

### 3 图像的基本操作：读取、显示和保存

#### 图像的基本操作：读取、显示和保存

欢迎来到OpenCV入门与基础课程的第三节，在这一节中，我们将深入探讨图像的基本操作，包括如何使用OpenCV读取、显示以及保存图像。掌握这些操作是进行更复杂图像处理任务的基础。

#### 图像读取

在OpenCV中，读取图像非常简单，使用 `cv2.imread()` 函数就可以实现。这个函数的第一个参数是图像的路径，第二个参数是读取图像的方式。OpenCV提供了三种读取模式：

- `cv2.IMREAD_COLOR`：读取彩色图像，默认模式，值为1。
- `cv2.IMREAD_GRAYSCALE`：以灰度模式读取图像，值为0。
- `cv2.IMREAD_UNCHANGED`：包括图像的alpha通道，值为-1。

```
import cv2

# 以彩色模式读取图像
image_color = cv2.imread('path_to_image.jpg', cv2.IMREAD_COLOR)

# 以灰度模式读取图像
image_gray = cv2.imread('path_to_image.jpg', cv2.IMREAD_GRAYSCALE)
```

#### 图像显示

读取图像后，我们通常希望查看图像内容。OpenCV提供了 `cv2.imshow()` 函数用于显示图像。此函数的第一个参数是窗口名称，第二个参数是图像变量。

```
cv2.imshow('Image Window', image_color)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

这里，`cv2.waitKey()` 是一个键盘绑定函数，参数是以毫秒为单位的时间。该函数等待任何键盘事件指定的毫秒数。如果你设置为0，它将无限期地等待一个键击。`cv2.destroyAllWindows()` 将销毁所有我们创建的窗口。如果要销毁任何特定窗口，可以使用 `cv2.destroyWindow()`，在括号内传递你想销毁的窗口名称。

#### 图像保存

修改图像或者读取图像之后，你可能想要保存图像。OpenCV通过 `cv2.imwrite()` 函数实现图像的保存。第一个参数是文件名，第二个参数是图像变量。

```
cv2.imwrite('path_to_save_image.jpg', image_gray)
```

```
1 import cv2
2 image_color=cv2.imread(r'C:\Users\pc\Desktop\map.jpg',cv2.IMREAD_COLOR)
3 image_grey=cv2.imread(r'C:\Users\pc\Desktop\map.jpg',cv2.IMREAD_GRAYSCALE)
4
5 cv2.imshow('Image Window1', image_color)
6 cv2.imshow('Image Window2', image_grey)
7 cv2.imwrite(r'C:\Users\pc\Desktop\map.jpg', image_grey)
8 gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)# 转换到灰度
9 cv2.waitKey(10000)
10 cv2.destroyAllWindows()单位：毫秒
11
```

# 4 图像基础：像素、颜色空间转换

## 导言

在进入计算机视觉领域的入门之前，掌握图像的基础知识是至关重要的。这一节将从像素深入讲解，引导大家涉足图像处理的世界，并使我们了解和使用颜色空间转换这一关键技术。

### 4.1 像素：图像的基石

像素，通常被视为图像的基本单位，是构成数字图像的最小元素。在一个二维图像中，像素被排列成行和列，形成了一个矩阵。

- **例题1：寻找特定像素值**假设你有一张灰度图像（只有一个颜色通道），请编写一个Python脚本使用OpenCV来找到图像正中心像素的灰度值。

```
import cv2

# 读取图像
image = cv2.imread('path_to_image', cv2.IMREAD_GRAYSCALE)

# 获取图像中心像素坐标
height, width = image.shape
center_pixel_value = image[height // 2, width // 2]

print("中心像素的灰度值是:", center_pixel_value)
```

### 4.2 颜色空间：通往真实视觉的桥梁

在计算机视觉中，我们经常需要转换图像的颜色空间。颜色空间是描述颜色的方法，最常见的颜色空间包括RGB（红、绿、蓝）和HSV（色调、饱和度、亮度）。

- **RGB颜色空间**：在这个模型中，任何颜色都可以通过三种颜色光量的组合来创建，即红、绿、蓝。
- **HSV颜色空间**：相对于RGB颜色模型以人的视觉感知为基础，HSV模型描述了颜色的感知特性，包括色相、饱和度和明度。
- **例题2：颜色空间转换**构建一个Python脚本以使用OpenCV将图像从RGB颜色空间转换到HSV颜色空间。

```
import cv2

# 读取彩色图像
image = cv2.imread('path_to_image')

# 将颜色空间从BGR（默认为OpenCV颜色空间）转换为HSV
hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
```

CGColorspace放大镜

## 4.3 颜色空间的实际应用

颜色空间转换在图像处理中扮演着重要的角色。下面列举一些实际场景中的应用：

- **例题3：简单的颜色分割** 使用HSV颜色空间进行颜色分割时，我们可以轻松地分离出图像中颜色相近的区域。

