Experiment 10

517030910356 谢知晖

目录

1 实验准备

1.1 实验环境

本次实验环境依旧为 Ubuntu 平台 (版本号 18.04) 下的 Python(版本号 2.7)。
另外，实验中还用到的 Python 库有:

1. OpenCV(版本号 3.4.3)
2. NumPy(版本号 1.15.4)
3. Matplotlib(版本号 2.2.3)

1.2 实验目的

本次的实验较为简单，但作为图像处理技术的第一次实验，还是有许多启发性的内容。

实验的主题围绕两张简单的"png" 格式的图片展开。通过对图片进行读取，我们能够将图像转化为数字。而进一步地应用统计学原理，我们便能通过查看像素的分布从图片中提取出简单的特征，并得到诸如灰度、梯度直方图等直观的特征图像。

实验所指向的是图像处理技术的重点: 特征提取。虽然我们并不能从简单的实验中提取出十分重要的特征，但实验让我们了解到了图像在计算机中的表示和存储方式、图像的基本特征以及特征提取的大致流程。这对于后续的实验开展无疑是具有引导作用的。

1.3 实验原理

在这里，我将简单地介绍图像特征提取的一些基本概念。

什么是' 特征' 至今为止特征没有万能和精确的定义。特征的精确定义往往由问题或者应用类型决定。特征是一个数字图像中“有趣”的部分，它是许多计算机图像分析算法的起点。因此一个算法是否成功往往由它使用和定义的特征决定。因此特征提取最重要的一个特性是" 可重复性"：同一场景的不同图像所提取的特征应该是相同的。

几种常见特征 由于许多计算机图像算法使用特征提取作为其初级计算步骤，因此有大量特征提取算法被发展，其提取的特征各种各样，它们的计算复杂性和可重复性也非常不同。

边缘 边缘是组成两个图像区域之间边界 (或边缘) 的像素。一般一个边缘的形状可以是任意的，还可能包括交叉点。在实践中边缘一般被定义为图像中拥有大的梯度的点组成的子集。

角 角是图像中点似的特征，在局部它有两维结构。早期的算法首先进行边缘检测，然后分析边缘的走向来寻找边缘突然转向（角）。后来发展的算法不再需要边缘检测这个步骤，而是可以直接在图像梯度中寻找高度曲率。

区域 与角不同的是区域描写一个图像中的一个区域性的结构，但是区域也可能仅由一个像素组成，因此许多区域检测也可以用来监测角。一个区域监测器检测图像中一个对于角监测器来说太平滑的区域。区域检测可以被想象为把一张图像缩小，然后在缩小的图像上进行角检测。

特征提取 特征被检测后它可以从图像中被抽取出来。这个过程可能需要许多图像处理的计算机。其结果被称为特征描述或者特征向量。常见的图像特征有: 颜色特征、纹理特征、形状特征、空间关系特征等。在此，我仅对纹理特征进行展开介绍。

纹理特征的特点 纹理特征是一种全局特征，它描述了图像或图像区域所对应景物的表面性质。与颜色特征不同，纹理特征不是基于像素点的特征，它需要在包含多个像素点的区域中进行统计计算。在模式匹配中，这种区域性的特征具有较大的优越性，不会由于局部的偏差而无法匹配成功。作为一种统计特征，纹理特征常具有旋转不变性，并且对于噪声有较强的抵抗能力。

常用的纹理特征提取方法 纹理特征的提取分为基于结构的方法和基于统计数据的方法。一个基于结构的纹理特征提取方法是将所要检测的纹理进行建模，在图像中搜索重复的模式。该方法对人工合成的纹理识别效果较好。但对于交通图像中的纹理识别，基于统计数据的方法效果较好。

2 实验过程

本次的两个实验所选用的测试图片为:

图片 1

图片 2

2.1 Exp10-1

实验的第一个练习为颜色直方图的计算。

2.1.1 实验步骤

图片读取 由于颜色直方图统计的是 RGB 三种颜色分量占图片总能量的相对比例，我们必须用彩色方式读入图片。

```
image = cv2.imread(img_file, cv2.IMREAD_COLOR)
```

