# 1.改变颜色空间

## 学习目标

|  |
| --- |
| * 学习将图像从一个颜色空间转换为另一个颜色空间，例如BGR↔Gray，BGR↔HSV * 创建一个提取视频中某个特定彩色对象的应用程序 * 学习以下函数：cv.cvtColor()，cv.inRange() |

## 改变色彩空间

**OpenCV中有150多种颜色空间转换方法。我们只研究两种最广泛使用的转换方法，BGR↔Gray和BGR↔HSV。 对于颜色转换，使用函数cv.cvtColor(input\_image，flag)，其中flag确定转换类型。 对于BGR→Gray转换，我们使用标志cv.COLOR\_BGR2GRAY。类似地，对于BGR→HSV，我们使用标志cv.COLOR\_BGR2HSV。要获取其他标志，只需 执行下面这个小脚本**

|  |
| --- |
| **import cv2**  **flags = [i for i in dir(cv2) if i.startswith('COLOR\_')]**  **print( flags )** |

**注意：对于HSV色彩空间，Hue(色调)的取值范围是[0,179]，Saturation(饱和度)的取值范围是[0,255]，Value(明度)的取值范围是[0,255]。不同的软件可能使用不同的取值方式，因此，如果要将OpenCV的HSV值与其他软件的HSV值进行比较时，则需要对这些范围进行标准化**。

## 对象提取

现在我们知道如何将BGR图像转换为HSV，我们可以使用HSV色彩空间来提取彩色对象。在HSV中表示颜色比在BGR颜色空间中更容易。在我们的程序中，我们将尝试提取视频画面中的蓝色对象。下面是方法程序执行步骤：

* 获取视频中的每一帧
* 从BGR转换为HSV颜色空间
* 我们为HSV图像设定一系列的蓝色阈值
* 单独提取蓝色对象并显示，之后我们便可以对我们想要的图像做任何事情。

以下是详细评论的代码：

|  |
| --- |
| **import cv2 as cv**  **import numpy as np**  **cap = cv.VideoCapture(0)**  **while(1):**    **# Take each frame**  **\_, frame = cap.read()**    **# Convert BGR to HSV**  **hsv = cv.cvtColor(frame, cv.COLOR\_BGR2HSV)**    **# define range of blue color in HSV**  **lower\_blue = np.array([110,50,50])**  **upper\_blue = np.array([130,255,255])**    **# Threshold the HSV image to get only blue colors**  **mask = cv.inRange(hsv, lower\_blue, upper\_blue)**    **# Bitwise-AND mask and original image**  **res = cv.bitwise\_and(frame,frame, mask= mask)**    **cv.imshow('frame',frame)**  **cv.imshow('mask',mask)**  **cv.imshow('res',res)**  **k = cv.waitKey(5) & 0xFF**  **if k == 27:**  **break**    **cv.destroyAllWindows()**  下面的图片展示了我们提取蓝色对象后的效果：  [image1](https://camo.githubusercontent.com/07dcac94d104e2b34396a743b425487f9b797049895f992a2fc41d45a9f515f7/68747470733a2f2f646f63732e6f70656e63762e6f72672f342e302e302f6672616d652e6a7067) |

### 注意：图像中有一些噪音，我们将在后面的章节中看到如何删除它们。这是对象提取中最简单的方法。一旦你学习了轮廓的功能，你就可以做很多事情，比如找到这个物体的重心并用它来追踪物体，只需在镜头前移动你的手以及许多其他有趣的东西来绘图。如果没有摄像头，用图片来做也是可以的

|  |
| --- |
|  |

## 如何去查找确定HSV值

这是我们在stackoverflow.com中常见的问题。其实解决这个问题非常简单，你可以使用相同的函数cv.cvtColor()。你只需传递所需的BGR值，而不是传递图像。例如，要查找绿色的HSV值，在Python终端中输入以下命令：

>>> green = np.uint8([[[0,255,0 ]]])

>>> hsv\_green = cv.cvtColor(green,cv.COLOR\_BGR2HSV)

>>> print( hsv\_green )

[[[ 60 255 255]]]

