[Python+Opencv进行识别相似图片](http://blog.csdn.net/feimengjuan/article/details/51279629)

http://blog.csdn.net/feimengjuan/article/details/51279629

在网上看到python做图像识别的相关文章后，真心感觉python的功能实在太强大，因此将这些文章总结一下，建立一下自己的知识体系。

当然了，图像识别这个话题作为计算机科学的一个分支，不可能就在本文简单几句就说清，所以**本文只作基本算法的科普向**。

看到一篇博客是介绍这个，但他用的是PIL中的Image实现的，感觉比较麻烦，于是利用Opencv库进行了更简洁化的实现。

相关背景

要识别两张相似图像，我们从感性上来谈是怎么样的一个过程？首先我们会区分这两张相片的类型，例如是风景照，还是人物照。风景照中，是沙漠还是海洋，人物照中，两个人是不是都是国字脸，还是瓜子脸（还是倒瓜子脸……哈哈……）。

那么从机器的角度来说也是这样的，先识别图像的特征，然后再相比。

很显然，在没有经过训练的计算机(即建立模型)，那么计算机很难区分什么是海洋，什么是沙漠。但是计算机很容易识别到图像的像素值。

因此，在图像识别中，颜色特征是最为常用的。（其余常用的特征还有纹理特征、形状特征和空间关系特征等）

其中又分为

* 直方图
* 颜色集
* 颜色矩
* 聚合向量
* 相关图

直方图计算法

这里先用直方图进行简单讲述。

先借用一下恋花蝶的图片，

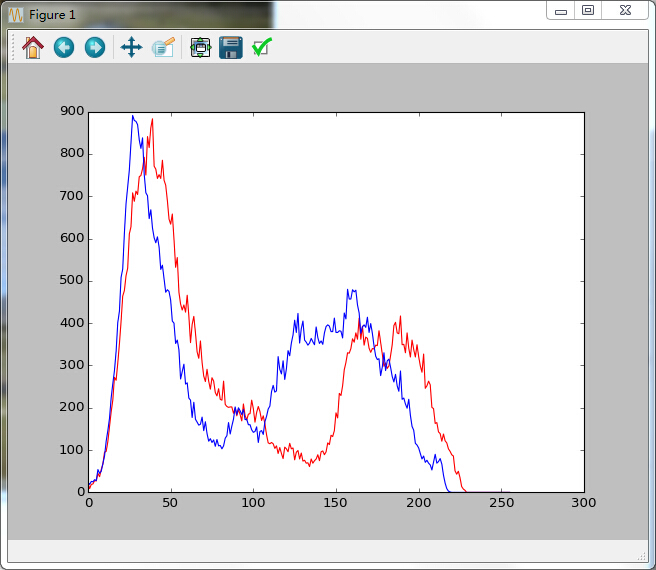




从肉眼来看，这两张图片大概也有八成是相似的了。

在[**Python**](http://lib.csdn.net/base/11)中利用opencv中的calcHist()方法获取其直方图数据，返回的结果是一个列表，使用matplotlib，画出了这两张图的直方图数据图

如下：



是的，我们可以明显的发现，两张图片的直方图还是比较重合的。所以利用直方图判断两张图片的是否相似的方法就是，计算其直方图的重合程度即可。

计算方法如下：

https://segmentfault.com/image?src=http://p.blog.csdn.net/images/p_blog_csdn_net/lanphaday/277551/o_gs.PNG&objectId=1190000004467183&token=3d8d05393711ef93069741d8586f7c47

