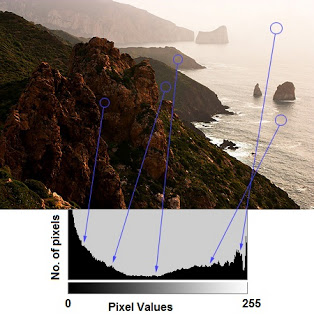
# 1.查找，绘画，分析

直方图是什么？你可以将直方图视为图形或绘图，它可以让你全面了解图像的强度分布。它是在X轴上具有像素值（范围从0到255，并非总是）的图和在Y轴上的图像中的对应像素数。

这只是理解图像的另一种方式。通过查看图像的直方图，你可以直观了解该图像的对比度，亮度，强度分布等。今天几乎所有的图像处理工具都提供了直方图的功能。以下是来自Cambridge in Color网站的图片，我建议你访问该网站了解更多详情。

[](https://camo.githubusercontent.com/a130dd7c71a8bc0bf3386e432ad3db654dc61a27902b7de4da4d083a6e89dc45/68747470733a2f2f646f63732e6f70656e63762e6f72672f342e302e302f686973746f6772616d5f73616d706c652e6a7067)

你可以看到图像及其直方图。（请记住，此直方图是为灰度图像绘制的，而不是彩色图像）。直方图的左区域显示图像中较暗像素的数量，右区域显示较亮像素的数量。从直方图中，你可以看到暗区域不仅仅是更亮的区域，中间色调的数量（中间区域的像素值，比如大约127）非常少。

## 查找直方图

现在我们知道什么是直方图，我们可以研究如何找到它。 OpenCV和Numpy都具有内置功能。在使用这些功能之前，我们需要了解与直方图相关的一些术语。

**BINS**：上面的直方图显示了每个像素值的像素数，即从0到255.即你需要256个值来显示上面的直方图。但是考虑一下，如果你不需要分别找到所有像素值的像素数，但像素值区间的像素数是多少呢？例如，你需要找到介于0到15之间，然后是16到31，......，240到255之间的像素数。你只需要16个值来表示直方图。这就是OpenCV教程中直方图中给出的示例。

所以你要做的只是将整个直方图分成16个子部分，每个子部分的值是其中所有像素数的总和。每个子部分称为“BIN”。在第一种情况下，bin的数量是256（每个像素一个），而在第二种情况下，它只有16. BINS由OpenCV docs中的术语histSize表示。

**DIMS**：这是我们收集数据的参数数量。在这种情况下，我们只收集强度值有关的数据，所以这里是1。

**RANGE**：这是要测量的强度值范围。通常，它是[0,256]，即所有强度值。

### 1. OpenCV中的直方图计算

使用cv.calcHist()函数来查找直方图。让我们熟悉一下这个函数及其参数：

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

1. images：它是uint8或float32类型的源图像。它应该用方括号表示，即“[img]”。
2. channels：它也在方括号中给出。它是我们计算直方图的通道索引。例如，如果输入是灰度图像，则其值为[0]。对于彩色图像，你可以通过[0]，[1]或[2]分别计算蓝色，绿色或红色通道的直方图。
3. mask：掩模图像。要查找完整图像的直方图，它将显示为“无”。但是，如果要查找图像特定区域的直方图，则必须为其创建蒙版图像并将其作为蒙版。 （稍后我会举一个例子。）
4. histSize：这代表我们的BIN计数。需要在方括号中给出。对于满量程，我们通过[256]。
5. ranges：范围。通常是[0,256]。

那么让我们从一个示例图像开始吧。只需以灰度模式加载图像并找到其完整的直方图。

img = cv.imread('home.jpg',0)

hist = cv.calcHist([img],[0],None,[256],[0,256])

hist是256x1数组，每个值对应于该图像中具有相应像素值的像素数。

### 2. Numpy中的直方图计算

Numpy还为你提供了一个函数，np.histogram()。 因此，你可以尝试以下行代替calcHist()函数：

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

hist与我们之前计算的相同。但是bins中有257个元素，因为Numpy中bins计算为0-0.99,1-1.99,2-2.99等。所以最终范围是255-255.99。为了表示这一点，他们还在bins中追加了256。但我们不需要256.高达255就够了。

参考： Numpy有另一个函数，np.bincount()，它比np.histogram()快得多（大约10倍）。因此，对于一维直方图，你可以更好地尝试。不要忘记在np.bincount中设置minlength = 256。例如，hist = np.bincount（img.ravel（），minlength = 256）

注意 OpenCV函数比np.histogram()快（约40倍）。所以推荐使用OpenCV功能。

## 绘制直方图

有两种方法：

* 简短方法：使用Matplotlib绘图功能
* 复杂方法：使用OpenCV绘图功能

### 1. 使用Matplotlib

Matplotlib附带直方图绘图功能：matplotlib.pyplot.hist() 它直接找到直方图并绘制它。你无需使用calcHist()或np.histogram()函数来查找直方图

