Lab 3 Report

Ruijie Song

Feb.11.2021

Section 1 Report:

1.

Figure 1. img22gd2.tif.

2.

Figure 2. the image showing the connected set for s = (67, 45), and T = 2.

3.

Figure 3. the image showing the connected set for s = (67, 45), and T = 1.

4.

Figure 4. the image showing the connected set for s = (67, 45), and T = 3.

Section 2 Report:

1.
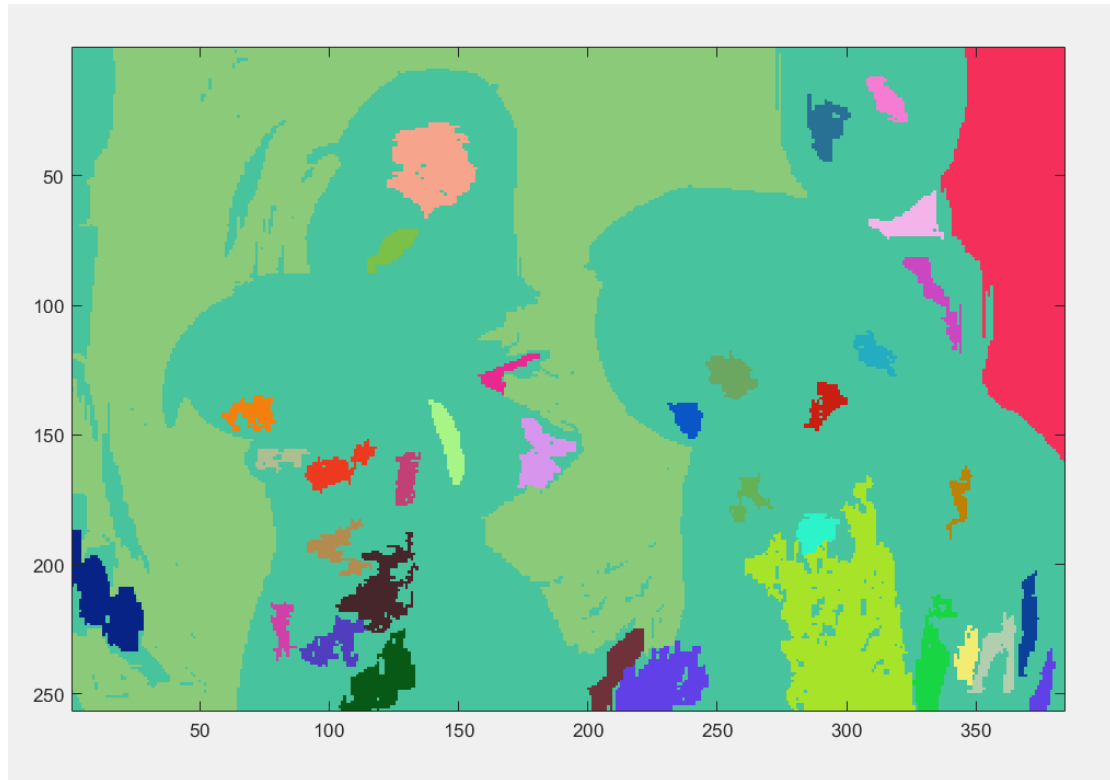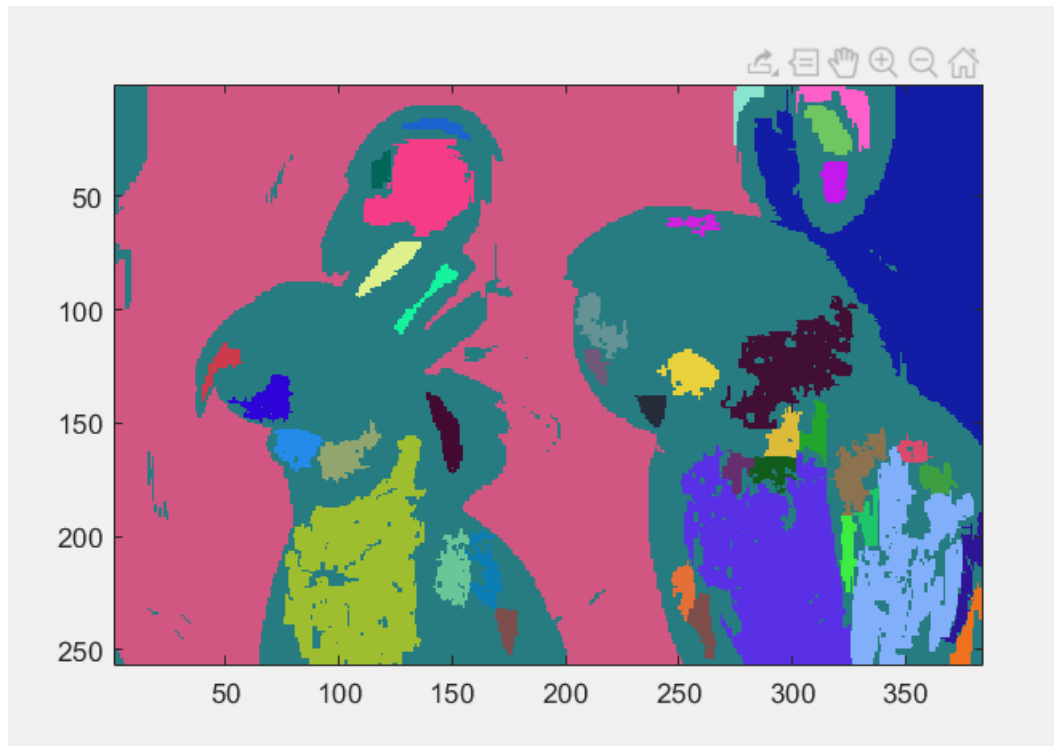