现在分别将[H-10,100,100]和[H+10,255,255]作为下限和上限。除了这种方法，你可以使用任何图像编辑工具如GIMP，或任何在线转换器来查找这些值，但不要忘记调整HSV范围。

# 2. 图像的几何变换

## 目标

* **将不同的几何变换应用于图像，如平移，旋转，仿射变换**
* **学习函数：cv.getPerspectiveTransform**

## 转换

OpenCV提供了两个转换函数cv.warpAffine和cv.warpPerspective，你可以使用它们进行各种转换。cv.warpAffine采用2x3变换矩阵作为参数输入，而cv.warpPerspective采用3x3变换矩阵作为参数输入。

## 缩放

缩放只是调整图像大小，OpenCV有一个函数cv.resize()，可以手动指定图像的大小以及缩放系数，可以使用不同的插值方法，常用的插值方法是用于缩小的cv.INTER\_AREA和用于缩放的cv.INTER\_CUBIC（慢）和cv.INTER\_LINEAR。默认情况下，使用的插值方法是cv.INTER\_LINEAR，它用于所有调整大小的操作。你可以使用以下方法之一调整输入图像的大小：

缩放只是调整图像大小，OpenCV有一个函数cv.resize()，可以手动指定图像的大小以及缩放系数，可以使用不同的插值方法，常用的插值方法是用于缩小的cv.INTER\_AREA和用于缩放的cv.INTER\_CUBIC（慢）和cv.INTER\_LINEAR。默认情况下，使用的插值方法是cv.INTER\_LINEAR，它用于所有调整大小的操作。你可以使用以下方法之一调整输入图像的大小：

**import numpy as np**

**import cv2 as cv**

**img = cv.imread('messi5.jpg')**

**res = cv.resize(img,None,fx=2, fy=2, interpolation = cv.INTER\_CUBIC)**

**#OR**

**height, width = img.shape[:2]**

**res = cv.resize(img,(2\*width, 2\*height), interpolation = cv.INTER\_CUBIC)**

## 缩放的完整例子

|  |
| --- |
|  |

### 效果：

|  |
| --- |
|  |

## 平移

平移是对象位置的移动。如果你知道像素点(x，y)要位移的距离，让它为变为($$ t\_x,t\_y $$)，你可以创建变换矩阵**M**，如下所示：

𝑀=[10𝑡𝑥01𝑡𝑦]

可以将其设置为np.float32类型的Numpy数组，并将其传递给cv.warpAffine()函数。下面的示例演示图像像素点整体进行(100,50)位移：

|  |
| --- |
|  |

### 效果

|  |
| --- |
|  |

**注意：cv.warpAffine()函数的第三个参数是输出图像的大小，它应该是(宽，高)的形式, width=列数，height=行数**

## 旋转

通过改变图像矩阵实现图像旋转角度θ

|  |
| --- |
|  |

OpenCV提供了可调旋转中心的缩放旋转，这样你可以在你喜欢的任何位置进行旋转。修正的变换矩阵由下式给出：

|  |
| --- |
|  |

其中：

𝛼=𝑠𝑐𝑎𝑙𝑒⋅𝑐𝑜𝑠Θ

𝛽=𝑠𝑐𝑎𝑙𝑒⋅𝑠𝑖𝑛Θ

为了找到这个转换矩阵，OpenCV提供了一个函数cv.getRotationMatrix2D。以下示例将图像相对于中心旋转90度而不进行任何缩放。

## 图像旋转的完整例子

|  |
| --- |
|  |

### 效果：

|  |
| --- |
|  |

## 仿射变换

在仿射变换中，原始图像中的所有平行线在输出图像中依旧平行。为了找到变换矩阵，我们需要从输入图像中得到三个点，以及它们在输出图像中的对应位置。然后cv.getAffineTransform将创建一个2x3矩阵，最后该矩阵将传递给cv.warpAffine。

参考以下示例，并查看选择的点（以绿色标记）：

## 仿射变换例子

|  |
| --- |
|  |

### 效果：

|  |
| --- |
|  |

## 透视变换

对于透视变换，需要一个3x3变换矩阵。即使在转换之后，直线仍是直线。要找到此变换矩阵，需要在输入图像上找4个点，以及它们在输出图像中的对应位置。在这4个点中，其中任意3个不共线。然后可以通过函数cv.getPerspectiveTransform找到变换矩阵，将cv.warpPerspective应用于此3x3变换矩阵。