### 代码

|  |
| --- |
|  |

#### 效果：

|  |
| --- |
|  |

#### 或者你可以使用matplotlib的通用画法，这对BGR图有好处。 为此，你需要首先找到直方图数据。

### 代码

|  |
| --- |
|  |

#### 注意：这个calcHist的参数多数是列表

#### 效果

|  |
| --- |
|  |

### 2.使用opencv

在这里你可以调整直方图的值及其bin值，使其看起来像x，y坐标，这样你就可以使用cv.line()或cv.polyline()函数绘制它，以生成与上面相同的图像。

### 蒙板的应用

我们使用cv.calcHist()来查找完整图像的直方图。 如果要查找图像某些区域的直方图，该怎么办？ 只需在要查找直方图的区域上创建一个白色的蒙版图像，其余为黑色，然后将其作为蒙板传递。

### 实例

|  |
| --- |
|  |

#### 效果

|  |
| --- |
|  |

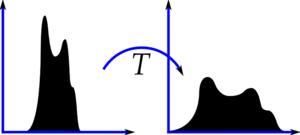
# 2.均衡直方图

## 目标

* 学习直方图均衡化的概念，并利用它来提高图像对比度

## 理论

考虑一个像素值仅限于某些特定值范围的图像。 例如，较亮的图像将所有像素限制为高值。 但是，良好的图像将具有来自图像的所有区域的像素。 所以你需要将这个直方图拉伸到两端（如下图所示，来自维基百科），这就是直方图均衡所做的，通常可以改善图像的对比度。

[](https://camo.githubusercontent.com/1d60080af5fafea318e8f472b12d92d74dd5b3ee3e0f74f6c360185396c5b230/68747470733a2f2f646f63732e6f70656e63762e6f72672f342e302e302f686973746f6772616d5f657175616c697a6174696f6e2e706e67)

建议阅读直方图均衡的维基百科页面，了解更多相关细节。 它有一个非常好的解释和解决的例子，所以在阅读之后几乎可以理解所有内容。 在这里有它的Numpy实现，之后还有OpenCV函数实现。

## 代码

|  |
| --- |
|  |

### 效果

|  |
| --- |
|  |

你可以看到直方图位于更亮的区域。 我们需要全谱，为此我们需要一个转换函数，它将较亮区域中的输入像素映射到整个区域中的输出像素。 这就是直方图均衡所做的。

## numpy直方图均衡化操作

现在我们找到最小直方图值（不包括0）并应用维基页面中给出的直方图均衡化方程。 但我在这里使用了Numpy的蒙板数组。 对于蒙板数组，所有操作都在非蒙板元素上执行。

cdf\_m = np.ma.masked\_equal(cdf,0)

cdf\_m = (cdf\_m - cdf\_m.min())\*255/(cdf\_m.max()-cdf\_m.min())

cdf = np.ma.filled(cdf\_m,0).astype('uint8')

现在我们有了查找表，它提供了关于每个输入像素值的输出像素值的信息。我们只要应用这个变换

img2 = cdf[img]

## 转换的完整代码

|  |
| --- |
|  |

### 我们把转换后的图片保存了下来。名字叫做wiki\_enhanced.png

## 然后我们来绘制新图片的直方图

## 代码如下

|  |
| --- |
|  |

### 效果：

|  |
| --- |
|  |

另一个重要特征是，即使图像是较暗的图像（而不是我们使用的更亮的图像），在均衡后我们将得到几乎与我们相同的图像。 结果，这被用作“参考工具”以使所有图像具有相同的照明条件。 这在许多情况下很有用。 例如，在面部识别中，在训练面部数据之前，将面部图像均衡化以使它们全部具有相同的照明条件。

