ECE 6310

Introduction to Computer Vision

Fall 2022

Lab 3

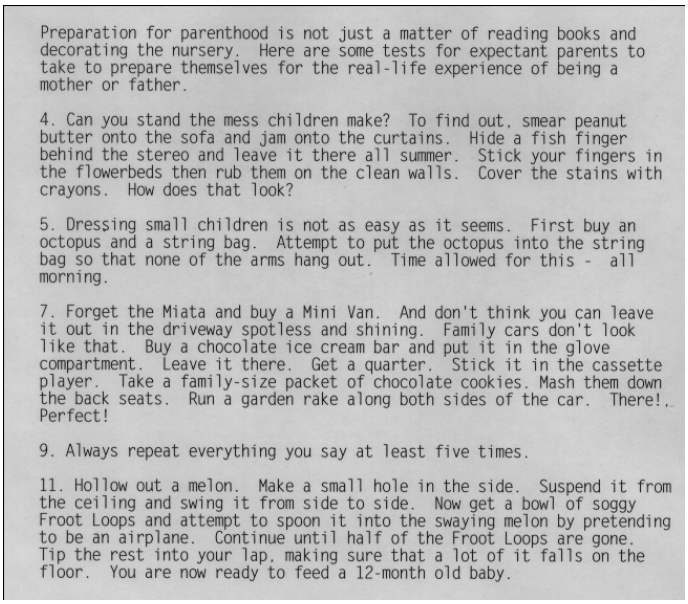
Letters [Optical Character Recognition with thinning and Skeletonization]

Harshal B. Varpe

Introduction

This lab builds upon the technique known as matched spatial filtering used in lab2. In this lab, students were to implement thinning, branchpoint and endpoint detection. Thinning is a process where an image in converted from its original form to simple skeletonized image. This is image retains the original feature of an image. Thinning is essential when dealing with the optical character recognition. By using skeletonization along with matched spatial filtering, we are able to reduce the number of false positives.

The students were provided an image of a book excerpt, a ground truth text file containing all the letters in the excerpt and their respective pixel coordinates, and a template of a letter to be recognized. The general approach includes looking at each pixel and surrounding area of 9x15 pixels and matching that to the template letter 'e.'



Preparation for parenthood is not just a matter of reading books and decorating the nursery. Here are some tests for expectant parents to take to prepare themselves for the real-life experience of being a mother or father.

4. Can you stand the mess children make? To find out, smear peanut butter onto the sofa and jam onto the curtains. Hide a fish finger behind the stereo and leave it there all summer. Stick your fingers in the flowerbeds then rub them on the clean walls. Cover the stains with crayons. How does that look?

5. Dressing small children is not as easy as it seems. First buy an octopus and a string bag. Attempt to put the octopus into the string bag so that none of the arms hang out. Time allowed for this - all morning.

7. Forget the Miata and buy a Mini Van. And don't think you can leave it out in the driveway spotless and shining. Family cars don't look like that. Buy a chocolate ice cream bar and put it in the glove compartment. Leave it there. Get a quarter. Stick it in the cassette player. Take a family-size packet of chocolate cookies. Mash them down the back seats. Run a garden rake along both sides of the car. There!. Perfect!

9. Always repeat everything you say at least five times.

11. Hollow out a melon. Make a small hole in the side. Suspend it from the ceiling and swing it from side to side. Now get a bowl of soggy Froot Loops and attempt to spoon it into the swaying melon by pretending to be an airplane. Continue until half of the Froot Loops are gone. Tip the rest into your lap, making sure that a lot of it falls on the floor. You are now ready to feed a 12-month old baby.