这样，image 得到的便是一个 $W \times H \times 3$ 的 NumPy 数组对象。其中第三维为 RGB 颜色空间。

统计比例 得益于功能强大的 NumPy 库，统计工作十分简单。我分别对第三维的三种像素求总能量，并得到各自的比例。

```
b = np.sum(image[:, :, 0])
g = np.sum(image[:, :, 1])
r = np.sum(image[:, :, 2])
total = r + g + b

per_b = float(b) / total
per_g = float(g) / total
per_r = 1 - per_b - per_g

result = [per_b, per_g, per_r]
```

作出直方图 在得到数据后，我们可以通过 Matplotlib 作出直方图。

首先，我设定横坐标的刻度，并对图像命名 (根据文件名)。

```
ind = np.arange(3)
plt.xticks(ind, ('B', 'G', 'R'))
plt.title("Color Histogram of " + img_file.split('.')[0])
```

然后，我们根据数据作出条形图。条形图的绘制是通过函数'bar' 实现的。

```
plt.bar(ind, result, width=1, color=['blue', 'green', 'red'])
plt.show()
```

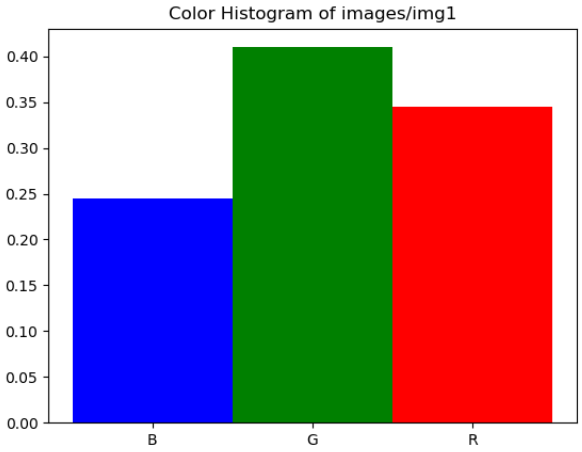
get_color_his 最后将上述功能封装为函数"get_color_his"。其输入为图片文件，返回结果为三种颜色分量相对比例的列表，并将直方图显示出来。

```
def get_color_his(img_file):
    image = cv2.imread(img_file, cv2.IMREAD_COLOR)

    b = np.sum(image[:, :, 0])
    g = np.sum(image[:, :, 1])
    r = np.sum(image[:, :, 2])
    total = r + g + b

    per_b = float(b) / total
```

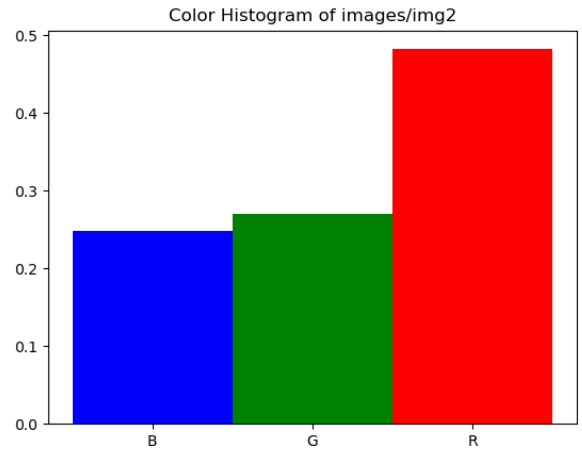
```
        per_g = float(g) / total
        per_r = 1 - per_b - per_g

        result = [per_b, per_g, per_r]

        ind = np.arange(3)
        plt.xticks(ind, ('B', 'G', 'R'))
        plt.title("Color Histogram of " + img_file.split('.')[0])
        plt.bar(ind, result, width=1, color=['blue', 'green', 'red'])

        plt.show()

        return result
```