## OpenCV中的直方图均衡

OpenCV有一个函数来执行此操作，cv.equalizeHist()。 它的输入只是灰度图像，输出是我们的直方图均衡图像。

## 实例：下面的例子可以把原图和均衡化后的图片拼接在一起生成一幅对比图片显示并且保持起来

|  |
| --- |
|  |

### 效果：

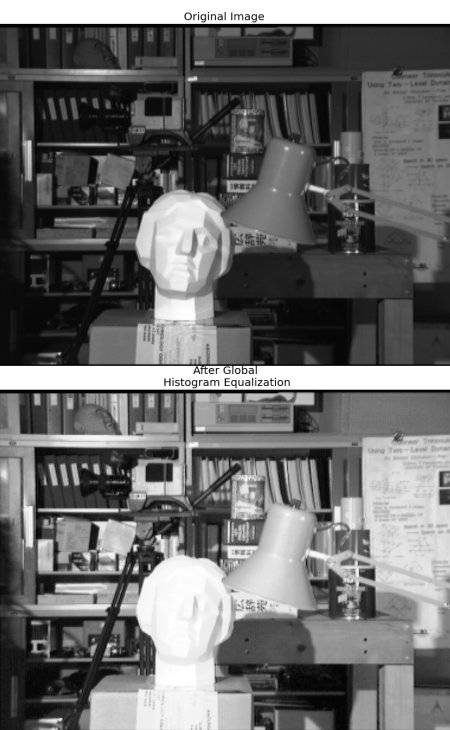
|  |
| --- |
|  |

### 同时在当前文件夹下面生成一幅result.png图片

|  |
| --- |
|  |

## CLAHE（Contrast Limited Adaptive Histogram Equalization）有限对比度自适应直方图均衡化

我们刚看到的第一个直方图均衡，考虑了图像的全局对比度。 在许多情况下，这不是一个好主意。 例如，下图显示了全局直方图均衡后的输入图像及其结果。

[](https://camo.githubusercontent.com/00abcd7caba717a88839efbeb419d347ccc09d2d29ea6135f014558e6701e0ba/68747470733a2f2f646f63732e6f70656e63762e6f72672f342e302e302f636c6168655f312e6a7067)

确实，直方图均衡后背景对比度有所改善。但比较两个图像中的雕像的脸。由于亮度过高，我们丢失了大部分信息。这是因为它的直方图并不局限于特定区域，正如我们在之前的案例中看到的那样（尝试绘制输入图像的直方图，你将获得更多的直觉）。

因此，为了解决这个问题，使用自适应直方图均衡。在此，图像被分成称为“tiles”的小块（在OpenCV中，tileSize默认为8x8）。然后这些块中的每一块都像往常一样进行直方图均衡化。所以在一个小的区域，直方图会局限在一个小的区域(除非有噪声)。如果有噪音，它会被放大。为避免这种情况，应用对比度限制。如果任何直方图区间高于指定的对比度限制（在OpenCV中默认为40），则在应用直方图均衡之前，将这些像素剪切并均匀分布到其他区间。均衡后，为了去除图块边框中的瑕疵，应用双线性插值。

## CLAHE实例

|  |
| --- |
|  |

### 效果

|  |
| --- |
|  |

# 3.2D直方图

## 目标

* 学习寻找和绘制二维直方图

## 介绍

我们学习计算并绘制了一维直方图。 它之所以被称为一维，是因为我们只考虑一个特征，即像素的灰度强度值。 但在二维直方图中，需要考虑两个特征。 通常，它用于查找颜色直方图，其中两个特征是每个像素的色调值和饱和度值。

有一个python样本（samples / python / color\_histogram.py）已经用于查找颜色直方图。 我们将尝试了解如何创建这样的颜色直方图，它将有助于理解直方图反投影等其他主题。

## OpenCV中的2D直方图

它很简单，使用相同的函数cv.calcHist()计算。 对于颜色直方图，我们需要将图像从BGR转换为HSV。（记住，对于1D直方图，我们从BGR转换为灰度）。 对于2D直方图，其参数将修改如下：

* channels = [0,1]因为我们需要处理H和S平面。
* b = H平面为[180,256] 180，S平面为256。
* range = [0,180,0,256] Hue值介于0和180之间，饱和度介于0和256之间。

## 实例

|  |
| --- |
|  |

### 效果

|  |
| --- |
|  |

## Numpy中的2D直方图

Numpy还为此提供了一个特定的功能：np.histogram2d()。（请记住，对于1D直方图，我们使用np.histogram()）。

## numpy实例

|  |
| --- |
|  |

### 效果：

|  |
| --- |
|  |

### 在直方图中，你可以看到H = 100和S = 200附近的一些高值。它对应于天空的蓝色。 类似地，在H = 25和S = 100附近可以看到另一个峰值。它对应于宫殿的黄色。 你可以使用任何图像编辑工具（如GIMP）对其进行验证。

## 2D直方图实例2

|  |
| --- |
|  |

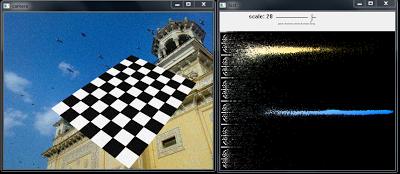
### 效果：

|  |
| --- |
|  |

## OpenCV样本风格!!

在OpenCV-Python2样本中有一个颜色直方图的示例代码（samples / python / color\_histogram.py）。 如果运行代码，则可以看到直方图也显示相应的颜色。 或者只是输出颜色编码的直方图。 它的结果非常好（虽然你需要添加额外的一堆线）。

