电类工程导论(C 类)实验 10 报告

贾萧松 516030910548

一、实验概述

在 Lab10 中，我们完成了 opencv2.4.11 的配置，并学习了其基本读取图像方法，然后学习了图像特征抽取的相关知识，使用其完成了颜色直方图，灰度直方图，灰度梯度直方图的绘制（我使用了 matplotlib 在 Python 中完成绘图）。

二、实验环境

Ubuntu14.04+Pythton2.7+opencv2.4.11+numpy1.13.3+matplotlib2.1.0

三、实验内容

1. Ex1

a. 理论知识：

彩色图像的每个像素由红色(R)、绿色(G)、蓝色(B)三个颜色分量构成，从而能够呈现多种色彩。一幅宽为W，高为H的彩色图像在计算机中用一个W×H×3的矩阵存储，其中最后一个维度表示RGB颜色空间。

$$I_{\mathrm{RGB}}(x,y,c)$$
$$1 \le x \le W$$
$$1 \le y \le H$$
$$c = \{0,1,2\}$$
$$0 \le I_{\mathrm{RGB}}(x,y,c) \le 255$$

彩色图像I(x, y, c)的颜色直方图是它三个颜色分量总能量的相对比例。

某一颜色分量的总能量：

$$E(c) = \sum_{x=0}^{W-1} \sum_{y=0}^{H-1} I(x,y,c)$$

某一颜色分量的能量的相对比例：

$$H(c) = \frac{E(c)}{\sum_{i=0}^{2} E(i)}$$

b. 代码实现：

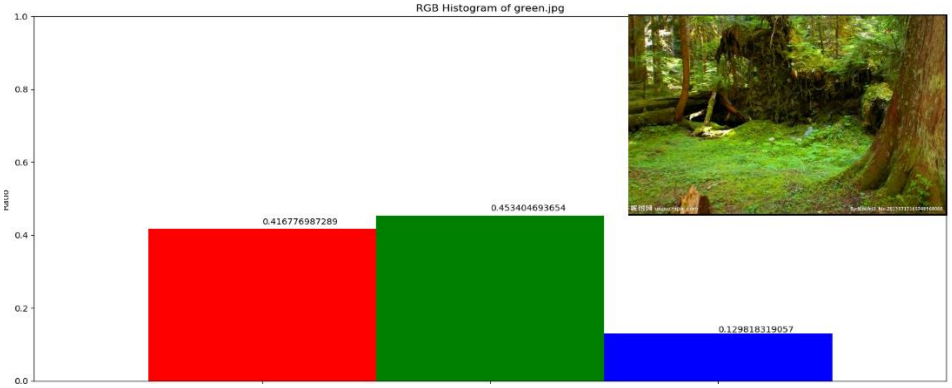
```
def drawRGB_hist(filename):
    img = cv2.imread(filename, cv2.IMREAD_COLOR)
    B, G, R = cv2.split(img)
    B = B.sum()
    G = G.sum()
    R = R.sum()
    total = float(R+G+B)
    R = R/total
    G = G/total
    B = B/total

    draw_RGB(R, G, B, filename)
```

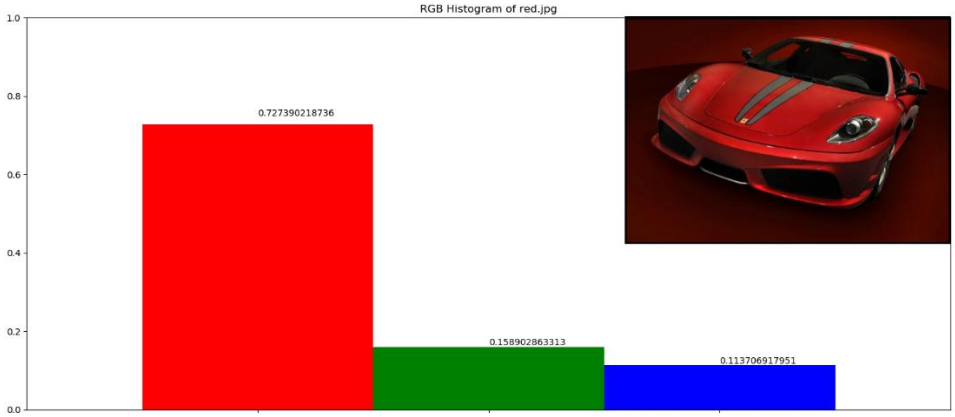
这里直接只用 cv2 中的 split 将 BGR 三个通道分开，然后用 Numpy 的 sum 方法求和，最后根据公式绘图。

