ECE 6310 Introduction to Computer Vision

Lab 3 – Letters

Prajval Vaskar

C20664702

September 29, 2020

CLEMSON® UNIVERSITY

Your program should perform the following steps:

1. Read the input image, your msf image, and ground truth file.

2. Loop through the following steps for a range of T:

a. Loop through the ground truth letter locations.

i. Check a 9 x 15 pixel area centered at the ground truth location. If any pixel in the msf image is greater than the threshold, consider the letter "detected". If none of the pixels in the 9 x 15 area are greater than the threshold, consider the letter "not detected".

ii. If the letter is "not detected" continue to the next letter.

iii. Create a 9 x 15 pixel image that is a copy of the area centered at the ground truth location (center of letter) from the original image.

iv. Threshold this image at 128 to create a binary image.

v. Thin the thresholded image down to single-pixel wide components.

vi. Check all remaining pixels to determine if they are branchpoints or endpoints.

vii. If there are not exactly 1 branchpoint and 1 endpoint, do not further consider this letter (it becomes "not detected").

b. Count up the number of FP (letters detected that are not 'e') and TP (number of letters detected that are 'e'). c. Output the total TP and FP for each T. Using any desired program, you must create an ROC curve from the program output

Solution:

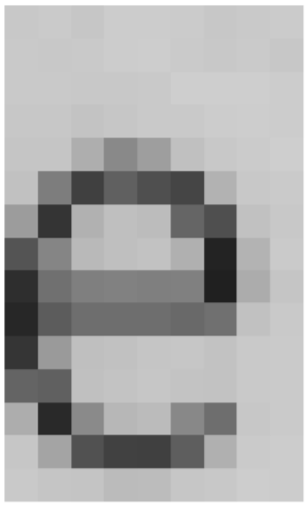

*Figure 1 Copy of area centered by ground truth location (15*9 pixels)*

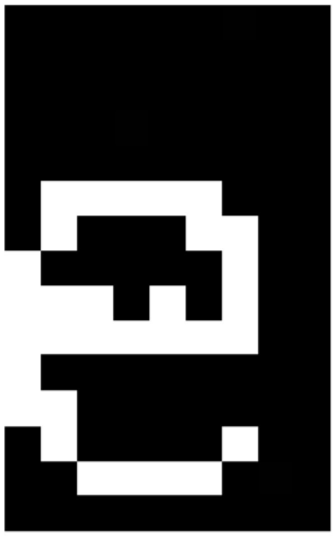

Figure 2 Thresholded image

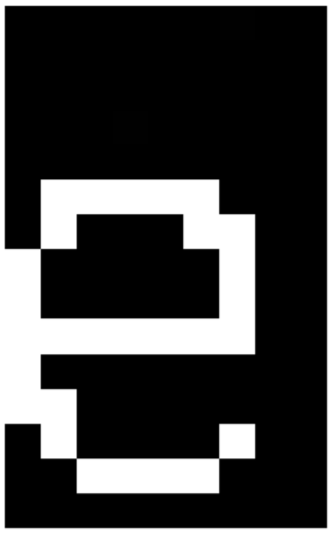

Figure 3 Image after thinning

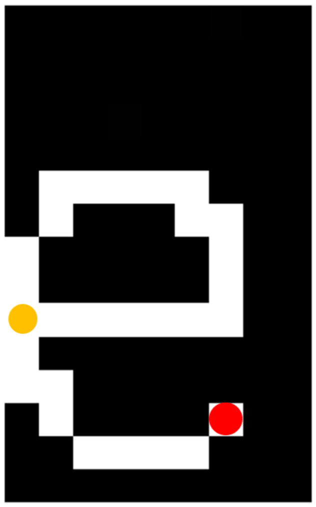

Figure 4 Image highlighting end point and branch point

In above image, the pixel denoted in red is edge point because it has only one edge to non-edge transition. The pixel in yellow is a branch point because it has 3 edge to non-edge transition. The letter e has only one edge point and branch point.

