**William Benton**

**Lab 2: Optical Character Recognition**

**09/19/2019**

**Background**

The purpose of this lab was to scan an image of text and locate all of a certain letter throughout the entirety of the text. This was done by using an image of the letter to find in that font and using template matching on the image of the text. Template matching will look at each pixel and compare the surrounding pixels to the template. In the case of this lab, the template is the letter ‘e’.

**Implementation**

In order to match a pixel with the template in varying light conditions, it is important to use a Zero Mean Filter on the template first. This filter simply calculates the average of every pixel in the template and subtracts that amount from each pixel. This is done to compare the pattern of the filter to the input image instead of just comparing the input image to the pixel values of the image used for template matching. From here, the input image and filtered template are convolved to create a Matched Spatial Filter, MSF. With this convolution, the range of the pixels will be significantly greater than the 8 bits used to create the PPM. The result of the convolution is stored in an array of integers instead of unsigned characters in order to account for this wider range. In order to get back to an 8-bit range, the MSF image will have to be normalized. There are different ways to normalize these values. The equation I used can be seen in equation 1.

This equation was found online and the source can be found in the reference 1. It was modified slightly to work with a range of 0-255. With this normalized MSF image, you can see small bright white pixels where there are ‘e’s. Thresholding the MSF image pulls these pixels out. Determining the best thresholding value isn’t something that is easily done ahead of time. In order to determine the best thresholding value, I read in the ground truth table into an array of a structure that holds the row and column location of every letter and the letter associated with those coordinates. Once the ground truth had been read in, the thresholding values were tested. Looping from 0 through 255, each value was used to threshold the MSF image. To test the quality of the threshold, I looped through the array used to store the ground truth information. At each coordinate, a 9x15 area around the coordinate was looked at to determine if there were any pixels that were “on,” meaning if a pixel was equal to 255 after thresholding. If a pixel was on, then an e was detected. If something was found, then the letter at that location was looked at to determine if it really was an ‘e’ or if it was a different letter. If it was really an ‘e’, then that was a true positive. If it wasn’t an ‘e’, then it was a false positive. If there wasn’t a pixel on in the 9x15 area around one of the coordinates, then the letter was compared. If the letter was an ‘e’, that was a false negative, meaning that the ‘e’ was missed. If the letter wasn’t an ‘e’, that was a true negative, meaning that an ‘e’ should not have been found. The totals for the true positives, false positives, true negatives, and false negatives were totaled for each threshold value. These values were written to a .csv value along with the thresholding value used. The .csv file was opened in excel. Using the values for the true positives, false positives, true negatives, and false negatives, a ROC curve was generated. The equations used to calculate this curve was found online and can be found in reference 2. In order to determine the best thresholding value, the distance to the corner was calculated where the corner was 1 for the true positive rate and 0 for the false positive rate. This equation was found in reference 3. After the best thresholding value was found, the code was edited to print out the result of the best threshold.

**Results**

The ROC Curve can be seen in Figure 1. This curve shows a very good true positive rate vs false positive rate.

