MALLIKA GAUR

Texture Segmentation

Uniform Local Binary Pattern

Second edition

This book was professionally typeset on Reedsy.
Find out more at reedsy.com

Contents

1

Acknowledgment

This book is dedicated to my Parents, they introduce me to this life and explained me the purpose to live life happily. They inspire me in each and every turn of my life that problems are the experiences to step forward.

This book is dedicated to the great mentor of Image Analysis Center(IAC, DRDO). I specially honor our Mentor for carrying this light to me by their generous sharing of the knowledge. To each i offer this book with my humble gratitude.

2

Introduction

What is digital image processing?

Digital image processing is use of a digital computer to process digital images through an algorithm. Digital image processing as a broad spectrum of application, such as remote sensing via satellite and other space crafts. A single digital image can present a very large amount of information in a compact and easily interpreted form.

Many of the techniques of digital image processing, or digital picture processing has developed in the 1960s at the Jet Propulsion Laboratory, MIT, Bell Labs, University of Maryland as application of satellite imagery.

Wire photo standard conversion, medical imaging, videophone, character recognition and photo enhancement, and the cost of processing was fairly high with the computing equipment of that era. In the 1970s, digital image processing proliferated, when cheaper computers and dedicated hardware became available. Images could then be processed in real time, for some dedicated problems as television standard conversion.

As general purpose computers became faster, they started to take over the role of dedicated hardware for all but the most specialized and computer intensive operations. With fast computers and signal processors available in the 2000s, digital image processing has become the most common form of image processing, and is generally used because it is the most versatile method.

Digital image processing allows the use of much more complex algorithms for image processing, and hence can offer more sophisticated performance at simple tasks, and the implementation of methods which would be impossible by analog means.

In accordance with the methodology of image analysis, digital image processing is the theoretical and practical approach for the Classification , Feature Extraction and Pattern Recognition of the image.

3

Texture

Texture can be broadly defined as the visual or tactile surface characteristics and appearance of something. Textures can consist of very small elements like sand, or big elements like tree canopy in a tree.

Texture can also be formed by a single surface via variations in shape, illumination, shadows, absorption, and reluctance. Distance and Scale play a big role in the formation of texture at the visual level and it is being concluded that "texture regions give different interpretations at different distances and different degrees of visual attention".

The notion of texture depends on three ingredients.

a. some local 'order' is repeated over a region that is large in comparison to the order's size.

b. The order consists of the non-random arrangement of elementary parts.

c. The parts are roughly uniform entities having approximately the same dimensions everywhere within the textured region.

4

Properties of Texture

Properties of texture

1. Texture is a property of areas; the texture of a point is undefined. So, the texture is a neighborhood of contextual property and its definition must involve gray values in a spatial.
2. The size of this neighborhood depends upon the texture type or the size of the primitives defining the texture.
3. In an Image Texture can be perceive at different scales or levels of resolution For example, consider the texture represented in a brick wall. At a coarse resolution, the texture is perceived as formed by the individual bricks in the wall. At a higher resolution, when only a few bricks are in the field of view, the perceived texture shows the details in the brick.
4. A region is perceived to have texture when the number of primitive objects in the region is large. If only a few primitive objects are present, then a group of countable objects is perceived instead of a textured image.

5

Texture extraction

In extracting texture is to Identify perceived qualities of texture present in an image The intensity variations in an image which characterize texture are generally due to some underlying physical variation in the scene (such as pebbles on a beach or waves in water). Modeling this physical variation is very difficult, so texture is usually characterized by the two-dimensional variations in the intensities present in the image.

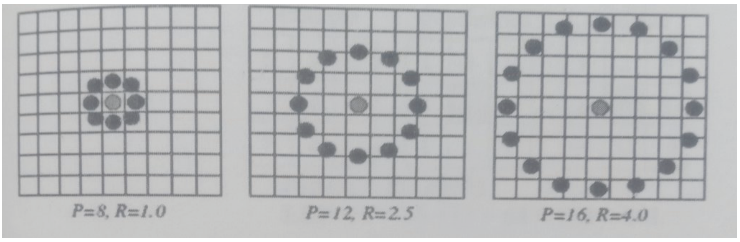
In order to characterize any texture, there are different parameters:

1. Local Binary Pattern,
2. Contrast,
3. Rotational in-variance,
4. Uniform Local Binary Pattern,
5. Texture vector

6

Texture extraction: Local Binary Pattern

Texture T in a local neighborhood of a gray scale image is defined as the joint distribution of the gray levels of P+1 (P>0) image pixels: T-3303P-1) where ge corresponds to the gray value of the center pixel of a local neighborhood. gp(p 0,..., P-1) correspond to the gray values of P equally spaced pixels on a circle of radius R (R>0) that form a circularly symmetric set of neighbors. This set of P+1 pixels is la denoted by GP. P=12 R-25 P:16 R=4.0 If the value of the center pixel is subtracted from the values of the neighbors, the local texture can be represented without losing information-as a joint distribution of the



value of the center pixel and the differences:

Although invariant against gray scale shifts, the differences are affected by scaling. To achieve invariance with respect to any monotonic transformation of the gray scale, only the signs of the differences are considered:

$$T = t(g_c, g_0 - g_c, \ldots, g_{P-1} - g_c).$$

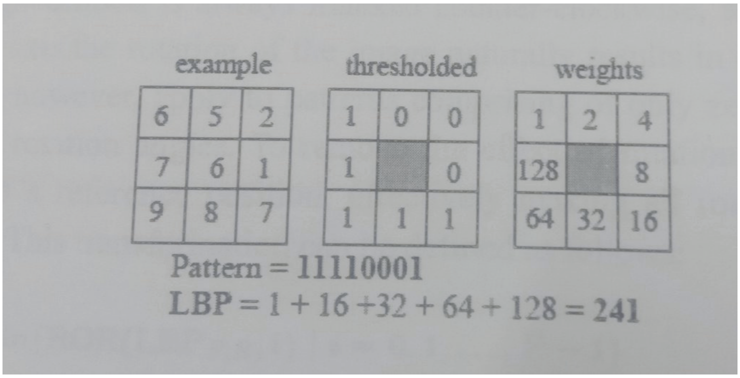
$$T \approx t(s(g_0 - g_c), \ldots, s(g_{P-1} - g_c)),$$

where

$$s(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \,. \end{cases}$$

$$\mathrm{LBP}_{P,R}(x_c, y_c) = \sum_{p=0}^{P-1} s(g_p - g_c) 2^P .$$

Example The basic idea of this approach is demonstrated in fig.We consider a 3X3 neighborhood around each pixel. All neighbor that have values higher than the value of the central pixel are given value 0.The eight binary number associated with the eight neighbor are then read sequentially in the clockwise direction to form a binary number. The binary number (or its equivalent in the decimal system) may be assigned to the central pixel and it may be used to characterize the local texture.



Pattern = 11110001

LBP = 1 + 16 + 32 + 64 + 128 = 241

relative position with respect to the central pixel These weight are 2 22,2′ 2″ 2′ 2′ and 2 With the first assigned to the neighbor which contributes the most significant

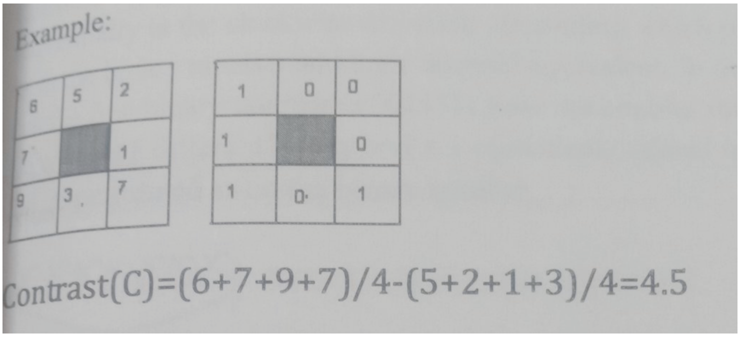
7

Texture extraction: Contrast

Contrast:

Contrast of those pixels which have value 1 and those which have value 0 digit, the second assigned to the neighbor which contributes the second most significant digit, and To address the contrast of the texture, local binary pattern(LBP) is combined with a simple contrast measure contrast(C),which is difference between the average gray-level so on.

Example:



Example:

6	5	2
7	▨	1
9	3	7

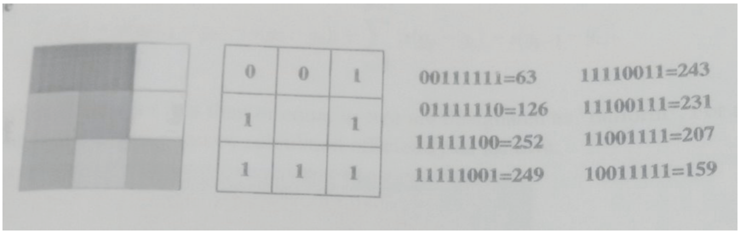
1	0	0
1	▨	0
1	0	1

Contrast(C)=(6+7+9+7)/4-(5+2+1+3)/4=4.5

8

Texture extraction: Rotation In-variance

Due to the circular sampling of neighborhoods, it is fairly straightforward to make LBP codes invariant with respect to rotation of the image domain. “Rotation in-variance” here does not however account for textural differences caused by changes in the relative positions of a light source and the target object.