```
1 import cv2
2 import numpy as np
3 image = cv2.imread('path_to_image')
4 #use cvt to change the picture to its grey form
5 hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
6 #in this part we set the range of "blue" using 3 symbols of the HSV
7 #Hue 色相: The type of color, like red, green, or blue, represented by a degree on the color wheel (0° to 360°).
8 #Saturation 饱和度: The intensity of the color, from gray (0%) to full color (100%).
9 #Value 明度: The brightness of the color, from black (0%) to full brightness (100%).
10
11 #see the arrays like a matrix,(110,130)_Hue (50,255)_Saturation (50,255)_Value
12 lower_blue = np.array([110,50,50])
13 upper_blue = np.array([130,255,255])
14 #use line 15 to select all the "blue" pixels in the image
15 mask = cv2.inRange(hsv_image, lower_blue, upper_blue)
16 #do the bitwise operation of AND between the mask(selected part) and the whole image to separate them off
17 blue_object = cv2.bitwise_and(image, image, mask=mask)
18
19 cv2.imshow('Original image', image)
20 cv2.imshow('Blue object', blue_object)
21 cv2.waitKey(0)
22 cv2.destroyAllWindows()
```

## 5 绘制函数与图像基础操作

```
1 import cv2
2 import numpy as np
3
4 # create an image 'zeros' meaning all initial values are 0==black
5 #the tuple is a symbol of RGB image
6 #(height,width,3) np.uint8 is a datatype for pixels
7 img = np.zeros((512, 512, 3), np.uint8)
8
9 ###Create certain shapes
10 #LINE: image_name,starting point, ending point, color in RGB, thickness of the line
11 cv2.line(img, (0, 0), (511, 511), (255, 255, 255), 5)
12 #RECTANGLE: lefttop point, rightlow point,color,thickness
13 cv2.rectangle(img, (384, 0), (510, 128), (255, 0, 0), 3)
14 #CIRCLE: center points, radius __(-1 means the thickness is whole part inside)
15 cv2.circle(img, (256, 256), 100, (0, 0, 255), -1)
16 #ELLIPSE: centerpoint, length of axeses(longer,shorter),
17 #angle, startAngle, endAngle,-1as before
18 #angle meaning the angle between the x-axis and the long axis of the ellipse
19 #the two angles behind start and end
20 cv2.ellipse(img, (256, 256), (100, 50), 0, 50, 360, 255, -1)
21 #POLYGON:to create a polygon, we need each points' x and y value_in int form
22 #and that ought to be set in a list(pts)
23 #np.reshape()我暂时没看懂。。
24 #name.reshape([image,[list],whether the polygon is a shut one,(RGB),thickness of the pixel]
25 pts = np.array([[10, 5], [20, 30], [70, 20], [50, 10]], np.int32)
26 pts = pts.reshape((-1, 1, 2))
27 cv2.polylines(img, [pts], True, (0, 255, 0), 3)
28 #TEXT:cv2.putText(img, 内容, 文字起始位置, 文字格式, 文字大小, color, thickness = 1, lineType = cv2.LINE_AA)
29 font = cv2.FONT_HERSHEY_SIMPLEX
30 cv2.putText(img, 'OpenCV', (10, 500), font, 4, (255, 255, 255), 2, cv2.LINE_AA)
31
32 ###Edit figures(shapes)
33 px = img[100, 100] #acquire pixel
34 img[100, 100] = [255, 255, 255]#change pixel
35 print(img.shape) # return a tuple representing the shape of the figure
36 print(img.size) # return the number of the pixels in a shape
37 print(img.dtype) # return the datatype of a figure
38 # cut the ROI(certain areaof a figure)and copy to sth else
39 face = img[100:200, 115:188]
40 img[0:100, 0:73] = face
41
42 cv2.imshow('Line', img)
43 cv2.waitKey(0)
44 cv2.destroyAllWindows()
45
```

## 6 图像的算术运算与逻辑运算

### (1) 算数运算

截断在 (0, 255) 内, using cv2.add(); cv2.subtract() [simply add or minus the value of the pixels]

using cv2.addWeighted(image1,a,image2,b,...,imagei,i)(a+b+...=1) 权重之和

```
import cv2
import numpy as np
x = np.uint8([250])
y = np.uint8([10])
print(cv2.add(x, y))
```

```
x = np.uint8([10])
y = np.uint8([250])
print(cv2.subtract(x, y))
```

# 图像权重相加示例