Figure 1: Input image (parenthood.ppm)

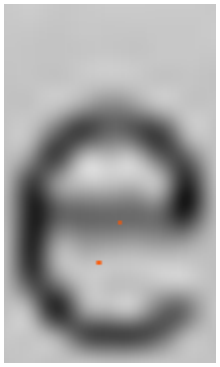


Figure 2: Template ‘e’

Implementation / Methods

As previously mentioned, Lab 3 builds upon the previous lab. In this lab, we use the normalized matched spatial filtered (henceforth, mentioned as normalized msf image) image obtained in lab 2. Let us dive into more details. The normalized msf image shows us the the pixel locations where the letter ‘e’ is detected as bright spots. We then copy this letter location from an original image and threshold at 128. Figure 4 shows an thresholded letter ‘e’. Once the thresholding is done, we move on to the process of determining edge 2 non-edge transitions, endpoints ,and bracnhpoints. We iteratively delete the pixels from the image, intil single pixel wide representation of the letter remains.

For different pixel intensities ranging from 0 to 255, the pixel values of normalized msf image are compared. If the pixel values is above certain threshold, then we find the neighbors of said pixel to decide the number of edge to non-edge transitions, endpoints and branchpoints. If a pixel has only one edge to non-edge transition, between 2-6 neighbors, and if the pixel to the north or east or (south and west) is

not an edge, then the said pixel is marked for erasure. This process is an iterative one. Figure 5 shows an thinned letter ‘e’.

We know that for letter ‘e’, we have exactly one endpoint and one branchpoint. We can determine the number of endpoints and branchpoints by the number of edge to non-edge transitions. For an endpoint, there is only one edge 2 non-edge transition. For a branchpoint there are two or more edge to non-edge transitions. Figure 6 shows the endpoint and branchpoint for letter ‘e’ marked in red.