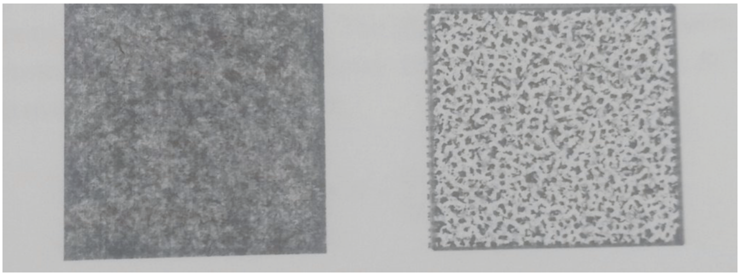


It also does not consider artifacts that are caused by digitizing effects. Each pixel is considered a rotation center, which seems to be the convention in deriving rotation invariant operators. With the assumptions stated above, the rotation invariant LBP can be derived as follows. When an image is rotated, the gray values gp in a circular neighbor set move along the perimeter of a circle centered at gc.

$$U(G_P) = |s(g_{P-1} - g_c) - s(g_0 - g_c)| + \sum_{p=1}^{P-1} |s(g_p - g_c) - s(g_{p-1} - g_c)|.$$

$$\mathrm{LBP}_{P,R}^{riu2} = \begin{cases} \sum_{p=0}^{P-1} s(g_p - g_c) & U(G_P) \leq 2 \\ P+1 & \text{otherwise} . \end{cases}$$

Since the neighborhood is always indexed counter-clockwise, starting in the direction of the positive x axis,the rotation of the image naturally results in a different LBP: P,R value. This does not, however, apply to patterns comprising of only zeros or ones which remain constant at all rotation angles. To remove the effect of rotation, each LBP code must be rotated back to a reference position, effectively making all rotated versions of a binary code the same. This transformation can be defined as follows: LBP: PR = min(ROR(LBPPR)| = 0, 1, P-1)



$$T_i = [t_0, t_1, \ldots, t_8, t_9] \qquad t_i = \frac{m_i}{n^2}$$

Consider the pixel values around the central pixel in the matrix If we read these values sequentially in the clockwise direction,depending which pixel is the starting pixel, we may form the binary number and their decimal equivalent. In the above example, we can see that the first binary number is 0011111,now the circular rotation performed on it 8 times and we will get 8 different binary and it's equivalent decimal number: circular right operation can be performed as on the binary number:

When 00111111(63) is circularly rotated it will become 011111110 which is equivalent to 126 Similarly this operation performed on the bit,eight times .In the 3X3 matrix actually these bits are rotated in 45° step. In the 8-different constructed number, we shall choose the smallest number, which is rotational invariant representation. In the given example this number is 63.

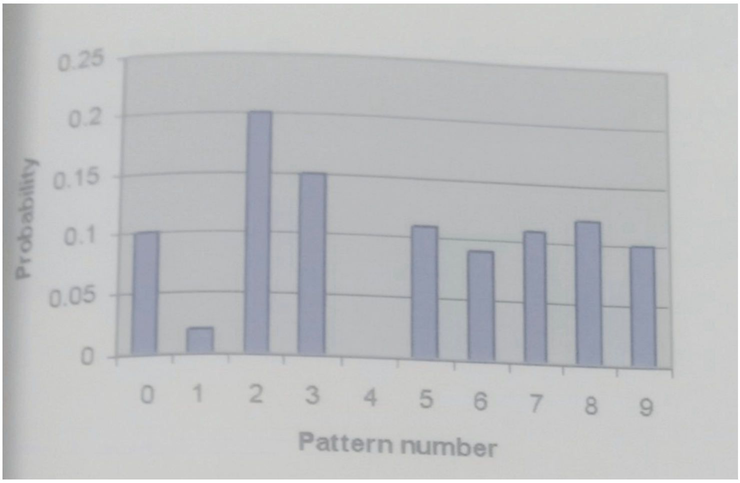
9

Texture extraction: Uniform local binary pattern

UNIFORM LOCAL BINARY PATTERN

The concept of “uniform” patterns was formed when it was observed that certain patterns seem to capture fundamental properties of texture, occurring in the vast majority of patterns, sometimes over 90%. This proposition was further confirmed with a large amount of data. These patterns called “uniform” because they have one thing in common: at most two one-to-zero or zero-to-one transitions in the circular binary code. The LBP codes shown in Figure are all uniform. To formally define the “uniformity” of a neighborhood G a uniformity measure U is needed:

The total number of patterns with U (GP) 2 is P (P-1)+2. When “uniform” codes are ted to their minimum values, the total number of patterns becomes P+1. The rotation vant uniform (riu2) pattern code for any “uniform” pattern is calculated by simply ing ones in the binary number. All other patterns are labeled “miscellaneous” and lapsed into one value:

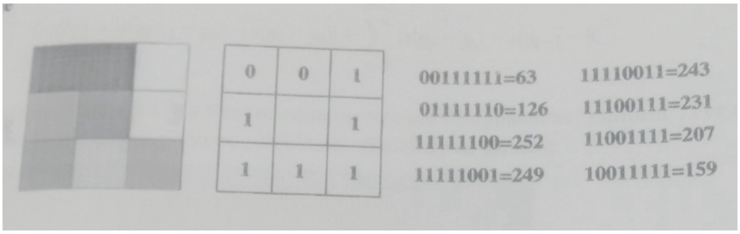

0.25
0.2
0.15
0.1
0.05
0
Probability
0
1
2
3
4
5
6
7
8
9
Pattern number

10

Texture Extraction: Texture Vectors

Texture vectors are being calculated using the uniform local binary pattern (ULBP), we define texture vector for each pixel in its local neighborhood as T = [44] Where t, is the probability of "uniform texture i" in its symmetric n x n neighborhood (t, is the total number of non-uniform patterns).

Distance between two texture vectors : The Bhattacharya measure can be used to compare the similarity between two histogram Let [ri. 2. Fn] and [S1, S2, Sn] denote the normalized frequencies of bins 1 to n in histograms R and S, respectively. The Bhattacharya similarity metric, BCH, between and S histograms is defined as follows: BCH is between 0 and 1. BCH close to 1 show that the two histograms are similar.



For the case of two identical histograms BCH equals to 1.As mentioned, if two histogram are very similar then BCH is very close to 1. Therefore using Bhattacharya distance difference can be redefined as

$$Diff^2_{texture_{i,j}} = 1 - \sum_{h=0}^{9} \left(\sqrt{t_{i,h}} \sqrt{\hat{t}_{j,h}} \right)$$

11

Cluster Analysis: k-means

Cluster Analysis

Cluster analysis is one of the basic tools for exploring the underlying structure of a given data set and is being applied in a wide variety of engineering and scientific disciplines such as medicine, psychology, biology, sociology, pattern recognition, and image processing.

The primary objective of cluster analysis is to partition a given data set of multidimensional vectors (patterns) into so-called homogeneous clusters such that patterns within a a cluster are more similar to each other than patterns belonging to different clusters.

K-means Clustering One of the earliest clustering techniques in the literature is the K-means clustering method. In this technique, clustering is based on the identification of K elements in the data set that can be used to create an initial representation of clusters. These K elements form the chatter seeds. The remaining elements in the data set are then assigned to one of these clusters. At each iteration cluster centers are recomputed. The algorithm terminates when re-computing cluster-centroids do not alter the cluster centroids with whom cluster membership v was calculated.

K-means is one of the simplest unsupervised learning algorithms that solve the well known clustering problem. The procedure follows a simple and easy way to classify a given data set through a certain number of clusters (assume k clusters) fixed a priority.

The idea is to define k centroids, one for each cluster. These centroids should be placed in cunning way because of different location causes different result. So, the better choice is place them as much as possible far away from each other.

The next step is to take each point belonging to a given data set and associate it to the nearest centroid. When no point is pending the first step is completed and an early group age is done. At this point we need to calculate k new centroids. After we have these k new centroids, a new binding has to be done between the same data set points and the nearest new centroid.

A loop has been generated. As a result of this loop we may notice that the k centroids change their location step by step until no more changes are done. In other words centroids do not move any Basic Algorithm for K-means clustering

1) Select K points as initial cluster centroids at random

2) Repeat 1) Form K Cluster by assigning all pixel one by one to nearest cluster using the Bhattacharya distance

4) Re-estimate the centroids of the clusters assuming that the data memberships obtained in

3)are correct, producing new model;

