Lab#2 – Optical Character Recognition

Name: Saroj Kumar Dash
Course: ECE-6310
Course Name: Intro to Computer Vision

Problem Statement:

To implement an image with MSF image filter applied on it and the TPR, FPR value is calculated for each Threshold value and an ROC curve is plotted to find out the knee of the operation i.e. the best place where the TPR and FPR are optimum.

Solution :-

In below code I compute the matched Spatial Filter image of an input

Code:-

```
/*
Explanation: The below function takes the array of pixels(integer type) of size inC*inR and then normalizes its pixels to 0 to 255 which is a unisgned char array of size inC*inR
*/

unsigned char *normalizePixels(int *iOldPix,int inC,int inR){

	//find oldmin an old max
	int oldMin = iOldPix[0];
	int oldMax = iOldPix[0];

	for(int i = 0;i<(inC*inR);i++){
		if(iOldPix[i] > oldMax)
			oldMax = iOldPix[i];
		else if(iOldPix[i] < oldMin)
			oldMin = iOldPix[i];
	}

	//use the oldMin and oldMax to get the normalized value
	unsigned char *opPixels = new unsigned char[inR*inC];
	for(int r=0;r<inR;r++){
		for(int c=0;c<inC;c++){
			int ithPix = r*inC+c;
			float normFactor = (float)(iOldPix[ithPix]-oldMin)/(float)(oldMax-oldMin);
			opPixels[ithPix] = (int)((float)normFactor*255);
		}
	}

	return opPixels;
}
```

```
/*
Explanation:- The main convolution operation is done here
whoever uses this method should delete the returned pointer to free the memory
We take the image and the kernel image and the MSF converted kernel image
This function returns a MSF kernel image convoluted over the main input image.
*/
unsigned char *convolveSimple(image &iImg,kernel &k,int **kMSF_t){

   unsigned char *opPixels = NULL;
   int *tmpPixels = NULL;
   int r,c;
   int sum;
   int ir=iImg.getRows(),ic= iImg.getCols(),kr = k.getRows(),kc =
k.getCols();

   tmpPixels = new int[iImg.getRows() * iImg.getCols()];


   //Do convolve operation here
  for(r=(kr/2); r< (ir-(kr/2)); r++){
         for(c=(kc/2) ; c<(ic-(kc/2)); c++){
               tmpPixels[r*ic+c] = kernel2dConv(iImg,c,r,kMSF_t,k);
         }
  }

  opPixels = normalizePixels(tmpPixels,ic,ir);

  delete tmpPixels;
  return opPixels;

}

/*
Explanation: This Function takes as input the kernel image and returns the matched spatial
Filter image of the kernel image
*/

int **computeMSFMeanKernel(kernel &ker){

      unsigned int kerMean = 0;
      unsigned char **ppkPixels = ker.getPixels2dArray();
      int **ppOut = new int*[ker.getRows()];
      unsigned int sum = 0;

      //calculate the kernel mean
      for(int r=0; r<ker.getRows();r++){
            for(int c=0; c<ker.getCols();c++){
                  sum += ppkPixels[r][c];
            }
      }

      kerMean = sum/(ker.getRows()*ker.getCols());
```

```
	//std::cout<<"Kernel Sum = "<<sum<<std::endl;

	//subtract each cell of the kernel with the mean
	for(int r=0; r<ker.getRows();r++){
		ppOut[r] = new int[ker.getCols()];
		for(int c=0; c<ker.getCols();c++){
			ppOut[r][c] = ppkPixels[r][c]-kerMean;
			//std::cout<<ppOut[r][c]<<"   ";
		}
		//std::cout<<std::endl;
	}

	return ppOut;
}
```

/*
EXPLANATION: This function takes as input a kernel or template image and the original image and also threshold value.

Computes the MSFKernel of the template image.

Applies the convolution on the kernel from the output MSF kernel that we get.

After that we use the threshold value to obtain the binary image.