```
img1 = cv2.imread('image1.jpg')
img2 = cv2.imread('image2.jpg')
result = cv2.addWeighted(img1, 0.7, img2, 0.3, 0)
```

### (2) 逻辑运算

'bitwise' computing

#### AND运算

使用 `cv2.bitwise_and()` 函数可以对两幅图像进行AND运算。这通常用于提取图像的特定部分。

```
## AND运算示例
mask = np.zeros_like(img1)
# 定义一个ROI
mask[100:150, 100:150] = 255
result = cv2.bitwise_and(img1, img1, mask=mask)
```

这里我们创建了一个与 `img1` 大小相同的掩码，并在掩码的某个区域内赋值为255（白色），然后应用AND运算。结果图像 `result` 只会在掩码的白色区域内保留 `img1` 的内容。

#### OR运算

使用 `cv2.bitwise_or()` 可以对两图像进行OR运算，这可以用来将两个图像合并。

```
## OR运算示例
result = cv2.bitwise_or(img1, img2)
```

#### NOT运算

使用 `cv2.bitwise_not()` 函数可以对图像进行NOT运算，通常用于图像的反色处理。

```
## NOT运算示例
result = cv2.bitwise_not(img1)
```

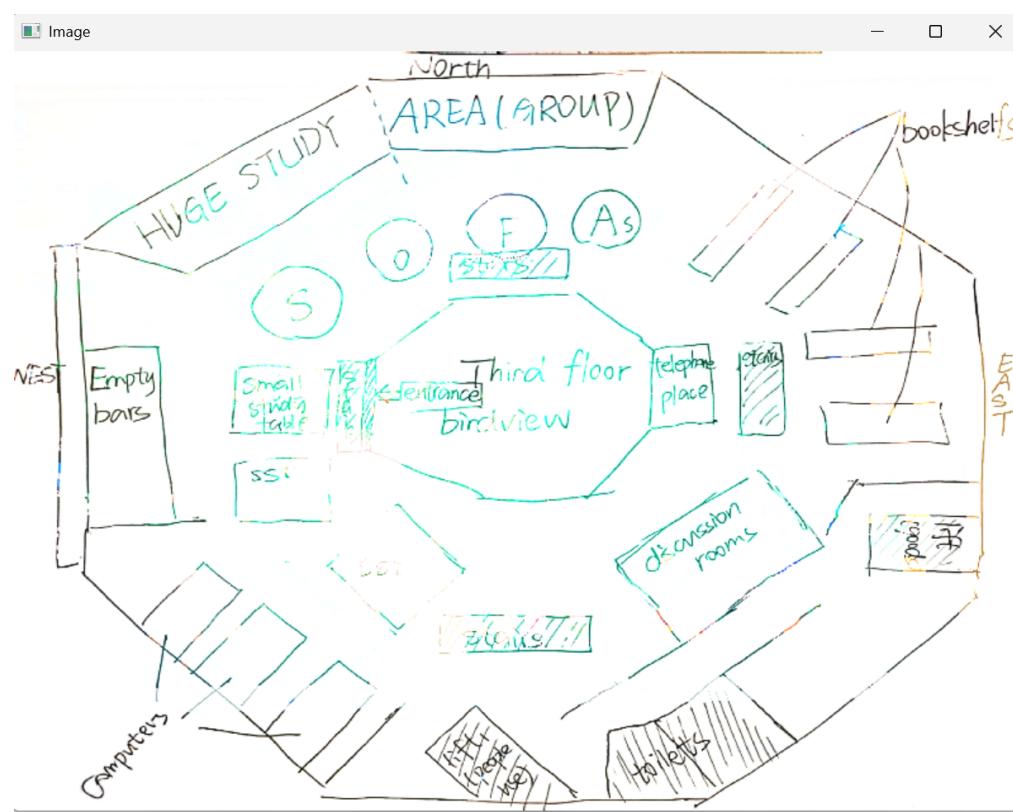
#### XOR运算

`cv2.bitwise_xor()` 函数用于执行XOR运算。