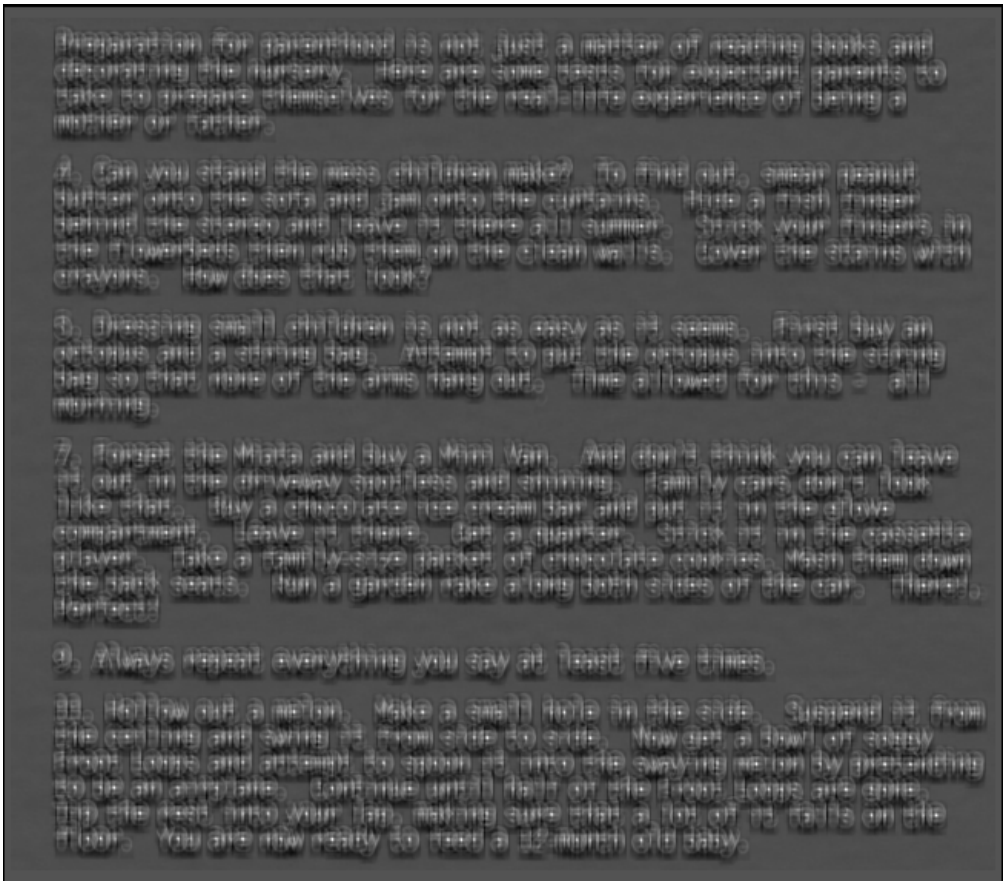


Figure 3: Normalized Image

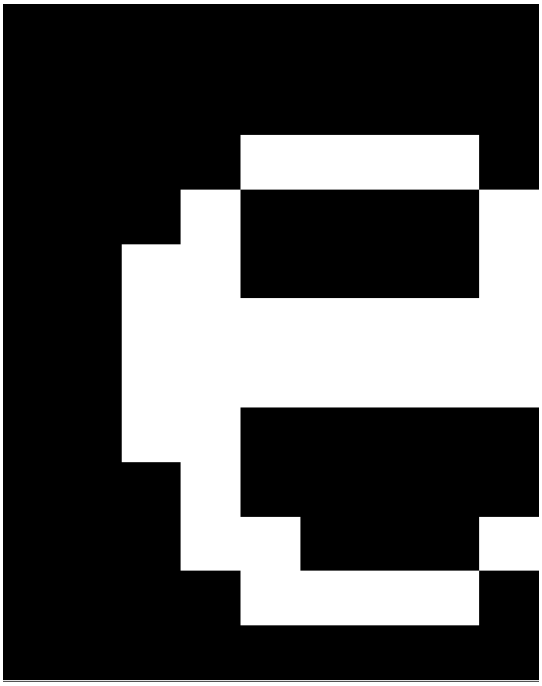


Figure 4: Thresholded letter ‘e’

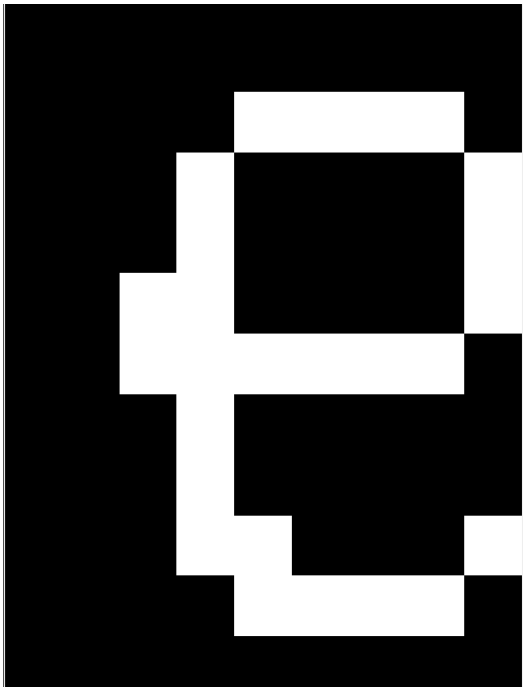


Figure 5: Thinned letter ‘e’

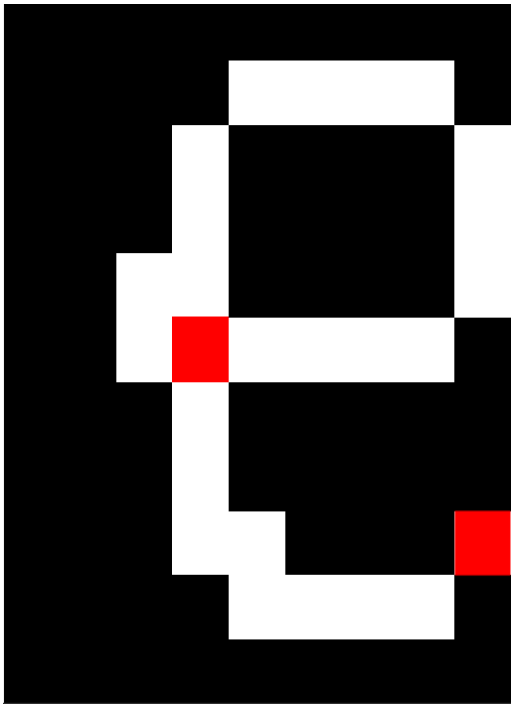


Figure 6: Thinned letter ‘e’