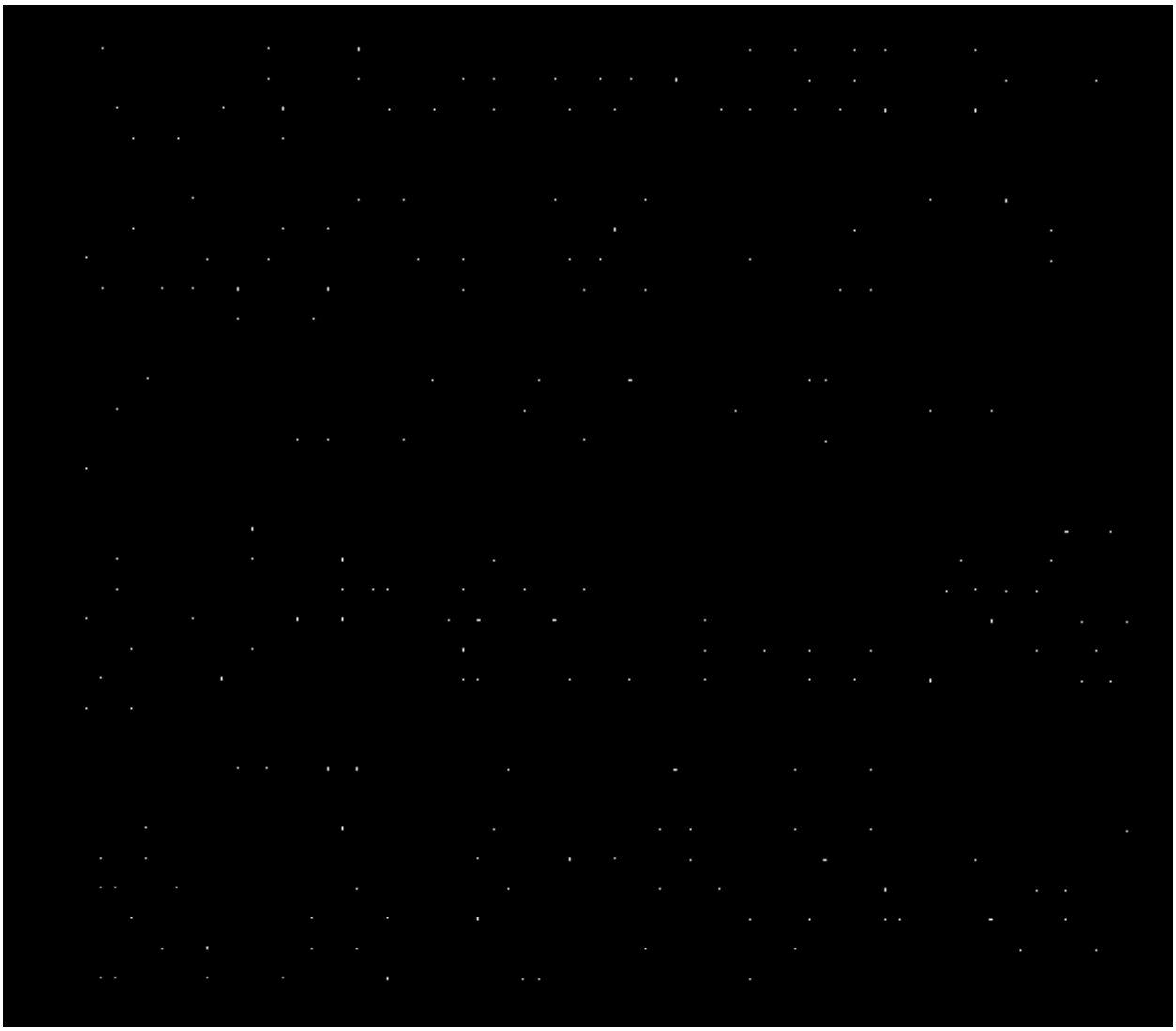
The final *image returned is a binary image.*
*/

```
unsigned char *getMSFImage(kernel &ker,image &iImg,int iT){

	//get the MSFtemplate mean kernel matrix
	int **ppMeanKernel = computeMSFMeanKernel(ker);

	//convolve the MSFMeankernel with the image
	unsigned char *pMSFImage = convolveSimple(iImg,ker,ppMeanKernel);

	//Debug: Without threshold MSFImage
	//saveImage(pMSFImage,iImg.getCols(),iImg.getRows());

	//loop through the image and make it binary
	for(int i=0;i<(iImg.getCols()*iImg.getRows());i++){

		//if the returned pixel is greater than the threshold
		if(pMSFImage[i]>=iT)
			pMSFImage[i]=255; //then set 255 on output image
		else
			pMSFImage[i]=0; //else set to 0
	}
```