此外，我了解到 cv2.split 的速度比直接索引(形如：B=img[:, :, 0])要慢,但 cv2.split 返回的是拷贝,直接索引返回的是引用，加上为了代码的简洁，我使用了 cv2.split()。

c. 结果：

绘图我使用的 matplotlib，由于代码类似，在最后统一说。



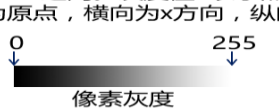


从结果看出，第一张确实绿色最多（由于色调感觉偏红，红色也很多），而第二张图中，红色占绝对优势。

2. Ex2 P1

a. 理论知识：

图像的基本元素是“像素(pixel)”。一幅宽为W，高为H的灰度图像在计算机中用一个W×H的矩阵存储。矩阵的每个元素是图像对应位置像素的灰度值，范围在0到255之间。灰度值0表示黑色，灰度值255表示白色。图像坐标系以左上角为原点，横向为x方向，纵向为y方向。



$$I_{gray}(x,y)$$
$$1 \le x \le W$$
$$1 \le y \le H$$
$$0 \le I_{gray}(x,y) \le 255$$

灰度图像I(x,y)的灰度直方图定义为各灰度值像素数目的相对比例。图像中灰度值为i的像素总个数为：

$$N(i) = \sum_{x=0}^{W-1} \sum_{y=0}^{H-1} I(x,y) == i ? 1 : 0$$

灰度直方图：

$$H(i) = \frac{N(i)}{\sum_{j=0}^{255} N(j)}, i = 0, \cdots 255$$

灰度直方图反映了图像明暗程度

b. 代码实现：

```
def Gray_hist(filename):
    img = cv2.imread(filename, cv2.IMREAD_GRAYSCALE)
    hist = cv2.calcHist([img], [0], None, [256], [0, 256]).flatten()
    total = hist.sum()
    hist = hist / total
```

我使用 cv2 中内置的 calcHist 直接得到了灰度的直方图
其使用方法如下：

cv2.calcHist(images, channels, mask, histSize, range[,hist[,accumulate]])

其中：

images:原图像（图像格式为 uint8 或 float32）。当传入函数时应该用中括号 [] 括起来，例如：[img]。

channels: 同样需要用中括号括起来，它会告诉函数我们要统计那幅图像的直方图。如果输入图像是灰度图，它的值就是 [0]；如果是彩色图像的话，传入的参数可以是 [0]， [1]， [2] 它们分别对应着通道 B， G， R。

