Lab 3 - Character Recognition via Thinning

John Lawler

September 29, 2020

INTRODUCTION

The foundation of Lab 3 stems from the results and code used for Lab 2 in character optical recognition. Lab 3 utilizes the MSF (matched-spatial filter) image found in Lab 2 for endpoint and branchpoint detection on each character provided from the ground truth file. For this program, the original MSF image provided from Lab 2 is thresholded at varying thresholds to generate a binary image. Students implement the method of thinning an image to break down the pixels of the binary image until it cannot be thinned further. Once the image is fully thinned, the image is analyzed again to mark the amount of branchpoints and endpoints contained in the image. The amount of branchpoints and endpoints will determine whether the letter is actually the letter of interest or not. This report details the functionality of my program: how the thinning process is implemented and how the true positives and false positives are acquired via analysis of the outputting thinned image.

FUNCTIONALITY

The necessary files are first scanned into the program to be read and handled: the msf image provided by Lab 2, the original text image parenthood.ppm, the ground truth file, and the template file. The ground truth file provides the row and column of the center pixel of each letter to test for. The msf image is then checked at the ground truth location for a detection; once a letter is detected, a 9 column by 15 row square pixel template of that location (the center pixel containing the ground truth location) is copied to a temporary pointer. This image is thresholded again at a value of 128 to generate the binary image to be thinned. If the value of the image is greater than 128, then the resultant binary image will display a value of 255 (white), or 0 (black) if the specified pixel is less than 128.

Once the binary image has been generated, A second binary image is created with an outlying border of 255 (white) pixel values (this helps later on during the thinning process). The temporary binary image shown in Figure 1 is 17 rows by 11 columns pixels wide. By adding a border around the 9 col by 15 row binary image, the for loop parameters for iterations must be adjusted. Beforehand, the original binary image data began at row 0 and column 0 (index of [0]). After adjusting to the binary temporary array, the starting data now begins on row 1 column 1 (indexing shown in the picture below), extending to column index 9 and row index 15 of the temporary binary image.

```
// set bin temp
for (r = 0; r < TROWS; r++)
    for (c = 0; c < TCOLS; c++){
        binTemp[(TCOLS+2)*(r+1) + (c+1)] = binImage[TCOLS*r + c];
    }
```

Setting Temporary Binary Image

An overview of the loops before detailing image thinning: the aforementioned and following steps are performed on every letter found in the image. The ground truth file will be scanned a total of 1262 times until the overall while loop finishes, and this loop is encased within a for-loop that performs the process at different thresholds. Within the while loop iterating through the ground truth file, once a letter is detected, its sub-image is copied from the original image to generate the binary image. Two for-loops will loop through the binary image that checks each pixel on whether or not it may be erased (thinned). These for-loops are placed within a while loop that checks whether the image has been fully thinned or not; if the image passes through the double for-loops and no erasures are made, then the binary image has been fully thinned.

The binary image is thinned in a series of 3 tasks that rinse and repeat until the image meets requirements. To check whether or not a specified edge pixel may be sliced off (erased/set equal to 255), a 3 by 3 temporary array of the surrounded pixels (with the pixel at interest in its center) is used. The surrounding pixels are analyzed clockwise around the center pixel for edge (pixel value of 0) to non-edge (255) transitions. If there is exactly 1 transition, then the next parameter proceeds to be checked. The neighboring edge pixels of the center pixel are checked to be within values 2 and 6, inclusive. If that holds true, then the left, right, top, and bottom sides of the center pixel are checked for edges. If the pixel value above, or to the right, or both the left bottom (i.e. N or E or (S and W)) contain non edge pixels (value of 255) then all checks pass and that pixel of interest is erased.