```
## XOR运算示例
result = cv2.bitwise_xor(img1, img2)
```

## 实例操作：

```
1 import cv2
2 import numpy as np
3 foreground = cv2.imread('map.jpg')
4 background = cv2.imread('picture1.jpg')
5 mask = cv2.imread('map.jpg', 0)
6 #前景; 背景, mask掩码 as mask approaches to 0 the part of the background appears
7 #more accurate, as it approaches to 255 the frontground appears more clearly
8 reference_size = foreground.shape[:2]
9 # .shape returns a tuple(height,width,channels(if colors,==3, grey ==1))
10 # and in this line it choose the size not the channels
11 background = cv2.resize(background, (reference_size[1], reference_size[0]))
12 mask = cv2.resize(mask, (reference_size[1], reference_size[0]))
13 #cv2.resize(image, dsize(width,height), fx=0, fy=0,
14 #interpolation插值方式 (怎么样resize) =cv2.INTER_LINEAR)
15 #cv2.INTER_NEAREST快速但是品质不佳
16 #cv2.INTER_LINEAR线性插值, 适用于缩放
17 #cv2.INTER_AREA区域插值, 适用于缩小
18 #cv2.INTER_CUBIC三次样条插值, 适用于放大
19 #cv2.INTER_LANCZOS4超级精细化
20 print("Foreground shape:", foreground.shape)
21 print("Background shape:", background.shape)
22 print("Mask shape:", mask.shape)
23 #debug
24 if foreground is None or background is None or mask is None:
25     print("Error: One or more images could not be loaded. Check file paths.")
26     exit()
27 if foreground.shape[:2] != background.shape[:2] or foreground.shape[:2] != mask.shape[:2]:
28     print("Error: Images or mask dimensions do not match.")
29     exit()
30
31 foreground = cv2.bitwise_and(foreground, foreground, mask=mask)
32 #mask!=0 ->save the result as and
33 #to extract the frontground
34 background = cv2.bitwise_and(background, background, mask=cv2.bitwise_not(mask))
35 result = cv2.add(foreground, background)
36
37 cv2.namedWindow('Image', cv2.WINDOW_NORMAL)
38 cv2.imshow('Image', result)
39 cv2.resizeWindow('Image', 800, 600)
40 cv2.waitKey(0)
```



## 7 图像的几何变换基础：缩放、平移

### (1) 缩放 cv2.resize()

fx,fy means the 系数 for appendix

```
11 background = cv2.resize(background, (reference_size[1], reference_size[0]))
12 mask = cv2.resize(mask, (reference_size[1], reference_size[0]))
13 #cv2.resize(image, dsize(width,height), fx=0, fy=0,
14 #interpolation插值方式 (怎么样resize) =cv2.INTER_LINEAR)
15 #cv2.INTER_NEAREST快速但是品质不佳
16 #cv2.INTER_LINEAR线性插值，适用于缩放
17 #cv2.INTER_AREA区域插值，适用于缩小
18 #cv2.INTER_CUBIC三次样条插值，适用于放大
19 #cv2.INTER_LANCZOS4超级精细化
```

### (2) 平移 cv2.warpAffine()

(actually 仿射变换 (affine transformation) 实现图像的平移、旋转、缩放、倾斜)

cv2.warpAffine(image, M, dsize, flags=1, borderMode=cv2.BORDER\_CONSTANT, borderValue=(0, 0, 0))

flag means the interpolation before(default=linear)

borderMode:边界处理方法 borderValue: 超出边界填充颜色

cv2.BORDER\_CONSTANT填充常数，需要borderValue

cv2.BORDER\_REFLECT反射边缘像素 (1,2,3) -> (3,2,1)

cv2.BORDER\_REPLICATE复制边缘像素

M is a very complex matrix 2\*3

The matrix M looks like this:

$$M = \begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix}$$

Where:

- $a, b, d, e$ : Define rotation, scaling, and shearing.
- $c, f$ : Define translation (shifting in the  $x$  and  $y$  directions).

For each pixel in the output image, its position  $(x', y')$  is calculated as:

$$x' = a \cdot x + b \cdot y + c$$

$$y' = d \cdot x + e \cdot y + f$$

平移

缩放

旋转

错切 (成倒影)

Where  $t_x$  and  $t_y$  are the translation distances.

$$M = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \end{bmatrix}$$

Where  $s_x$  and  $s_y$  are the scale factors in the  $x$  and  $y$  directions.

$$M = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & t_x \\ \sin(\theta) & \cos(\theta) & t_y \end{bmatrix}$$

Where:

- $\theta$ : Rotation angle in radians.
- $t_x, t_y$ : Adjustments to keep the rotated image properly aligned.

Matrix:

$$M = \begin{bmatrix} 1 & k_x & 0 \\ k_y & 1 & 0 \end{bmatrix}$$

Where:

- $k_x$ : Shear factor along the  $x$ -axis.
- $k_y$ : Shear factor along the  $y$ -axis.

## 8 图像的几何变换进阶：旋转、仿射变换

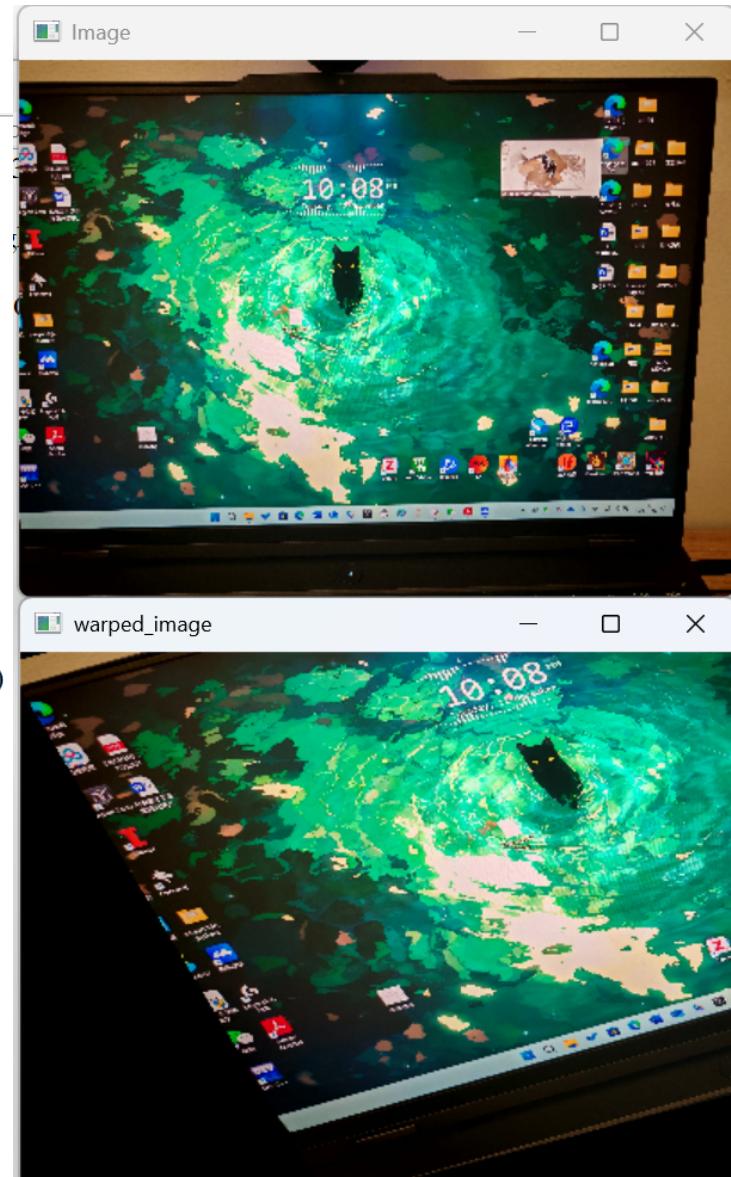
### (1) Rotation 旋转

