**ECE 6310**

**Introduction to Computer Vision**

**Fall 2022**

Lab 2

Optical Character Recognition

Harshal B. Varpe

**Introduction**

In this project/lab, the students were asked to implement a matched filter, otherwise known as normalized cross-correlation or matched spatial filtering, to recognize letters in an image of text. 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.'

Text, letter

Description automatically generated A picture containing blur

Description automatically generated

Figure 1: Input image (parenthood.ppm) Figure 2: Template ‘e’

**Implementation**

In character recognition problems, it is possible that the template and the input image may not have the same brightness. In such cases, we rely upon matched spatial filtering. In this method, we first create a zero-mean centered template, then we convolve this template with the given image. In the last step, we find a suitable threshold to find actual matches. The equation to create a matched spatial filter is given as –

A picture containing diagram

Description automatically generated

During the process's convolution step, the pixels' range increases greater than 8-bit. Hence the result of such convolution cannot be stored in unsigned characters. Instead, we store the result of convolution in an array of int. However, to get back to the 8-bit format, we normalize. A general equation to normalize the given data is –

Xnormalized = (X-Xmin) \*255 / (Xmax-Xmin)

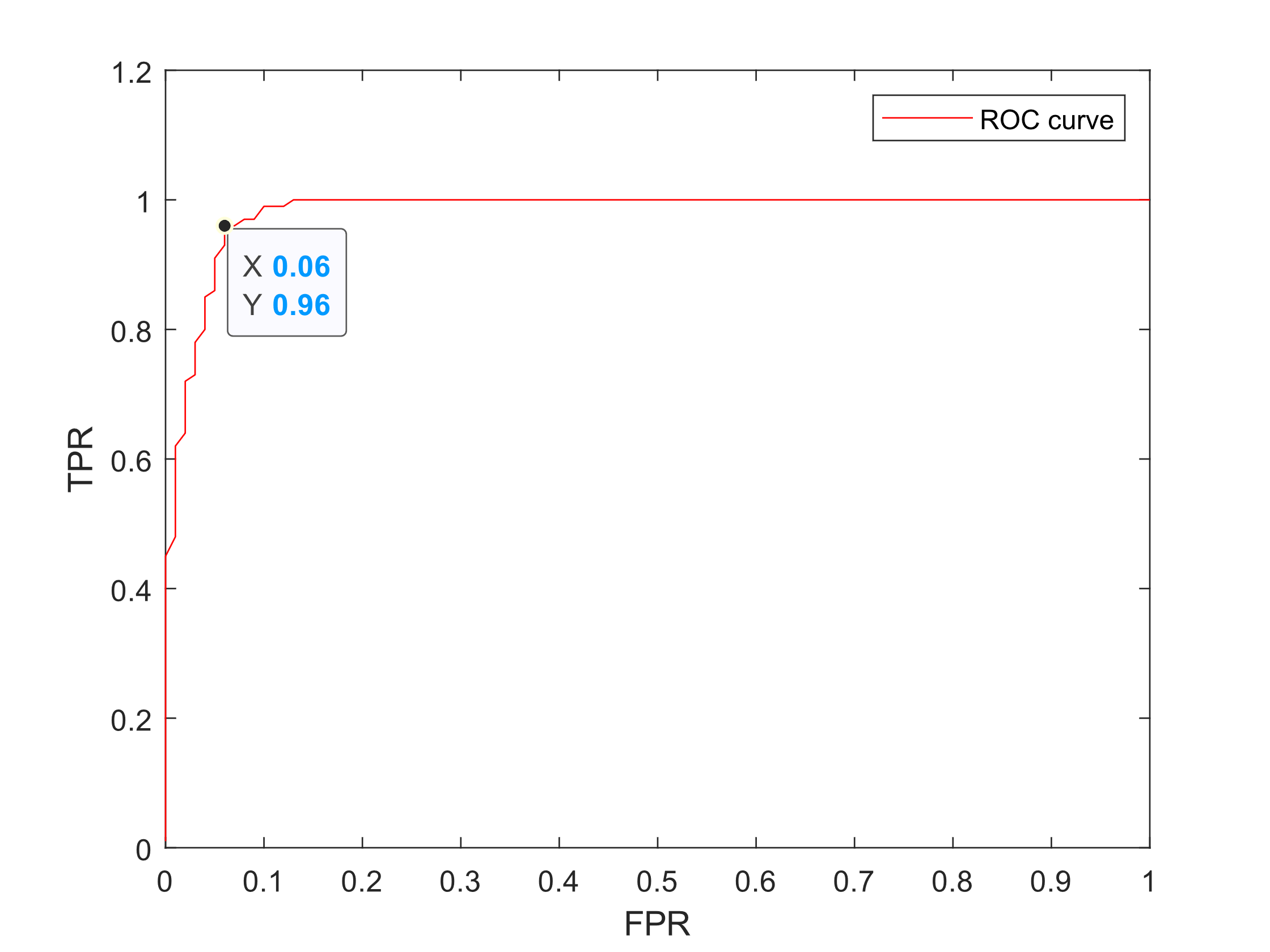
Text

Description automatically generated  
Figure 3: Normalized Image

We can see the tiny bright pixels. These bright pixels are all the possible locations of the letter 'e.' There is a problem here – How to be sure that the bright spot is the letter 'e'? It is possible that some of the bright spots are not the letter 'e.' Here, we rely on thresholding and satisfying a condition where we check if the letter from ground truth and letter 'e' match.