Once the branchpoints and endpoints were found, we check if the letter detected is indeed letter ‘e’. Depending on this condition, we calculate True Positive, True Negative, False Positive, False Negative, and True Positive Rate (TPR), and False Positive Rate (FPR). We create a ROC curve from TPR and FPR.

Figure 7 shows the ROC curve for lab 2 in comparison with ROC curve for lab 3. From the figure, we can clearly see that the We have reduced our FPR in lab 3 as compared to the FPR in lab 2. This reduction is evident by the shifting of red line to left. However, this reduction in FPR also means small reduction in our TPR, which is also evident from the plot below.

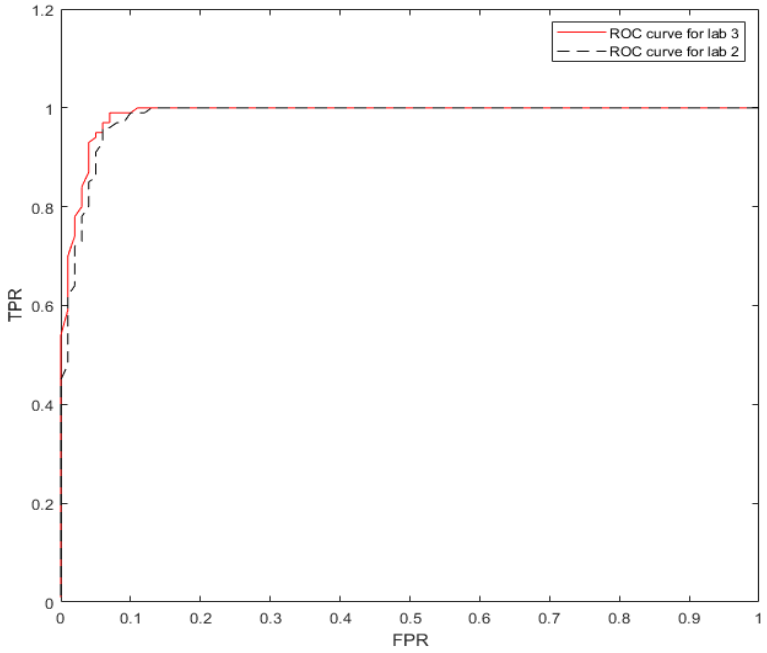


Figure 7: ROC curve comparison between lab2 and lab3

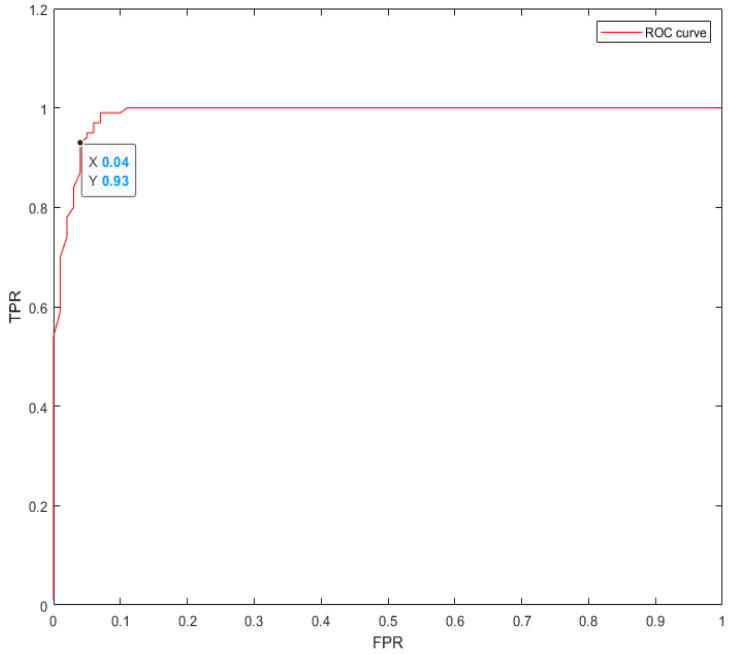


Figure 8: ROC curve Lab 3

The figure 8 above shows that the behavior of the ROC curve TPR (0.93) and FPR (0.04) changes rapidly. This point is known as the knee of the ROC curve. In an average case like ours, we can select the knee of the curve as the best tradeoff performance. The Threshold value at the knee is 212. At the optimal threshold, there were 142 True positives and 53 False positives. While the number of true positives is the same as observed in lab2, the number of false positives is reduced by 14.