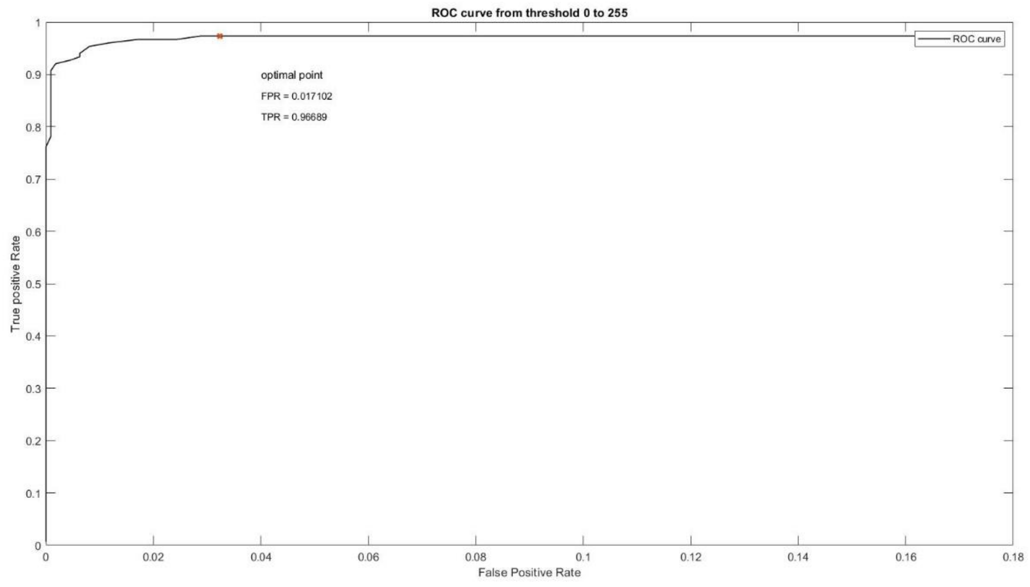

Figure 5 ROC curve for 0 to 255 threshold value

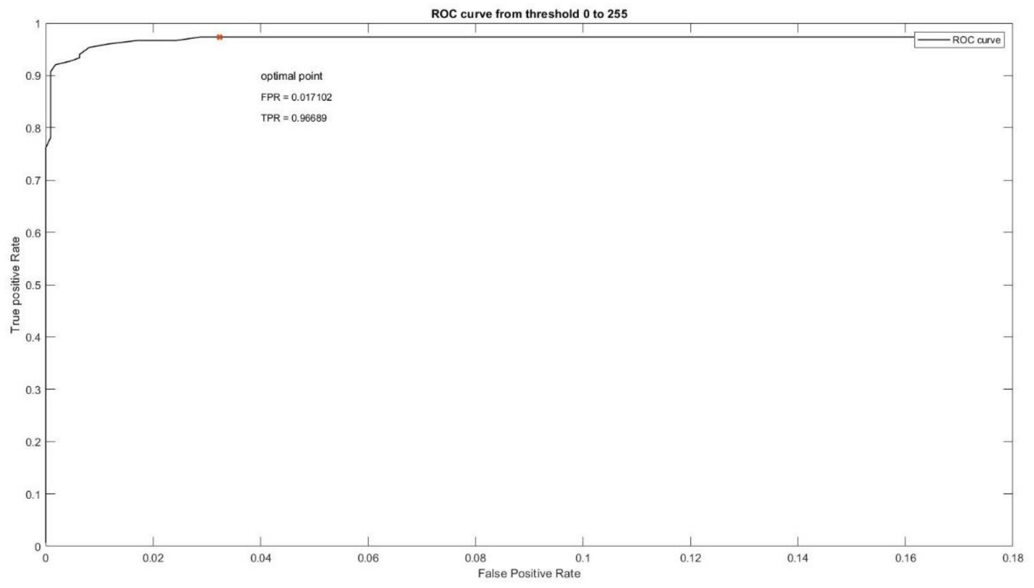

Figure 6 ROC curve for 170 to 255 threshold value

For the optimum value of threshold from the ROC curve, Euclidian distance is calculated from (0,1) i.e. FPR = 0 and TPR = 1 to point of each point on ROC curve. The point with minimum Euclidian distance is taken as optimum point and following result is found.

The minimum distance comes out to be 0.0373 and respective threshold value is 203.

Optimum Point

T = 203.

