Lab#3 – Letters

Course Name: Intro to Computer Vision

Course: ECE-6310

Name: Saroj Kumar Dash

Problem Statement:

To implement thinning, branchpoint and endpoint detection to recognize letters in an image of text

Solution: -

For the solution of the problem I built upon my code in my previous lab.

Preparation for my operation in the images:
I used my code to read the below images and store in a class named image for further use or operations in the future,

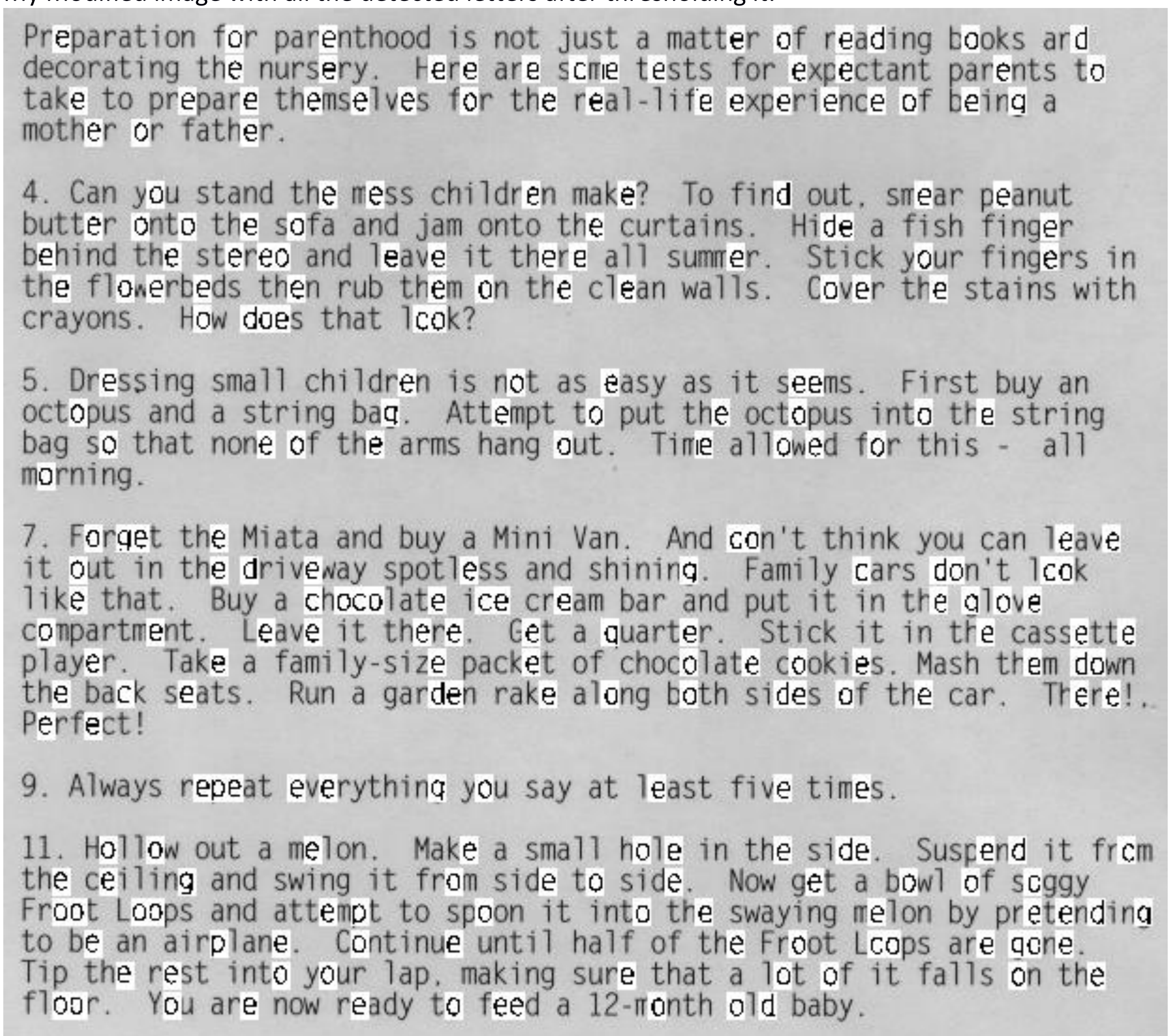
- msf_e.ppm → to use our previous algorithm to detect the initial detection of the matching images
- parenthood.ppm → To use it to do all our operations
- parenthood_gt.txt → To read it for our ground truth computation, I stored this in vector of points and the ground truth character value.