Code

```
// ECE 6310 Introduction to CV
// Lab 3 Fall 22
// Harshal Varpe
// Clemson University
# include <stdio.h>
# include <stdlib.h>
# include <string.h>

int main (int argc, char *argv[]){
    FILE *fpt, *temp_file, *g_truth, *out;
    int r,c,gt_row,gt_col,detected,not_detected,T,TP,FP,FN,TN,i,j,k;
    int min,max,r1,c1,ROWS,COLS,BYTES,temp_R, temp_C,temp_B;
    char header[320],gt_letter[320],letter;
    unsigned char *input,*template_img,*img_msf_norm,*final, *binary, *copy,
*thin_img;
    float TPR,FPR,mean; int *temp_img;
    double *img_msf,sum;
    // New variable decalaration
    int del_counter;
    unsigned char neighbor[9] = {0};
    int e2ne,flag1,flag2,flag3,num_neighbors,endpt,branchpt;

if(argc != 2){printf("Wrong number or arguments! \n Usage : [executable_name]
[letter]");
exit(0);}
letter = argv[1][0];

/* Task 1 - Create a Zero mean centered template from the template image */
fpt = fopen("parenthood.ppm","rb");
if (fpt == NULL)
  {
  printf("Unable to open %s for reading\n","parenthood.ppm");
  exit(0);
  }
fscanf(fpt,"%s %d %d %d",header,&COLS,&ROWS,&BYTES);
input = (unsigned char *)calloc(ROWS*COLS,sizeof(unsigned char));
fread(input,1,ROWS*COLS,fpt);
fclose(fpt);

fpt = fopen("parenthood_e_template.ppm","rb");
if (fpt == NULL)
  {
  printf("Unable to open %s for reading\n","parenthood_e_template.ppm");
  exit(0);
```

```
  }
fscanf(fpt,"%s %d %d %d",header,&temp_C,&temp_R,&temp_B);
template_img = (unsigned char *)calloc(temp_R*temp_C,sizeof(unsigned char));
fread(template_img,1,temp_R*temp_C,fpt);
fclose(fpt);

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


fpt = fopen("normalized_msf.ppm","rb");
if(fpt == NULL){printf("File %s could not be
opened!\n","normalized_msf.ppm");exit(0);}
fscanf(fpt,"%s %d %d %d",header,&COLS,&ROWS,&BYTES);
fread(img_msf_norm,1,ROWS*COLS,fpt);
fclose(fpt);

// Open the ground truth file
g_truth = fopen("parenthood_gt.txt","rb");
if (g_truth == NULL){
    printf("The ground truth file could not be opened. Check if the name is
parenthood_gt.txt and location is correct.");
    exit(0);
}
// /* creating a text file to write the threshold and ROC curve data*/
out  = fopen("confusion_mat.txt","a"); // make sure to delete the previous data
from text file.
fprintf(out,"T TP FP FN TN TPR FPR \n");
// printf("%c \n",letter);


for(T=0;T<=255;T++){
    printf("%d\n",T);
    // for point 4.a
    binary = (unsigned char *)calloc(ROWS*COLS,sizeof(unsigned char));
    TP=TN=FP=FN=0;
    while(fscanf(g_truth,"%s %d %d",gt_letter,&gt_col,&gt_row) != EOF){
        // k = fscanf(g_truth,"%s %d %d",gt_letter,&gt_col,&gt_row);
        // printf("%d",k);
        detected = 0;
        not_detected = 0;
        // if(k!=3){break;}
        // printf(" detected %d \n",detected);
        for(r=gt_row-7;r<=gt_row+7;r++){
            for(c=gt_col-4;c<=gt_col+4;c++){
                if(img_msf_norm[r*COLS+c]>=T){detected = 1;}
                else{not_detected = 1;}
                }
```

```
            }
        // printf(" detected %d \n",detected); // delete this ; eveything works
up until this point if you comment everything below.

        if (detected == 1){
            // printf(" inside detected loop \n");
            int l = 0;
            // printf("%d %d \n",gt_col,gt_row);
            // getchar();
            copy = (unsigned char *)calloc(temp_R*temp_C,sizeof(unsigned char));