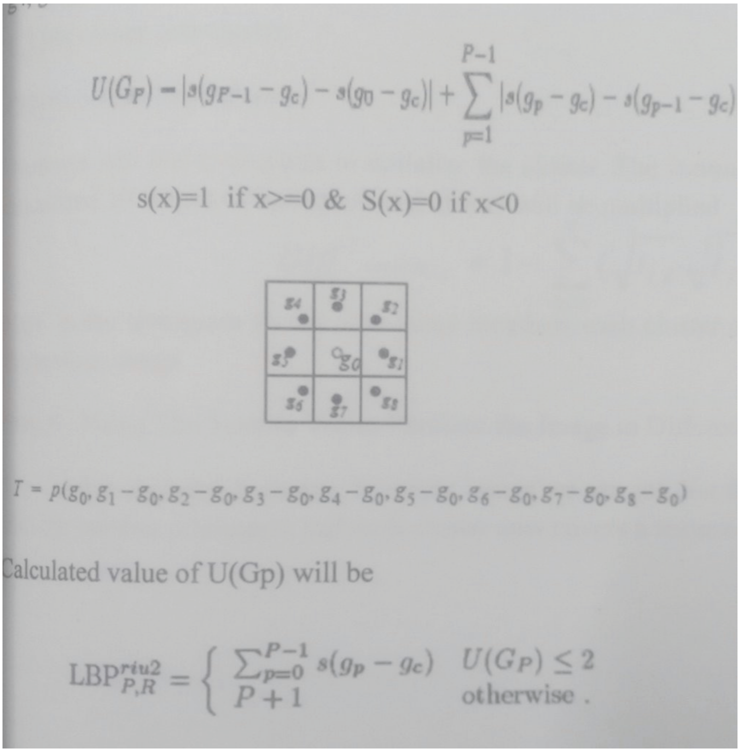
5) Until Centroids do not change.

12

Image Analysis methodology

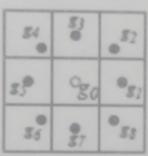
Step 1: Read Image of 256×256 size from RAW file format In this step we use the texture image of grayscale of 8 bit. The gray image also known as the one band image It is basically 2D image in which each pixel shows the value between 0 to 255. We use the malloc() function to dynamically allocate th memory to read 2D Texture image. We read the file of the image through a function gray fread(). Similarly we use the fwrite() function to write the output image in which we hav stored the calculated parameter.

Step 2: For each pixel of Image, Calculate ULBP and store it an array Calculate neighbour value in clockwise direction, in the 3×3 matrix we could be ab to read these values in 45°. The central element is go and neighbour element is gl, g2, g 4. g5, 86, 87, g8 in the given fig(a).



$$U(G_P) = |s(g_{P-1} - g_c) - s(g_0 - g_c)| + \sum_{p=1}^{P-1} |s(g_p - g_c) - s(g_{p-1} - g_c)$$

s(x)=1 if x>=0 & S(x)=0 if x<0



$$T = p(g_0 \cdot g_1 - g_0 \cdot g_2 - g_0 \cdot g_3 - g_0 \cdot g_4 - g_0 \cdot g_5 - g_0 \cdot g_6 - g_0 \cdot g_7 - g_0 \cdot g_8 - g_0)$$

Calculated value of U(Gp) will be

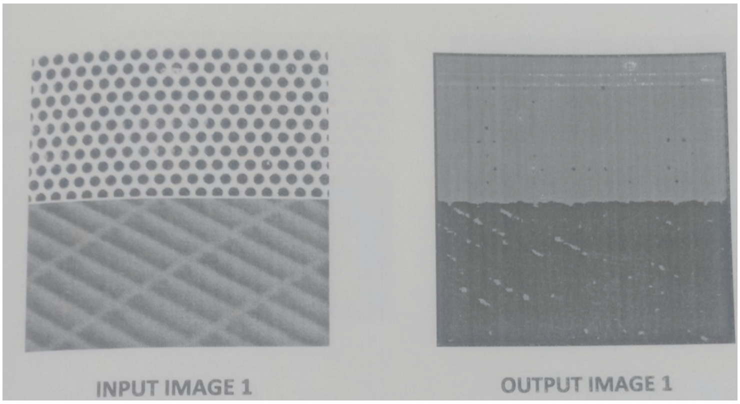
$$\mathrm{LBP}_{P,R}^{riu2} = \begin{cases} \sum_{p=0}^{P-1} s(g_p - g_c) & U(G_P) \leq 2 \\ P+1 & \text{otherwise .} \end{cases}$$

step 3: With the help of calculated ULBP we can calculate the texture vector it can be calculated in 5×5 window The texture vector can be calculated with the help of formula is where T- To store the values of Texture vector,we initialize an array of 10 elements

