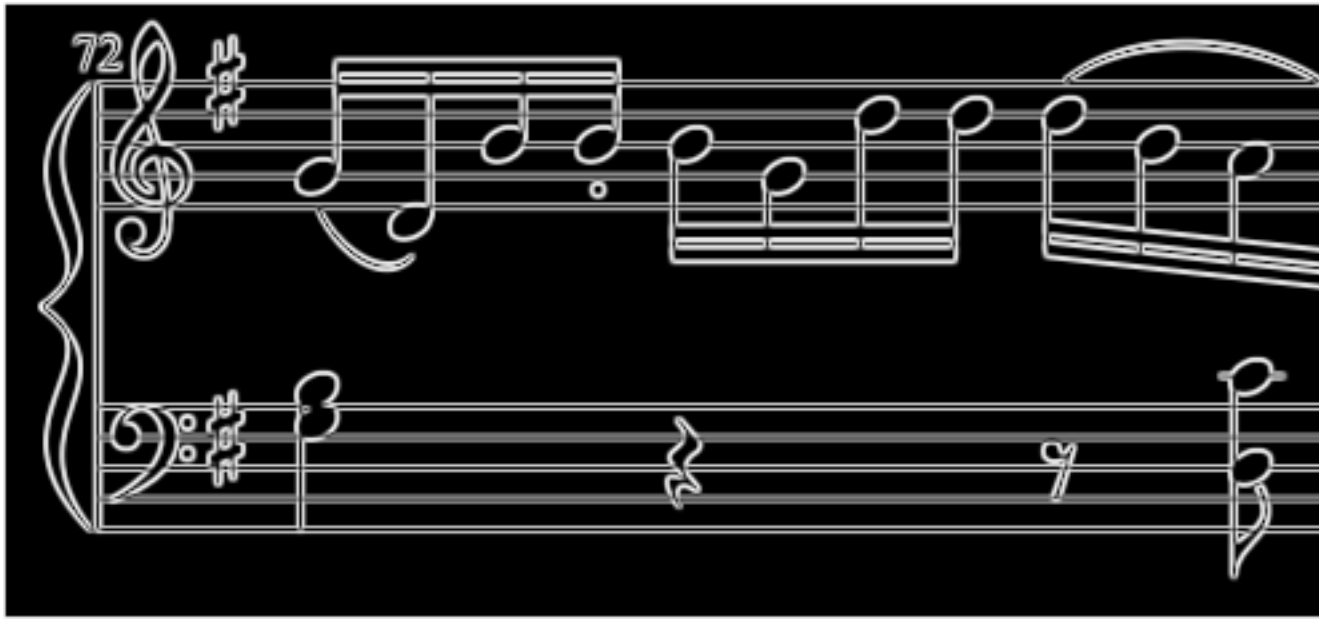


Figure a. Image edge map

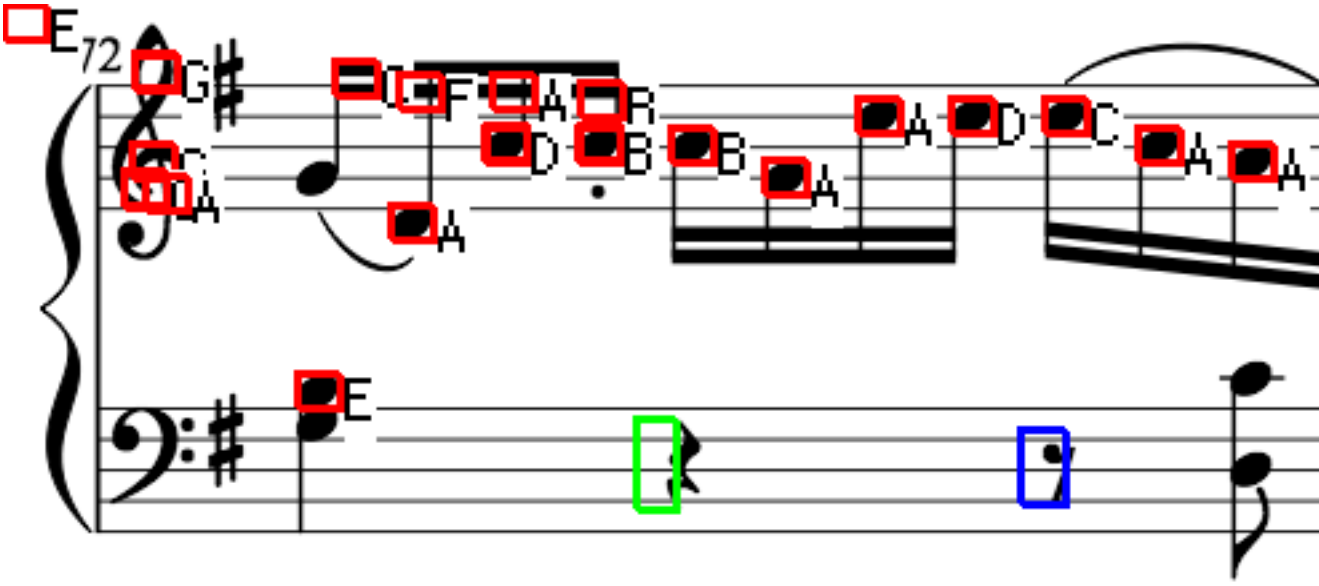


Figure b. Detected symbols after this step

5. Apply the Hough transform to find the groups of five equally-spaced lines.

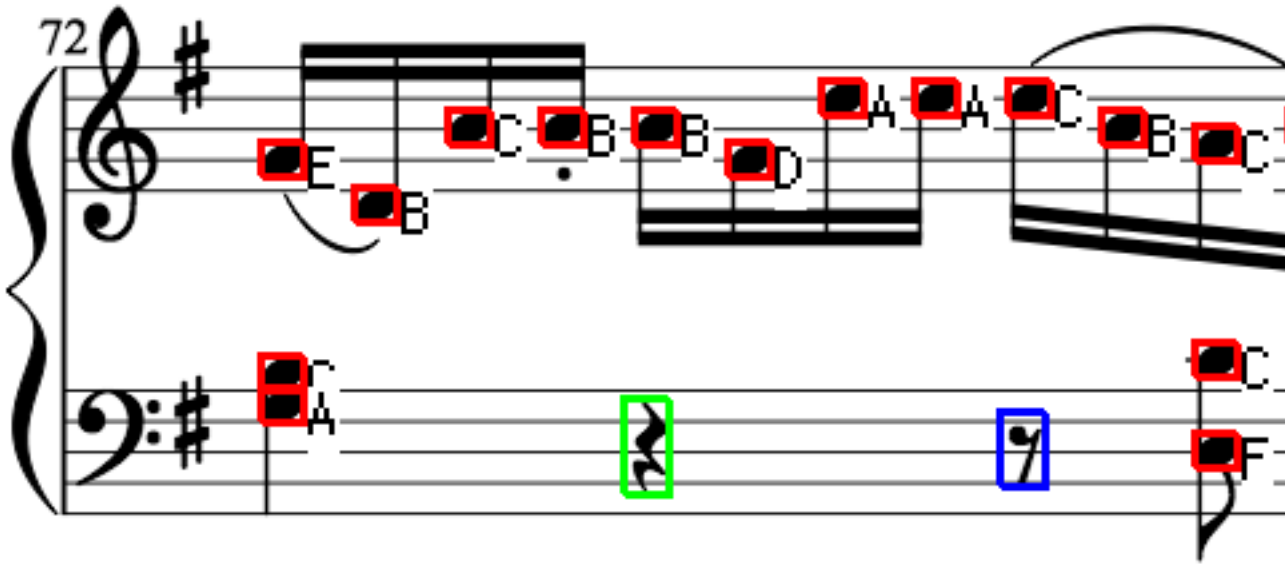
   - We first tried the standard hough approach to find points in r and Θ space and map it to a line in the image space. We were successful in finding the lines but had trouble in finding the distance between the horizontal lines.
   - As a second approach, we used only binary of an edge image and tried to implement an algorithm which looks vaguely like,
     if(pixel[x][y]>0), then

temp = pixel[x][y] + pixel[x][y+1*space] + pixel[x][y+2*space] + pixel[x][y+3*space] + pixel[x][y+4*space];
Accumulator_array[y,space]++;

- This accumulator array of thresholding should have ideally given set of 5 lines.
- However we couldn't implement it exactly how we wanted to. Instead, we implemented code that uses accumulator array concept to give single horizontal lines only. This code works only for lines like staves which are considerably long.

6. Combine the above techniques together to implement a simple OMR system.

   - We have used the approach of step3 as our best solution to detect the notes and eight and quarter rests in the image.



Detected symbols by following the approach used in step3

   - We were able to detect the staves(five equally-spaced lines) using Hough transform method explained above. The image below represents the staves for "music4.png"

Detection of staves for music4.png

- We implemented note detection using brute force approach. We were able to successfully detect few notes but not all were at the accurate positions. We take the relative distances using the hough transform and the code to implemented this can be found in the main() function.
- We haven't rescaled the image and hence the note detection will not be very accurate.

7. Accuracy report:

   - Mean average precision for filled_note = 1.00
     Mean average precision for eight_rest = 1.00
     Mean average precision for quarter_rest = 0.00 (We have a question regarding this accuracy as the quarter rest nodes are matching pretty well in the detected image.)
     Overall Mean average precision = 0.67
     Fraction of correct note_head detections with correct pitch = 0.167

8. Extra:

   - In order to run the code, you need to copy all the contents from the Master branch of our repository(vavula-jagadiss-tpkhot-a1) to the CS Linux machines and execute the 2 commands-

make
./omr music1.png

- The design design decisions and any assumptions made are explained in detail under each problem section.
- We have tried to maintain the efficiency of our code by implementing dynamic programming and reducing the number of loops wherever possible. However we hit few roadblocks in our hough transformation implementation due to which we were unable to rescale the image. Our project could be improved with a better implementation of hough transform.