Figure 5. randomly colored segmentation for T = 1
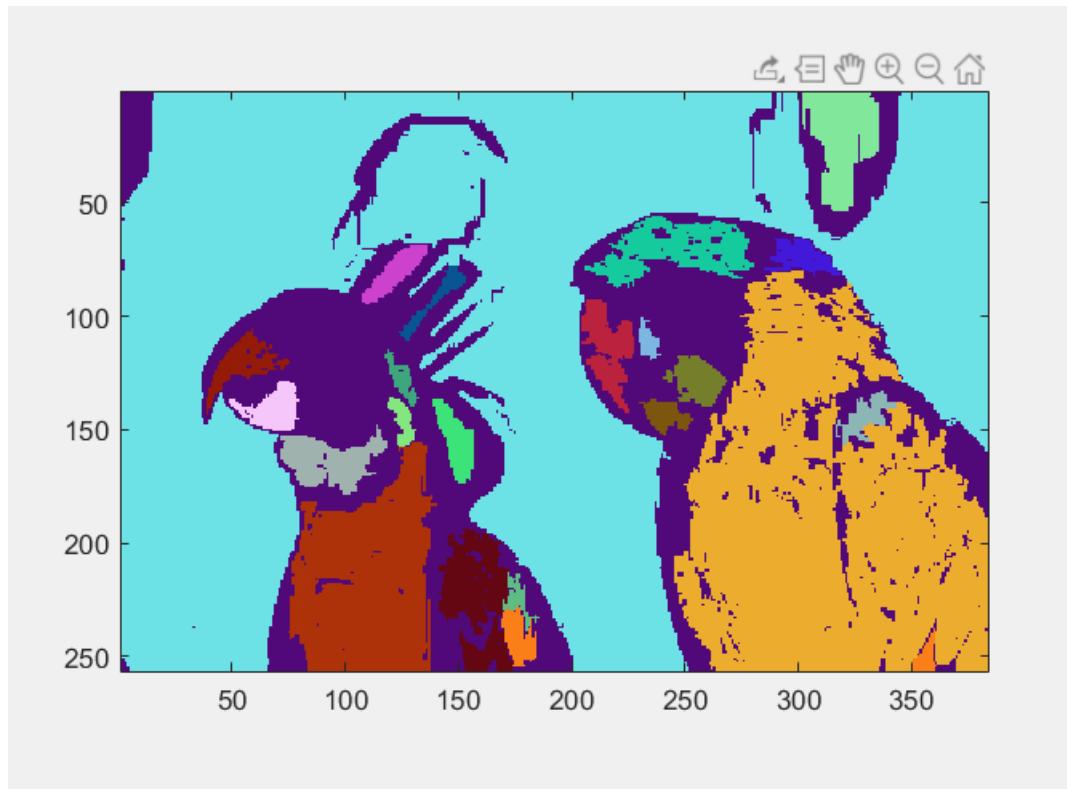
Figure 6. randomly colored segmentation for T = 2

Figure 7. randomly colored segmentation for T = 3

2.