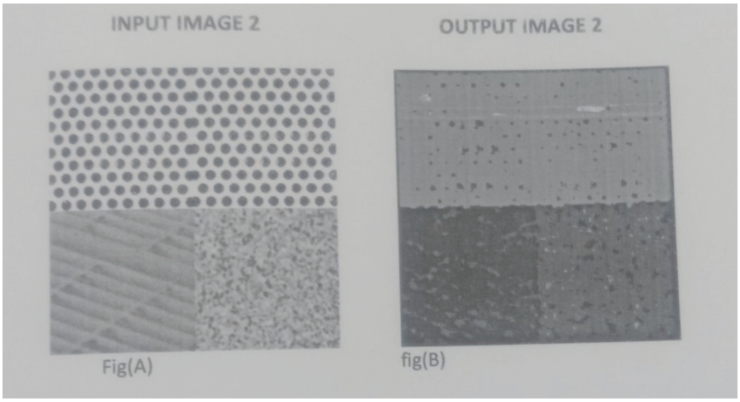
Step 4: Start K-means Algorithm We can randomly select the initial seed points (k-2 or k-3), now we will assign every pixel a label after finding the nearest cluster. After each iteration, cluster centroids will be recalculated based on membership and assignment of every pixel to newly formed cluster will start over again until the cluster calculated after iteration do not change from the clusters with which we started the iteration. C[2] it makes two cluster. C3)-it makes three cluster. Now we will make structure to initialize the cluster The summation of square root of the calculated 10 values of histogram and cluster will be multiplied Diff texture =1-(√ √ -0 Here is the histogram shows how many members each cluster contains and ij is limit for the texture image

Step 5: Using The Texture Vector Partition the Image in Different clusters After applying K-means, in output image we can see that the image is partitioned into the number of clusters and each cluster now covers a texture region.

Results 1 The following input/output images shows result Result In A) we can see that it is 2 class texture image. we apply the UBLP and k-means algorithm. According to the methodology described in section 4, the Output Image is produced (fig B). It clearly shows that image was divided into two segments, but there is small percentage of pixels mislabeled.



Result 2 The input image has three distinct texture regions. We applied k-means on texture vector and the output clearly snows that image is divided into three regions. Also seen some percentage of pixels mislabeled into other regions. Here we can see that the output image is segmented into three parts