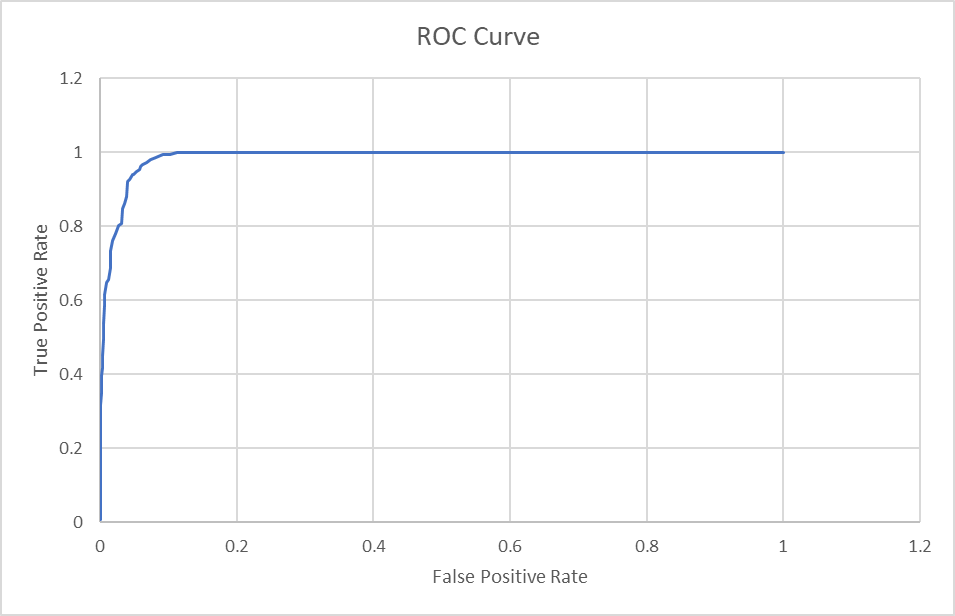
****After calculating the distance for every threshold value, it was determined that the best threshold value was 206 with 146 true positives and 69 false positives. The distance from the corner at [0,1] on the curve for this threshold value was 0.070382. Figure 2 shows the starting image and figure 3 shows the result of the MSF filter after normalization. Figure 4 shows the output of the optical character recognition program if the threshold value is set to 206, the best threshold value for this specific use case.

Figure : ROC Curve

**A screenshot of a cell phone

Description automatically generatedA close up of a newspaper

Description automatically generated**

Figure 3: Normalized MSF Image

Figure : Input Image

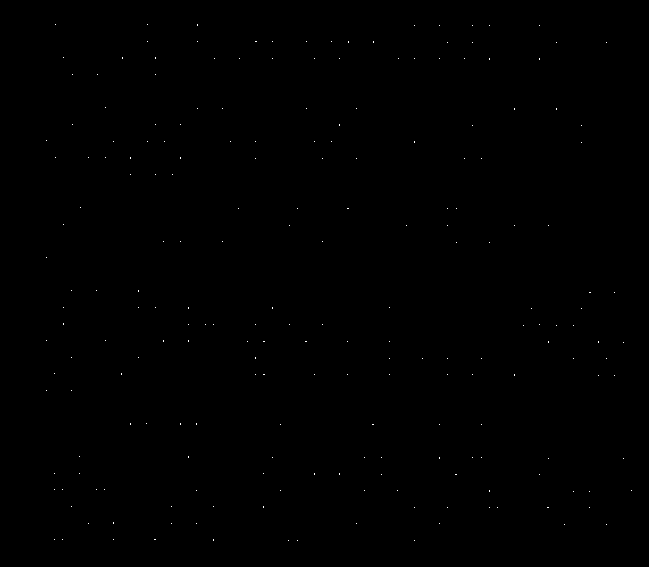
****

Figure 4: Output at Threshold of 206

**Code**

charRecog.h

struct coordinates\_s{

int Column;

int Row;

char Letter;

};

void Calc\_MSF(unsigned char \*image, unsigned char \*letter, int ROWS, int COLS, int T\_ROWS, int T\_COLS, unsigned char \*MSF);

void ZMean(int \*temp\_image, int ROWS, int COLS);

void Normalize(int \*temp\_image, int ROWS, int COLS);

void find\_letter(unsigned char \*image, int ROWS, int COLS, int T\_ROWS, int T\_COLS, struct coordinates\_s \*coordinates, int CurrentCount);

charRecog.c

/\* William Benton

\* Lab 2: Optical Character Recognition

\* This program will take an image of text and look for a specific character

\* based on an image of the specific letter.

\*/

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <stdbool.h>

#include "charRecog.h"

int main()

{

FILE \*fpt;

unsigned char \*letter;

unsigned char \*input\_image;

char header[320];

int ROWS, COLS, BYTES;

// Template Header

char T\_header[320];

// Template Rows, Columns, and Bytes

int T\_ROWS, T\_COLS, T\_BYTES;

unsigned char \*MSF;

// Open input image

if ((fpt=fopen("parenthood.ppm","rb")) == NULL)

{

printf("Unable to open image for reading.\n");

exit(0);

}

fscanf(fpt, "%s %d %d %d", header, &COLS, &ROWS, &BYTES);

if (strcmp(header, "P5") != 0 || BYTES != 255)