```
1 import cv2
2
3 image = cv2.imread('map.jpg')
4 rows, cols = image.shape[:2]
5
6 # 旋转中心点为图像中心，旋转角度为90度，不改变图像尺寸
7 M = cv2.getRotationMatrix2D((cols / 2, rows / 2), 90, 1)
8 rotated_image = cv2.warpAffine(image, M, (cols, rows))
9 #cv2.getRotationMatrix2D(center, angle, scale)
10 #when processing normal M for cv2.warpAffine
11 #We use like this:
12 #M = np.float32( [ [a,b,c],
13 #                  [d,e,f] ] )
14
15 cv2.namedWindow('Image', cv2.WINDOW_NORMAL)
16 cv2.imshow('Image', rotated_image)
17 cv2.resizeWindow('Image', 800, 600)
18 cv2.waitKey(0)
19 cv2.destroyAllWindows()
```

### (2) Affine Transformation 仿射变换

```
1 import cv2
2 import numpy as np
3
4 image = cv2.imread('picture1.jpg')
5 rows, cols, ch = image.shape
6
7 # 原图中的三个点以及它们在输出图像中的位置
8 pts1 = np.float32([[50, 50], [200, 50], [50, 200]])
9 pts2 = np.float32([[10, 100], [200, 50], [100, 250]])
10 # 生成仿射变换矩阵
11 M = cv2.getAffineTransform(pts1, pts2)
12 # 应用仿射变换
13 warped_image = cv2.warpAffine(image, M, (cols, rows))
14 # 显示结果
15 cv2.namedWindow('Image', cv2.WINDOW_NORMAL)
16 cv2.imshow('Image', image)
17 cv2.namedWindow('warped_image', cv2.WINDOW_NORMAL)
18 cv2.imshow('warped_image', warped_image)
19 cv2.resizeWindow('Image', 800, 600)
20 cv2.resizeWindow('warped_image', 800, 600)
21 cv2.waitKey(0)
22 cv2.destroyAllWindows()
```

These two methods could be combined together as follows.