13

Chapter 13

Coding of the project(Uniform Local Binary Pattern)

```
// Texture segmentation using Uniform Local Binary Pattern

#include
   #include
   #include
   #include
   #include

struct cluster
   {
   /* data */
   float i[10];
   float cumi[10];
   int memcount;
   };

//variable declaration
   unsigned char **allocateMem(int,int);
   void applyKmeans(unsigned char **,int,int, unsigned char
**);
   void calculateTexturevector(unsigned char **, int,int,float
*);
   void initializecluster(unsigned char **, struct * c);
   void calculateTexture(unsigned char **,int,int);
   void uniformPattern(unsigned char **, unsigned char
**,int,int,int,int);
   void localbinary(unsigned char **, unsigned char**,int,int,int,int);
   void contrast(unsigned char **, unsigned char **,int,int,int,int);
   void invariant(unsigned char **, unsigned char **,int,int);

int valassignment(int);
   float abslocal(float);
   float roundof(float);

// begining of main

void main()
   {
   int i, row, col,k,l;
   FILE *in,*out;
   unsigned char **p,**q,**lblImage;

// file open operation
   in = fopen("C:\texture_segmentation/raw.png","rb");
   if(in=NULL)
   {
   printf("there is no such file exists:\n");

}
```

```
   printf("take value for p");
   scanf("%d%d", &raw, &col);

//function call for memory alloaction
   p= allocateMem(row,col);
   q= allocateMem(row,col);

lblImage= allocateMem(row,col);

for(i=0;i
   {
   fread(p[i],sizeof(unsigned char), col,in);

}

uniformPattern(p,q,row,col,k,l);
   applyKmeans(q,row, col,k,lblImage);
   calculateTexture(q,row,col);
   invariant(p,q,col,row);
   contrast(p,q,row,col,k,l);
   localbinary(p,q,,row,col,k,l);

fclose();
   //file open operation
   out=fopen("C:\texture_segmentation/cluster2.raw","wb");

//filewrite operation

for(i=0;i
   {
   fwrite(lblImage[i], sizeof[char],col,out);
   }
   fclose(out);

//memorydeletion

for(i=0;i
   {
   free(p[i]);
   free(q[i]);
   free(lblImage[i]);

}

free(p);
   free(q);
   free(lblImage);


}

// end of main function

//rotine for memory allocation
   unsigned char ** allocateMem(int r,int c);
   unsigned char **m;
   int i,j;
   m=(unsigned char **) malloc(c * sizeof(unsigned char *));

if(m=null)
   {
   printf("1.Error");
   }
   for(i=0;i
   {
   m[i]=(unsignes char *) malloc(c * sizeof(unsignes char));
   if(m[i]==null)
   printf("2. Error");
   }
   for(i=0;i
   {
   for(j=0;j

{
   m[i][j]=0;

}
   }
   return m;
   }


// routineffor k mean cluster

void applyKmeans(unsigned char ** q,int row,int col, unsigned
char ** lblImage)
   {
   unsigned int consstatus;
   int clstLbl;
   int i,j,s,m;
   int cnt;
   float minDist, dist, delta, sigma, temp, error;
   float r[10]={0,0,0,,0,0,0};
   struct cluster clst[2];
   cnt=0;
   initializecluster(q, &clst[0]);
   conStatus=0;

while(! conStatus)
   {
   for=(i=2;i
   {
   for(j=2;j
   {
   for(s=0;s<10;s++)
   r[s]=0.0;
   //printf("\ndata points");
   calculateTexturevector(q,i,j,&r[0]);
   minDist=999.0;
   for (size_t m = 0; m < 2; m++)
   {
   /* code */
   delta = 0.0; dist=0.0;
   for (cnt = 0; cnt < 10; cnt++)
   {
   /* code */
   {sigma= r[cnt]* clst[m].i[cnt];
   delta +=pow(sigma, 0.5);
   }
   dist= pow(delta, 0.5);
   //printf("\n dist=%f", dist);

if(dist
   {
    clstLbl=m;
    minDist=dist;
    }
    }
    //printf("%f and label %d, minDist, clstLbl");
    for(cnt=0; cnt<10; cnt++)
    {
    clst[clstLbl.cumi[cnt]+= r[cnt];
    }
    clst[clstLbl].memcount +=1;
    lblImage[i][j]=(unsigned char) pow(2, clstLbl+6);
    }//inner for

}//outer for
    conStatus=1; temp=0.0;
    for(m=0;m<2;m++)
    {
    printf("\n cluster %d has %d points",m, clst[m].memcount);
    for(cnt=0;cnt<10;cnt++)
    {
    temp= clst[m].cumi[cnt];
    ///new mean center
    clst[m].cumi[cnt]= temp/clst[m].memcount;
    error = clst[m].cumi[cnt]-clst[m].i[cnt];

 if(abslocal(error)<0.0001)
    conStatus =1;;
    else
    conStatus =0;
```

```
}
   }
   // refresh cluster center

for(m=0;m<2;m++)
   {
   for (cnt=0; cnt<10; cnt++)
   {
   /* code */
   clst[m].i[cnt]= clst[m].cumi[cnt];
   clst[m].cumi[cnt]=0.0;
   }
   clst[m].memocount=0;
   }

}//end of while
   }//end of function
   //routine for initialize cluster
   void initializecluster(unsigned char **q, struct cluster * c)
   {
   int m,s,i,,j;
   float r[10]={0,0,0,0,0,0,0,0,0,0};
   //int xcoord[10]={};
   //int ycoord[10]={};
   int xcoord[2]={,};
   int xcoord[2]={,};
   for(m=0;m<2;m++)
   {
   i=xcoord[m];
   j=ycoord[m];
   calculateTexturevector(q,i,j,&r[0]);
```

```
    for(s=0;s<10;s++)
    {
    c[m].i[s]=r[s];
    //printf(”\t%0.3f”, c[m].i[s]);
    r[s]=0.0;
    c[m].cumi[s]=0.0;

}//inner for
    c[m].memcount=0;
    }//outer for