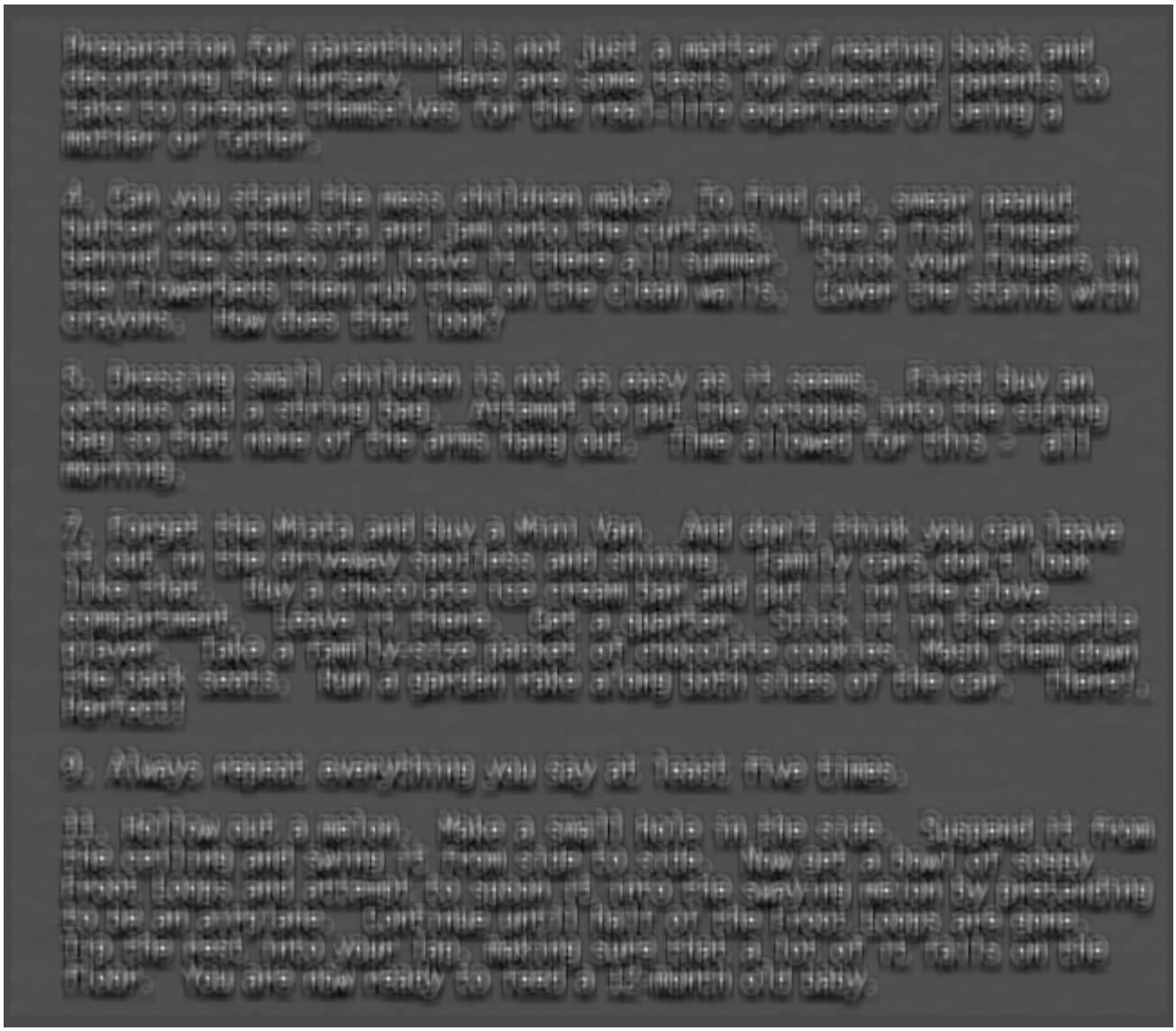
```
    return pMSFImage;
}
```

Output:-

Below is the Binary image that I get at threshold value of 210.



Below is the image after thte MSF filter is applied but the thresholding of the image is not done.



