**灰度共生矩阵**

        灰度共生矩阵定义为像素对的联合分布概率，是一个对称矩阵，它不仅反映图像灰度在相邻的方向、相邻间隔、变化幅度的综合信息，但也反映了相同的灰度级像素之间的位置分布特征，是计算纹理特征的基础。

       设f(x,y)为一幅数字图像，其大小为M×N，灰度级别为Ng，则满足一定空间关系的灰度共生矩阵为：

http://img.my.csdn.net/uploads/201302/07/1360205488_7580.gif

        其中#(x)表示集合x中的元素个数，显然P为Ng×Ng的矩阵，若(x1,y1)与(x2,y2)间距离为d,两者与坐标横轴的夹角为θ，则可以得到各种间距及角度的灰度共生矩阵(i,j,d,θ)。其中元素(i,j)的值表示一个灰度为i，另一个灰度为j的两个相距为d的像素对在角的方向上出现的次数。

在计算得到共生矩阵之后，往往不是直接应用计算的灰度共生矩阵，而是在此基础上计算纹理特征量，我们经常用反差、能量、熵、相关性等特征量来表示纹理特征。

        (1) 反差：又称为对比度，度量矩阵的值是如何分布和图像中局部变化的多少，反应了图像的清晰度和纹理的沟纹深浅。纹理的沟纹越深，反差越大，效果清晰；反之，对比值小，则沟纹浅，效果模糊。

http://img.my.csdn.net/uploads/201302/07/1360205775_8290.gif

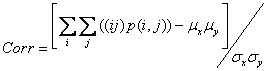
        (2) 能量：是灰度共生矩阵各元素值的平方和，是对图像纹理的灰度变化稳定程度的度量，反应了图像灰度分布均匀程度和纹理粗细度。能量值大表明当前纹理是一种规则变化较为稳定的纹理。

http://img.my.csdn.net/uploads/201302/07/1360205806_5194.gif

        (3) 熵：是图像包含信息量的随机性度量。当共生矩阵中所有值均相等或者像素值表现出最大的随机性时，熵最大；因此熵值表明了图像灰度分布的复杂程度，熵值越大，图像越复杂。

http://img.my.csdn.net/uploads/201302/07/1360205863_1033.gif

        (4) 相关性：也称为同质性，用来度量图像的灰度级在行或列方向上的相似程度，因此值的大小反应了局部灰度相关性，值越大，相关性也越大。



**应用**

    由上面的叙述知道，可以根据各种间距和角度计算灰度共生矩阵，下面程序中给定了间距，根据传入的参数计算：

**[cpp]** [view plain](http://blog.csdn.net/cxf7394373/article/details/6988229) [copy](http://blog.csdn.net/cxf7394373/article/details/6988229)