Once the pixel is marked to be erased, a count of erased pixels is kept for each pass through of the temporary binary image, and the original binary image's pixel is erased. The original binary image was adjusted to add a white border to simply indexing when parsing the image to generate the 3 by 3 array on every pixel. However, this may be a cause to the original pixel being off by one row and column from the original binary image. This is adjusted back properly when erasing the pixel. Once the image has been thinned for one pass-through, the original binary image storing the erased pixels is copied back to the temporary binary image and the thinning process starts again, until zero erasures.

```
} /* if detected */

if (detected == 1) {
   if (endpt == 1 && branchpt == 1 && gt_letter == 'e'){ TP++;
   }else if (endpt != 1 || branchpt != 1){
       if (gt_letter != 'e') FP++;
       else TP++;
   } else if (gt_letter != 'e') { FP++;
   } else TN++;


} else if (gt_letter == 'e') { FN++;
} else if (gt_letter != 'e') { TN++;
}
```

Checking for TP and FP logic implementation

The illustration above displays the logic behind determining the true and false positive and negative values corresponding to each threshold the thinning is performed at. Once the image has been finally thinned, the image is parsed through again using a temporary 3 x 3 array to check for surrounding edge pixel transitions. Endpoints and branchpoints are determined by the number of edge to non-edge transitions: exactly 1 transition is counted as an endpoint and 3 or more transitions is counted as a branchpoint.

DISCUSSION OF RESULTS

The output of my program is shown in Figure 1; a threshold value range of 190 to 220 was chosen to thin the image at each consecutive value (i.e. 30 loop iterations). The ROC curve is depicted in Figure 4 and the data plotted is shown in Figure 5. Shown in Figure 2 and 3 are examples of unthinned and thinned binary images. These images are actually off-center by 1 column; this error caused quite a headache. The ground truth column always seemed to be off-centered one to the right; this, however, did not effect the outcome of the thinned image greatly, but did effect the resulting endpoint and branchpoint detection. I adjusted for this discrepancy by increasing the found ground truth column by 1 prior to entering the conditional for “if detected = 1”. This effect might be a workaround to an issue caused by oversetting bounds with the temporary binary image having the extra border. Figure 6 represents a thinned binary image that is properly centered after adjustment–however I cannot explain the origination of this issue.

Another main obstacle dealt with fixing unending loops; the main culprit being that I, at first, performed thinning on every pixel (edge and non-edge). This caused a never-ending loop as the image never reached zero erasures. To fix this issue, a conditional was set to verify, prior to transition checking, that the pixel was indeed an edge (value of 0).

Shown in Figure 3, is an example of a single branchpoint and endpoint in the letter of interest to detect. Since the letter “e” possesses only one branchpoint (a pixel having more than

two edge transitions in surrounding pixels) and only one endpoint (contains exactly 1 edge transition), the true positive detection is based on branchpoint and endpoint detection. True positives, displayed as data in Figure 5, are counted only if the thinned image contains exactly 1 branchpoint and one endpoint and the ground truth letter is in fact an “e”. Figure 5 also displays optimal and suboptimal values for false positives and negatives. The optimal range of thresholds to detect and verify “e” would be within 209 to 214, about at 213, with 49 false positives and 139 true positives. Depending on the goal of detection, the optimal value would vary according to whether the user is looking to minimize false positives or false negatives. Choosing lower thresholds optimize detecting every possible “e”; however, it is also at the expense of also detecting many letters that are not “e”. Higher threshold values optimize for certainty in letter detection; if an “e” is detected there is a higher percent chance that that letter is an “e”. However, high thresholds also yield high false negatives and many “e”s may go undetected.

CONCLUSION

This lab demonstrates verification in letter recognition using branchpoint and endpoint detection; the resultant true and false positive and negative values displaying how varied recognition is when varying thresholds. Branchpoint and endpoint detection also are not perfect algorithms; as shown prior, before I adjusted the ground truth column location, this can lead to varied counts of endpoints and branchpoints. What if the 9 by 15 area surrounding the ground truth location picked up a few pixels that are not attached at all to the letter of interest? They might still be counted, but are not connected to the actual letter in the template image. Image and letter detection may always display forms of slight discrepancies here or there, and varying the threshold values help to account for that.