## 透视变换的实例

|  |
| --- |
|  |

### 效果：

|  |
| --- |
|  |

### 注意：图片大小不一样，效果不一样

# 3.图像阈值

## 目标：

* 学习简单的阈值处理，自适应阈值处理，Otsu's的阈值处理等
* 学习函数：cv.threshold，cv.adaptiveThreshold等

## 简单阈值处理

这种阈值处理的方法是简单易懂的。如果像素值大于阈值，则为其分配一个值（可以是白色），否则为其分配另一个值（可以是黑色）。使用的函数是cv.threshold。函数第一个参数是源图像，它应该是灰度图像。第二个参数是用于对像素值进行分类的阈值。第三个参数是maxVal，它表示如果像素值大于（有时小于）阈值则要给出的值。OpenCV提供不同类型的阈值，由函数的第四个参数决定。不同的类型有：

* cv.THRESH\_BINARY
* cv.THRESH\_BINARY\_INV
* cv.THRESH\_TRUNC
* cv.THRESH\_TOZERO
* cv.THRESH\_TOZERO\_INV

文档清楚地解释了每种类型的含义。

函数将获得两个输出。第一个是retavl，将在后面解释它的作用。第二个输出是我们的阈值图像。

参考以下代码：

|  |
| --- |
|  |

### 效果：

|  |
| --- |
|  |

## 自适应阈值处理

在上面，我们使用全局值作为阈值，但在图像在不同区域具有不同照明条件的所有条件下可能并不好。在那种情况下，我们进行自适应阈值处理，算法计算图像的小区域的阈值，所以我们对同一幅图像的不同区域给出不同的阈值，这给我们在不同光照下的图像提供了更好的结果。

这种阈值处理方法有三个指定输入参数和一个输出参数。

**Adaptive Method** - 自适应方法，决定如何计算阈值。

* cv.ADAPTIVE\_THRESH\_MEAN\_C：阈值是邻域的平均值。
* cv.ADAPTIVE\_THRESH\_GAUSSIAN\_C：阈值是邻域值的加权和，其中权重是高斯窗口。

**Block Size** - 邻域大小，它决定了阈值区域的大小。

**C** - 它只是从计算的平均值或加权平均值中减去的常数。

下面的代码比较了具有不同照明的图像的全局阈值处理和自适应阈值处理：

|  |
| --- |
|  |

### 效果

|  |
| --- |
|  |

## Otsu's 二值化

### 原理

OTSU算法又称为最大类间方差法。这是我们很常见的一种求阈值的方法，它的思路就是通过统计整个图像的直方图特性来实现全局阈值T的自动选取。  
该算法进行前先设一个i作为分类阈值，通过归一化的直方图，统计图像中各个灰度级的像素占整幅图像的比例w0，并统计前景像素和背景像素的平均灰度u0和u1，和背景像素所占比例w1；通过公式g = w0w1(u0-u1) (u0-u1)计算前景像素和背景像素的方差 ，每进行完一次循环i值加一，直道i=256时停止循环，并将最大的g值作为图像阈值。

#### **缺点**：对图像噪声敏感；只能针对单一目标分割；当目标和背景大小比例悬殊、类间方差函数可能呈现双峰或者多峰，这个时候效果不好

在第一节中，我只告诉你另一个参数是retVal，但没告诉你它的作用。其实，它是用来进行Otsu's二值化。

在全局阈值处理中，我们使用任意值作为阈值，那么，我们如何知道我们选择的值是好还是不好？答案是，试错法。但如果是双峰图像（简单来说，双峰图像是直方图有两个峰值的图像）我们可以将这些峰值中间的值近似作为阈值，这就是Otsu二值化的作用。简单来说，它会根据双峰图像的图像直方图自动计算阈值。（对于非双峰图像，二值化不准确。）

为此，使用了我们的cv.threshold()函数，但是需要多传递一个参数cv.THRESH\_OTSU。这时要吧阈值设为零。然后算法找到最佳阈值并返回第二个输出retVal。如果未使用Otsu二值化，则retVal与你设定的阈值相同。