T	1	2	3
The number of regions	36	41	23

```
1  /* ----------------------SECTION 1--------------------*/
2  #include <math.h>
3  #include "tiff.h"
4  #include "allocate.h"
5  #include "randlib.h"
6  #include "typeutil.h"
7  #include <stdio.h>
8
9  void error(char *name);
10
11 struct pixel {
12     int m,n; /* m=row, n=col */
13 };
14
15 void ConnectedNeighbors(
16     struct pixel s,
17     double T, /* threshold*/
18     unsigned char **img, /* 2D array of pixels */
19     int width,
20     int height,
21     int *M, /* a pointer to the number of neighbors connected to the pixel s */
22     struct pixel c[4]); /* This is an array containing the M connected neighbors to
   the pixel s. */
23
24 void ConnectedSet(
25     struct pixel s,
26     double T,
27     unsigned char **img,
28     int width,
29     int height,
30     int ClassLabel, /* s the integer value that will be used to label any pixel
   which is connected to s. */
31     uint8_t **seg,
32     int *NumConPixels); /* the number of pixels which were found to be connected to
   s. */
33
34 int main (int argc, char **argv)
35 {
36     // Define parameters
37     FILE *fp;
38     struct TIFF_img input_img, color_img;
39     struct pixel s;
40     s.m = 45;
41     s.n = 67;
42     double T = 3.0;
43     int ClassLabel = 1;
44     int NumConPixels = 0;
45
46     /* accepts a command line argument specifying the value of rho */
47     // scanf("%lf", &rho);
48
49     if ( argc != 2 ) error( argv[0] );
50
51     /* open image file */
52     if ( ( fp = fopen ( argv[1], "rb" ) ) == NULL ) {
53     fprintf ( stderr, "cannot open file %s\n", argv[1] );
54     exit ( 1 );
55     }
56
57     /* read image */
```

```
 58        if ( read_TIFF ( fp, &input_img ) ) {
 59        fprintf ( stderr, "error reading file %s\n", argv[1] );
 60        exit ( 1 );
 61        }
 62
 63        /* close image file */
 64        fclose ( fp );
 65
 66        /* check the type of image data */
 67        if ( input_img.TIFF_type != 'g' ) {
 68        fprintf ( stderr, "error:  image must be 24-bit color\n" );
 69        exit ( 1 );
 70        }
 71
 72        // // Define seg
 73        // unsigned int **seg;
 74        // int i;
 75        // int j;
 76        // for ( i = 0; i < input_img.height; i++ ) {
 77        //      for ( j = 0; j < input_img.width; j++ ) {
 78        //          seg[i][j] = 0;
 79        //      }
 80        // }
 81
 82        /* set up structure for output color image */
 83        /* Note that the type is 'g' rather than 'c' */
 84        get_TIFF ( &color_img, input_img.height, input_img.width, 'g' );
 85
 86        // printf("reading success\n");
 87
 88        int i;
 89        int j;
 90        for ( i = 0; i < input_img.height; i++ ) {
 91            for ( j = 0; j < input_img.width; j++ ) {
 92                color_img.mono[i][j] = 0;
 93            }
 94        }
 95
 96        // printf("output init success\n");
 97
 98        ConnectedSet(
 99            s,
100            T,
101            input_img.mono,
102            input_img.width,
103            input_img.height,
104            ClassLabel, /* s the integer value that will be used to label any pixel
     which is connected to s. */
105            color_img.mono,
106            &NumConPixels); /* the number of pixels which were found to be connected to
     s. */
107
108        // printf("process success\n");
109
110        i = 0;
111        j = 0;
112        for ( i = 0; i < input_img.height; i++ ) {
113            for ( j = 0; j < input_img.width; j++ ) {
114                if (color_img.mono[i][j] == 0) {
115                    color_img.mono[i][j] = 255;
```

```
116                }
117            else {
118                color_img.mono[i][j] = 0;
119            }
120        }
121    }
122
123    /* open color image file */
124    if ( ( fp = fopen ( "color.tif", "wb" ) ) == NULL ) {
125        fprintf ( stderr, "cannot open file color.tif\n");
126        exit ( 1 );
127    }
128
129    /* write color image */
130    if ( write_TIFF ( fp, &color_img ) ) {
131        fprintf ( stderr, "error writing TIFF file %s\n", argv[2] );
132        exit ( 1 );
133    }
134
