LAB#2 OPTICAL CHARACTER RECOGNITION

ECE 4610/6310 INTRODUCTION TO COMPUTER VISION

MOHAMMAD ANAS IMAM KHAN
C17566828
September 15, 2020

1. Calculation of Matched-Spatial Filter:

i. To account for variation in intensity values, the given template was converted into a zero-mean center template.
   a. Calculate the sum of the pixel values in the template.
   b. Average the sum by dividing by the total number of rows and columns in the template image.
   c. Subtract each pixel intensity value of the template from the mean value to obtain a zero mean center template.

ii. Convolve the template over the given image using Matched-Spatial Filter.

$$MSF[r,c] = \sum_{dr=-\frac{W_r}{2}}^{+\frac{W_r}{2}} \sum_{dc=-\frac{W_c}{2}}^{+\frac{W_c}{2}} \left[I[r+dr, c+dc] * T_0 \left[dr + \frac{W_r}{2}, dc + \frac{W_c}{2} \right] \right]$$

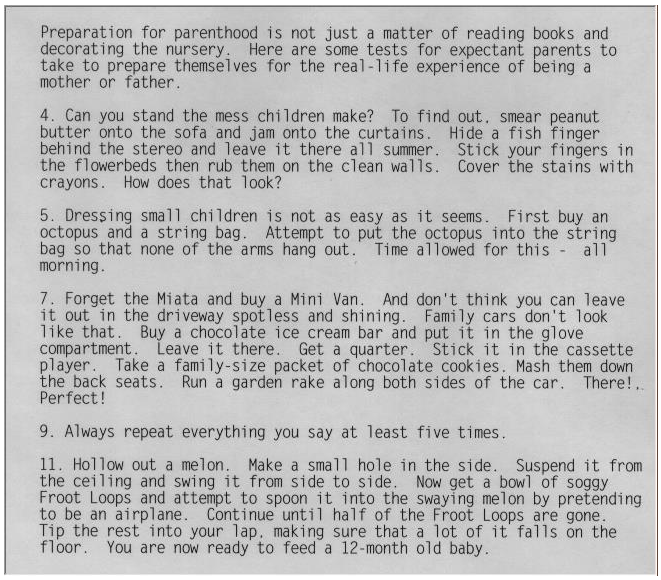
where I is the image, r is the row-number of the image, c is the column number, T_0 is the zero mean center template, W_r is the number of rows of the template image, and W_c is the number of columns of the template image.

2. Normalization of Matched-Spatial Filter:

i. The pixel values that were obtained using MSF were too large to be directly written to an image file. Thus, normalization was performed to change the range of pixel intensity values to fit between 0 to 255.
   a. Find the minimum and maximum pixel intensity value in the intermediate MSF image ("pixel values" in the code). The memory allocated for the intermediate MSF image was of type float to not lose the precision of the intensity values.
   b. The formula used for normalization was:

$$MSF_{normalized} = (I - \text{Minimum}) * \frac{\text{newMax} - \text{newMin}}{Maximum - Minimum} + newMin$$

newMax and newMin are 255 and 0 respectively. Minimum and Maximum are the intermediate MSF pixel intensity values.