mask: 掩模图像。要统计整幅图像的直方图就把它设为 None。但是如果你想统计图像某一部分的直方图的话，你就需要制作一个掩模图像，并使用它。

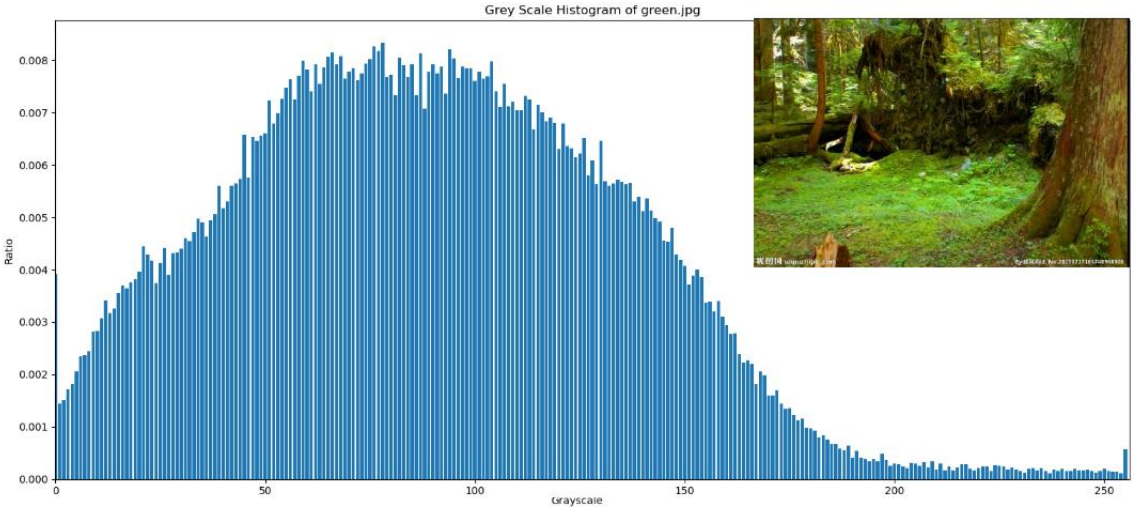
histSize:BIN 的数目。也应该用中括号括起来，例如： [256]。

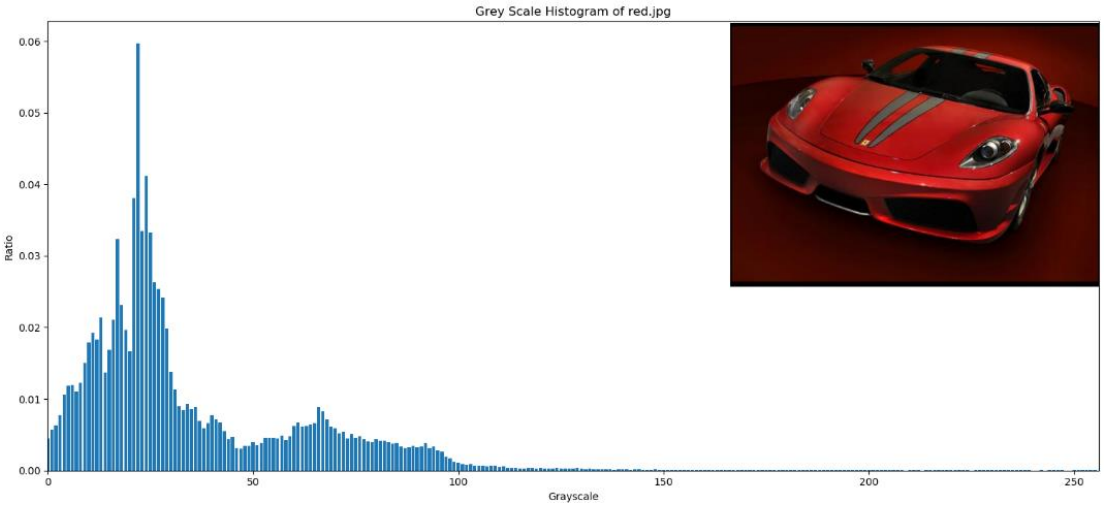
ranges: 像素值范围，通常为 [0， 256]

然后同样根据公式求和、作比，最后绘图就可以了。

c. 结果：

绘图我使用的 matplotlib，由于代码类似，在最后统一说。





可以看出，在图一中，灰度分布在中段区域，因此画面看起来较明亮；而在图二中灰度分布在较小的区域，因此画面看起来较黑暗。

3. Ex2 P2

a. 理论知识：

假设I(x, y)表示一幅灰度图像，则它X方向的梯度定义为：

$$I_x(x,y)=\frac{\partial I(x,y)}{\partial x}=I(x+1,y)-I(x-1,y)$$

Y方向的梯度定义为：

$$I_y(x,y)=\frac{\partial I(x,y)}{\partial y}=I(x,y+1)-I(x,y-1)$$

梯度强度定义为

$$M(x,y)=\sqrt{I_x(x,y)^2+I_y(x,y)^2}$$

* 图像边界处的梯度无法根据上述梯度定义求出，并且边界的梯度的定义常常随不同应用场景有改变，如有的定义边界像素的梯度为零，有的定义边界像素的梯度等于它临近的非边界像素的梯度。本实验中涉及梯度的地方均不考虑边界像素的梯度。