DATA TABLES AND FIGURES

```
lawler6@DESKTOP-66UA8JA:~/git/clemson/4310-vision/labs/3$ gcc -Wall -o run lawler6-Lab3.c
lawler6@DESKTOP-66UA8JA:~/git/clemson/4310-vision/labs/3$ ./run
 Threshold: 190 TP: 151         FP: 215         TN: 896         FN: 0
 Threshold: 191 TP: 151         FP: 200         TN: 911         FN: 0
 Threshold: 192 TP: 151         FP: 187         TN: 924         FN: 0
 Threshold: 193 TP: 151         FP: 180         TN: 931         FN: 0
 Threshold: 194 TP: 151         FP: 169         TN: 942         FN: 0
 Threshold: 195 TP: 151         FP: 162         TN: 949         FN: 0
 Threshold: 196 TP: 151         FP: 148         TN: 963         FN: 0
 Threshold: 197 TP: 151         FP: 141         TN: 970         FN: 0
 Threshold: 198 TP: 151         FP: 133         TN: 978         FN: 0
 Threshold: 199 TP: 151         FP: 124         TN: 987         FN: 0
 Threshold: 200 TP: 150         FP: 113         TN: 998         FN: 1
 Threshold: 201 TP: 150         FP: 108         TN: 1003        FN: 1
 Threshold: 202 TP: 150         FP: 101         TN: 1010        FN: 1
 Threshold: 203 TP: 149         FP: 90          TN: 1021        FN: 2
 Threshold: 204 TP: 148         FP: 82          TN: 1029        FN: 3
 Threshold: 205 TP: 147         FP: 75          TN: 1036        FN: 4
 Threshold: 206 TP: 146         FP: 69          TN: 1042        FN: 5
 Threshold: 207 TP: 145         FP: 65          TN: 1046        FN: 6
 Threshold: 208 TP: 144         FP: 64          TN: 1047        FN: 7
 Threshold: 209 TP: 143         FP: 59          TN: 1052        FN: 8
 Threshold: 210 TP: 142         FP: 54          TN: 1057        FN: 9
 Threshold: 211 TP: 142         FP: 52          TN: 1059        FN: 9
 Threshold: 212 TP: 140         FP: 48          TN: 1063        FN: 11
 Threshold: 213 TP: 139         FP: 45          TN: 1066        FN: 12
 Threshold: 214 TP: 133         FP: 43          TN: 1068        FN: 18
 Threshold: 215 TP: 133         FP: 42          TN: 1069        FN: 18
 Threshold: 216 TP: 130         FP: 40          TN: 1071        FN: 21
 Threshold: 217 TP: 128         FP: 37          TN: 1074        FN: 23
 Threshold: 218 TP: 122         FP: 34          TN: 1077        FN: 29
 Threshold: 219 TP: 121         FP: 30          TN: 1081        FN: 30
 Threshold: 220 TP: 118         FP: 25          TN: 1086        FN: 33
 Threshold: 221 TP: 115         FP: 20          TN: 1091        FN: 36
 Threshold: 222 TP: 111         FP: 17          TN: 1094        FN: 40
 Threshold: 223 TP: 104         FP: 16          TN: 1095        FN: 47
 Threshold: 224 TP: 99   FP: 14         TN: 1097        FN: 52
 Threshold: 225 TP: 98   FP: 10         TN: 1101        FN: 53
 Threshold: 226 TP: 93   FP: 7   TN: 1104        FN: 58
 Threshold: 227 TP: 89   FP: 7   TN: 1104        FN: 62
 Threshold: 228 TP: 81   FP: 5   TN: 1106        FN: 70
 Threshold: 229 TP: 75   FP: 5   TN: 1106        FN: 76
 Threshold: 230 TP: 68   FP: 4   TN: 1107        FN: 83
lawler6@DESKTOP-66UA8JA:~/git/clemson/4310-vision/labs/3$
```