135    /* close color image file */
136    fclose ( fp );
137
138    /* de-allocate space which was used for the images */
139    free_TIFF ( &(input_img) );
140    free_TIFF ( &(color_img) );
141
142    return(0);
143 }
144
145 void error(char *name)
146 {
147    printf("usage:  %s  image.tiff \n\n",name);
148    printf("this program reads in a 24-bit color TIFF image.\n");
149    printf("It then horizontally filters the green component, adds noise,\n");
150    printf("and writes out the result as an 8-bit image\n");
151    printf("with the name 'green.tiff'.\n");
152    printf("It also generates an 8-bit color image,\n");
153    printf("that swaps red and green components from the input image");
154    exit(1);
155 }
156
157 void ConnectedNeighbors(
158    struct pixel s,
159    double T, /* threshold*/
160    unsigned char **img, /* 2D array of pixels */
161    int width,
162    int height,
163    int *M, /* a pointer to the number of neighbors connected to the pixel s */
164    struct pixel c[4]) /* This is an array containing the M connected neighbors to
   the pixel s. */
165 {
166    if ((s.m > height) || (s.n > width)) {
167        printf("error, the pixel s exceeds the width & height of the image\n");
168    }
169    // *M = 0;
170    if (((s.m-1)>=0) && (abs(img[s.m][s.n] - img[s.m-1][s.n])<=T)) {
171        c[*M].m = s.m - 1;
172        c[*M].n = s.n;
173        *M = *M + 1;
174        // printf("1 \n");
```

```c
175        }
176        if (((s.m+1)<=height-1) && (abs(img[s.m][s.n] - img[s.m+1][s.n])<=T)) {
177            c[*M].m = s.m + 1;
178            c[*M].n = s.n;
179            *M = *M + 1;
180            // printf("2 \n");
181        }
182        if (((s.n-1)>=0) && (abs(img[s.m][s.n] - img[s.m][s.n-1])<=T)) {
183            c[*M].n = s.n - 1;
184            c[*M].m = s.m;
185            *M = *M + 1;
186            // printf("3 \n");
187        }
188        if (((s.n+1)<=width-1) && (abs(img[s.m][s.n] - img[s.m][s.n+1])<=T)) {
189            c[*M].n = s.n + 1;
190            c[*M].m = s.m;
191            *M = *M + 1;
192            // printf("4 \n");
193        }
194        return;
195 }
196
197 void ConnectedSet(
198        struct pixel s,
199        double T,
200        unsigned char **img,
201        int width,
202        int height,
203        int ClassLabel, /* s the integer value that will be used to label any pixel
    which is connected to s. */
204        uint8_t **seg,
205        int *NumConPixels) /* the number of pixels which were found to be connected to
    s. */
206 {
207
208        // printf("processing started\n");
209
210        int M = 0;
211        *NumConPixels = *NumConPixels + 1;
212        struct pixel c[4];
213        int i=0;
214
215        // printf("piazza code starts\n");
216        seg[s.m][s.n]=1;
217        // Doing recursive is simpler than link-list (idea from Piazza)
218        ConnectedNeighbors(s,T,img,width,height,&M,c);
219        //base case:
220        if(M==0){return;}//no neighbors
221        // if(seg[c[i].m][c[i].n]==1 for all i in range(M)){return;} //all neighbors
    already visited
222        for(i=0;i<M;i++){
223            if(seg[c[i].m][c[i].n]!=1) {
224                ConnectedSet(c[i],T,img,width,height,ClassLabel,seg,NumConPixels);
225            }
226            else{
227                continue;
228            }
229        }
230        return;
231 }
```

```c
  1 /* ------------------------SECTION 2---------------------*/
  2 #include <math.h>
  3 #include "tiff.h"
  4 #include "allocate.h"
  5 #include "randlib.h"
  6 #include "typeutil.h"
  7 #include <stdio.h>
  8
  9 void error(char *name);
 10
 11 struct pixel {
 12     int m,n; /* m=row, n=col */
 13 };
 14
 15 void ConnectedNeighbors(
 16     struct pixel s,
 17     double T, /* threshold*/
 18     unsigned char **img, /* 2D array of pixels */
 19     int width,
 20     int height,
 21     int *M, /* a pointer to the number of neighbors connected to the pixel s */
 22     struct pixel c[4]); /* This is an array containing the M connected neighbors to
    the pixel s. */
 23
 24 void ConnectedSet(
 25     struct pixel s,
 26     double T,
 27     unsigned char **img,