```
# 旋转后的仿射变换
M_rot = cv2.getRotationMatrix2D((cols / 2, rows / 2), 45, 1)
rotated_image = cv2.warpAffine(image, M_rot, (cols, rows))
```

```
# 仿射变换的三点
pts1 = np.float32([[100, 100], [200, 100], [100, 200]])
pts2 = np.float32([[80, 150], [200, 130], [120, 220]])
```

```
M_affine = cv2.getAffineTransform(pts1, pts2)
warped_image = cv2.warpAffine(rotated_image, M_affine, (cols, rows))
```

## 9 图像的阈值处理：全局阈值与自适应阈值

阈值处理是图像处理中非常基础且关键的步骤，它的主要目的是为了简化图像的分析和处理，将图像二值化，即将像素点的灰度值设置为0或255。阈值处理通常用于图像分割、去除或识别对象等各种任务中。本节课程将详细介绍两种常见的阈值处理方法：全局阈值（Global Thresholding）和自适应阈值（Adaptive Thresholding）。

`cv2.threshold` 函数是OpenCV中用于进行图像阈值处理的函数，它能够将图像转换为二进制形式或根据设定的阈值进行分割。以下是该函数的基本语法：

```
retval, dst = cv2.threshold(src, thresh, maxval, type)
```

其中各参数的含义如下：

- `src`：输入图像，即待处理的原始图像，必须是单通道灰度图。
- `thresh`：设定的阈值，用于对图像进行分割。
- `maxval`：分割后的像素值，当像素值超过阈值时，设定的像素值。
- `type`：阈值处理的类型，包括以下几种取值：
  - `cv2.THRESH_BINARY`：超过阈值的像素设为 `maxval`，否则设为0。
  - `cv2.THRESH_BINARY_INV`：超过阈值的像素设为0，否则设为 `maxval`。
  - `cv2.THRESH_TRUNC`：超过阈值的像素设为阈值，其他像素不变。