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 Given Image

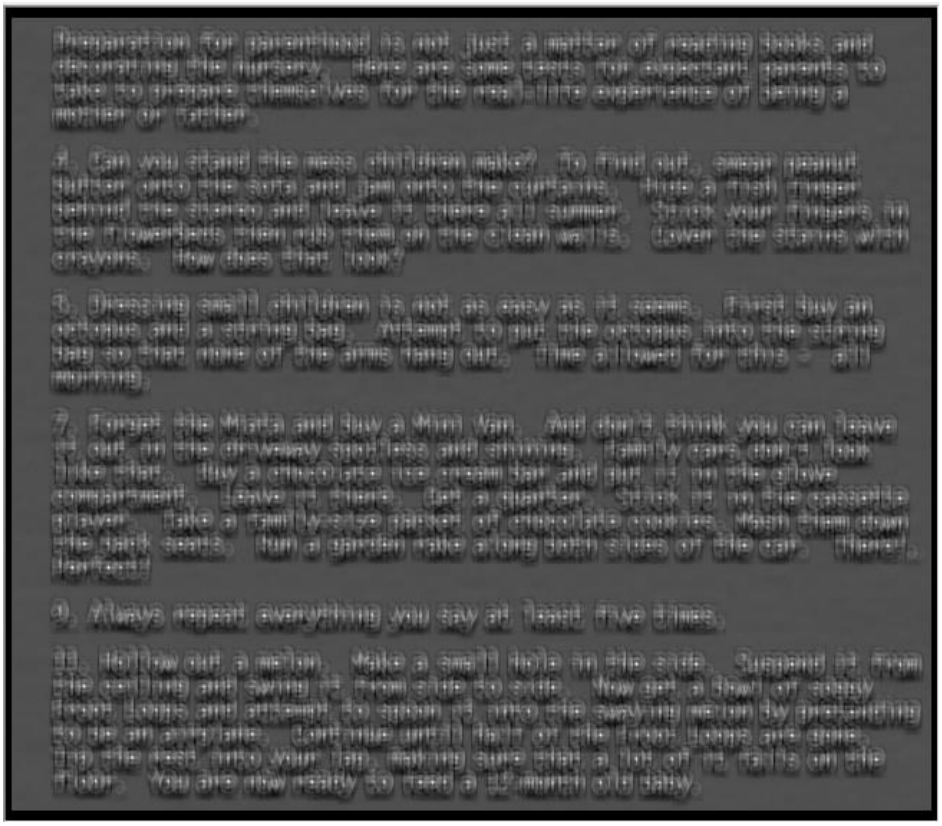


Figure 2 Normalized MSF Image

- The brightest white dots seen in the above image is the center location that the MSF filter calculated of the template letter ‘e’.
- It can also be seen that the MSF calculates the position of those letters which were the ‘e’ according to the filter but is a different filter.
- To be able to characterize the performance of the matched spatial curve, a ROC curve was generated.

3. ROC Curve

i. ROC curve is the plot of False Positive Rate vs True Positive Rate which is a measure of how well the algorithm detected a certain template (letter ‘e’) in the image.

ii. To be able to calculate those values the True Positive, True Negative, False Positive, and False Negative were generated.

iii. A detection is true positive if the detection of the position of the letter by the algorithm matches against the position of the letter in the ground truth. True negative is when there is the algorithm certainly detects that the letter isn’t an e at that position against the ground truth location.

iv. False Positive is when the algorithm detected an e at that position but the ground truth location does not mention an e, and false negative is when the algorithm failed to detect the letter e but the ground truth indicates the presence of letter e at that position.

v. To calculate these values, ground truth letter locations were looped through. A $9 * 15$ pixel area centered at the ground truth location was checked.