2.1.2 实验结果

对于图片 1，由于图片中有大片草地，我设想图片中的 G 能量最高；而对于图片 2，女子的头发为红色，R 的能量应该是最高的。而我得到了直方图结果也印证了这样的直觉。



img1 的颜色直方图



img2 的颜色直方图

2.2 Exp10-2

在练习一的基础上，我们进一步提取图像的灰度直方图和梯度直方图。

2.2.1 实验步骤

灰度直方图 灰度直方图的实现和颜色直方图没有太大的区别，大致流程不再赘述。只不过这次我们统计的是各个灰度值占所有像素灰度值的比例。

为此，我用函数"get_gray_his" 实现此功能。函数仍然接受一个图片文件，输出各灰度值像素数目的相对比例，并显示直方图。

```
def get_gray_his(img_file):
    image = cv2.imread(img_file, cv2.IMREAD_GRAYSCALE)
    count_gray = [0] * 256
    total = float((image.shape[0] - 2) * (image.shape[1] - 2))

    for i in range(image.shape[0]):
        for j in range(image.shape[1]):
            count_gray[image[i, j]] += 1 / total

    ind = np.arange(256)
    plt.title("Gray Histogram of " + img_file.split('.')[0])
    plt.bar(ind, count_gray, width=1)

    plt.show()

    return count_gray
```

梯度直方图 梯度的计算同样是对于图像的灰度值进行的。但每个像素的梯度并不直接由它的灰度决定，而是由它相邻的上下左右四个像素决定。

函数"get_grad_his" 中，对于每个非最外围像素，我分别计算了它们的梯度强度，并统计在 count_grad 中。直方图的显示与灰度直方图类似。

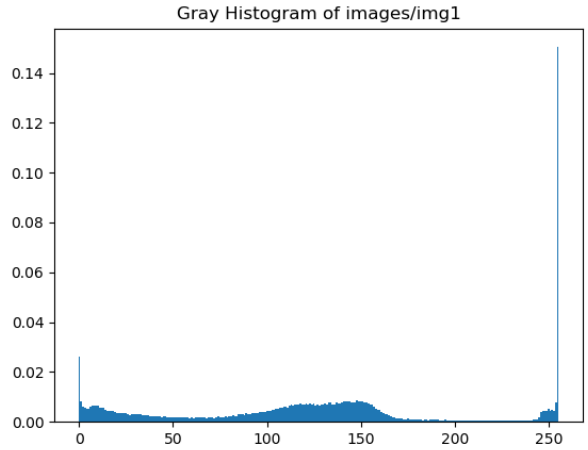
```
def get_grad_his(img_file):
    image = cv2.imread(img_file, cv2.IMREAD_GRAYSCALE)
    count_grad = [0] * 361
    total = float((image.shape[0] - 2) * (image.shape[1] - 2))

    for i in range(1, image.shape[0] - 1):
        for j in range(1, image.shape[1] - 1):
            i_x = int(image[i + 1, j]) - int(image[i - 1, j])
            i_y = int(image[i, j + 1]) - int(image[i, j - 1])

            seth = int((i_x ** 2 + i_y ** 2) ** 0.5)
            count_grad[seth] += 1 / total

    ind = np.arange(361)
    plt.title("Grad Histogram of " + img_file.split('.')[0])
    plt.bar(ind, count_grad, width=1)

    plt.show()
```

```
    return count_grad
```

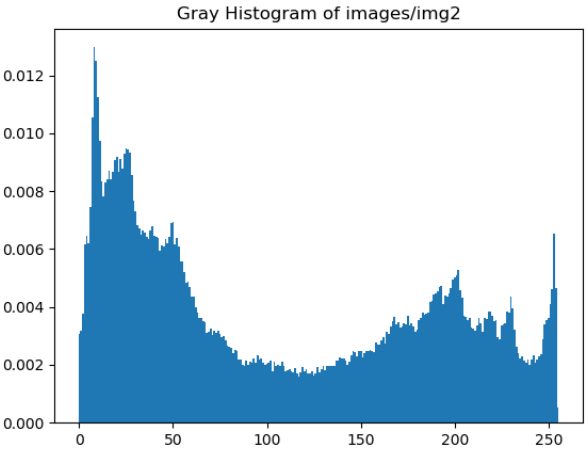
2.2.2 实验结果

灰度直方图 灰度直方图反映了图像明暗程度。

图片 1 中的草地和树林在人的观感上偏暗偏密，而灰度图显示有很大比例的灰度值都在 255 附近。图片 2 中既有肤色的缓和，也有发色的暗处，灰度分布应该更广，结果也证实了这样的猜测。



