CLEMSON UNIVERSITY INTERNATIONAL CENTER FOR AUTOMOTIVE RESEARCH

ECE 6310 Introduction to Computer Vision

Fall 2020

Lab 2 – Optical character recognition

Submitted By –

Siddhant Srivastava,

C17547202

Matched Spatial Filter:

Matched spatial filter or MSF is best way to find a template matched image in case of convolution. The reason being that this filter is not affected by the brightness of the image.

The template image that is zero mean centered and then it is applied in the filter using the equation:

$$MSF[r,c] = \sum_{dr=-Wr/2}^{+Wr/2} \sum_{dc=-Wc/2}^{+Wc/2} [I[r+dr, c+dc] * T[dr + Wr\ /\ 2, dc + Wc\ /\ 2]]$$

However, the MSF image obtained is not a bit image. Hence, this image is normalized to obtain a bit image.

The obtained MSF image after normalization in lab 2 is shown below.

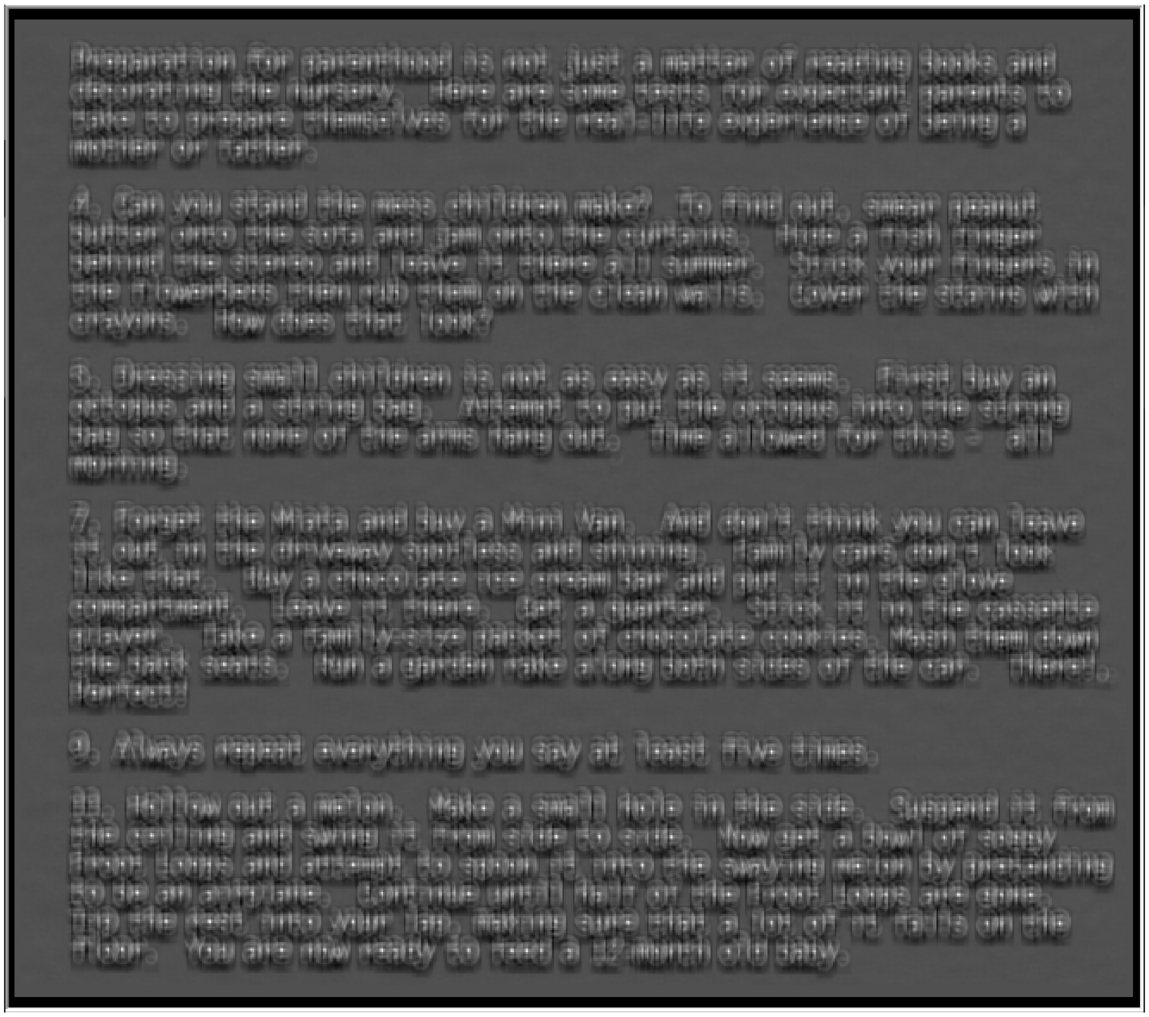
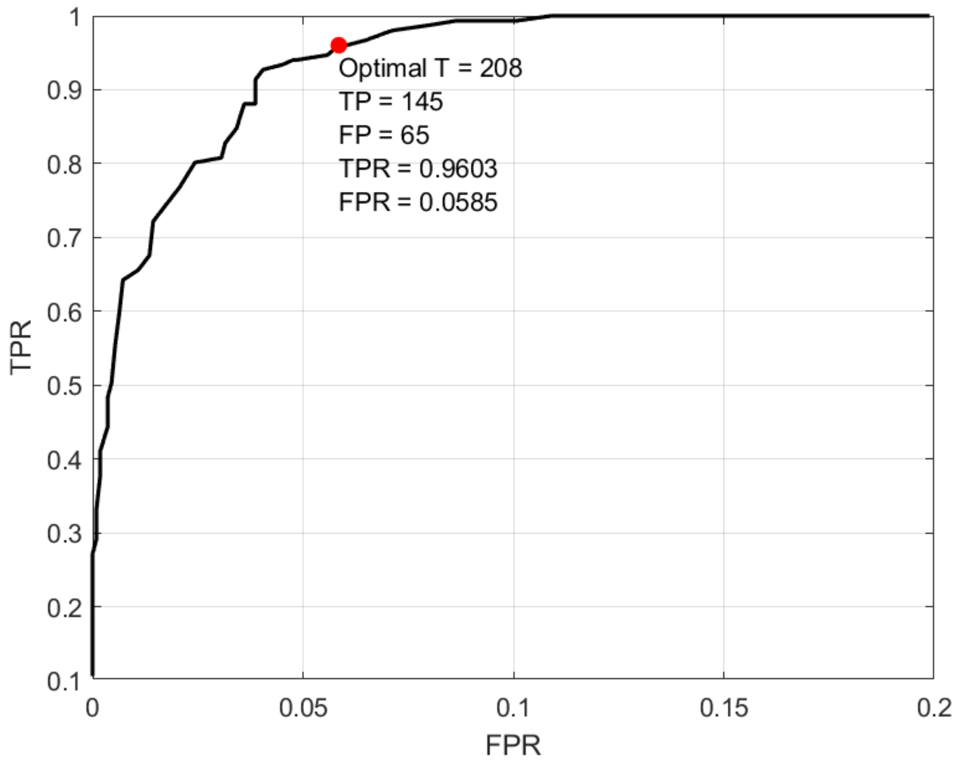