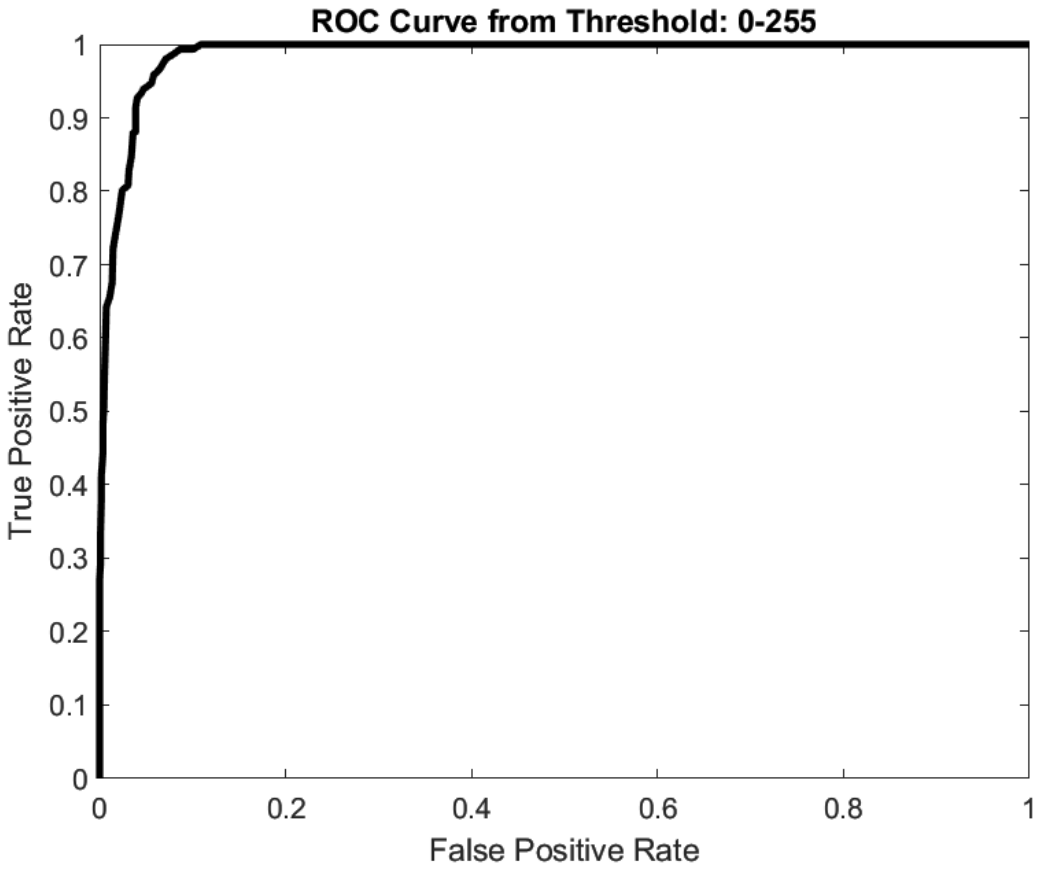
vi. The MSF image was compared against the ground truth location using a range of threshold values. If any pixel in the MSF image was greater than the threshold, the letter was considered detected. If none of the pixels in the $9 * 15$ areas were greater than the threshold, the letter was considered not detected.

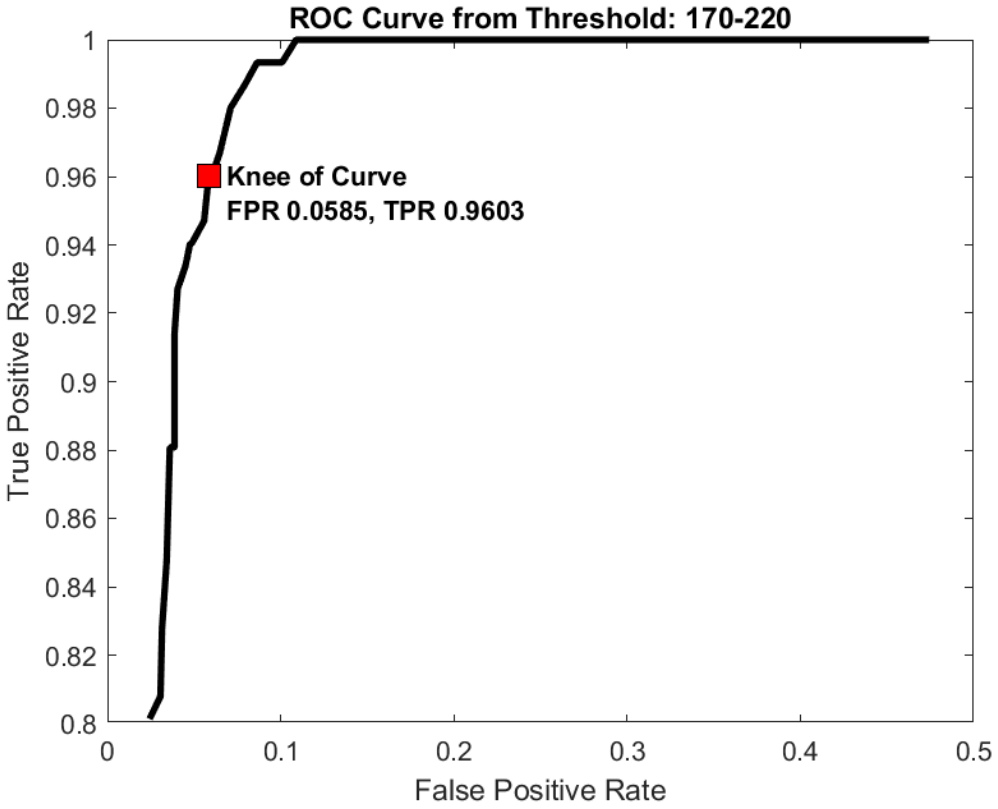
vii. The full 0-to-255 range of values was considered for the threshold to observe the evolution of the ROC curve as the threshold gets closer to 255.