梯度的范围：

$$-255\le I_x, I_y\le 255$$

梯度强度的范围：

$$0\le M(x,y)\le 255\sqrt{2}\approx 360.625$$

把梯度强度均匀分成361个区间，(x, y)处的像素所在区间为：

$$B(x,y)=i,\ \text{if}\ i\le M(x,y)<i+1,\ 0\le i\le 360$$

落在第i个区间总的像素数目为：

$$N(i)=\sum_{x=1}^{W-2}\sum_{y=1}^{H-2}B(x,y)==i\,?\,1:0$$

比例为：

$$H(i)=\frac{N(i)}{\sum_{j=0}^{360}N(j)}$$

灰度图像的梯度强度分布情况，反映了图像的纹理的疏密程度。

b. 代码实现：

方法一：一开始没有在 Opencv 中查到直接的函数时，我根据公式，直接实现了一个：

```
def my_Graygrad_hist(filename):
    img = cv2.imread(filename, cv2.IMREAD_GRAYSCALE)
    height, width = img.shape

    #store the grayg gradient magnitude
    res = np.zeros(361)
    for i in range(1, height-1):
        for j in range(1, width-1):
            gx = int(img[i][j+1]) - int(img[i][j-1])
            gy = int(img[i+1][j]) - int(img[i-1][j])
            res[int(sqrt(gx*gx+gy*gy))] += 1
    total = res.sum()
    res /= total
```

方法很原始，就是遍历点，计算该点梯度强度，然后取整后在 res 中进行统计。

方法二：后来想用矩阵的知识实现，经过在思考与查询，我用矩阵算子实现来求灰度梯度：

```
def matrix_Graygrad_hist(filename):
    img = cv2.imread(filename, cv2.IMREAD_GRAYSCALE)
    height, width = img.shape
    #matrix operator
    mx = np.array([[-1, 0, 1]])
    my = np.array([[-1, 0, 1]]).T
    #get the gradient through the api filter2D
    gx = cv2.filter2D(img, cv2.CV_32F, mx)
    gy = cv2.filter2D(img, cv2.CV_32F, my)

    res = np.zeros(361)
    for i in range(1, height-1):
        for j in range(1, width-1):
            res[int(sqrt(gx[i][j]*gx[i][j]+gy[i][j]*gy[i][j]))] += 1
    total = res.sum()
    res /= total
```

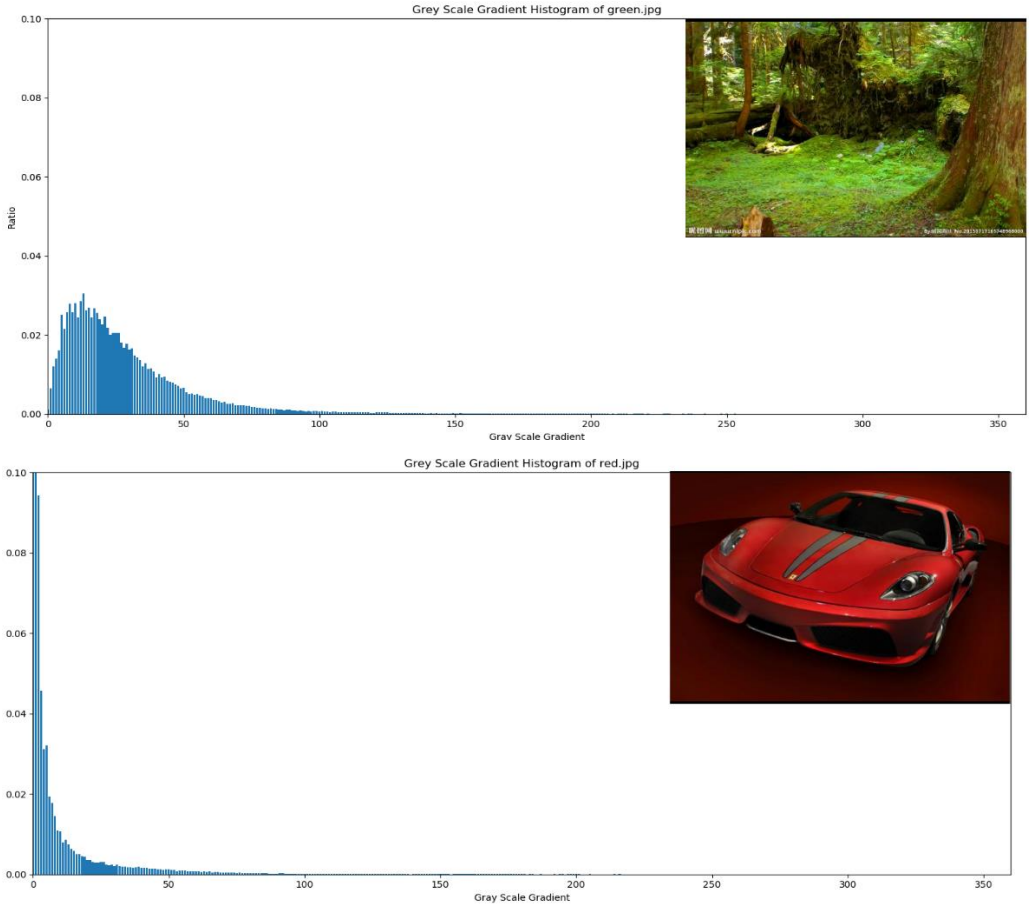
代码中的 mx，my 是矩阵算子。得出的思路很简单：观察到给出的灰度梯度公式，就是让内核加前后（上下）与算子做一个卷积运算。