### 示例。

输入图像是嘈杂的图像。在第一种情况下，我将全局阈值应用为值127。在第二种情况下，我直接应用了Otsu的二值化。在第三种情况下，我使用5x5高斯卷积核过滤图像以消除噪声，然后应用Otsu阈值处理。来看看噪声过滤如何改善结果。

|  |  |
| --- | --- |
|  | **import cv2 import numpy as np from matplotlib import pyplot as plt """  在第一节中，我只告诉你另一个参数是retVal，但没告诉你它的作用。其实，它是用来进行Otsu's二值化。  在全局阈值处理中，我们使用任意值作为阈值，那么，我们如何知道我们选择的值是好还是不好？答案是，试错法。 但如果是双峰图像（简单来说，双峰图像是直方图有两个峰值的图像）我们可以将这些峰值中间的值近似作为阈值，这就是Otsu二值化的作用。简单来说， 它会根据双峰图像的图像直方图自动计算阈值。（对于非双峰图像，二值化不准确。）  为此，使用了我们的cv.threshold()函数，但是需要多传递一个参数cv.THRESH\_OTSU。这时要吧阈值设为零。然后算法找到最佳阈值并返回第二个输出retVal。 如果未使用Otsu二值化，则retVal与你设定的阈值相同。  请查看以下示例。输入图像是嘈杂的图像。在第一种情况下，我将全局阈值应用为值127。在第二种情况下，我直接应用了Otsu的二值化。在第三种情况下，我使用5x5 高斯卷积核过滤图像以消除噪声，然后应用Otsu阈值处理。来看看噪声过滤如何改善结果。 """ img = cv2.imread('../mydata/noisy2.png', 0) *# 填写图片名称* ret,th1 = cv2.threshold(img,127,255,cv2.THRESH\_BINARY) ret,th2 = cv2.threshold(img,0,255,cv2.THRESH\_BINARY+cv2.THRESH\_OTSU) blur = cv2.GaussianBlur(img,(5,5),0) *#高斯模糊处理* ret,th3 = cv2.threshold(blur,0,255,cv2.THRESH\_BINARY+cv2.THRESH\_OTSU) *# 使用cv2.ADAPTIVE\_THRESH\_GAUSSIAN\_C处理的效果很好  # plot all the images and their histograms* images = [img, 0, th1,  img, 0, th2,  blur, 0, th3]  titles = ['Original Noisy Image','Histogram','Global Thresholding (v=127)',  'Original Noisy Image','Histogram',"Otsu's Thresholding",  'Gaussian filtered Image','Histogram',"Otsu's Thresholding"]  for i in range(3):  plt.subplot(3,3,i\*3+1),plt.imshow(images[i\*3],'gray')  plt.title(titles[i\*3]), plt.xticks([]), plt.yticks([])  plt.subplot(3,3,i\*3+2),plt.hist(images[i\*3].ravel(),256)  plt.title(titles[i\*3+1]), plt.xticks([]), plt.yticks([])  plt.subplot(3,3,i\*3+3),plt.imshow(images[i\*3+2],'gray')  plt.title(titles[i\*3+2]), plt.xticks([]), plt.yticks([])  plt.show() cv2.waitKey(0) cv2.destroyAllWindows()** |

### 效果：

|  |
| --- |
|  |

## ostu算法 python实现

import numpy as np

import cv2 as cv

#将图片转为灰度图

# https://blog.csdn.net/weixin\_43635647/article/details/99625105

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

cv.imshow("img", img)

cv.waitKey()

def OTSU(img\_gray, GrayScale):

assert img\_gray.ndim == 2, "must input a gary\_img" #shape有几个数字, ndim就是多少

img\_gray = np.array(img\_gray).ravel().astype(np.uint8)

u1=0.0#背景像素的平均灰度值

u2=0.0#前景像素的平均灰度值

th=0.0

#总的像素数目

PixSum=img\_gray.size

#各个灰度值的像素数目

PixCount=np.zeros(GrayScale)

#各灰度值所占总像素数的比例

PixRate=np.zeros(GrayScale)

#统计各个灰度值的像素个数

for i in range(PixSum):