Figure 1: MSF Image

After obtaining the MSF image, the ROC curve was obtained for range of threshold values (T) from 190 to 245. The ROC curve was plotted after outputting the TP, FP, TN, FN, TPR and FPR values for each T in a .csv file named Outputs.csv. The ROC curve was plotted using MATLAB.



The optimal T, FP and TP values can be obtained if the x-coordinate (FPR value) of the point on ROC curve is close to 0 (zero) and the y-coordinate (TPR value) of the same point is close to 1. This is calculated by finding the minimum of all the Euclidean distances of FPR and TPR values for each T.

For the given image, the optimal T is 208. The corresponding TP value is 145 and FP value is 65, as shown in the ROC curve above.

C code:

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

int main()
{
    FILE    *fpt;
    FILE    *fpt1;
    unsigned char *image;
    unsigned char *otemplate;
    float *template;
    float *msf_bit;
    unsigned char *msf;
    unsigned char *binaryimage;
    char Header[320], header[320], gt;
    int ROWS,COLS,BYTES,rows,cols,bytes,sysrow,syscol,tp,fp,tn,fn,thres,i,detecte
d,not_detected;
    int r,r2,c,c2;
    double sum,sumtemp,omin,omax,tpr,fpr;

        /* read 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);

        /* read template image */
    if ((fpt=fopen("parenthood_e_template.ppm","rb")) == NULL)
        {
        printf("Unable to open parenthood_e_template.ppm for reading\n");
```

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

        /* Zero mean template */
    template=(float *)calloc(rows*cols,sizeof(float));
    sumtemp = 0.0;

    for (r2=0; r2<rows*cols; r2++) {
        sumtemp+=otemplate[r2];
    }

    for (r2=0; r2<rows*cols; r2++) {
        template[r2]=otemplate[r2]-(sumtemp/(rows*cols));
        }

        /* allocate memory for msf bits version of image */
    msf_bit=(float *)calloc(ROWS*COLS,sizeof(float));

    omax = 0.0;
    omin = 9999999999.0;

        /* msf bits version of image */
    for (r=0; r<ROWS; r++) {
        if (r<7 || r>=ROWS-7)
            continue;
        else {
            for (c=0; c<COLS; c++)
            {
                if(c<4 || c>=COLS-4)
                    continue;
                else {
                    sum=0.0;
                    for (r2=-(rows/2); r2<=(rows/2); r2++)
                        for (c2=-(cols/2); c2<=(cols/2); c2++)
```

```
                            sum+=(image[(r+r2)*COLS+(c+c2)]*template[(r2+(rows/2)
)*cols+(c2+(cols/2))]);
                }
            if (omax<=sum)
                omax = sum;
            if (omin>=sum)
                omin = sum;
            msf_bit[r*COLS+c]=sum;
            }
        }
    }

        /* normalising the image to obtain msf image */
    msf=(unsigned char *)calloc(ROWS*COLS,sizeof(unsigned char));

    for (r=0; r<ROWS; r++) {
        for (c=0; c<COLS; c++)
        {
            msf[r*COLS+c] = round((msf_bit[r*COLS+c]-omin)*(255.0/(omax-omin)));
        }
    }

        /* Calculating ROC for range of Threshold values */
    printf("Creating Outputs.csv file\n");
    fpt1=fopen("Outputs.csv","w");
    fprintf(fpt1,"Threshold,TP,FP,TPR,FPR\n");

    for (thres=190; thres<=245; thres++) {

        binaryimage = (unsigned char *)calloc(ROWS*COLS,sizeof(unsigned char));
        for (r=0; r<ROWS; r++){
            for(c=0;c<COLS;c++){
                if(msf[r*COLS+c]>thres){
                    binaryimage[r*COLS+c] = 255;
                }
                else if(msf[r*COLS+c]<=thres){
                    binaryimage[r*COLS+c] = 0;
                }
            }
        }

            /* Reading the ground truth data */
        if ((fpt=fopen("parenthood_gt.txt","r")) == NULL)
            {
            printf("Unable to open parenthood_gt.txt for reading\n");
            exit(0);
            }

        tp=fp=tn=fn=0;
        while (1)
        {
            detected=0;
            not_detected=0;
            i = fscanf(fpt,"%s %d %d",&gt,&syscol,&sysrow);
            if (i!=3)
                break;
            else {
                for (r=-(rows/2); r<=(rows/2); r++) {
                    for (c=-(cols/2); c<=(cols/2); c++) {
                        if (msf[(sysrow+r)*COLS+(syscol+c)]>thres)
                            detected=1;
                    }
                }
                if(detected==0)
                    not_detected = 1;

                if (gt=='e' && detected==1)
                    tp++;
                else if (gt!='e' && detected==1)
                    fp++;
                else if (gt!='e' && not_detected==1)
                    tn++;
                else if (gt=='e' && not_detected==1)
                    fn++;
            }
        }
        fclose(fpt);

        tpr = (double)(tp)/(tp+fn);
        fpr = (double)(fp)/(fp+tn);

            /* Outputting TP, FP, TPR and FPR for each Threshold (T) in Outputs.c
sv */
        fprintf(fpt1,"%d,%d,%d,%lf,%lf\n",thres,tp,fp,tpr,fpr);
    }
    printf("Outputs.csv file created");
    fclose(fpt1);

        /* write out MSF image to see result */
    fpt=fopen("msf.ppm","wb");
    fprintf(fpt,"P5 %d %d 255\n",COLS,ROWS);
    fwrite(msf,COLS*ROWS,1,fpt);
    fclose(fpt);
}
```