1. #define GLCM\_DIS 3  //灰度共生矩阵的统计距离
2. #define GLCM\_CLASS 16 //计算灰度共生矩阵的图像灰度值等级化
3. #define GLCM\_ANGLE\_HORIZATION 0  //水平
4. #define GLCM\_ANGLE\_VERTICAL   1  //垂直
5. #define GLCM\_ANGLE\_DIGONAL    2  //对角
6. **int** calGLCM(IplImage\* bWavelet,**int** angleDirection,**double**\* featureVector)
7. {
8. **int** i,j;
9. **int** width,height;
11. **if**(NULL == bWavelet)
12. **return** 1;
14. width = bWavelet->width;
15. height = bWavelet->height;
17. **int** \* glcm = **new** **int**[GLCM\_CLASS \* GLCM\_CLASS];
18. **int** \* histImage = **new** **int**[width \* height];
20. **if**(NULL == glcm || NULL == histImage)
21. **return** 2;
23. //灰度等级化---分GLCM\_CLASS个等级
24. uchar \*data =(uchar\*) bWavelet->imageData;
25. **for**(i = 0;i < height;i++){
26. **for**(j = 0;j < width;j++){
27. histImage[i \* width + j] = (**int**)(data[bWavelet->widthStep \* i + j] \* GLCM\_CLASS / 256);
28. }
29. }
31. //初始化共生矩阵
32. **for** (i = 0;i < GLCM\_CLASS;i++)
33. **for** (j = 0;j < GLCM\_CLASS;j++)
34. glcm[i \* GLCM\_CLASS + j] = 0;
36. //计算灰度共生矩阵
37. **int** w,k,l;
38. //水平方向
39. **if**(angleDirection == GLCM\_ANGLE\_HORIZATION)
40. {
41. **for** (i = 0;i < height;i++)
42. {
43. **for** (j = 0;j < width;j++)
44. {
45. l = histImage[i \* width + j];
46. **if**(j + GLCM\_DIS >= 0 && j + GLCM\_DIS < width)
47. {
48. k = histImage[i \* width + j + GLCM\_DIS];
49. glcm[l \* GLCM\_CLASS + k]++;
50. }
51. **if**(j - GLCM\_DIS >= 0 && j - GLCM\_DIS < width)
52. {
53. k = histImage[i \* width + j - GLCM\_DIS];
54. glcm[l \* GLCM\_CLASS + k]++;
55. }
56. }
57. }
58. }
59. //垂直方向
60. **else** **if**(angleDirection == GLCM\_ANGLE\_VERTICAL)
61. {
62. **for** (i = 0;i < height;i++)
63. {
64. **for** (j = 0;j < width;j++)
65. {
66. l = histImage[i \* width + j];
67. **if**(i + GLCM\_DIS >= 0 && i + GLCM\_DIS < height)
68. {
69. k = histImage[(i + GLCM\_DIS) \* width + j];
70. glcm[l \* GLCM\_CLASS + k]++;
71. }
72. **if**(i - GLCM\_DIS >= 0 && i - GLCM\_DIS < height)
73. {
74. k = histImage[(i - GLCM\_DIS) \* width + j];
75. glcm[l \* GLCM\_CLASS + k]++;
76. }
77. }
78. }
79. }
80. //对角方向
81. **else** **if**(angleDirection == GLCM\_ANGLE\_DIGONAL)
82. {
83. **for** (i = 0;i < height;i++)
84. {
85. **for** (j = 0;j < width;j++)
86. {
87. l = histImage[i \* width + j];
89. **if**(j + GLCM\_DIS >= 0 && j + GLCM\_DIS < width && i + GLCM\_DIS >= 0 && i + GLCM\_DIS < height)
90. {
91. k = histImage[(i + GLCM\_DIS) \* width + j + GLCM\_DIS];
92. glcm[l \* GLCM\_CLASS + k]++;
93. }
94. **if**(j - GLCM\_DIS >= 0 && j - GLCM\_DIS < width && i - GLCM\_DIS >= 0 && i - GLCM\_DIS < height)
95. {
96. k = histImage[(i - GLCM\_DIS) \* width + j - GLCM\_DIS];
97. glcm[l \* GLCM\_CLASS + k]++;
98. }
99. }
100. }
101. }
103. //计算特征值
104. **double** entropy = 0,energy = 0,contrast = 0,homogenity = 0;
105. **for** (i = 0;i < GLCM\_CLASS;i++)
106. {
107. **for** (j = 0;j < GLCM\_CLASS;j++)
108. {
109. //熵
110. **if**(glcm[i \* GLCM\_CLASS + j] > 0)
111. entropy -= glcm[i \* GLCM\_CLASS + j] \* log10(**double**(glcm[i \* GLCM\_CLASS + j]));
112. //能量
113. energy += glcm[i \* GLCM\_CLASS + j] \* glcm[i \* GLCM\_CLASS + j];
114. //对比度
115. contrast += (i - j) \* (i - j) \* glcm[i \* GLCM\_CLASS + j];
116. //一致性
117. homogenity += 1.0 / (1 + (i - j) \* (i - j)) \* glcm[i \* GLCM\_CLASS + j];
118. }
119. }
120. //返回特征值
121. i = 0;
122. featureVector[i++] = entropy;
123. featureVector[i++] = energy;
124. featureVector[i++] = contrast;
125. featureVector[i++] = homogenity;
127. **delete**[] glcm;
128. **delete**[] histImage;
129. **return** 0;
130. }