#默认灰度图像的像素值范围为GrayScale

Pixvalue=img\_gray[i]

PixCount[Pixvalue]=PixCount[Pixvalue]+1

#确定各个灰度值对应的像素点的个数在所有的像素点中的比例。

for j in range(GrayScale):

PixRate[j]=PixCount[j]\*1.0/PixSum

Max\_var = 0

#确定最大类间方差对应的阈值

for i in range(1,GrayScale):#从1开始是为了避免w1为0.

u1\_tem=0.0

u2\_tem=0.0

#背景像素的比列

w1=np.sum(PixRate[:i])

#前景像素的比例

w2=1.0-w1

if w1==0 or w2==0:

pass

else:#背景像素的平均灰度值

for m in range(i):

u1\_tem=u1\_tem+PixRate[m]\*m

u1 = u1\_tem \* 1.0 / w1

#前景像素的平均灰度值

for n in range(i,GrayScale):

u2\_tem = u2\_tem + PixRate[n]\*n

u2 = u2\_tem / w2

#print(u1)

#类间方差公式：G=w1\*w2\*(u1-u2)\*\*2

tem\_var=w1\*w2\*np.power((u1-u2),2)

#print(tem\_var)

#判断当前类间方差是否为最大值。

if Max\_var<tem\_var:

Max\_var=tem\_var#深拷贝，Max\_var与tem\_var占用不同的内存空间。

th=i

return th

th = OTSU(img,256)

print("使用numpy的方法：" + str(th)) # 结果为 135

**直接使用 opencv的函数**

import numpy as np

import cv2 as cv

#该函数返回的第一个值就是输入的thresh值，第二个就是处理后的图像

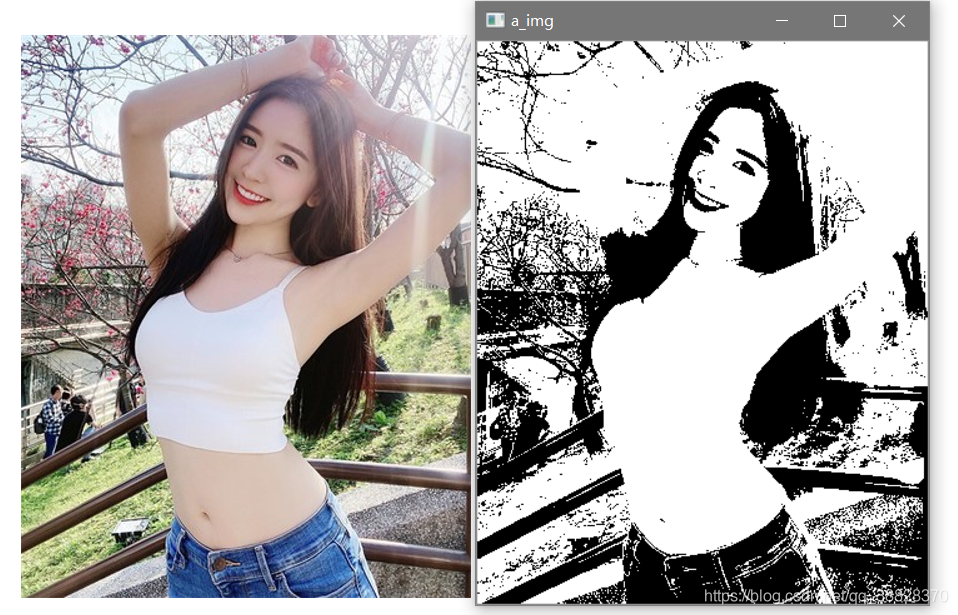
img = cv.imread('test.jpg',0)

retVal, a\_img = cv.threshold(img, 0, 255, cv.THRESH\_OTSU)

print("使用opencv函数的方法：" + str(retVal)) # 结果为 134

cv.imshow("a\_img",a\_img)

cv.waitKey()