            for(r=gt_row-7; r<=gt_row+7; r++ ){
                for(c=gt_col-4 ;c<=gt_col+4; c++){
                    // printf("%d\n",l);
                    copy[l] = input[r*COLS+c]; // check indexing here !! copy
index may be wrong
                    l++;
                }
            }
            // 2.iv thresholding at 128
            for (i=0;i<(temp_R*temp_C);i++){
                if(copy[i] > 128){
                    copy[i] = 0;}
                else{
                    copy[i] = 255;
                }
            }
            // if(*gt_letter == letter){
            //     fpt = fopen("thresholded.ppm","wb");
            //     fprintf(fpt,"P5 %d %d 255\n",temp_C,temp_R);
            //     fwrite(copy,1,temp_R*temp_C,fpt);
            //     fclose(fpt);
            //     getchar();
            // } //  writing letter


            del_counter = 1;

            while(del_counter > 0){
                del_counter = 0;
                thin_img = (unsigned char *)calloc(temp_R*temp_C,sizeof(unsigned
char));
                for(i=0; i<(temp_R*temp_C) ;i++){
                    thin_img[i] = 0;
                }
                for(r=0;r<15;r++){
                    for(c=0;c<9;c++){
                        num_neighbors = 0;
```

```
                        e2ne = 0;
                        flag1 = 0;
                        flag2 = 0;
                        flag3 = 0;
                        if(copy[r*temp_C+c] == 255)
                        { // neighbor check
                            if(r==0 || c == 0){neighbor[0]=0;}
                            else{neighbor[0] = copy[(r-1)*temp_C+(c-1)];}

                            if(r==0){neighbor[1]=0;}
                            else{neighbor[1] = copy[(r-1)*temp_C+(c)];}

                            if(r==0 || c == 8){neighbor[2]=0;}
                            else{neighbor[2] = copy[(r-1)*temp_C+(c+1)];}

                            if(c == 8){neighbor[3]=0;}
                            else{neighbor[3] = copy[(r)*temp_C+(c+1)];}

                            if(r== 14 || c == 8){neighbor[4]=0;}
                            else{neighbor[4] = copy[(r+1)*temp_C+(c+1)];}

                            if(r==14){neighbor[5]=0;}
                            else{neighbor[5] = copy[(r+1)*temp_C+(c)];}

                            if(r==14 || c == 0){neighbor[6]=0;}
                            else{neighbor[6] = copy[(r+1)*temp_C+(c-1)];}

                            if(c == 0){neighbor[7]=0;}
                            else{neighbor[7] = copy[(r)*temp_C+(c-1)];}

                            // edge to non-edge transition condition
                            for(i=0;i<7;i++){if(neighbor[i]==255 &&
neighbor[i+1]==0){e2ne++;}}
                            if(neighbor[7]==255 && neighbor[0]==0){e2ne++;}
                            // printf("%d\n",e2ne);
                            if(e2ne==1){flag1 = 1;}

                            // number of neighbor condition
                            for(i=0;i<8;i++){if(neighbor[i] ==
255){num_neighbors++;}}
                            if(num_neighbors >= 2 && num_neighbors <=6){flag2 =
1;}
                            // printf("%d\n",num_neighbors);
                            // north or east or (west AND south) is not edge