TP = 145

FP = 13

TN = 1098

FN = 6

TPR = 0.9603

FPR = 0.0171

Appendix

C code

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
```

```
FILE    *fpt, *file;
unsigned char *image,*temp, *copy_t,*copy, *copy_threshold, *normal_image, *final
_image, *msf_e, *temporary,*branch;
char    header[320], header_1[320],letter[10],gt_letter[10];
int   ROWS,COLS,bytes,rows,cols,BYTES;
int   r,c,i,j,c1,c2,r1,r2,n,T,or,oc,tp,tn,fp,fn,detected=0;



/* Creates copy and thresholds */
unsigned char* img_copy(int r, int c)
{
  copy=(unsigned char *)calloc(ROWS*COLS,sizeof(unsigned char));
  i = 0;
  for (r1 = r-7; r1 <= (r + 7); r1++)
  {
    j = 0;
    for (c1 = c-4; c1 <= (c + 4); c1++)
    {
      copy[i*COLS+j] = image[r1*cols+c1];
      j++;
    }

    i++;
  }

  for (i = 0; i < ROWS*COLS; i++)
  {
    if (copy[i] < 128)
    {
      copy_threshold[i] = 255;
    }
    else
    {
      copy_threshold[i] = 0;
    }
  }

  return copy_threshold;
}

void thinning(unsigned char *thinned)
{
  int row, col, i, neighbors, edge_to_nonedge;
  bool erase;
bool check;


erase = true;
while (erase)
{
  erase = false;
  for (row = 0; row < 15; row++)
  {
    for (col = 0; col < 9; col++)
    {
      i = row*9+col;
      /* Checking if pixel is on*/
      if ((thinned[i] > 0))
      {
        edge_to_nonedge = 0;
        neighbors = 0;
        /* North */
        if (thinned[i-9] == 255)
        {
          neighbors++;
          check = true;
        }
        else
        {
          check = false;
        }
        /* North East */
        if (thinned[i-9+1] == 255)
        {
          neighbors++;
          check = true;
        }
        else
        {
          if (check)
          {
            edge_to_nonedge++;
          }
          check = false;
        }
        /* East */
        if (thinned[i+1] == 255)
        {
          neighbors++;
```

```
            check = true;
          }
          else
          {
            if (check)
            {
              edge_to_nonedge++;
            }
            check = false;
          }
          /* South East */
          if (thinned[i+1+9] == 255)
          {
            neighbors++;
            check = true;
          }
          else
          {
            if (check)
            {
              edge_to_nonedge++;
            }
            check = false;
          }
          /* South */
          if (thinned[i+9] == 255)
          {
            neighbors++;
            check = true;
          }
          else
          {
            if (check)
            {
              edge_to_nonedge++;
            }
            check = false;
          }
          /* South West */
          if (thinned[i+9-1] == 255)
          {
            neighbors++;
            check = true;
          }
          else
```

```
{
  if (check)
  {
    edge_to_nonedge++;
  }
  check = false;
}
/* West */
if (thinned[i-1] == 255)
{
  neighbors++;
  check = true;
}
else
{
  if (check)
  {
    edge_to_nonedge++;
  }
  check = false;
}
/* North West */
if (thinned[i-1-9] == 255)
{
  neighbors++;
  check = true;
}
else
{
  if (check)
  {
    edge_to_nonedge++;
  }
  check = false;
}
/* loop to check edge to non edge for north west to north*/
if ((thinned[i-9] == 0) && check)
{
  edge_to_nonedge++;
}

/* loop to erase the pixel */
if (edge_to_nonedge == 1)
{
  if (2 <= neighbors && neighbors <= 6)
```

```
            {
              if ((thinned[i-9] == 0) || (thinned[i+1] == 0) || ((thinned[i-
1] == 0) && (thinned[i+9] == 0)))
              {
                thinned[i] = 0;
                erase = true;
              }
            }
          }
        }
      }
    }
  }
  return;
}