viii. After obtaining the TP, FP, TN, and FN values the True Positive Rate and the False Positive Rate were calculated as:

$$TPR = \frac{TP}{TP + FN}$$

$$FPR = \frac{FP}{FP + TN}$$





- The ideal point would be where the True Positive Rate is 1 and False Positive Rate is 0. This would indicate that the algorithm did not misclassify any detections.
- However, that is only an ideal case. To obtain the point closest to FPR=0 and TPR=1 which is the "Knee of Curve", the Euclidean distance was calculated as follows:

$$Distance = \sqrt{(TPR - 1)^2 + (FPR - 0)^2}$$

$$Best\ Distance = \min_{\text{argwhere}} distance$$

Threshold	*Best Distance*	*Best TPR*	*Best FPR*	*TP Count*	*FP Count*
208	**0.0707**	**0.9603**	**0.0585**	**145**	**65**

Appendix

MATLAB Code for plotting ROC Curve

```
clc
clear all
close all
data = load('TPR_FPR.txt');
data1 = load('TP_FP.txt');
TP = data1(:,2);
FP = data1(:,3);
TPR = data(:,1);
FPR = data(:,2);

for i=1:length(FPR)
    dist(i) = sqrt((TPR(i) - 1)^2 + (FPR(i) - 0)^2);
end

mindist = min(dist);
index = find(mindist == dist)
TPR_min = TPR(index)
FPR_min = FPR(index)
TP_min = TP(index)
FP_min = FP(index)
Actual_Index = index-1

figure()
plot(FPR, TPR,'linewidth',2.5,'color','black');
xlabel('False Positive Rate')
ylabel('True Positive Rate')
title('ROC Curve from Threshold: 0-255')
figure()
plot(FPR(170:221), TPR(170:221),'linewidth',2.5,'color','black');
hold on
plot(FPR(index), TPR(index),
'square','MarkerSize',11,'MarkerEdgeColor','black', 'MarkerFaceColor','red')
xlabel('False Positive Rate')
ylabel('True Positive Rate')
title('ROC Curve from Threshold: 170-221')
% xlim([0 1]);
% ylim([0 1]);
textspec2='Knee of Curve';
Text_on_plot2=sprintf(textspec2);
text(FPR(index)+0.01,TPR(index),Text_on_plot2,'color','black','FontSize',10,
'FontWeight','bold')
textspec23='FPR %.4f, TPR %.4f';
Text_on_plot23=sprintf(textspec23,FPR(index),TPR(index));
text(FPR(index)+0.01,TPR(index)-
0.005,Text_on_plot23,'color','black','FontSize',10,'FontWeight','bold')
```

C Code

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

int main(int argc, char *argv[]){
    FILE *fpt, *temp_fpt;
    unsigned char *image;
    unsigned char *Template;
    unsigned char *MSF_Image;
    unsigned char *BinaryImage;
    float *ZeroMeanTemplate;
    float *pixelvalue;
    char header[80], header_temp[80];
    int TEMP_COLS, TEMP_ROWS, TEMP_BYTES;
    int COLS,ROWS,BYTES;
    float sum;
    int r,c,iter_row=7,iter_col=4,dr,dc,i;
    int final, initial, num_values, step;
    int *ThresholdValues;

    if (argc!=4){
        printf("Error - Pass Arguments: Executable file | Image File | Templat
e Image File | Ground Truth Text File \n");
        exit(0);
    }

    /* read input image */
    if ((fpt=fopen(argv[1],"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(" Parenthood 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);
```