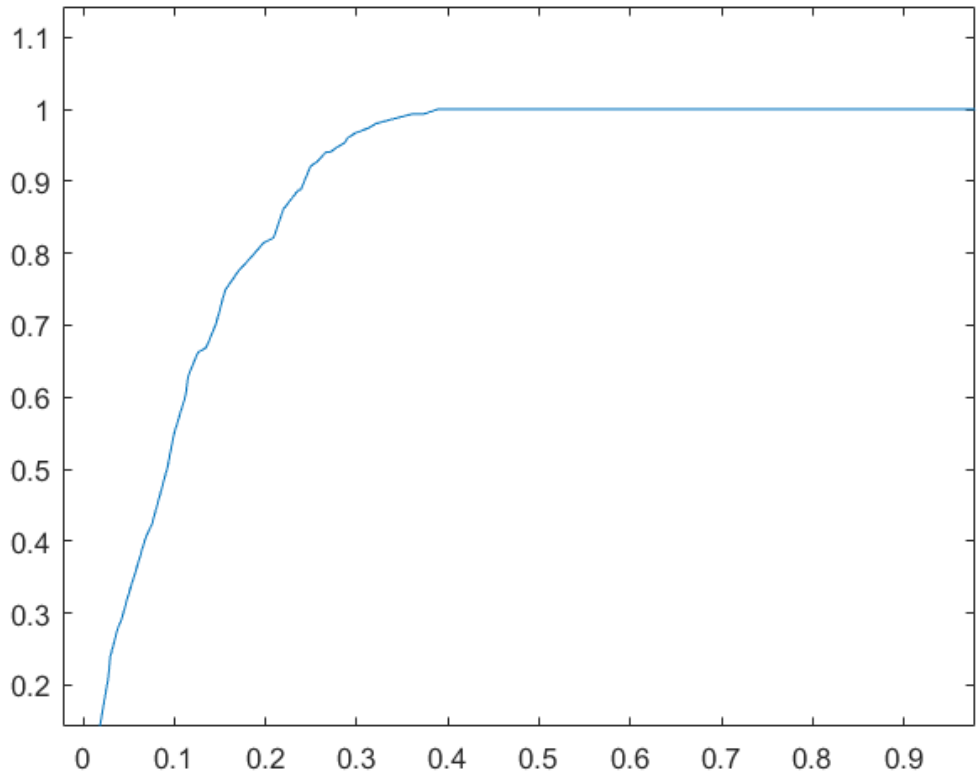
/*
TESTING of MY DETECTION:- After the image pixels are detected as ‘e’ or not ‘e’ we count the value of True Positive(TP) and False Positve(FP). The TP tells us how many e’s it detected and the value of FP tells how wrong is our program detecting the letter ‘e’. Since we can go through the whole ground-truth

text file so based on our detection we can also calculate the FN and TN values. Thus I calculated the TPR and FPR values and plotted the image based on the values that I obtained in this below function.