The possible values for threshold T range from 0 to 255. For a single value of threshold T, we create a binary image. If a pixel value in a normalized MSF image is above the said threshold, then for the same pixel in a binary image value is set to 255. Otherwise, the value is set to 0. Then, if the pixel value in binary image is above threshold T for a specific range of pixels, then we set the variable "detected" to 1. Otherwise, we set the variable "non\_detected" value to 1. Along with the variables "detected" and "non-detected," If the given letter, in our case the letter 'e,' and the letter read from the ground truth file match, 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.

Background pattern

Description automatically generated  
Figure 4: Binary Image at Threshold 206  
  
Figure 5: ROC curve

The figure above shows that the behavior of the ROC curve TPR (0.96) and FPR (0.06) 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 206. At the optimal threshold, there were 142 True positives and 67 False positives.

**Code**

// ECE 6310 Introduction to CV

// Lab 2 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;

    int min,max,sum,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;

    float TPR,FPR,mean; int \*temp\_img;

    double \*img\_msf;

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);

// printf("%d this is temp\_r \n",temp\_R);

// printf("%d \n",template\_img[12]);

/\* Calculating a ZMC template \*/

mean = 0;

for(j=0;j<(temp\_R\*temp\_C);j++){

    mean += (float)template\_img[j];

}

mean = mean / (float)(temp\_R \* temp\_C);

// printf("%d",(int)mean);

/\* We will store the zmc template.\*/

temp\_img = (int \*)calloc(temp\_R\*temp\_C,sizeof(int));

// temp\_img = (double \*)calloc(temp\_R\*temp\_C,sizeof(double));

// temp\_img = (float \*)calloc(temp\_R\*temp\_C,sizeof(float));

for(j=0;j<(temp\_R\*temp\_C);j++){

    temp\_img[j] = (int)((float)template\_img[j] - mean);

}

// for(int l=0;l<ROWS\*COLS;l++){printf("%d \n",temp\_img[l]);}

// printf("%f is mean \n",mean);

// /\* Lets us create an MSF version of original image. \*/

img\_msf = (double \*)calloc(ROWS\*COLS,sizeof(double));

for(r = (temp\_R/2);r<=(ROWS-(temp\_R/2));r++){

    for(c = (temp\_C/2);c<=(COLS-(temp\_C/2));c++){

        sum=0;

        for(r1 = -(temp\_R/2);r1<=(temp\_R/2);r1++){

            for(c1 = -(temp\_C/2);c1<=(temp\_C/2);c1++){

                sum += (temp\_img[(r1+(temp\_R/2))\*temp\_C+(c1+(temp\_C/2))] \* input[(r+r1)\*COLS+(c+c1)]);

            }

        }

        img\_msf[r\*COLS+c] = sum;

    }

}

/\* normalization of the msf image \*/

min =  5000; max = 0;

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

    if(img\_msf[j]>max){max = img\_msf[j];}

    else if(img\_msf[j]<min){min = img\_msf[j];}

}

// printf("%d is min \n",min);

// printf("%d is max \n",max);

img\_msf\_norm = (unsigned char \*)calloc(ROWS\*COLS,sizeof(unsigned char));

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

    img\_msf\_norm[j] = ((img\_msf[j]-min)\*255)/(max-min);

}

fpt = fopen("normalized\_msf.ppm","w");

fprintf(fpt,"%s %d %d 255",header,COLS,ROWS);

fwrite(img\_msf\_norm,1,ROWS\*COLS,fpt);

fclose(fpt);

// /\* creating a text file to write the threshold and ROC curve data\*/

out  = fopen("confusion\_mat.txt","a");

fprintf(out,"T TP FP FN TN TPR FPR \n");

// printf("%c \n",letter);

for(T=0;T<=255;T++){

    // for point 4.a

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

    for(j=0;j<(ROWS\*COLS);j++){

        if (img\_msf\_norm[j]>=T){binary[j]=255;}

        else{binary[j]=0;}

    }

    // Although in the

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

    TP=TN=FP=FN=0;

    while(1){

        i = fscanf(g\_truth,"%s %d %d",gt\_letter,&gt\_col,&gt\_row);

        detected = not\_detected = 0;

        if(i!=3){break;}

        else {

            for(r=gt\_row-7;r<=gt\_row+7;r++){

                for(c=gt\_col-4;c<=gt\_col+4;c++){

                    if(binary[r\*COLS+c]>=T){detected = 1;}

                    else{not\_detected = 1;}

                }

            }

            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 ;}

        }

    }

    // printf("%c is gt \n", gt\_letter);

    // printf("%c is input \n",letter);

    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);

    fclose(g\_truth);

    // printf("%d",T);

}

fclose(out);

printf("This is the end my friend!");

/\* binary image with chosen threshold \*/

// The variable final is used to store a binary image

T = 206; // Knee point of ROC

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

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

        if (img\_msf\_norm[j]>=T){final[j]=255;}

        else{final[j]=0;}

    }

fpt = fopen("final\_output.ppm","w");

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

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

fclose(fpt);

}