Figure 1: Terminal Output

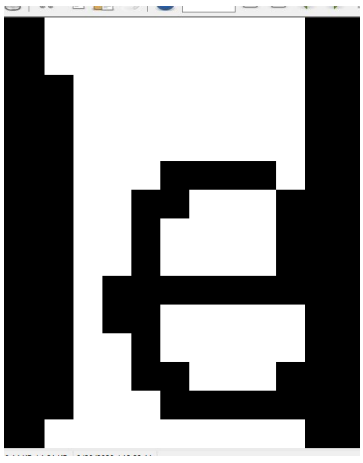

Figure 2: Thresholded Binary Image

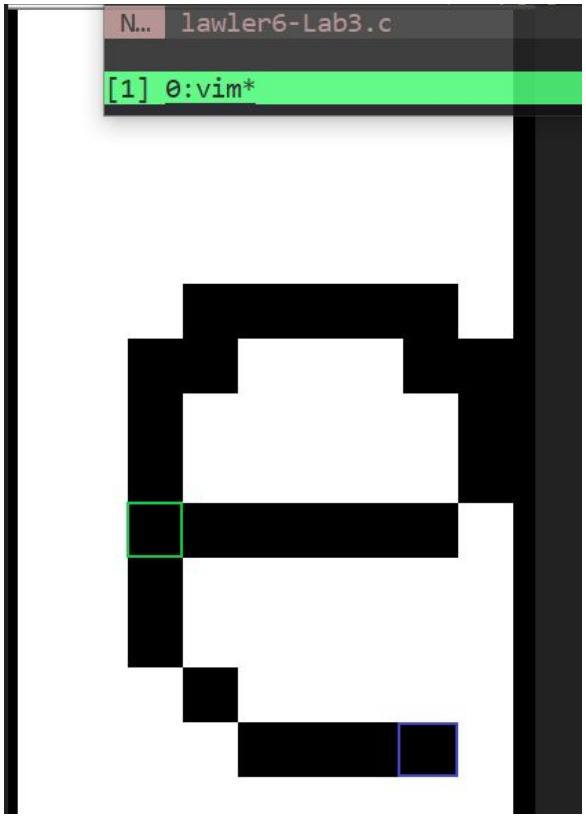

Figure 3: Thinned Binary Image with Blue Endpoint and Green Branchpoint

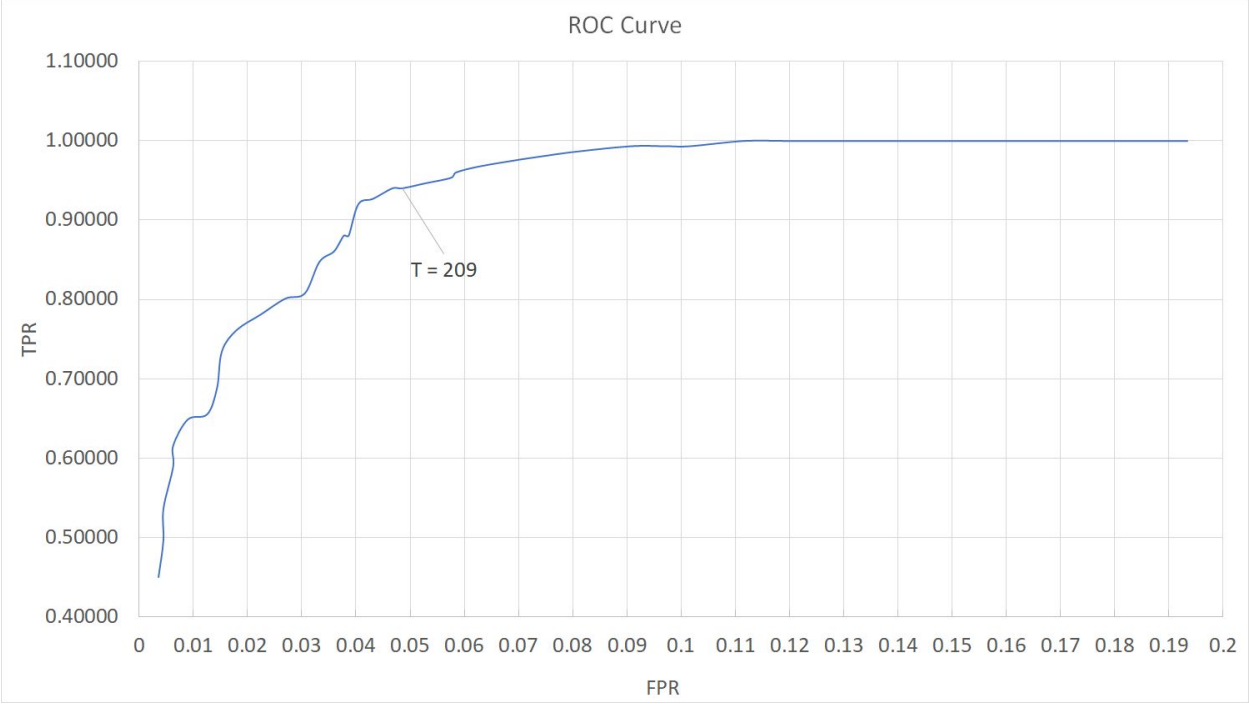

Figure 4: ROC Curve

Threshold	FP	TP	FN	TN	TPR	FPR
190	215	151	0	896	1.00000	0.19352
191	200	151	0	911	1.00000	0.18002
192	187	151	0	924	1.00000	0.16832
193	180	151	0	931	1.00000	0.16202
194	169	151	0	942	1.00000	0.15212
195	162	151	0	949	1.00000	0.14581
196	148	151	0	963	1.00000	0.13321
197	141	151	0	963	1.00000	0.12772
198	133	151	0	978	1.00000	0.11971
199	124	151	0	987	1.00000	0.11161
200	113	150	1	998	0.99338	0.10171
201	108	150	1	1003	0.99338	0.09721
202	101	150	1	1010	0.99338	0.09091
203	90	149	2	1021	0.98675	0.08101
204	82	148	3	1029	0.98013	0.07381
205	75	147	4	1036	0.97351	0.06751
206	69	146	5	1042	0.96689	0.06211
207	65	145	6	1046	0.96026	0.05851