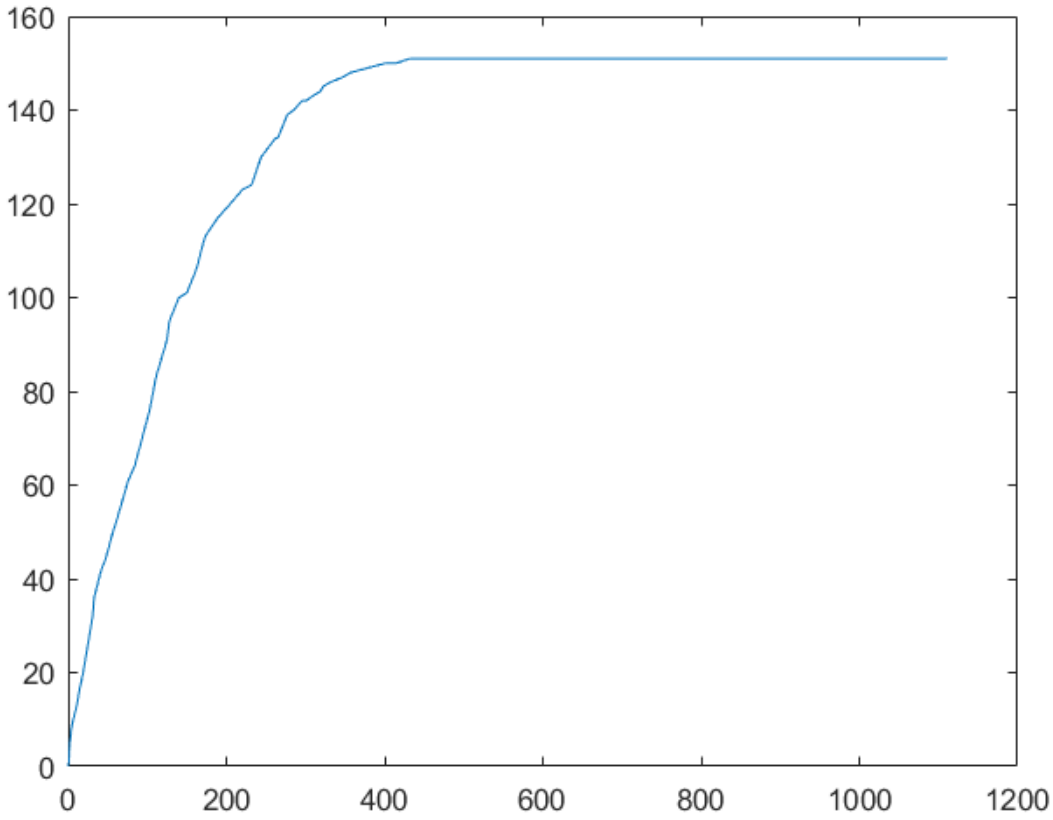
```
*/
void TestMSFImage(unsigned char *pMSFImage,char infileName[30],image
&img,kernel &ker){

	int imgR=img.getRows(),imgC=img.getCols();
	int nbTP=0,nbFP=0;
	int nbTN=0,nbFN=0;
	char letter;
	int gtR,gtC;

	//read GroudTruth file
	FILE *fpt = fopen(infileName,"rb");

	while(1){
		//Get GroudTruth value from file line by line
		int rc = fscanf(fpt,"%c %d %d\n",&letter,&gtC,&gtR);
		if(EOF == rc) break;

		bool isDetected = false;
		for(int r=gtR-ker.getRows();r<gtR+ker.getRows();r++){
			for(int c=gtC-ker.getCols();c<gtC+ker.getCols();c++){

				if(pMSFImage[(r*imgC)+c] == 255){
					isDetected=true;
					r=gtR+ker.getRows(); //this will stop the loop
					c=gtC+ker.getCols();
				}
			}
		}

		if(isDetected && ('e' == letter)) nbTP+=1;   //True Positive
		if(isDetected && ('e' != letter))      nbFP+=1; //Falses Positive
		if(!isDetected && ('e' != letter))  nbTN+=1; //True Negative
		if(!isDetected && ('e' == letter))  nbFN+=1;  //False Negative

	}

	fclose(fpt);
	//Make observation of TP and FP for plotting Graph
	float TPR,FPR;
	TPR = (float)nbTP/float(nbTP+nbFN);
	FPR = (float)nbFP/float(nbFP+nbTN);

std::cout<<nbTP<<","<<nbFP<<","<<nbTN<<","<<nbFN<<","<<TPR<<","<<FPR<<std::end
l;
	return;
}
```

OUTPUT:

Below is the TPR-FPR plot as shown below:-



Below is the TP-FP plot:-




```
/*
The Main function from where I call all the operations:-
*/
int main(int argc, char *argv[]){

  FILE    *fpt;
  unsigned char *pOutImg = NULL;
  char fileName[30];
  struct timespec tp1,tp2;
  int thres;

  //check the command line argument for the file name
  strcpy(fileName,"img/parenthood_e_template.ppm");
  kernel ker(fileName);

  strcpy(fileName,"img/parenthood.ppm");
  image img(fileName);

  thres = 210; //default value can be set to anything
  if(argc==2){ //is command line argument given
     thres = atoi(argv[1]);
  }
  unsigned char *pMSFImage = NULL;
  for(thres = 0;thres<=256;thres++){
     std::cout<<thres<<",";
     pMSFImage=getMSFImage(ker,img,thres);
```

```
      strcpy(fileName,"parenthood_gt.txt");
      TestMSFImage(pMSFImage,fileName,img,ker);
      }
  saveImage(pMSFImage,img.getCols(),img.getRows());

  return 0;

}
```