得到算子后使用 cv2 中的 filter2D（内置的利用内核实现对图像的卷积运算的函数），其中：第一个参数为原图像矩阵，第二个参数是设置输出矩阵数据类型为 32 位浮点（原来的 uint8 范围为 0~255，做减法可能会溢出），第三个是矩阵算子。

这样就可以得到 gx，gy 两个梯度矩阵，然后根据公式计算出梯度强度，取整后在 res 中进行统计。

c. 结果

两种方法的得到的结果相同，但是计算一个 6.5mb 的 png 图片，第一种用时：12.9304349422s， 第二种用时：9.63080191612s，说明了矩阵运算在 opencv、numpy 是有优化的，我们在处理图像时，应该尽量从矩阵角度考虑，而不是单纯的遍历。



可以看出，图一纹理比图二复杂，其灰度梯度强度图相比图二在中间分布多。

4. 输出直方图

由于本实验需要在 Python 中输出图表，经过在网上查询，我找到了 matplotlib 这个库可以很方便的完成任务。

以第二个实验为例：绘制横坐标为灰度梯度强度，纵坐标为所占总点数百分比的柱状图，就可以得到所需的直方图。

```
# draw the histogram
x = np.linspace(0, 360, num=361)
plt.bar(x, res, align="center")
plt.title('Grey Scale Gradient Histogram of ' + filename)
plt.xlim(0, 360)
plt.ylim(0, 0.1)
plt.ylabel("Ratio")
plt.xlabel("Gray Scale Gradient")
plt.show()
```

四、问题与解决

1. 求灰度直方图效率问题

经过学习，发现有三种求灰度直方图的方法，我用一个 6.5mb 大小的 png 图片对三种方法进行了测试：

方法一：

```
hist = cv2.calcHist([img], [0], None, [256], [0, 256]).flatten()
```

这个方法上面介绍过。

用时：0.00531601905823s

方法二：

```
hist,bins = np.histogram(img.ravel(),256, [0,256])
```

使用 Numpy 内置的求直方图方法，其中 img.ravel()表示把[height, width] 的二维图片转化成一维的 [height * width,]大小。

用时：0.105583190918s

方法三：

```
hist=np.bincount(img.ravel(), minlength=256)
```

使用 Numpy 中的 bincount，该函数返回[0,minlength)的整数在参数一的矩阵中出现的次数所组成的矩阵。

用时：0.0292339324951s

可以看出，cv2 中内置的方法效率是最高的，因而我选用了方法一。

2. 矩阵的数据类型问题

在计算灰度梯度过程中，由于要对数据做减法，却发现计算出来的结果很奇怪，并有 RuntimeWarning: overflow encountered in ubyte_scalars 这个警告。

经过学习得知，imread 默认读进来的数据以 uint8 的类型储存（0-255），而作减法可能得到负数也就溢出了。

解决方法:储存时候设置其他储存类型(如 int64)，或者计算前将其转换为 int。

五、总结

在本实验中，我学会了使用 opencv 来读取图片，并学习了图像特征相关的知识，并结合 Numpy 这个强大的库，来完成提取、处理。

除此之外，我还学到了一个可以在 Python 中绘图表的库 matplotlib。

最后，衷心感谢老师和助教们的精心准备和辛勤付出，你们整理完善的 ppt 和到位的答疑大大提高了我学习图像处理知识的效率，感谢~

贾萧松 516030910548
2017.11.25