  - `cv2.THRESH_TOZERO`：超过阈值的像素不变，小于阈值的像素设为0。
  - `cv2.THRESH_TOZERO_INV`：超过阈值的像素设为0，小于阈值的像素不变。

函数返回值包括：

- `retval`：计算出的阈值，用于后续分析。
- `dst`：处理后的输出图像，即阈值处理后的图像。

全局阈值处理是用一个值处理，适用于光线均匀分布，对比度清晰的场景

自适应阈值处理是用在光线不均匀分布的地方，但由于阈值与每一个像素点的周围的多个点有关，可以更加精确的分割图像的前景。

对图像进行预处理，可以分辨出前景和后景  
(operate) 有点像4.3

### 1. 均值法 (Adaptive Mean Thresholding)

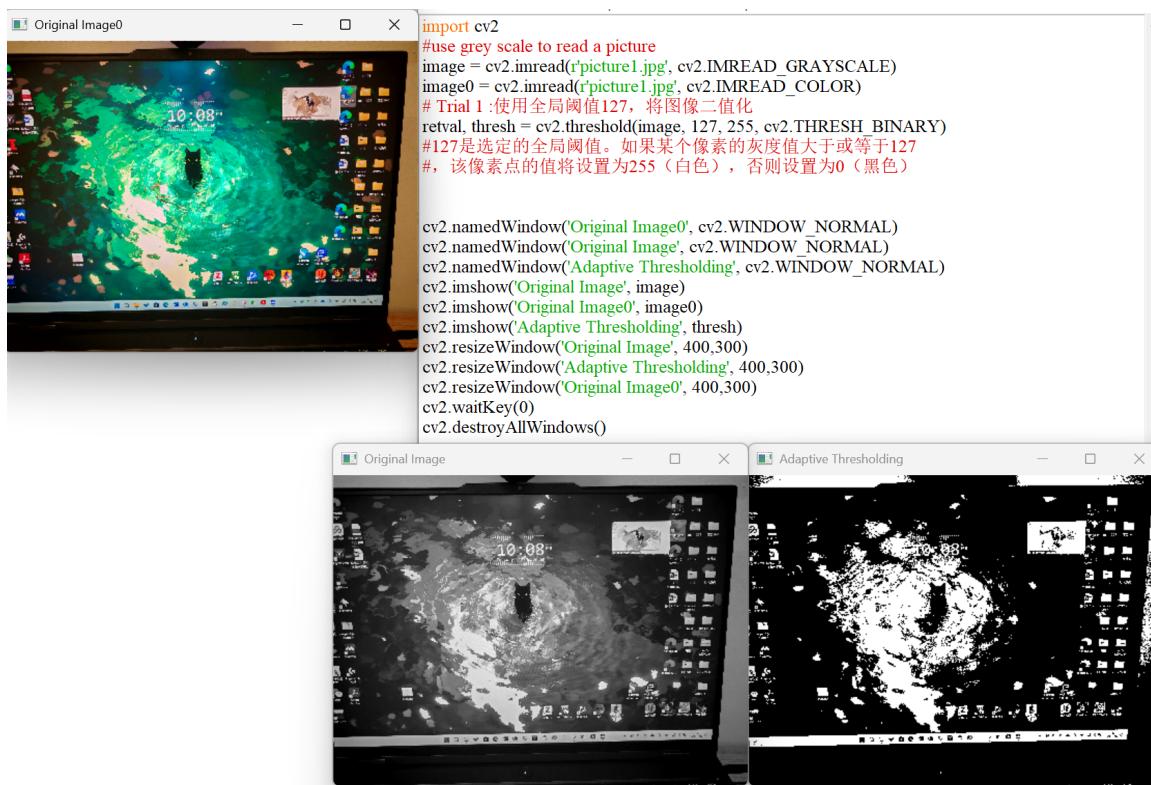
#### • 计算方式：

每个像素的阈值是其局部邻域（以方形窗口表示）的像素灰度值的均值，再减去一个常量  $C$ 。

$$T(x, y) = \text{mean}(N(x, y)) - C$$

其中  $N(x, y)$  是像素  $(x, y)$  的邻域， $C$  是人为设定的偏移值，用于调整阈值的灵敏度。

### THE FIRST TRIAL OF 全局 阈值处理



## 1. 均值法 (Adaptive Mean Thresholding)

- 计算方式:

每个像素的阈值是其局部邻域（以方形窗口表示）的像素灰度值的均值，再减去一个常量  $C$ 。

$$T(x, y) = \text{mean}(N(x, y)) - C$$

其中  $N(x, y)$  是像素  $(x, y)$  的邻域， $C$  是人为设定的偏移值，用于调整阈值的灵敏度。

## 2. 高斯法 (Adaptive Gaussian Thresholding)

- 计算方式:

每个像素的阈值是其局部邻域像素值的加权平均值，权重由高斯函数决定，再减去常量  $C$ 。

$$T(x, y) = \text{GaussianWeightedSum}(N(x, y)) - C$$

这种方法给邻域中心的像素赋予更大的权重，周围像素的影响逐渐减小。

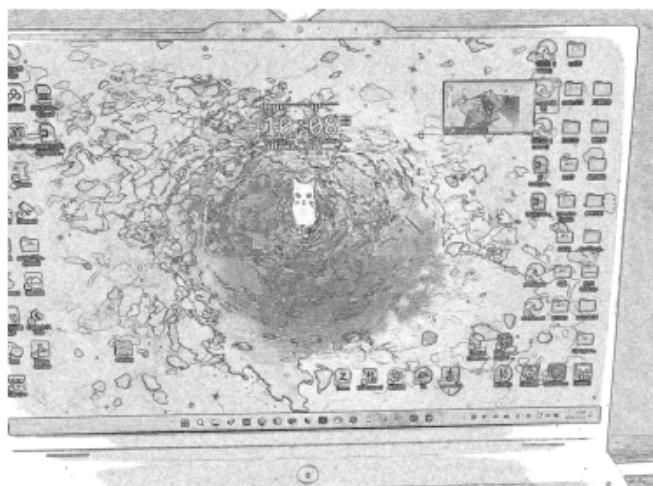
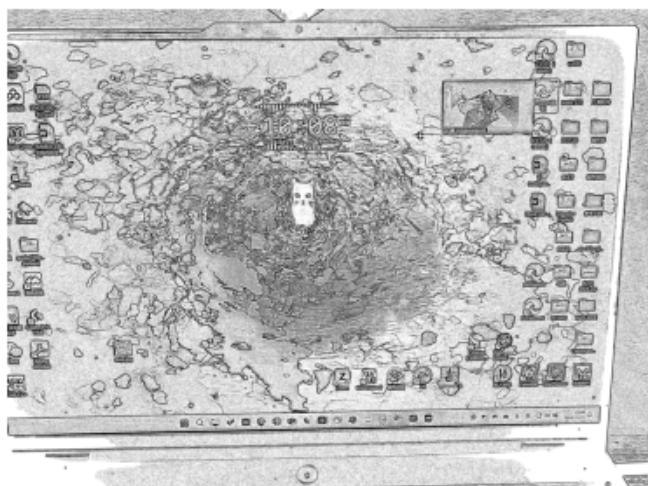
```