208	64	144	7	1047	0.95364	0.05761
209	59	143	8	1052	0.94702	0.05311
210	54	142	9	1057	0.94040	0.0486
211	52	142	9	1059	0.94040	0.0468
212	48	140	11	1063	0.92715	0.0432
213	45	139	12	1066	0.92053	0.0405
214	43	133	18	1068	0.88079	0.0387
215	42	133	18	1069	0.88079	0.0378
216	40	130	21	1071	0.86093	0.036
217	37	128	23	1074	0.84768	0.0333
218	34	122	29	1077	0.80795	0.0306
219	30	121	30	1081	0.80132	0.027
220	25	118	33	1086	0.78146	0.0225
221	20	115	36	1091	0.76159	0.018
222	17	111	40	1094	0.73510	0.0153
223	16	104	47	1095	0.68874	0.0144
224	14	99	52	1097	0.65563	0.0126
225	10	98	53	1101	0.64901	0.009
226	7	93	58	1104	0.61589	0.0063
227	7	89	62	1104	0.58940	0.0063
228	5	81	70	1106	0.53642	0.0045
229	5	75	76	1106	0.49669	0.0045
230	4	68	83	1107	0.45033	0.0036

Figure 5: Data Values

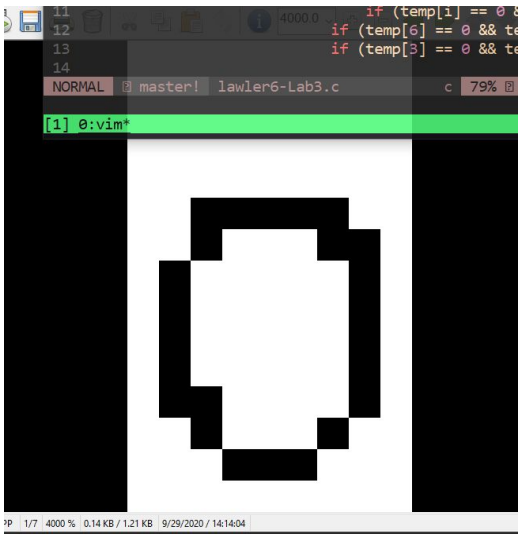

Figure 6: Example of a Properly Aligned and Thinned Binary Image