I loop through all the threshold values, iT is starting from 0 to 255. For each threshold value I compare the pixel value in the msf_e.ppm if the value of a pixel at a location r,c is greater than the iT then I consider the pattern ‘e’ to be detected or else I continue in my logic.

Cropping and Thresholding of the image:

For every ground-truth coordinates I draw a 9x15 window around the coordinate in the original image and make it a binary image using a threshold value, where all pixels that are greater than the threshold value are made 255 and below the threshold value is made 0. It is said to crop the image from the main image, I instead changed the value in the original image. The cropped and threshold separated binary image appeared as below.

My Modified image with all the detected letters after thresholding it:-



Preparation for parenthood is not just a matter of reading books ard decorating the nursery. Here are scme 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 lcok?

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 con't think you can leave it out in the driveway spotless and shining. Family cars don't lcok 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 frcm 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 Lcops 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.

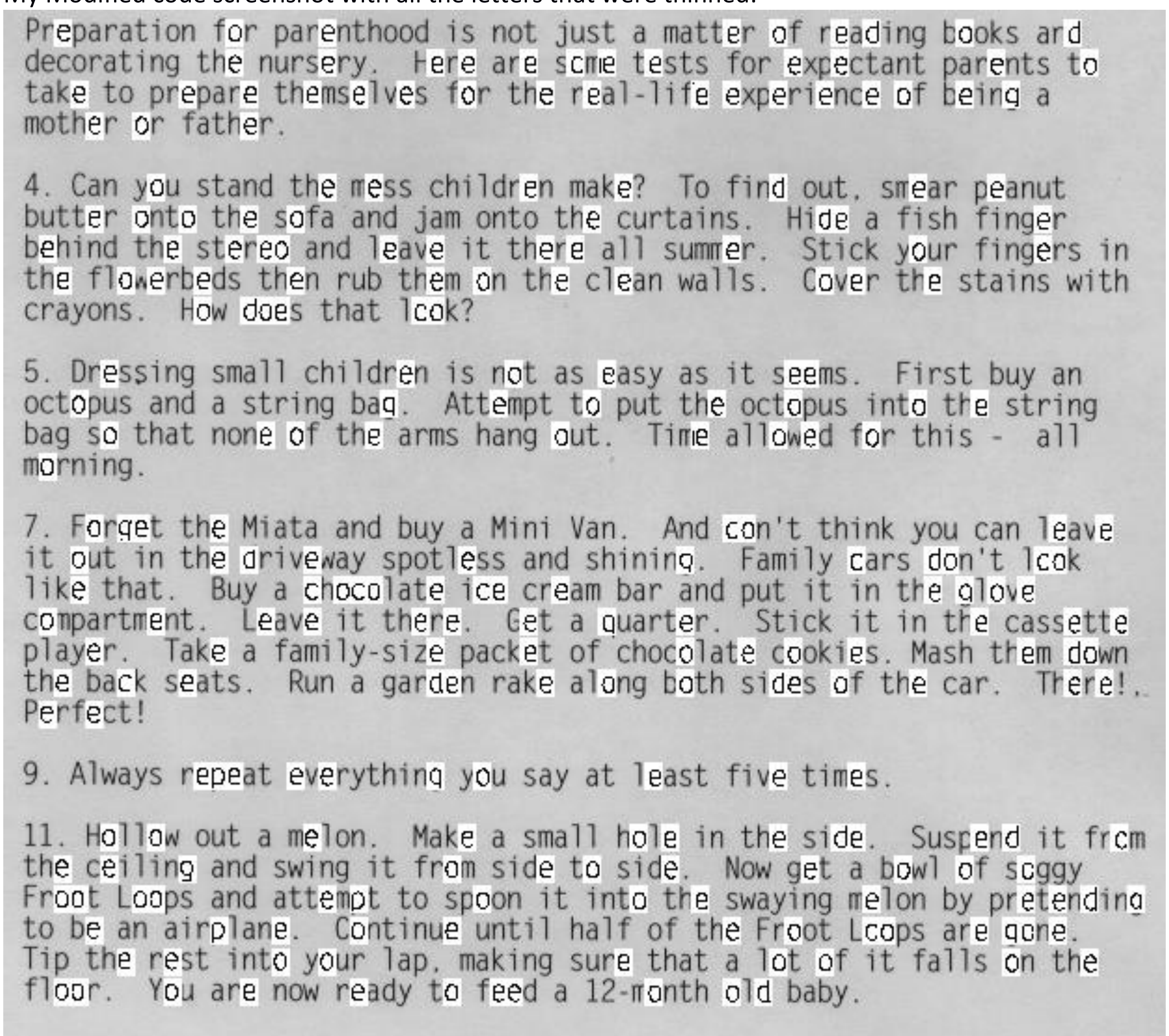
The Thinning algorithm:

For a better edge recognizing algorithm we need to thin the thick sections of the detected image. Since we have a binary image now so thinning the image will improve the detection a lot in finding the number of end points and branch points in the image. For thinning I used the ground-truth coordinates obtained to make a window in the original image at which we can use the thinning algorithm (used from class notes). The output of this operation is shown below in the image.

The marking of pixels image condition:-

```
//the thinning condition to mark the pixel location
if((1 == nbE2NE) && ((3<=nbE)&&(7>=nbE)) && ( N | E | (W&S) )) {
            struct point p;
            p.c = c;
            p.r = r;
            vErasePoints.push_back(p);
}
```

My Modified code screenshot with all the letters that were thinned: -



Preparation for parenthood is not just a matter of reading books ard decorating the nursery. Here are scme 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 lcok?

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 lcok 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 frcm the ceiling and swing it from side to side. Now get a bowl of scggy Froot Loops and attempt to spoon it into the swaying melon by pretending to be an airplane. Continue until half of the Froot Lcops 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.

Edge Properties of the thinned image:

Next was to find the important pixels in the thinned structure which are the endpoints. I found the 8 neighbor pixels of the pixel of operation (i.e. only edge pixels or whose value was 0) and then checked for edge to non-edge transitions. The most important thing I think was to count the edge to non-edge pixel transition in a clockwise operation. Then using it to calculate the end and branch points in the image, below was my operation to do this.

To count the edge to non-edge transition: -

```
nbr = getNeighBors(img,c,r);
for(int i=0;i<8;i++){
//count edge to non-edge
if( (nbr[i]==0) && (nbr[i+1]==255))
      nb_E2NE+=1;
}
```

To count the branch point and endpoint in our analysis window:-

```
if( 1 == nb_E2NE)
      nb_ep+=1;

if( 2 < nb_E2NE)
      nb_bp+=1;
```

To check for our letter ‘e’ I check for 1 ep and 1 branchpoint in the analysis window or letter:-

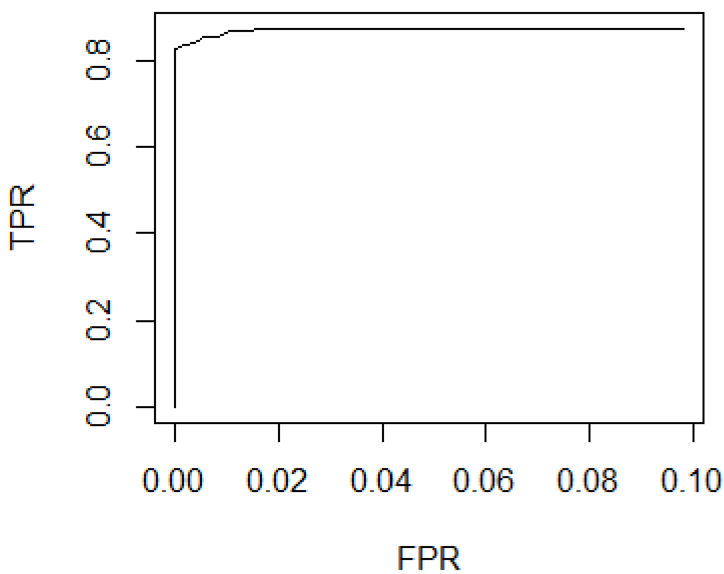
```
if((nb_bp ==1) && (nb_ep==1))
      isE = true;
```