# 4.图像滤波

## 目标：

* 使用各种低通滤波器模糊图像
* 将自定义滤波器应用于图像（2D卷积）

## 2D卷积（图像过滤）

与一维信号一样，图像也可以使用各种低通滤波器（LPF），高通滤波器（HPF）等进行滤波。LPF有助于消除噪声，模糊图像等。HPF滤波器有助于找到图片的边缘。

OpenCV提供了一个函数cv.filter2D()来将卷积核与图像进行卷积。例如，我们将尝试对图像进行平均滤波。下面是一个5x5平均滤波器的核：

|  |
| --- |
|  |

**操作步骤如下：将此核放在一个像素A上，求与核对应的图像上 25（5x5）个像素的和，取其平均值并用新的平均值替换像素A的值。重复以上操作直到将图像的每一个像素值都更新一遍。**

## 实例

|  |
| --- |
|  |

### 效果

|  |
| --- |
|  |

## 图像模糊（图像平滑）

通过将图像与低通滤波器卷积核卷积来实现平滑图像。它有助于消除噪音，从图像中去除了高频内容（例如：噪声，边缘）。因此在此操作中边缘会模糊一点。（有的平滑技术也不会平滑边缘）。OpenCV主要提供四种平滑技术。

### 1. 均值滤波

这是由一个归一化卷积框完成的。它取卷积核区域下所有像素的平均值并替换中心元素。这是由函数cv.blur()或cv.boxFilter()完成的。查看文档以获取有关卷积核的更多详细信息。我们应该指定卷积核的宽度和高度，3x3标准化的盒式过滤器如下所示：

|  |
| --- |
|  |

### 注意：如果不想使用规范化的框过滤器，请使用cv.boxFilter()。将参数normalize = False传递给函数

### blur实例

|  |
| --- |
|  |

### 效果：

|  |
| --- |
|  |

### boxFilter实例，有2种情况，normalize=True和normalize=False

|  |
| --- |
|  |

### 效果

|  |
| --- |
|  |

### 2. 高斯滤波

下面把卷积核换成高斯核。它是通过函数cv.GaussianBlur()完成的。我们应该指定卷积核的宽度和高度，它应该是正数并且是奇数。我们还应该分别指定X和Y方向的标准偏差sigmaX和sigmaY。如果仅指定了sigmaX，则sigmaY与sigmaX相同。如果两者都为零，则根据卷积核大小计算它们。高斯模糊在从图像中去除高斯噪声方面非常有效。

如果需要，可以使用函数cv.getGaussianKernel()创建高斯卷积核。

上面的代码可以修改为高斯模糊：

|  |
| --- |
|  |

效果：

|  |
| --- |
|  |

### 3. 中值滤波

顾名思义，函数cv.medianBlur()取卷积核区域下所有像素的中值，并用该中值替换中心元素。这对去除图像中的椒盐噪声非常有效。有趣的是，在上述滤波器中，中心元素是新计算的值，其可以是图像中的像素值或新值。但在中值模糊中，中心元素总是被图像中的某个像素值替换,它有效地降低了噪音。其卷积核大小应为正整数。

在这个演示中，我为原始图像添加了50％的噪点并应用了中值模糊。

### 实例：

|  |
| --- |
|  |

### 效果：

|  |
| --- |
|  |

### 我们也可以用高斯滤波来做同一幅图片和medianBlur对比一下

|  |
| --- |
|  |

### 效果：

|  |
| --- |
|  |

#### 效果没有多大变化

### 4. 双边过滤

cv.bilateralFilter()在降低噪音方面非常有效，同时保持边缘清晰。但与其他过滤器相比，操作速度较慢。我们已经看到高斯滤波器采用像素周围的邻域并找到其高斯加权平均值。该高斯滤波器仅是空间的函数，即在滤波时考虑附近的像素。它没有考虑像素是否具有几乎相同的强度。它不考虑像素是否是边缘像素。所以它也模糊了边缘，我们不想这样做。

双边滤波器在空间中也采用高斯滤波器，但是还有一个高斯滤波器是像素差的函数。空间的高斯函数确保仅考虑附近的像素用于模糊，而强度差的高斯函数确保仅考虑具有与中心像素相似的强度的像素用于模糊。因此它保留了边缘，因为边缘处的像素将具有较大的强度变化。

### 下面的示例显示使用双边过滤器

|  |
| --- |
|  |

#### 效果：

|  |
| --- |
|  |