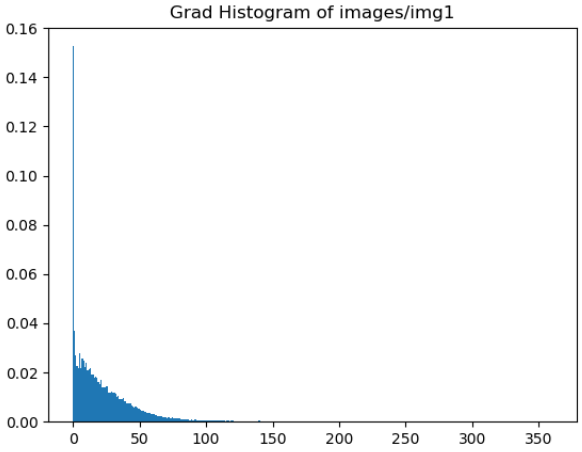
img1 的灰度直方图



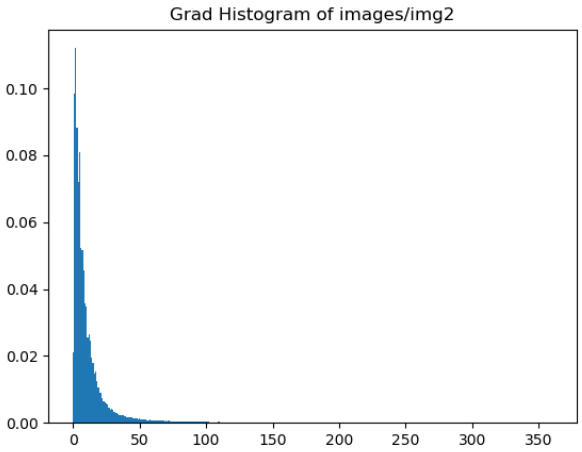
img2 的灰度直方图

梯度直方图 梯度直方图侧重于图片纹理的体现。

从整体上看，两张图片中都没有太多横竖的纹理，除了图片 1 中高尔夫球表面的纹理较为复杂外，大部分的图像过度都是比较自然的。而我得到的梯度直方图显示两张图片确实在梯度分布上都集中在较小范围内，不过图片 1 的范围相对较广。



img1 的梯度直方图



img2 的梯度直方图

3 实验感想

本次实验内容虽然不多，但还是有要总结的地方的。

由于 OpenCV 与 NumPy 两库之间的关联度极大，在进行图像特征提取时，我们应该尽可能地发挥 NumPy 的优势，以提高效率。

如当我计算 RGB 分别的总能量时，我使用了 NumPy 中的方法'sum'。

```
b = np.sum(image[:, :, 0])
g = np.sum(image[:, :, 1])
r = np.sum(image[:, :, 2])
```

类似的累加工作当然也能完成目的。但是考虑到 NumPy 的内部优化，这样的效率是肯定不如使用 NumPy 的方法来的快的。

事实上，更进一步的讨论来自于我在知乎上浏览过的一篇文章[1]。文章中讨论的是彩色图像转黑白图像这样一个简单的问题。其核心公式为 $Y = 0.299R + 0.587G + 0.114B$。作者针

[1] https://www.zhihu.com/question/287421003/answer/528275532?utm_source=qq&utm_medium=social&utm_oi=801142017282445312

对这个式子不断进行优化，从去浮点运算到并行计算，到最后竟将原本 120s 的程序优化到仅需 0.5s。

这样的尝试不禁引发我的思考。在图像处理中往往会面对规模庞大的数据，每步操作效率的细小差别将会被无限放大。效率在图像处理技术中应该时刻得到我们的关注。

References

[1] 图像特征提取: https://www.jianshu.com/p/b520e11e7d4a

[2] 纹理特征提取: https://blog.csdn.net/abcjennifer/article/details/7425483