Since it’s a grey scale image the edge point and branch point could not colored. So I plotted the pixel value and changed the values of edge pixels to 1 and branchpoint pixels to 2. Below is a screenshot of the debug program that I have run. The below dump is of an ‘e’ window at gorundTruth location 139,357. Code is present in Appendix for reference.:-

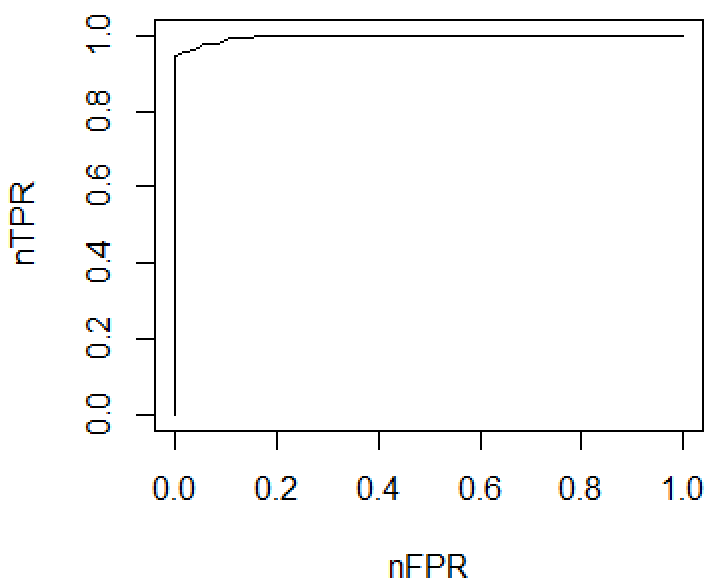
```
saroj@saroj-Inspiron-5559:/media/saroj/Studies/Clemson/SemesterStudies/Fall2017/Computer Vision/Code/CV-Lab/3-Letters$ ./a.out
 T  TP  FP
 200  202  198  204  204  203  199  201  203 nb_ep=0 nb_bp=0 gtC = 139 gtR = 357
 255  255  255  255  255  255  255  255  255 nb_ep=0 nb_bp=0 gtC = 139 gtR = 357
 255  255  255  255  255  255  255  255  255 nb_ep=0 nb_bp=0 gtC = 139 gtR = 357
 255  255  255  255  255  255  255  255  255 nb_ep=0 nb_bp=0 gtC = 139 gtR = 357
 255  255  255  255  255  255  255  255  255 nb_ep=0 nb_bp=0 gtC = 139 gtR = 357
 255    0    0    0    0    0  255  255  255 nb_ep=0 nb_bp=0 gtC = 139 gtR = 357
 255    0  255  255  255    0    0  255  255 nb_ep=0 nb_bp=0 gtC = 139 gtR = 357
   0  255  255  255  255  255    0  255  255 nb_ep=0 nb_bp=0 gtC = 139 gtR = 357
   0  255  255  255  255  255    0  255  255 nb_ep=0 nb_bp=0 gtC = 139 gtR = 357
   2    0    0    0    0    0  255  255  255 nb_ep=0 nb_bp=1 gtC = 139 gtR = 357
   0  255  255  255  255  255  255  255  255 nb_ep=0 nb_bp=1 gtC = 139 gtR = 357
   0    0  255  255  255  255  255  255  255 nb_ep=0 nb_bp=1 gtC = 139 gtR = 357
 255    0  255  255  255  255    1  255  255 nb_ep=1 nb_bp=1 gtC = 139 gtR = 357
 255  255    0    0    0    0  255  255  255 nb_ep=1 nb_bp=1 gtC = 139 gtR = 357
 255  255  255  255  255  255  255  255  255 nb_ep=1 nb_bp=1 gtC = 139 gtR = 357
```

Plotting my TP and FP counts for my ROC curve: -

I missed some TP values as the size of the window that I took brought some of the side image data too, but with the missing data and the data that I obtained the ROC curve that I obtained was steeper than my previous lab, thus using this algorithm the letter detection improved. By my observation the TP and FP were optimized at a point of **threshold = 202** which is highlighted in my table.