```
    /* Read template image*/
    if ((temp_fpt=fopen(argv[2],"rb")) == NULL)
    {
    printf("Unable to open parenthood_e_template.ppm for reading\n");
    exit(0);
    }

    fscanf(temp_fpt,"%s %d %d %d ",header_temp,&TEMP_COLS,&TEMP_ROWS,&TEMP_BYT
ES);
    if (strcmp(header_temp,"P5") != 0 ||  TEMP_BYTES != 255 )
    {
    printf("Parenthood template Not a greyscale 8-bit PPM image\n");
    exit(0);
    }

    Template=(unsigned char *)calloc(TEMP_ROWS*TEMP_COLS,sizeof(unsigned char)
);
    // header_temp[0]=fgetc(temp_fpt);  /* read white-
space character that separates header */
    fread(Template,1,TEMP_COLS*TEMP_ROWS,temp_fpt);
    fclose(temp_fpt);

    MSF_Image = (unsigned char *)calloc(ROWS*COLS,sizeof(unsigned char));
    pixelvalue = (float *)calloc(ROWS*COLS, sizeof(float));
    ZeroMeanTemplate = (float *)calloc(TEMP_ROWS*TEMP_COLS,sizeof(float));

    // Zero Mean Center Template
    float sum_template = 0.0;
    float mean_template = 0.0;

    for(int m=0; m<TEMP_ROWS*TEMP_COLS;m++){
        sum_template+= Template[m];
    }
    mean_template = sum_template/(TEMP_COLS*TEMP_ROWS);

    for(int m=0; m<TEMP_ROWS*TEMP_COLS;m++){
        ZeroMeanTemplate[m] = Template[m] - mean_template;
    }
    // printf("%f \n", mean_template);

    // Computing Cross Correlation
    float minPixelValue = 99999999999999;
    float maxPixelValue = 0;
    for(r=0; r<ROWS; r++){
        if((r-iter_row)>=0 && (r+iter_row)<ROWS){
            for(c=0;c<COLS; c++){
                sum=0.0;
                if((c-iter_col)>=0 && (c+iter_col)<COLS){
```

```
                    for(dr=-TEMP_ROWS/2; dr<=TEMP_ROWS/2; dr++){
                        for(dc=-TEMP_COLS/2; dc<=TEMP_COLS/2; dc++){
                            sum+= (image[(r+dr)*COLS+(c+dc)]*ZeroMeanTemplate[
(dr+TEMP_ROWS/2)*TEMP_COLS+(dc+TEMP_COLS/2)]);
                        }
                    }
                    pixelvalue[r*COLS+c] = sum;
                    if(sum > maxPixelValue){
                        maxPixelValue=sum;
                    }
                    if(sum < minPixelValue){
                        minPixelValue=sum;
                    }
                }
            }
        }
    }

    // Normalization
    float PixelRange=maxPixelValue-minPixelValue;
    float newmax=255.0;
    float newmin=0.0;
    float newrange = newmax - newmin;
    float normalized;
    float factor;

    for(r=0; r<ROWS;r++){
        for(c=0; c<COLS; c++){
            if(r-iter_row>=0 && r+iter_row<ROWS && c-
iter_col>=0 && c+iter_col<COLS){
                factor = newrange / PixelRange;
                normalized = (pixelvalue[r*COLS+c] - minPixelValue)*factor;
                int f = round(normalized);
                MSF_Image[r*COLS+c] = f;
            }
        }
    }

    // Get array of threshold values. Outputs 256 values from 0-to-255
    initial = 0;
    final = 255;
    num_values = 255;
    ThresholdValues = (int *)calloc(num_values,sizeof(int));
    step = (final - initial) / num_values;
    int j = 0;
    int ct = 0;
    ThresholdValues[j] = initial;
    for(j = 1; j<=num_values; j++){
```

```
        ThresholdValues[j] = ThresholdValues[j-1] + step;
        // printf("%d  ", ThresholdValues[j]);
        ct+=1;
    }
    // printf("%d \n",ct);