1 import cv2
2 import matplotlib.pyplot as plt
3 #use grey scale to read a picture
4 image = cv2.imread(r'picture1.jpg', cv2.IMREAD_GRAYSCALE)
5
6 # mean
7 adaptive_mean = cv2.adaptiveThreshold(image, 255, cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY, 11, 2)
8 # gaussian
9 adaptive_gaussian = cv2.adaptiveThreshold(image, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 11, 2)
10 #cv2.adaptiveThreshold(image,value of the pixel when processed,method,
11 # #if||if not,the final slight change)
12 #increase the '2'constant C to reduce the noises
13 # SHOW THE CONTRASTING PICTURES
14 plt.figure(figsize=(8, 4))#size of the figure created(width, height)
15
16 plt.subplot(1, 2, 1)#generate many figures at the same time
17 plt.title('Adaptive Mean')#one row, three column, the first one
18 plt.imshow(adaptive_mean, cmap='gray')
19 plt.axis('off')
20
21 plt.subplot(1, 2, 2)
22 plt.title('Adaptive Gaussian')
23 plt.imshow(adaptive_gaussian, cmap='gray')
24 plt.axis('off')
25
26 plt.tight_layout()#automatically set the space \
27 #between the pictures to make it more beautiful
28 plt.show()

```

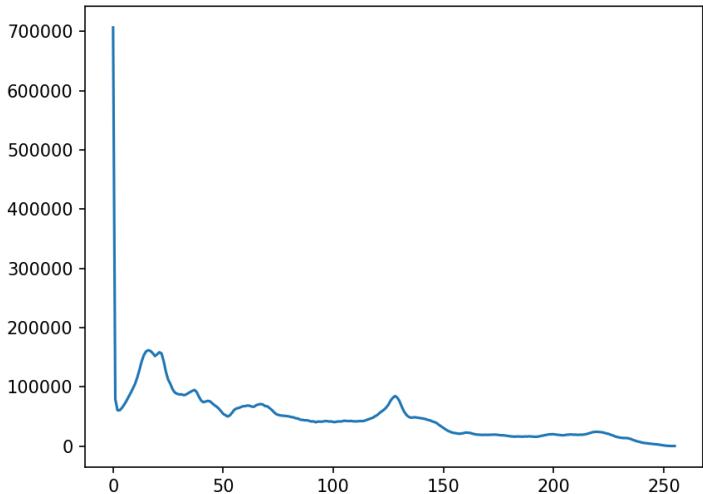
Adaptive Mean

Adaptive Gaussian



# 10 图像的基础分析：直方图与掩模操作

```
1 import cv2
2 import matplotlib.pyplot as plt
3 import numpy as np
4
5 image_GREY = cv2.imread(r'picture1.jpg', cv2.IMREAD_GRAYSCALE)
6 image_COLOR = cv2.imread(r'picture1.jpg', cv2.IMREAD_COLOR)
7 #直方图cv2.calHist(image,tunnel,mask,xrange,yrange)
8 #tunnel:在灰度图像中为[0],如果是彩色图像,可以传入[0], [1] 或 [2] 来分别计算B、G、R通道的直方图
9 #mask:choose specific area pf the image to operate,255mean not operate,OR NONE
10 mask = np.zeros(image_GREY.shape[:2], dtype="uint8")
11 histogram = cv2.calcHist([image_GREY], [0], None, [256], [0, 256])
12
13 plt.plot(histogram)
14 plt.show()
```



掩码操作见过往的的python程序，已经用到过了，这里不再重复涉及

## 11. 图像平滑技术：模糊与去噪

图像平滑，也称为图像模糊或去噪，主要目的是去除图像中的随机噪声。这些噪声可能是由图像捕获设备（如相机）或传输过程中引入的。噪声可以减少图像的质量，影响后续的图像处理任务，比如特征检测、图像分类等。

```
1 import cv2
2 import matplotlib.pyplot as plt
3 import numpy as np
4 image = cv2.imread(r'picture6.jpg',cv2.IMREAD_GRAYSCALE)
5 image_mean = cv2.blur(image,(5, 5))#均值滤波 (image,(a,b)size of the kernel)
6 image_Gaussian = cv2.GaussianBlur(image,(5,5),100)#高斯滤波(image,(a,b)kernel,c标准差)
7 image_median = cv2.medianBlur(image,5)#中值滤波(image,kernel)
8
9 #extra statements:
10 #均值滤波和高斯滤波是线性滤波，采用均值或加权求和来进行计算，可二维定义核
11 #中值滤波是对于周围的像素值排序取中值，是非线性的所以采用正方形核
12 plt.figure(figsize=(8, 4))#size of the figure created(width, height)
13
14 plt.subplot(1, 3, 1)#generate many figures at the same time
15 plt.title('mean')#one row, three column, the first one
16 plt.imshow(image_mean,cmap='grey')
17 plt.axis('off')
18
19 plt.subplot(1, 3, 2)
20 plt.title('Gaussian')
21 plt.imshow(image_Gaussian,cmap='grey')
22
23 plt.axis('off')
24
25 plt.subplot(1, 3, 3)
26 plt.title('median')
27 plt.imshow(image_median,cmap='grey')
28 plt.axis('off')
29
30 plt.tight_layout()#automatically set the space \
31 #between the pictures to make it more beautiful
32 plt.show()
33 cv2.waitKey(0)
34 cv2.destroyAllWindows()
```