 28     int width,
 29     int height,
 30     int ClassLabel, /* s the integer value that will be used to label any pixel
    which is connected to s. */
 31     uint8_t **seg,
 32     int *NumConPixels); /* the number of pixels which were found to be connected to
    s. */
 33
 34 int main (int argc, char **argv)
 35 {
 36     // Define parameters
 37     FILE *fp;
 38     struct TIFF_img input_img, color_img, segmentation;
 39     // struct pixel s;
 40     // s.m = 45;
 41     // s.n = 67;
 42     double T = 3.0;
 43     int ClassLabel = 1;
 44     int NumConPixels = 0;
 45     int NOR = 0;
 46
 47     /* accepts a command line argument specifying the value of rho */
 48     // scanf("%lf", &rho);
 49
 50     if ( argc != 2 ) error( argv[0] );
 51
 52     /* open image file */
 53     if ( ( fp = fopen ( argv[1], "rb" ) ) == NULL ) {
 54     fprintf ( stderr, "cannot open file %s\n", argv[1] );
 55     exit ( 1 );
 56     }
 57
```

```
58      /* read image */
59      if ( read_TIFF ( fp, &input_img ) ) {
60      fprintf ( stderr, "error reading file %s\n", argv[1] );
61      exit ( 1 );
62      }
63
64      /* close image file */
65      fclose ( fp );
66
67      /* check the type of image data */
68      if ( input_img.TIFF_type != 'g' ) {
69      fprintf ( stderr, "error:  image must be 24-bit color\n" );
70      exit ( 1 );
71      }
72
73      /* set up structure for output color image */
74      /* Note that the type is 'g' rather than 'c' */
75      get_TIFF ( &color_img, input_img.height, input_img.width, 'g' );
76
77      // printf("reading success\n");
78
79      /* create a 2D matrix to record segmentations */
80      get_TIFF ( &segmentation, input_img.height, input_img.width, 'g' );
81
82      /* Initialize the Output */
83      int i;
84      int j;
85      for ( i = 0; i < input_img.height; i++ ) {
86          for ( j = 0; j < input_img.width; j++ ) {
87              color_img.mono[i][j] = 0;
88          }
89      }
90
91      /* Initialize the segmentations */
92      for ( i = 0; i < input_img.height; i++ ) {
93          for ( j = 0; j < input_img.width; j++ ) {
94              segmentation.mono[i][j] = 0;
95          }
96      }
97
98      // printf("output init success\n");
99
100     i = 0;
101     j = 0;
102     int ii = 0; int jj = 0;
103     for ( i = 0; i < input_img.height; i++ ) {
104         for ( j = 0; j < input_img.width; j++ ) {
105             struct pixel s;
106             s.m = i;
107             s.n = j;
108             if (color_img.mono[i][j] == 0) {
109                 ConnectedSet(
110                     s,
111                     T,
112                     input_img.mono,
113                     input_img.width,
114                     input_img.height,
115                     ClassLabel, /* s the integer value that will be used to label
    any pixel which is connected to s. */
116                     segmentation.mono,
```

```c
117                     &NumConPixels); /* the number of pixels which were found to be
       connected to s. */
118                     // printf("%d\n", NumConPixels);
119                     if (NumConPixels > 100) {
120                         NOR = NOR + 1;
121                         ClassLabel = ClassLabel + 1;
122                         for ( ii = 0; ii < input_img.height; ii++ ) {
123                             for ( jj = 0; jj < input_img.width; jj++ ) {
124                                 color_img.mono[ii][jj] = segmentation.mono[ii][jj] +
       color_img.mono[ii][jj];
125                             }
126                         }
127                         ii = 0; jj = 0;
128                     }
129                 for ( ii = 0; ii < input_img.height; ii++ ) {
130                     for ( jj = 0; jj < input_img.width; jj++ ) {
131                         segmentation.mono[ii][jj] = 0;
132                     }
133                 }
134                 NumConPixels = 0;
135                 }
136             }
137         }
138
139     printf("%d\n", NOR);
140
141     // printf("process success\n");
142
143     // i = 0;
144     // j = 0;
145     // for ( i = 0; i < input_img.height; i++ ) {