    // ROC Curves
    FILE *gt_fpt;
    int GT_R,GT_C;
    char letter;
    float TPR=0.0;
    float FPR=0.0;
    float *TPR_Array = (float *)calloc(num_values+1,sizeof(float));
    float *FPR_Array = (float *)calloc(num_values+1,sizeof(float));
    int *TP_Array = (int *)calloc(num_values+1,sizeof(int));
    int *FP_Array = (int *)calloc(num_values+1,sizeof(int));
    int *T = (int *)calloc(num_values+1,sizeof(int));

    for (int k=initial; k<=final;k++){
        // Create Binary Image
        BinaryImage = (unsigned char *)calloc(ROWS*COLS,sizeof(unsigned char))
;
        for (r=0; r<ROWS; r++){
            for(c=0;c<COLS;c++){
                if(MSF_Image[r*COLS+c]>ThresholdValues[k]){
                    BinaryImage[r*COLS+c] = 255;
                }
                else if(MSF_Image[r*COLS+c]<=ThresholdValues[k]){
                    BinaryImage[r*COLS+c] = 0;
                }
            }
        }
        int TP=0;
        int FP=0;
        int TN=0;
        int FN=0;
        gt_fpt = fopen(argv[3],"rb");
        if (gt_fpt == NULL)
        {
        printf("Unable to open ocr.txt for reading\n");
        exit(0);
        }
        // Calculate TPR and FPR
        while(1){
            int detected=0;
            int not_detected=0;
            i=fscanf(gt_fpt,"%s %d %d",&letter,&GT_C,&GT_R);
            if(i!=3){
```

```
                break;
            }
            else{
                for(dr=-TEMP_ROWS/2; dr<=TEMP_ROWS/2; dr++){
                // If a detection was already found before finishing looping t
hrough the 9*15 window, then it skips the rest of looping
                    if (detected==0){
                        for(dc=-TEMP_COLS/2; dc<=TEMP_COLS/2; dc++){
                                if(MSF_Image[(GT_R+dr)*COLS+(GT_C+dc)] > Thres
holdValues[k]){
                                    detected=1;
                                }
                            }
                        }
                    }
                // If no detections found
                if(detected==0){
                    not_detected = 1;
                }
                if(detected==1 && letter=='e'){
                    TP++;
                }
                else if(detected==1 && letter!='e'){
                    FP++;
                }
                else if(not_detected==1 && letter!='e'){
                    TN++;
                }
                else if(not_detected==1 && letter=='e'){
                    FN++;
                }
            }
        }
        fclose(gt_fpt);
        TPR = TP/(float)(TP+FN);
        FPR = FP/(float)(FP+TN);
        TP_Array[k] = TP;
        FP_Array[k] = FP;
        T[k] = k;
        TPR_Array[k] = TPR;
        FPR_Array[k] = FPR;

        // printf("\n True Positive Rate|%f \n", TPR);
        // printf("False Positive Rate|%f \n", FPR);
        // printf("T|%d True Positive|%d False Positive|%d True Negative|%.d F
alse Negative|%d \n",k,TP,FP,TN,FN);

    }
```

```
    fpt=fopen("TPR_FPR.txt","wb");
    if (fpt == NULL)
    {
    printf("Unable to open TPR_FPR.txt for writing\n");
    exit(0);
    }
    for(int t=initial; t<=final;t++){
        fprintf(fpt,"%f %f\n",TPR_Array[t], FPR_Array[t]);
        // printf("%f  ", TPR_Array[t]);
    }
    fclose(fpt);


    fpt=fopen("TP_FP.txt","wb");
    if (fpt == NULL)
    {
    printf("Unable to open FP_TP.txt for writing\n");
    exit(0);
    }
    for(int t=initial; t<=final;t++){
        fprintf(fpt,"%d %d %d\n",T[t], TP_Array[t], FP_Array[t]);
        // printf("%f  ", FPR_Array[t]);
    }
    fclose(fpt);

    //MSF Image
    fpt=fopen("MSF_Image.ppm","wb");
    if (fpt == NULL)
    {
    printf("Unable to open temp.ppm for writing\n");
    exit(0);
    }
    fprintf(fpt,"P5 %d %d 255\n",COLS,ROWS);
    fwrite(MSF_Image,1,ROWS*COLS,fpt);
    fclose(fpt);
}
```