In normalized values of TPR and FPR values: -



Below are my TP and FP values using my detection method:-

T	TP	FP	T	TP	FP	T	TP	FP	T	TP	FP	T	TP	FP	T	TP	FP
0	143	114	41	143	114	81	143	114	121	143	114	161	143	114	201	142	21
1	143	114	42	143	114	82	143	114	122	143	114	162	143	114	202	142	16
2	143	114	43	143	114	83	143	114	123	143	114	163	143	114	203	141	14
3	143	114	44	143	114	84	143	114	124	143	114	164	143	114	204	140	12
4	143	114	45	143	114	85	143	114	125	143	114	165	143	114	205	140	9
5	143	114	46	143	114	86	143	114	126	143	114	166	143	114	206	139	8
6	143	114	47	143	114	87	143	114	127	143	114	167	143	114	207	139	7
7	143	114	48	143	114	88	143	114	128	143	114	168	143	114	208	138	6
8	143	114	49	143	114	89	143	114	129	143	114	169	143	114	209	137	2
9	143	114	50	143	114	90	143	114	130	143	114	170	143	112	210	136	0
10	143	114	51	143	114	91	143	114	131	143	114	171	143	111	211	136	0
11	143	114	52	143	114	92	143	114	132	143	114	172	143	110	212	134	0
12	143	114	53	143	114	93	143	114	133	143	114	173	143	108	213	133	0
13	143	114	54	143	114	94	143	114	134	143	114	174	143	108	214	129	0
14	143	114	55	143	114	95	143	114	135	143	114	175	143	104	215	129	0
15	143	114	56	143	114	96	143	114	136	143	114	176	143	101	216	126	0
16	143	114	57	143	114	97	143	114	137	143	114	177	143	98	217	124	0
17	143	114	58	143	114	98	143	114	138	143	114	178	143	96	218	119	0
18	143	114	59	143	114	99	143	114	139	143	114	179	143	93	219	118	0
19	143	114	60	143	114	100	143	114	140	143	114	180	143	88	220	116	0
20	143	114	61	143	114	101	143	114	141	143	114	181	143	81	221	113	0

21	143	114
22	143	114
23	143	114
24	143	114
25	143	114
26	143	114
27	143	114
28	143	114
29	143	114
30	143	114
31	143	114
32	143	114
33	143	114
34	143	114
35	143	114
36	143	114
37	143	114
38	143	114
39	143	114
40	143	114

62	143	114
63	143	114
64	143	114
65	143	114
66	143	114
67	143	114
68	143	114
69	143	114
70	143	114
71	143	114
72	143	114
73	143	114
74	143	114
75	143	114
76	143	114
77	143	114
78	143	114
79	143	114
80	143	114

102	143	114
103	143	114
104	143	114
105	143	114
106	143	114
107	143	114
108	143	114
109	143	114
110	143	114
111	143	114
112	143	114
113	143	114
114	143	114
115	143	114
116	143	114
117	143	114
118	143	114
119	143	114
120	143	114

142	143	114
143	143	114
144	143	114
145	143	114
146	143	114
147	143	114
148	143	114
149	143	114
150	143	114
151	143	114
152	143	114
153	143	114
154	143	114
155	143	114
156	143	114
157	143	114
158	143	114
159	143	114
160	143	114

182	143	79
183	143	75
184	143	72
185	143	68
186	143	68
187	143	64
188	143	59
189	143	55
190	143	52
191	143	47
192	143	45
193	143	43
194	143	40
195	143	38
196	143	32
197	143	30
198	143	28
199	143	26
200	142	21

222	109	0
223	102	0
224	97	0
225	96	0
226	91	0
227	88	0
228	80	0
229	74	0
230	67	0
231	62	0
232	59	0
233	52	0
234	47	0
235	42	0
236	40	0
237	38	0
238	34	0
239	30	0
240	28	0

T	TP	FP
241	24	0
242	23	0
243	20	0
244	16	0
245	13	0
246	11	0
247	8	0
248	6	0
249	5	0
250	1	0
251	1	0
252	1	0
253	1	0
254	1	0
255	0	0