146     //     for ( j = 0; j < input_img.width; j++ ) {
147     //         if (color_img.mono[i][j] == 0) {
148     //             color_img.mono[i][j] = 255;
149     //         }
150     //         else {
151     //             color_img.mono[i][j] = 0;
152     //         }
153     //     }
154     // }
155
156     /* open color image file */
157     if ( ( fp = fopen ( "color.tif", "wb" ) ) == NULL ) {
158         fprintf ( stderr, "cannot open file color.tif\n");
159         exit ( 1 );
160     }
161
162     /* write color image */
163     if ( write_TIFF ( fp, &color_img ) ) {
164         fprintf ( stderr, "error writing TIFF file %s\n", argv[2] );
165         exit ( 1 );
166     }
167
168     /* close color image file */
169     fclose ( fp );
170
171     /* de-allocate space which was used for the images */
172     free_TIFF ( &(input_img) );
173     free_TIFF ( &(color_img) );
174
```

```c
175        return(0);
176  }
177
178  void error(char *name)
179  {
180        printf("usage:  %s  image.tiff \n\n",name);
181        printf("this program reads in a 24-bit color TIFF image.\n");
182        printf("It then horizontally filters the green component, adds noise,\n");
183        printf("and writes out the result as an 8-bit image\n");
184        printf("with the name 'green.tiff'.\n");
185        printf("It also generates an 8-bit color image,\n");
186        printf("that swaps red and green components from the input image");
187        exit(1);
188  }
189
190  void ConnectedNeighbors(
191        struct pixel s,
192        double T, /* threshold*/
193        unsigned char **img, /* 2D array of pixels */
194        int width,
195        int height,
196        int *M, /* a pointer to the number of neighbors connected to the pixel s */
197        struct pixel c[4]) /* This is an array containing the M connected neighbors to
     the pixel s. */
198  {
199        if ((s.m > height) || (s.n > width)) {
200            printf("error, the pixel s exceeds the width & height of the image\n");
201        }
202        // *M = 0;
203        if (((s.m-1)>=0) && (abs(img[s.m][s.n] - img[s.m-1][s.n])<=T)) {
204            c[*M].m = s.m - 1;
205            c[*M].n = s.n;
206            *M = *M + 1;
207            // printf("1 \n");
208        }
209        if (((s.m+1)<=height-1) && (abs(img[s.m][s.n] - img[s.m+1][s.n])<=T)) {
210            c[*M].m = s.m + 1;
211            c[*M].n = s.n;
212            *M = *M + 1;
213            // printf("2 \n");
214        }
215        if (((s.n-1)>=0) && (abs(img[s.m][s.n] - img[s.m][s.n-1])<=T)) {
216            c[*M].n = s.n - 1;
217            c[*M].m = s.m;
218            *M = *M + 1;
219            // printf("3 \n");
220        }
221        if (((s.n+1)<=width-1) && (abs(img[s.m][s.n] - img[s.m][s.n+1])<=T)) {
222            c[*M].n = s.n + 1;
223            c[*M].m = s.m;
224            *M = *M + 1;
225            // printf("4 \n");
226        }
227        return;
228  }
229
230  void ConnectedSet(
231        struct pixel s,
232        double T,
233        unsigned char **img,
```

```
234      int width,
235      int height,
236      int ClassLabel, /* s the integer value that will be used to label any pixel
     which is connected to s. */
237      uint8_t **seg,
238      int *NumConPixels) /* the number of pixels which were found to be connected to
     s. */
239  {
240
241      // printf("processing started\n");
242
243      int M = 0;
244      *NumConPixels = *NumConPixels + 1;
245      struct pixel c[4];
246      int i=0;
247
248      // printf("piazza code starts\n");
249      seg[s.m][s.n]=ClassLabel;
250      // Doing recursive is simpler than link-list (idea from Piazza)
251      ConnectedNeighbors(s,T,img,width,height,&M,c);
252      //base case:
253      if(M==0){return;}//no neighbors
254      // if(seg[c[i].m][c[i].n]==1 for all i in range(M)){return;} //all neighbors
     already visited
255      for(i=0;i<M;i++){
256          if(seg[c[i].m][c[i].n]!=ClassLabel) {
257              ConnectedSet(c[i],T,img,width,height,ClassLabel,seg,NumConPixels);
258          }
259          else{
260              continue;
261          }
262      }
263      return;
264  }
```