CSCI-B 657

Computer Vision

A1

Spring 2016

Indiana University Bloomington

Anand Saurabh Sharma | asaurabh@indiana.edu
Ghanshyam Malu | gmalu@indiana.edu
Rohit Nair | ronair@indiana.edu

February 9, 2016

Running the code:

make

./omr <image.png>

1.

We ran the code on tank.soic.indiana.edu using make file.

2.

Convolution has been implemented using brute force convolution methodology with a 2D filter. In this method only one pass is used for convolution. The time complexity of the methodology is $O(n^2k^2)$ where *n* is the rows/columns of input image and k is the rows/columns of the square filter matrix.

Two helper functions have been used:

- make_mean_filter: A nested loop giving each pixel of the filter a value of (1/(n*n)), where n is the filter dimension
- convolve_general:

    a. Convolve the image with the passed template except the boundary values.
    b. Boundary pixels do not have anything outside image dimensions for the convolution operation. So, we use a mirror reflection function called reflect, which returns the mirror value of the non-existent pixel coming into picture when we try to access the row or column or both of the image. For instance, reflect(image[-1][5]) will be image[0][5] and can be generalized for higher distance reflections easily.

    Below: i and j loop through the rows and columns of the image, m,n loop through the filter and r,c are rows and cols of the image

```
a. Top   : output[0][j] = sum(filter[m + 1][n + 1] * input[reflect(0 - m, r)][reflect(j - n, c)])
b. Bottom: output[r - 1][j] = sum(filter[m + 1][n + 1] * input[reflect(r-1-m, r)][reflect(j - n, c)])
c. Left  : output[i][0] = sum(filter[m + 1][n + 1] * input[reflect(i - m, r)][reflect(0 - n, c)])
d. Right : output[i][c - 1] = sum(filter[m + 1][n + 1] * input[reflect(i - m, r)][reflect(c-1-n, c)])
```

3.

In this problem we implemented convolution using separable filter by performing two 1D passes.

1. First filter is a column vector of size 3 and the second filter is a row matrix of size 3 Rows. Since convolution is associative this speeds up the process, hence the total running time for separable convolution is $O(n^2k)$.
2. Boundary cases are handled by ignoring half the length of the filter size in the image. For our case of filter size 3, ignoring one pixel on each edge of the image

4.

We performed the following steps for detecting the musical symbols.

Steps 1 - 2 are done for each and every template respectively:

1. Convolute the input and template images using a separable kernel. Convolution is done to blur the image so as to smear the transition between edges.
2. Using a threshold of 200 for every pixel we converted the input image and each template to binary. Pixels above 200 are set a value of 1 and pixels below are set a value of 0.
3. Calculate hamming distance between input image and the template, the output is a matrix of hamming distances for every pixel.
4. Every value greater than product of confidence threshold (90 for Notehead and 0.95 for other symbols) and the maximum pixel value in hamming distance matrix is considered as the topmost pixel of the musical symbol. The surrounding pixels equal to the dimensions of the template are then marked with a negative number. This is done to mark the symbol as detected so that it is not detected repeatedly.

Following steps are done for detecting the pitch of Notehead:

1. Line Detection: For each row, we average the values of 5 columns. If the average is below a confidence threshold we assume that the row is a Staff line.
2. Assigning pitch: A pitch value is assigned based on the position of the Notehead relative to the staff lines. Also, special care is taken to differentiate the higher pitch tones from the low pitch ones.

5.

Template matching scoring function is performed on the input and template images using edge maps as described below.

1. Convoluted the input image with Gaussian Kernel
2. Apply horizontal and vertical Sobel filter of size 3x3 to the input image individually.
3. Find magnitude using sqrt(Gx^2+Gy^2)
4. Converting it to binary gives us the edge map

We tried few other different approaches for edge detection with Sobel operator (applies to the input images as well as the templates)

1. Applied Sobel operator to the input image and then converted it to black and white to make the edges stand out
2. Applied Sobel operator on a blurred image and converted it to binary
3. Many online sources specified that it is optimal to apply Sobel on the blurred binary of the image
4. The distance function worked best with the third approach which is why we have selected the same

Also, we have tried edge thinning because thick edges and multiple staff lines nearby disrupted the staff line detection algorithm used later in the code

Calculate Distance matrix D

- Calculating Distance Matrix by a naive implementation was very expensive since it was of the order O(N4) and we were wasting 20 odd minutes each time we ran the code. Luckily we got an alternate idea to find this in $O(n^2)$ from an online resource[1]
- Initialize edges as 0 and others as infinity (10k) in the image edge map after the Sobel operation.
- We first calculate the minimum distance of each pixel with respect to the right pixel and the pixel below it. We do another run to calculate minimum distance of each pixel with respect to the left pixel and the pixel above it.
- This leaves out many pixels on the bottom and right boundaries, so we handle it separately. For the bottom row, we compare the pixel with the pixels to the left and below it and for the rightmost column, we compare the pixel with the pixels to the right and above it.

Calculate score function 'f'

- Score of each pixel is calculated using the provided formula i.e., sum of binary_template[k][l] * D[i+k][j+l]
- Also, tweaked threshold value differently for each template by trial and error
- Saved the results in HammingDistances objects and found the symbols using the method mentioned above (using the class HammingDistances, so that we didn't have to write a separate one)

6 & 7.

Hough transform:

We use a sobel operator to detect the edges from the input image. The image is then processed using Hough transformation detect the straight lines, there by detecting the staves in the input image.

- Every edge pixel from the input image is transformed into a different space where this pixel/point is represented by a line.
- An accumulator matrix is maintained to track the votes of every pixel.
- The peaks from this matrix represent the edges/staves in the original space.

Once we get the staves, the average distance between every staff is life is computed to get the scaling factor by comparing with the NOTEHEAD template.

The input image is then resized/rescaled using the Sub-Sampling technique. We implemented scaling using Sub-Sampling technique. Following are the steps involved:

- We create two sets of unique random numbers with sizes equal to the newHeight and and newWidth respectively. Random numbers are generated in the range of height and width of the original image.
- Resized image is created by selecting only the pixels that are present in the sets.

The newly scaled image is then processed using the Step 4 or Step 5 to obtain the detected symbols.

Results

Below are the results of algorithm for music1.png

- Mean average precision for filled_note = 0.758336
- Mean average precision for eighth_rest = 1.000000
- Mean average precision for quarter_rest = 0.000000
- OVERALL Mean average precision = 0.586112

The results are better for music1.png compared to music4.png. One of the reason is that music1.png is comparatively noise free.

Future Improvements:

- The edge detection that we employ gives thick edges. We did try thinning them and we were successful to a large extent, but there is still a margin of improvement.
- We use magic numbers as thresholds at multiple places depending on the type of template. We can improve it by having a better detection logic which requires lesser parameters.
- We could improve our line detection and scaling algorithms. Currently we've implemented an algorithm which does only downscaling and we plan to extend it to upscaling as well.

References.

- Q4. Discussed with Suhas Gulur about the hamming distance approach
- [1] Discussed with Suhas Gulur and referred slides by Dan Huttenlocher, Cornell University (http://www.cs.cornell.edu/courses/cs664/2008sp/handouts/cs664-7-dtrans.pdf)
- Q6. Discussed with Nihar Khetan and referred the lecture of Prof. William Hoff, Colorado School of Mines, Engineering Division (https://www.youtube.com/watch?v=o-n7NoLArcs and http://goo.gl/TxbGWi)