int branch_find(unsigned char *thinned)
{
  int edge_to_nonedge,row,col,branch_pt=0,end_pt=0,detected;
  bool check;
  /* funtion that return detected = 1 or 0 */
  for (row = 0; row < 15; row++)
    {
      for (col = 0; col < 9; col++)
      {
        i = row*9+col;
        /* Checking if pixel is on*/
        if ((thinned[i] > 0))
        {
          edge_to_nonedge = 0;
          /* North */
          if (thinned[i-9] == 255)
          {
            check = true;
          }
          else
          {
            check = false;
          }
          /* North East */
          if (thinned[i-9+1] == 255)
          {
            check = true;
          }
          else
          {
            if (check)
            {
              edge_to_nonedge++;
            }
            check = false;
          }
          /* East */
          if (thinned[i+1] == 255)
          {
            check = true;
          }
          else
          {
            if (check)
            {
              edge_to_nonedge++;
            }
            check = false;
          }
          /* South East */
          if (thinned[i+1+9] == 255)
          {
            check = true;
          }
          else
          {
            if (check)
            {
              edge_to_nonedge++;
            }
            check = false;
          }
          /* South */
          if (thinned[i+9] == 255)
          {
            check = true;
          }
          else
          {
            if (check)
            {
              edge_to_nonedge++;
            }
            check = false;
          }
          /* South West */
          if (thinned[i+9-1] == 255)
          {
            check = true;
          }
          else
          {
            if (check)
            {
              edge_to_nonedge++;
            }
            check = false;
          }
          /* West */
          if (thinned[i-1] == 255)
          {
            check = true;
          }
          else
          {
            if (check)
            {
              edge_to_nonedge++;
            }
            check = false;
          }
          /* North West */
          if (thinned[i-1-9] == 255)
          {
            check = true;
          }
          else
          {
            if (check)
            {
              edge_to_nonedge++;
            }
            check = false;
          }
          /* loop to check edge to non edge for north west to north*/
          if ((thinned[i-9] == 0) && check)
          {
            edge_to_nonedge++;
          }

          if (edge_to_nonedge == 1)
          {
            end_pt++;
          }
          if (edge_to_nonedge > 2)
          {
            branch_pt++;
          }
        }
      }
    }
  if (branch_pt == 1 && end_pt ==1)
  {
    detected = 1;
  }
  else
  {
    detected = 0;
  }

  return detected;
}

int main()

{
  /* read template */
  if ((fpt=fopen("parenthood_e_template.ppm","rb")) == NULL)
    {
    printf("Unable to open parenthood_e_template.ppm for reading\n");
    exit(0);
    }
  fscanf(fpt,"%s %d %d %d",header_1,&COLS,&ROWS,&bytes);
  if (strcmp(header_1,"P5") != 0  ||  bytes != 255)
    {
    printf("Not a greyscale 8-bit PPM image\n");
    exit(0);
    }

  /* Allocate dynamic memory for template and zero_min_template */
  temp=(unsigned char *)calloc(ROWS*COLS,sizeof(unsigned char));
  temporary=(unsigned char *)calloc(ROWS*COLS,sizeof(unsigned char));
  copy_t=(unsigned char *)calloc(ROWS*COLS,sizeof(unsigned char));
  copy_threshold=(unsigned char *)calloc(ROWS*COLS,sizeof(unsigned char));
  branch = (unsigned char *)calloc(ROWS*COLS,sizeof(unsigned char));

  header_1[0]=fgetc(fpt);           /* read white-
space character that separates header */
  fread(temp,1,ROWS*COLS,fpt);
  fclose(fpt);

  /* Reading parenthood image */
  if ((fpt=fopen("parenthood.ppm","rb")) == NULL)
    {
    printf("Unable to open parenthood.ppm 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);
    }

  image=(unsigned char *)calloc(rows*cols,sizeof(unsigned char));
  header[0]=fgetc(fpt); /* read white-space character that separates header */
  fread(image,1,cols*rows,fpt);
  fclose(fpt);


  /* Reading MSF_E image */
  if ((fpt=fopen("msf_e.ppm","rb")) == NULL)
    {
    printf("Unable to open msf_e.ppm 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);
    }

  msf_e = (unsigned char *)calloc(rows*cols,sizeof(unsigned char));
  header[0]=fgetc(fpt); /* read white-space character that separates header */
  fread(msf_e,1,cols*rows,fpt);
  fclose(fpt);

  /* Creating a data.csv file to store tn fn and other */

  file = fopen("Data.csv", "wb");
  fprintf(file, "Threshold,TP,FP,TN,FN,TPR,FPR\n");



  for (T =0; T < 256; T++)
  {
    /* Reading parenthood_gt file */
    fpt = fopen("parenthood_gt.txt","rb");
    tp = fp = fn = tn = 0;
    strcpy(letter, "e");