在这段代码中，作者用HSV创建了一个颜色映射。然后把它转换成BGR。得到的直方图图像与此颜色映射相乘。他还使用一些预处理步骤来删除小的孤立像素，从而得到一个良好的直方图。下面是与上面的图像相同的代码的输出

[](https://camo.githubusercontent.com/7cc79ff1612dc2a18263b9488a7a060e752beddb6ee376d42c68da3a04ee6d3a/68747470733a2f2f646f63732e6f70656e63762e6f72672f342e302e302f3264686973745f6f70656e63762e6a7067)

你可以清楚地看到直方图中有哪些颜色，蓝色，黄色，还有一些白色是由于棋盘的缘故

## 为了方便学习，我把它和它的依赖文件拷贝到笔记代码目录下面，把主文件改名color\_hist.py

|  |
| --- |
|  |

# 4. 反投影直方图

## 目标：

* 学习直方图的反向投影

## 理论

它由Michael J. Swain，Dana H. Ballard在他们的论文“Indexing via color histograms”中提出。

用简单的话来说，它到底是什么?它用于图像分割或查找图像中感兴趣的对象。简单地说，它创建了与输入图像大小相同(但是是单一通道)的图像，其中每个像素对应于该像素属于对象的概率。在更简单的世界中，输出的图像将使我们感兴趣的对象与其余部分相比有更多的白色。这是一个直观的解释。(我说得再简单不过了)。直方图反投影与camshift算法等相结合。我们怎么做呢?我们创建一个an的直方图

我们怎么做呢?我们创建一个图像的直方图，其中包含我们感兴趣的对象(在我们的例子中是地面、离开玩家和其他东西)。对象应该尽可能地填充图像，以获得更好的结果。与灰度直方图相比，颜色直方图更受欢迎，因为与灰度强度相比，对象的颜色是一种更好的定义对象的方法。然后我们将这个直方图“向后投射”到我们需要找到目标的测试图像上，换句话说，我们计算出属于地面的每个像素的概率并显示出来。在适当的阈值上得到的输出只给我们提供了基础

## Numpy中的算法

1. 首先，我们需要计算需要找到的对象(设为“M”)和要搜索的图像(设为“I”)的颜色直方图。

import numpy as np

import cv2 as cvfrom matplotlib import pyplot as plt

#roi is the object or region of object we need to find

roi = cv.imread('rose\_red.png')

hsv = cv.cvtColor(roi,cv.COLOR\_BGR2HSV)

#target is the image we search in

target = cv.imread('rose.png')

hsvt = cv.cvtColor(target,cv.COLOR\_BGR2HSV)

# Find the histograms using calcHist. Can be done with np.histogram2d also

M = cv.calcHist([hsv],[0, 1], None, [180, 256], [0, 180, 0, 256] )

I = cv.calcHist([hsvt],[0, 1], None, [180, 256], [0, 180, 0, 256] )

1. 求出比率 $$ R=\frac{M}{I} $$ 。然后反投影R，使用R作为调色板并创建一个新图像，每个像素作为其对应的目标概率。 即B（x，y）= R[h（x，y），s（x，y）]其中h是色调，s是（x，y）处像素的饱和度。 之后应用条件B（x，y）= min [B（x，y），1]。

h,s,v = cv.split(hsvt)

B = R[h.ravel(),s.ravel()]

B = np.minimum(B,1)

B = B.reshape(hsvt.shape[:2])

1. 现在对圆盘进行卷积，B = D \* B，其中D是盘卷积核。

disc = cv.getStructuringElement(cv.MORPH\_ELLIPSE,(5,5))

cv.filter2D(B,-1,disc,B)

B = np.uint8(B)

cv.normalize(B,B,0,255,cv.NORM\_MINMAX)

1. 现在最大强度的位置为我们提供了物体的位置。 如果我们期望图像中有一个区域，那么对适当值进行阈值处理会得到很好的结果。

ret,thresh = cv.threshold(B,50,255,0)

## OpenCV中的反投影

OpenCV提供了一个内置函数cv.calcBackProject()。 它的参数与cv.calcHist()函数几乎相同。 它的一个参数是直方图，它是对象的直方图，我们必须找到它。 此外，在传递给backproject函数之前，应该对象直方图进行规范化。 它返回概率图像。 然后我们将图像与光盘卷积核卷积并应用阈值。

### 实例1

|  |
| --- |
|  |

### 效果

|  |
| --- |
|  |

### 实例2

|  |
| --- |
|  |
|  |

### 效果：

|  |
| --- |
|  |

### 如果把图片改为下面这两个，

|  |  |
| --- | --- |
|  |  |

### 并且把拼接方式改为hstack，效果如下

|  |
| --- |
|  |