Utility Functions that I use:-

```
/*
The Image class. A data Structure that I use to access the image structure all the time.
*/
#include "image.h"
#include <cstring>
#include <cstdlib>

unsigned char *image::readImage(FILE *fpt){

           char ws;
           unsigned char *pImgPixels = new unsigned char[_rows*_cols];
           unsigned char **ppImgPixels = new unsigned char*[_rows];
           fread(pImgPixels,1,_rows*_cols,fpt);

           //Having a double array version for better operations
           for(int i=0;i<_rows;i++){
                ppImgPixels[i]=pImgPixels+(i*_cols);
           }

           _ppPixels = ppImgPixels;

           return pImgPixels;
}

image::image(FILE *fpt):_rows(0),_cols(0){

      fscanf(fpt,"%s %d %d %d\n",_header,&_cols,&_rows,&_bytes);

      _pPixels = readImage(fpt);
}

image::image(char iFileName[30]):_rows(0),_cols(0){

      FILE *fpt=NULL;
      if ((fpt=fopen(iFileName,"rb")) == NULL)
  {
    std::cout<<"Unable to open <kernelFileName>.ppm for reading\n";
    exit(0);
  }
```

```
  fscanf(fpt,"%s %d %d %d\n",_header,&_cols,&_rows,&_bytes);
      _pPixels = readImage(fpt);
      strcpy(_fileName,iFileName);

}

image::~image(){
      delete _ppPixels;
      delete _pPixels;
}


int image::getRows(){
      return _rows;
}

int image::getCols(){
      return _cols;
}

int image::getBytes(){
      return _bytes;
}

unsigned char *image::getPixels(){
      return _pPixels;
}

unsigned char **image::getppPixels(){
      return _ppPixels;
}


/*
The kernel class. In this class I save the image specific data for the kernel image.
*/

#include "kernel.h"
#include <iostream>
#include <cstdlib>
#include <string.h>

unsigned char *kernel::readKernelImage(FILE *fpt){

            char ws;
            unsigned char *pKPix = new unsigned char[_rows*_cols];
            unsigned char **ppKPixels = new unsigned char*[_rows];

            //ws = fgetc(fpt);  //kernel images doesn't have the whitepace
            fread(pKPix,1,_rows*_cols,fpt);

            //Having a double array version for better operations
```

```
            for(int i=0;i<_rows;i++){
                 ppKPixels[i]=pKPix+(i*_cols);
            }

            _ppPixels = ppKPixels;

            return pKPix;
}

kernel::kernel(FILE *fpt):_rows(0),_cols(0){

     fscanf(fpt,"%s %d %d %d\n",_header,&_cols,&_rows,&_bytes);

     _pPixels = readKernelImage(fpt);
}

kernel::kernel(char ifileName[30]):_rows(0),_cols(0){

     FILE *fpt=NULL;
     if ((fpt=fopen(ifileName,"rb")) == NULL)
  {
    std::cout<<"Unable to open <kernelFileName>.ppm for reading\n";
    exit(0);
  }


     fscanf(fpt,"%s %d %d %d\n",_header,&_cols,&_rows,&_bytes);

     _pPixels = readKernelImage(fpt);
     strcpy(_fileName,ifileName);

     fclose(fpt);
}

kernel::~kernel(){

     delete _ppPixels;
     delete _pPixels;
}


int kernel::getRows(){
     return _rows;
}

int kernel::getCols(){
     return _cols;
}

int kernel::getBytes(){
     return _bytes;
}
```

```
unsigned char *kernel::getPixels1dArray(){
      return _pPixels;
}

unsigned char **kernel::getPixels2dArray(){
      return _ppPixels;
}
```

Conclusion:-

After analysing my curve I feel the best place to have a threshold in case of my ‘e’ object recognition algorithm at 213. My observation as below in my readings:-

210,143,309,802,8,0.94702,0.278128

so,
Threshold = 210
True Positive = 143
False Positive = 309
True Negative = 802
False Negative = 8

The TPR Value drops sharply after this value of threshold I feel. This project gave me a good idea of how to use the MSF filter using the Zero-mean squared image of the template image. I also learnt on how to use a ground truth data to verify the best point of my analysis or my detection algorithm here.