    /* Checking pixel value above threshold */
    while((fscanf(fpt, "%s %d %d\n", gt_letter, &oc, &or)) != EOF)
    {
      for (r1 = or-7; r1 <= (or + 7); r1++)
      {
        for (c1 = oc-4; c1 <= (oc + 4); c1++)
        {
          if (msf_e[(r1 * cols) + c1] > T)
          {
            detected = 1;
          }
        }
      }

      if (detected == 1)
      {
        /* Creating a 9x15 copy of image(or,oc) and thresholding at value of 128
*/
        copy_t = img_copy(or,oc);

        /* Apply thining on copied image */
        thinning(copy_t);

        for (i=0;i<9*15;i++){
          branch[i] = copy_t[i];
        }

        detected = branch_find(branch);
      }

      /* Finding values of TP, FP, FN and TN */
      if ((detected == 1) && (strcmp(gt_letter, letter) == 0))
      {
        tp++;
      }
      if ((detected == 1) && (strcmp(gt_letter, letter) != 0))
      {
        fp++;
      }
      if ((detected == 0) && (strcmp(gt_letter, letter) == 0))
      {
        fn++;
      }
      if ((detected == 0) && (strcmp(gt_letter, letter) != 0))
      {
        tn++;
      }
      detected = 0;
    }


    printf("Threshold: %d, TP: %d, FP: %d, TN: %d, FN: %d, TPR: %lf, FPR: %lf\n",
T,tp,fp,tn,fn,(double)tp/(double)(tp+fn),(double)fp/(double)(fp+tn));
    fprintf(file, "%d,%d,%d,%d,%d,%lf,%lf\n",T,tp,fp,tn,fn,(double)tp/(double)(tp
+fn),(double)fp/(double)(fp+tn));
  }
  fclose(file);

  fpt=fopen("center.ppm","wb");
  fprintf(fpt,"P5 %d %d 255\n",COLS,ROWS);
  fwrite(copy,COLS*ROWS,1,fpt);
  fclose(fpt);


  fpt=fopen("thinned.ppm","wb");
  fprintf(fpt,"P5 %d %d 255\n",COLS,ROWS);
  fwrite(copy_t,COLS*ROWS,1,fpt);
  fclose(fpt);
}
```

MATLAB code for ROC generation.

```
%% Prajval Vaskar
% Computer Vision
% Lab 03 - Optical Character Recognition
% ROC curve plot

clear all;
clc;
close all;
M = csvread('data.csv',1,0);
X = M(:,7);
X =X';
Y = M(:,6);
Y = Y';
T = M(:,1);
plot(X',Y','k','LineWidth',1);
% axis equal
% xlim([-0.05 1.05]);ylim([-0.05 1.05]);
xlabel("False positive rate (FPR)");
ylabel("True positive rate (TPR)")
hold on
% Finding the optimal point

for i = 1:length(X)
   dist(i) = sqrt((1-Y(i))^2+(-X(i))^2);
end
[min,I] = min(dist);
plot(X(I),Y(I),"x",'LineWidth',2)
ylim([0 1])
legend("ROC curve", "Optimal point")
str = {'optimal point'};
text(0.04,0.90,str,'FontSize',10);
t = text(0.04,0.86,['FPR = ',num2str(X(I))]);
t.FontSize = 9;
t = text(0.04,0.82,['TPR = ',num2str(Y(I))]);
t.FontSize = 9;
title("ROC curve from threshold 170 to 255");
xlabel("False Positive Rate");
ylabel("True positive Rate");
legend('ROC curve','Optimum point');
xlim([0,0.3]);
% ylim([0.8,1]);
```

```
%%

x = X(170:255);
y = Y(170:255);
t = T(170:255);

figure()
plot(x,y,'k','LineWidth',1);
hold on
plot(x(30),y(30),"x",'LineWidth',2)
title("ROC curve from threshold 0 to 255");
xlabel("False Positive Rate");
ylabel("True positive Rate");
legend('ROC curve');
str = {'optimal point'};
text(0.04,0.90,str,'FontSize',10);
t = text(0.04,0.86,['FPR = ',num2str(X(I))]);
t.FontSize = 9;
t = text(0.04,0.82,['TPR = ',num2str(Y(I))]);
t.FontSize = 9;
```