{

printf("Not a greyscale 8-bit PPM image\n");

exit(0);

}

input\_image = (unsigned char \*)calloc(ROWS\*COLS, sizeof(unsigned char));

header[0] = fgetc(fpt); // Ensure the whitespace is skipped

fread(input\_image, 1, COLS\*ROWS, fpt);

fclose(fpt);

// Open Templatate

if ((fpt=fopen("parenthood\_e\_template.ppm","rb")) == NULL)

{

printf("Unable to open image for reading.\n");

exit(0);

}

fscanf(fpt, "%s %d %d %d", T\_header, &T\_COLS, &T\_ROWS, &T\_BYTES);

if (strcmp(T\_header, "P5") != 0 || T\_BYTES != 255)

{

printf("Not a greyscale 8-bit PPM image\n");

exit(0);

}

letter = (unsigned char \*)calloc(ROWS\*COLS, sizeof(unsigned char));

header[0] = fgetc(fpt); // Ensure the whitespace is skipped

fread(letter, 1, T\_COLS\*T\_ROWS, fpt);

fclose(fpt);

// Allocate Space for MSF image

MSF = (unsigned char \*)calloc(ROWS\*COLS, sizeof(unsigned char));

Calc\_MSF(input\_image, letter, ROWS, COLS, T\_ROWS, T\_COLS, MSF);

// Create the Normalized MSF image for output

fpt = fopen("MSF.ppm", "w");

fprintf(fpt, "P5 %d %d 255\n", COLS, ROWS);

fwrite(MSF, COLS\*ROWS, 1, fpt);

fclose(fpt);

// Ground Truth Processing

fpt = fopen("parenthood\_gt.txt", "rb");

if (fpt == NULL)

{

printf("Error opening ground truth\n");

exit(0);

}

struct coordinates\_s \*coordinates;

int CurrentCount = 0;

// Allocate space for the ground truth data

coordinates = (struct coordinates\_s\*)calloc(1300,sizeof(struct coordinates\_s));

// Loop through until the end of file and store data in array of structs

while (!feof(fpt))

{

fscanf(fpt, "%c %d %d\n", &coordinates[CurrentCount].Letter, &coordinates[CurrentCount].Column, &coordinates[CurrentCount].Row);

CurrentCount++;

}

find\_letter(MSF, ROWS, COLS, T\_ROWS, T\_COLS, coordinates, CurrentCount);

return(1);

}

void Calc\_MSF(unsigned char \*image, unsigned char \*letter, int ROWS, int COLS, int T\_ROWS, int T\_COLS, unsigned char \*MSF)

{

int i, j;

int i2, j2;

int r, c;

int \*temp;

int \*temp\_letter;

int pixel;

temp\_letter = (int \*)calloc(T\_ROWS \* T\_COLS, sizeof(int));

temp = (int \*)calloc(ROWS \* COLS, sizeof(int));

for (i = 0; i < T\_ROWS \* T\_COLS; i++)

{

temp\_letter[i] = (int)letter[i];

}

// Calculate the zero mean filter of the template

ZMean(temp\_letter, T\_ROWS, T\_COLS);

// Loop through the image but cut off the edge pixels that can't be centered at the template.

for (i = T\_ROWS/2; i < (ROWS-(T\_ROWS/2)); i++)

{

for (j = T\_COLS/2; j < (COLS-(T\_COLS/2)); j++)

{

pixel = 0;

r = 0;

// Convolve the zero mean filtered image with the input image

for (i2 = (i - T\_ROWS/2); i2 <= (i + T\_ROWS/2); i2++)

{

c = 0;

for (j2 = (j - T\_COLS/2); j2 <= (j + T\_COLS/2); j2++)

{

pixel += (int)image[i2\*COLS+j2] \* temp\_letter[r\*T\_COLS+c];

c++;

}

r++;

}

temp[i \* COLS + j] = pixel;

}

}

// Normalize the MSF image

Normalize(temp, ROWS, COLS);

// Set the "output" which is the pointer that was passed to this function

for (i = 0; i < ROWS\*COLS; i++)