Appendix for Code:-

```
void cropAndThreshold_9x15(image &inImg,int ix,int iy){

	//unsigned char *pOutCropPixels = new unsigned char[9*15];
	int ix0,ixn,iy0,iyn;
	ix0 = ix-(9/2)-1;
	ixn = ix+(9/2);//+1;
	iy0 = iy-(15/2);//-1;
	iyn = iy+(15/2);//+1;
	int T = 128;
	unsigned char **ppImgPix = inImg.getppPixels();

	int cnt = 0;
	for(int r=iy0;r<=iyn;r++){
		for(int c=ix0;c<=ixn;c++){

			//edgecases
			if((c==ix0) || (c==ixn) || (r==iy0) || (r==iyn))
				ppImgPix[r][c] = 255;

			if(ppImgPix[r][c] > T)
				ppImgPix[r][c] = 255;
			else
				ppImgPix[r][c] = 0;
		}
	}


}


unsigned char *getNeighBors(image &img,int c,int r){

	unsigned char *nbr = new unsigned char[9];
	int cnt = 0;
	int i =0;
	unsigned char **ppPixels = img.getppPixels();

	for(i = c-1;i<=(c+1);i++)
		nbr[cnt++] = ppPixels[r-1][i];

	nbr[cnt++] = ppPixels[r][c+1];

	for(i = c+1;i>=(c-1);i--)
		nbr[cnt++] = ppPixels[r+1][i];

	nbr[cnt++] = ppPixels[r][c-1];
```

```
        nbr[cnt++] = ppPixels[r-1][c-1];

        return nbr;


}



void thinning(image &img,int gtc,int gtr){

        //thinning has to be done for single pixel wide components

        int ic0,icn,ir0,irn;
        ic0 = gtc-(9/2);
        icn = gtc+(9/2);
        ir0 = gtr-(15/2);
        irn = gtr+(15/2);
        unsigned char **ppImgPix = img.getppPixels();
        std::vector<struct point> vErasePoints;
        int c =0;

        //loop through the thresholded image
        do{

         //erase the marked points
        for(int ithP=0;ithP < vErasePoints.size(); ithP++){
          //erase here by making them 255
          ppImgPix[vErasePoints[ithP].r][vErasePoints[ithP].c] = 255;
        }
        if(vErasePoints.size()) vErasePoints.clear();

        for(int r=ir0;r<=irn;r++){
                for(int c=ic0;c<=icn;c++){
                //Pass through the image looking at each pixel with value 0 i.e. edge pixels
                if( 0 == ppImgPix[r][c]){
                  //check for erasure
                  unsigned char *nbr = getNeighBors(img,c,r);
                  int nbE2NE = 0;
                 int N=0,E=0,W=0,S=0;
                 int nbE = 0;

                //check for all edge to non-edge transition
                for(int i =0;i<8;i++){
                  if( (nbr[i]==0) && (nbr[i+1]==255))
                        nbE2NE+=1;
                        //get neighbors
                        if( (i == 1) && (nbr[i]==255)) N=1;
                        if( (i == 3) && (nbr[i]==255)) E=1;
```

```
                        if( (i == 5) && (nbr[i]==255)) S=1;
                        if( (i == 7) && (nbr[i]==255)) W=1;
                        //get number of edge neighbors
                        if( 0 == nbr[i])
                              nbE+=1;

                        }

                        //the thinning condition to mark the pixel location
                        if( (1 == nbE2NE) && ((3<=nbE)&&(7>=nbE)) && ( N | E | (W&S) )) {
                              struct point p;
                              p.c = c;
                              p.r = r;
                              vErasePoints.push_back(p);
                        }

                        delete nbr;
                        }

                  }
            }
}while(vErasePoints.size());
return;
}

bool is_e_detected(image &img,int gtC,int gtR){
      bool isE = false;
      int nb_E2NE=0;
      int nb_ep=0,nb_bp=0;
      int ic0,icn,ir0,irn;
      ic0 = gtC-(9/2);
      icn = gtC+(9/2);
      ir0 = gtR-(15/2);
      irn = gtR+(15/2);
      unsigned char *nbr = NULL;

      for(int r=ir0;r<=irn;r++){
        for(int c=ic0;c<=icn;c++){
            //printf("%4d ",(img.getppPixels())[r][c]);
            if( 255 == (img.getppPixels())[r][c]) continue;

            nbr = getNeighBors(img,c,r);
            for(int i=0;i<8;i++){
              //count edge to non-edge
             if( (nbr[i]==0) && (nbr[i+1]==255))
                  nb_E2NE+=1;
            }

            if( 1 == nb_E2NE)
```

```
				nb_ep+=1;

			if( 2 < nb_E2NE)
				nb_bp+=1;

			nb_E2NE=0;


			delete nbr;
			}
		}

		if((nb_bp ==1) && (nb_ep==1))
			isE = true;

		return isE;
}
```