                            if(neighbor[1]==0 || neighbor[3]==0 || (neighbor[7]
== 0 && neighbor[5] == 0)){
                                flag3 = 1;
```

```
                    }
                    // printf("%d\n",flag3);
                    if(flag1 == 1 && flag2 == 1 && flag3 == 1){
                        del_counter = del_counter - 1;
                        thin_img[r*temp_C+c] = 255;
                        // printf("counter reset");
                        }
                } // if
            } // for
        }//for loop
        //  delete marked cells
        for(r=0;r<15;r++){
            for(c=0;c<9;c++){
                if(thin_img[r*temp_C+c] == 255){copy[r*temp_C+c]=0;}
                }
            }
        //  if(*gt_letter == letter){
        //     fpt = fopen("thresholded.ppm","wb");
        //     fprintf(fpt,"P5 %d %d 255\n",temp_C,temp_R);
        //     fwrite(copy,1,temp_R*temp_C,fpt);
        //     fclose(fpt);
        //     getchar();
        //     } //  writing letter
    } // second while loop
    endpt =0;
    branchpt = 0;
    for(r=0;r<15;r++){
            for(c=0;c<9;c++){
                if(copy[r*temp_C+c] == 255)
                { // neighbor check
                    e2ne = 0;
                    // printf("Checking B and E\n");
                    if(r==0 || c == 0){neighbor[0]=0;}
                    else{neighbor[0] = copy[(r-1)*temp_C+(c-1)];}

                    if(r==0){neighbor[1]=0;}
                    else{neighbor[1] = copy[(r-1)*temp_C+(c)];}

                    if(r==0 || c == 8){neighbor[2]=0;}
                    else{neighbor[2] = copy[(r-1)*temp_C+(c+1)];}

                    if(c == 8){neighbor[3]=0;}
                    else{neighbor[3] = copy[(r)*temp_C+(c+1)];}

                    if(r== 14 || c == 8){neighbor[4]=0;}
                    else{neighbor[4] = copy[(r+1)*temp_C+(c+1)];}
```

```
                            if(r==14){neighbor[5]=0;}
                            else{neighbor[5] = copy[(r+1)*temp_C+(c)];}

                            if(r==14 || c == 0){neighbor[6]=0;}
                            else{neighbor[6] = copy[(r+1)*temp_C+(c-1)];}

                            if(c == 0){neighbor[7]=0;}
                            else{neighbor[7] = copy[(r)*temp_C+(c-1)];}

                            // edge to non-edge transition condition

                            for(i=0;i<7;i++){if(neighbor[i]==255 &&
neighbor[i+1]==0){e2ne++;}}
                            if(neighbor[7]==255 && neighbor[0]==0){e2ne++;}
                            if(e2ne==1){endpt++;}
                            else if(e2ne>2){branchpt++;}
                        }
                    }
                }
            if(branchpt == 1 && endpt == 1){detected = 1;}
            else{not_detected = 0;}
            // detected if loop
        }

        if ((detected == 1) && (*gt_letter == letter)){TP += 1;}
        else if ((detected == 1) && (*gt_letter != letter)){FP += 1;}
        else if ((not_detected == 1) && (*gt_letter == letter)){FN += 1;}
        else if ((not_detected == 1) && (*gt_letter != letter)){ TN +=1 ;}

    }
    TPR = (float)TP / (float)(TP+FN);
    FPR = (float)FP / (float)(FP+TN);
    fprintf(out,"%d %d %d %d %d %f %f\n",T,TP,FP,FN,TN,TPR,FPR);
    rewind(g_truth);
    // fclose(g_truth);
 }
fclose(out);
fclose(g_truth);
printf("This is the end my friend!");
}
```