其中gi和si是分别指两条曲线的第i个点。

最后计算得出的结果就是就是其相似程度。

不过，这种方法有一个明显的弱点，就是他是按照颜色的全局分布来看的，无法描述颜色的局部分布和色彩所处的位置。

也就是假如一张图片以蓝色为主，内容是一片蓝天，而另外一张图片也是蓝色为主，但是内容却是妹子穿了蓝色裙子，那么这个算法也很可能认为这两张图片的相似的。

缓解这个弱点有一个方法就是利用Image的crop方法把图片等分，然后再分别计算其相似度，最后综合考虑。

图像指纹与汉明距离

在介绍下面其他判别相似度的方法前，先补充一些概念。第一个就是**图像指纹**

图像指纹和人的指纹一样，是身份的象征，而图像指纹简单点来讲，**就是将图像按照一定的哈希算法，经过运算后得出的一组二进制数字。**

说到这里，就可以顺带引出**汉明距离**的概念了。

假如一组二进制数据为101，另外一组为111，那么显然把第一组的第二位数据0改成1就可以变成第二组数据111，所以两组数据的汉明距离就为1

简单点说，汉明距离就是**一组二进制数据变成另一组数据所需的步骤数**，显然，这个数值可以**衡量两张图片的差异**，汉明距离越小，则代表相似度越高。汉明距离为0，即代表两张图片完全一样。

如何计算得到汉明距离，请看下面三种哈希算法

平均哈希法(aHash)

此算法是基于比较灰度图每个像素与平均值来实现的

一般步骤

* 1.缩放图片，一般大小为8\*8，64个像素值。
* 2.转化为灰度图
* 3.计算平均值：计算进行灰度处理后图片的所有像素点的平均值，直接用numpy中的mean()计算即可。
* 4.比较像素灰度值：遍历灰度图片每一个像素，如果大于平均值记录为1，否则为0.
* 5.得到信息指纹：组合64个bit位，顺序随意保持一致性。

最后比对两张图片的指纹，获得汉明距离即可。

感知哈希算法(pHash)

平均哈希算法过于严格，不够精确，更适合搜索缩略图，为了获得更精确的结果可以选择感知哈希算法，它采用的是DCT（离散余弦变换）来降低频率的方法

一般步骤：

* 缩小图片：32 \* 32是一个较好的大小，这样方便DCT计算
* 转化为灰度图
* 计算DCT：利用Opencv中提供的dct()方法，注意输入的图像必须是32位浮点型，所以先利用numpy中的float32进行转换
* 缩小DCT：DCT计算后的矩阵是32 \* 32，保留左上角的8 \* 8，这些代表的图片的最低频率
* 计算平均值：计算缩小DCT后的所有像素点的平均值。
* 进一步减小DCT：大于平均值记录为1，反之记录为0.
* 得到信息指纹：组合64个信息位，顺序随意保持一致性。

最后比对两张图片的指纹，获得汉明距离即可。

dHash算法

相比pHash，dHash的速度要快的多，相比aHash，dHash在效率几乎相同的情况下的效果要更好，它是基于渐变实现的。

步骤：

* 缩小图片：收缩到9\*8的大小，以便它有72的像素点
* 转化为灰度图
* 计算差异值：dHash算法工作在相邻像素之间，这样每行9个像素之间产生了8个不同的差异，一共8行，则产生了64个差异值
* 获得指纹：如果左边的像素比右边的更亮，则记录为1，否则为0.  
  最后比对两张图片的指纹，获得汉明距离即可。

整个的代码实现如下：

**[python]** [view plain](http://blog.csdn.net/feimengjuan/article/details/51279629) [copy](http://blog.csdn.net/feimengjuan/article/details/51279629)