{

MSF[i] = (unsigned char)temp[i];

}

free(temp\_letter);

free(temp);

return;

}

void ZMean(int \*temp\_image, int ROWS, int COLS)

{

int i;

int average;

int total = 0;

for (i = 0; i < ROWS\*COLS; i++)

{

total += temp\_image[i];

}

average = total/(ROWS \* COLS);

for (i = 0; i < ROWS\*COLS; i++)

{

temp\_image[i] -= average;

}

return;

}

void Normalize(int \*temp, int ROWS, int COLS)

{

int i;

int max, min;

// Determine Max and Min

max = 0;

min = 2000;

for (i = 0; i < ROWS \* COLS; i++)

{

if (temp[i] < min)

{

min = temp[i];

}

if (temp[i] > max)

{

max = temp[i];

}

}

// Function for nomalization found online

for (i = 0; i < ROWS\*COLS; i++)

{

temp[i] = (temp[i] - min)\*255/(max-min);

}

return;

}

void threshold(unsigned char \*image, unsigned char \*temp\_image, int ROWS, int COLS, int thresh\_val)

{

int i;

for (i = 0; i < ROWS\*COLS; i++)

{

if (image[i] > thresh\_val)

{

temp\_image[i] = 255;

}

else

{

temp\_image[i] = 0;

}

}

return;

}

void find\_letter(unsigned char \*image, int ROWS, int COLS, int T\_ROWS, int T\_COLS, struct coordinates\_s \*Coordinates, int CurrentCount)

{

int i, j, k;

unsigned char \*temp;

int thresh\_val;

FILE \*fpt1;

FILE \*fpt2;

int tp, fp, tn, fn;

bool found;

temp = (unsigned char \*)calloc(ROWS\*COLS, sizeof(unsigned char));

// Test every threshold value from 0 through 255

for (thresh\_val = 0; thresh\_val < 256; thresh\_val++)

{

threshold(image, temp, ROWS, COLS, thresh\_val);

// Prepare the .csv file for the data

fpt2 = fopen("ROC.csv", "w");

fprintf(fpt2, ",TP,FP,TN,FN\n");

fpt2 = fopen("ROC.csv", "a");

tp = 0;

fp = 0;

tn = 0;

fn = 0;

// Loop through the ground truth data

for (i = 0; i < CurrentCount; i++)

{

found = false;

// Loop through 9x15 area around the coordinates from the ground truth data

for (j = Coordinates[i].Row - 7; j <= Coordinates[i].Row + 7; j++)

{

for (k = Coordinates[i].Column - 4; k <= Coordinates[i].Column + 4; k++)

{

if (temp[j\*COLS+k] == 255)

{

found = true;

}

}

}

// Determine true positive, false positive, true negative, and false negative

if (found == true && Coordinates[i].Letter == 'e')

{

tp++;

}

else if (found == true && Coordinates[i].Letter != 'e')

{

fp++;

}

else if (found != true && Coordinates[i].Letter == 'e')

{

fn++;

}

else if (found != true && Coordinates[i].Letter != 'e')

{

tn++;

}

}

fprintf(fpt2, "%d,%d,%d,%d,%d\n", thresh\_val, tp, fp, tn, fn);

// Print the totals for the TP, FP, TN, and FN to the .csv file

}

// Added after calculating the best threshold value.

// Prints the best threshold image

threshold(image, temp, ROWS, COLS, 206);

fpt1 = fopen("out.ppm", "w");

fprintf(fpt1, "P5 %d %d 255\n", COLS, ROWS);

fwrite(temp, COLS\*ROWS, 1, fpt1);

fclose(fpt1);

free(temp);

return;

}

**References**

Normalization Function:

[1] <https://www.wallstreetmojo.com/normalization-formula/#targetText=The%20equation%20for%20normalization%20is,is%20divided%20by%20the%20latter.>

ROC Calculations:

[2] <https://www.medcalc.org/manual/roc-curves.php>

[3] <https://ncss-wpengine.netdna-ssl.com/wp-content/themes/ncss/pdf/Procedures/NCSS/One_ROC_Curve_and_Cutoff_Analysis.pdf>