    }//end of function
    //routine for calculate textutervector
    void calcullateTexturevector(unsigned char **q, int i, int j,
float * r)
    {
    int k,l;
    for(k=-2;k<3; k++)
    {
    for(l=-2;;l<3;l++)
    {
    r[q[i+k][j+l] += #/#;
    //printf(”%.3f”, r[q[i+k][j+l]]);
    }
    }
    }//end of function
    // routine for calculation of histogram-value
    void calculateTexture(unsiged char **q, int row,int col)
    {
    int i,j,k,l;
    float probability;
    float histo[10]={0,0,0,0,0,0,0,0,0,0};
    for(i=1; i
    {
    for(j=1; j
    {
    for(k=-1;k<2;k++)
    {
    for(l=-1;l<2;l++)
    {
    printf(”\nq value %d “,q[i+j][j+i]);
    histo[q[i+j][j+i]]+=1.0;

}
    }
    printf(”\n Texture vector for (%d,%d)“,i,j);
    for (
    int s = 0; s < 10; s++)
    {
    /* code */
    probability = histo[s]/9.0;
    printf(”\t %.3f”,probability);
    histo[s]=0.0;

}

}//inner for
    }//outer for
    }//end of function
    //routine for lbp

void uniformPattern(unsigned char **p, unsigned char **q, int
row, int col,int k, int l)
```

```
   {
   {
   int i,cv,cw, lbp;
   int j, lastEle, count, frstVal;
   float delta1, delta2, tempdelta1, temdelta2;
   float theta, anglestep;
   int uValTwo, uvalOne, uVal;
   }
   for(i=1;i
   {
   for (j = 0; j < col-2; j++)
   {
   /* code */
   uValOne = 0; uValTwo =0;
   uVal =0;
   count=0; delta1=0;delta2=0;
   theta=0;cw=0;
   anglestep =( / )* ( );
   theta = anglestep;

cv=p[i][j];

frstVal=p[i]+[j+1];
   lastEle = frstVal;
   //print
   while (count<7)
   {
   /* code */
   delta1 = tempdelta1 = sin(theta);
   delta2 = tempdelta2 = cos(theta);
```

```
if (abslocal1((abslocal(tempdelta1)(abslocal(tempdelta2))))<.01)
   {
   /* code */

delta1=tempdelta1/abslocal(delta1);
   delta2= tempdelta2/abslocal(delta2);

}
   cw = p[i+ (int)(roundof(delta1), (int) roundof(delta2));]
   uValTwo += abslocal ( valassignment(cw-cv)) - valassign-
ment( lastEle-cv));
   theta += angelstep;
   lastEle=cv;
   count ++;
   }

uValOne = abslocal( valassignment(cw-cv) - valassign-
ment(frstVal - cv));
   uVal = uValOne + uValTwo;
   lbp=0;
   if( uVal<=2)
   {
   for (k=-1; k< 2; k++)
   {
   /* code */
   for (l = -1; l < 2; l++)
   {
   /* code */
   if (!((k==0) && (l==0)))
   {
   /* code */
   lbp += valassignment(p[i+k][j+1] - p[i][j]);
   }

}//inner for


}//outer for
   q[i][j]= lbp;

}
   else
   q[i][j]=9;
   }

}//inner for

}//outer for
   }

// routine for lbp

void loacalbinary(unsigned char **p, unsigned char **q, int
row, int col, int i, int j)
   {
   int i, cv, temp=0,t,j, count;
   for (i= 1; i
   {
   /* code */
   for (j=1; j < col-2; j++)
   {
   /* code */
```

```
    temp=0;
    count=0;
    cv=p[i][j];
    for (k=-1; k <2; k++)
    {
    /* code */
    for (l=-1; l < 2; l++)
    {
    /* code */
    t=p[i+k][j+2];
    if (!((k==0)&&(l==0)))

{
    /* code */
    if (t>=cv)
    {
    /* code */
    temp = temp + pow(2, count);
    count++;
    }

}// inner for


} //outer for
    q[i + j] = temp;

}// inner for

}//outer for
```

```
return;
   }
   }

//routine for invariant

void invariant(unsigned char **p, unsigned char **q, int row ,
int col)
   {
   int i, cv, cw, t,j, count;
   unsigned char s, leftshift, rightshift, fv;
   float delta1, delta2, tempdelta1, tempdelta2;

double theta, anglestep;
   for( i=1; i< row-2; i++)
   {
   for( j=1; j< col-2; j++)
   {
   count=0;
   delta1=0.0; delta2=0.0; theta=0; cw=0;
   s=0*00;
   angelstep=(/)*( );

cv=p[i][j]
   while(count<8)
   {
   delta1=tempdelta1=sin(theta);
   delta2=tempdelta2=sin(theta);

if(abslocal((abslocal(tempdelta1)abslocal(tempdelta2)))<.01)
   {
```

```
   delta1=tempdelta1/abslocal(delta1);
   delta2=tempdelta2/abslocal(delta2);
   }

cw=p[i+(int) roundof(delta1)] [j+(int) roundoof(delta2)];
   if(cw>=cv)
   {
   t=1;
   }
   else
   {
   t=0;
   }

theta += angelstep;
   if(t==1)
   s= s | 0*01« count;
   count++;
   }
   int minval =256;
   for(int n=1;n<8;n++)
   {
   leftshift =s<
   rightshift = s»(8-n)

fv= rightshift | leftshift;

if(fv
   {
   q[i][j ]= fv;
   minval= fv;
   }
   }
   }//inner for

}//outer for
   }//end of function


// routine for contrast
   void contrast(unsigned char **p, unsigned **q, int row, int
col, int k, int l)
   {
   int temp1, temp2, temp, count1, count2, i, j, cv,t;
   float count;

for( i = 1; i
   {
   for(j=1;j
   {
   temp1=0;
   temp2=0;
   temp=0;
   count1=0;
   count2=0;
   cv=p[i][j];
   for(k=-1; k<2;k++)

{
   for( l=-1; l<2;l++)
   {
   t= p[i+k][j+l];
if(!((k==0)&&(l==0)))
{
if(t>=cv)
{
temp1=temp1+t;
count1++;
}
else
{
temp2= temp2+t;
count2++;
}
}
}//inner
}//outer
if(count1 !=0)
}
}
}
```

14

References

Image Analysis