1. # -\*- coding: utf-8 -\*-
2. #feimengjuan
3. # 利用python实现多种方法来实现图像识别
5. **import** cv2
6. **import** numpy as np
7. **from** matplotlib **import** pyplot as plt
9. # 最简单的以灰度直方图作为相似比较的实现
10. **def** classify\_gray\_hist(image1,image2,size = (256,256)):
11. # 先计算直方图
12. # 几个参数必须用方括号括起来
13. # 这里直接用灰度图计算直方图，所以是使用第一个通道，
14. # 也可以进行通道分离后，得到多个通道的直方图
15. # bins 取为16
16. image1 = cv2.resize(image1,size)
17. image2 = cv2.resize(image2,size)
18. hist1 = cv2.calcHist([image1],[0],None,[256],[0.0,255.0])
19. hist2 = cv2.calcHist([image2],[0],None,[256],[0.0,255.0])
20. # 可以比较下直方图
21. plt.plot(range(256),hist1,'r')
22. plt.plot(range(256),hist2,'b')
23. plt.show()
24. # 计算直方图的重合度
25. degree = 0
26. **for** i **in** range(len(hist1)):
27. **if** hist1[i] != hist2[i]:
28. degree = degree + (1 - abs(hist1[i]-hist2[i])/max(hist1[i],hist2[i]))
29. **else**:
30. degree = degree + 1
31. degree = degree/len(hist1)
32. **return** degree
34. # 计算单通道的直方图的相似值
35. **def** calculate(image1,image2):
36. hist1 = cv2.calcHist([image1],[0],None,[256],[0.0,255.0])
37. hist2 = cv2.calcHist([image2],[0],None,[256],[0.0,255.0])
38. # 计算直方图的重合度
39. degree = 0
40. **for** i **in** range(len(hist1)):
41. **if** hist1[i] != hist2[i]:
42. degree = degree + (1 - abs(hist1[i]-hist2[i])/max(hist1[i],hist2[i]))
43. **else**:
44. degree = degree + 1
45. degree = degree/len(hist1)
46. **return** degree
48. # 通过得到每个通道的直方图来计算相似度
49. **def** classify\_hist\_with\_split(image1,image2,size = (256,256)):
50. # 将图像resize后，分离为三个通道，再计算每个通道的相似值
51. image1 = cv2.resize(image1,size)
52. image2 = cv2.resize(image2,size)
53. sub\_image1 = cv2.split(image1)
54. sub\_image2 = cv2.split(image2)
55. sub\_data = 0
56. **for** im1,im2 **in** zip(sub\_image1,sub\_image2):
57. sub\_data += calculate(im1,im2)
58. sub\_data = sub\_data/3
59. **return** sub\_data
61. # 平均哈希算法计算
62. **def** classify\_aHash(image1,image2):
63. image1 = cv2.resize(image1,(8,8))
64. image2 = cv2.resize(image2,(8,8))
65. gray1 = cv2.cvtColor(image1,cv2.COLOR\_BGR2GRAY)
66. gray2 = cv2.cvtColor(image2,cv2.COLOR\_BGR2GRAY)
67. hash1 = getHash(gray1)
68. hash2 = getHash(gray2)
69. **return** Hamming\_distance(hash1,hash2)
71. **def** classify\_pHash(image1,image2):
72. image1 = cv2.resize(image1,(32,32))
73. image2 = cv2.resize(image2,(32,32))
74. gray1 = cv2.cvtColor(image1,cv2.COLOR\_BGR2GRAY)
75. gray2 = cv2.cvtColor(image2,cv2.COLOR\_BGR2GRAY)
76. # 将灰度图转为浮点型，再进行dct变换
77. dct1 = cv2.dct(np.float32(gray1))
78. dct2 = cv2.dct(np.float32(gray2))
79. # 取左上角的8\*8，这些代表图片的最低频率
80. # 这个操作等价于c++中利用opencv实现的掩码操作
81. # 在python中进行掩码操作，可以直接这样取出图像矩阵的某一部分
82. dct1\_roi = dct1[0:8,0:8]
83. dct2\_roi = dct2[0:8,0:8]
84. hash1 = getHash(dct1\_roi)
85. hash2 = getHash(dct2\_roi)
86. **return** Hamming\_distance(hash1,hash2)
88. # 输入灰度图，返回hash
89. **def** getHash(image):
90. avreage = np.mean(image)
91. hash = []
92. **for** i **in** range(image.shape[0]):
93. **for** j **in** range(image.shape[1]):
94. **if** image[i,j] > avreage:
95. hash.append(1)
96. **else**:
97. hash.append(0)
98. **return** hash

101. # 计算汉明距离
102. **def** Hamming\_distance(hash1,hash2):
103. num = 0
104. **for** index **in** range(len(hash1)):
105. **if** hash1[index] != hash2[index]:
106. num += 1
107. **return** num

110. **if** \_\_name\_\_ == '\_\_main\_\_':
111. img1 = cv2.imread('10.jpg')
112. cv2.imshow('img1',img1)
113. img2 = cv2.imread('11.jpg')
114. cv2.imshow('img2',img2)
115. degree = classify\_gray\_hist(img1,img2)
116. #degree = classify\_hist\_with\_split(img1,img2)
117. #degree = classify\_aHash(img1,img2)
118. #degree = classify\_pHash(img1,img2)
119. **print** degree
120. cv2.waitKey(0)