警务 图像分析技术

智慧警务学院　李智慧

12

Python 图像分割

目　录
contents

1

图像分割概述

图像分析概述

- 图像分割是将图像分成若干个具有独特性质的区域并提取感兴趣目标的技术与过程，是图像处理和图像分析的关键步骤。
- 主要分为基于阈值的分割方法、基于区域的分割方法、基于边缘的分割方法和基于特定理论的分割方法。
- 本章重点围绕图像处理实例，详细介绍各种图像分割的方法。

图像分析概述

- 图像分割 (image segmentation) 技术是计算机视觉领域的重要研究方向，是图像语义理解和图像识别的重要一环。它是指将图像分割成若干个具有相似性质的区域的过程。
- 研究方法包括基于阈值的分割方法、基于区域的分割方法、基于边缘的分割方法和基于特定理论的分割方法 (含图论、聚类、深度语义等) 。
- 该技术广泛应用于场景物体分割、人体背景分割、三维重建、车牌识别、人脸识别、无人驾驶、增强现实等行业 12-4] 。如图所示，可将鲜花颜色划分为四个层级。

图像分析概述

- 该技术广泛应用于场景物体分割、人体背景分割、三维重建、车牌识别、人脸识别、无人驾驶、增强现实等行业12-4]。如图所示，可将鲜花颜色划分为四个层级。

图像分析概述

- 图像分割是根据图像中的物体将图像的像素分类，并提取感兴趣的目标。
- 从数学角度来看，图像分割是将数字图像划分成互不相交的区域的过程。图像分割的过程也是一个标记过程，即把属于同一区域的像素赋予相同的编号。
- 图像分割主要依据图像中像素的亮度及颜色，但计算机在自动处理分割时，会遇到各种困难，如光照不均匀、噪声影响、图像中存在不清晰的部分以及阴影等。同时，随着深度学习和神经网络的发展，基于深度学习和神经网络的图像分割技术有效地提高了分割的准确率，能够较好地解决图像中噪声和不均匀问题。

2

基于阈值的图像分割

12.2 基于阈值的图像分割

- 最常用的图像分割方法是将图像灰度分为不同的等级，然后用设置灰度门限的方法确定有意义的区域或欲分割的物体边界。
- 图像阈值化 (binarization) 旨在去除图像中一些低于或高于一定值的像素，从而提取图像中的物体，将图像的背景和噪声区分开。图像阈值化可以理解为一个简单的图像分割操作，阈值又称为临界值，目的是确定出一个范围，然后这个范围内的像素点使用同一种方法处理，而阈值之外的部分则使用另一种处理方法或保持原样。

12.2 基于阈值的图像分割

- 阈值化处理可以将图像中的像素划分为两类颜色，常见的阈值化算法如下面公式所示，当某个像素点的灰度 Gray (i, j) 小于阈值 T 时，其像素设置为 0, 表示黑色；当灰度 Gray (i, j) 大于或等于阈值 T 时，其像素值为 255，表示白色。

$$\mathrm{Gray}(i,j)=\begin{cases}255, & \mathrm{Gray}(i,j)\geqslant T\\ 0, & \mathrm{Gray}(i,j)<T\end{cases}$$

12.2 基于阈值的图像分割

- 在 Python 的 OpenCV 库中，提供了固定阈值化函数 threshold () 和自适应阈值化函数 adaptiveThreshold ()，将一幅图像进行阈值化处理，第 6 章已详细介绍了图像阈值化处理方法，下面的代码对比了不同阈值化算法的图像分割结果。
- # 阈值化处理 (核心程序)

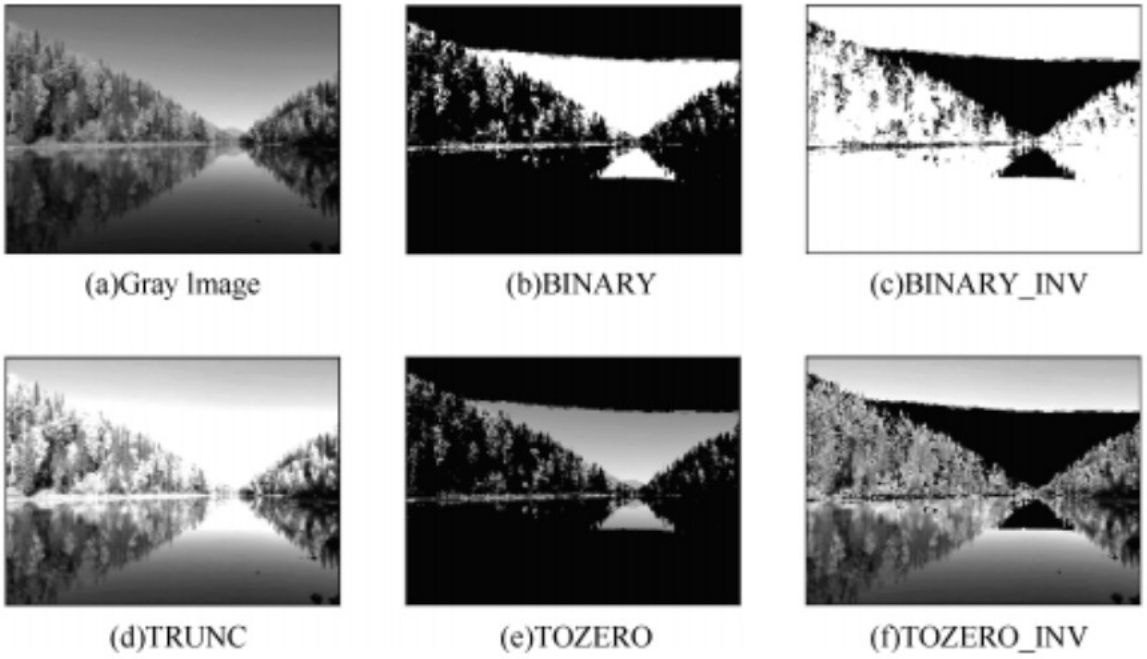
```
ret, thresh1=cv2. threshold (grayImage, 127, 255, cv2. THRESH_BINARY)
ret, thresh2=cv2. threshold (grayImage, 127, 255, cv2.
THRESH_BINARY_INV)
ret, thresh3=cv2.threshold (grayImage, 127, 255, cv2. THRESH_TRUNC)
ret, thresh4=cv2.threshold (grayImage, 127, 255, cv2. THRESH_TOZERO)
ret, thresh5=cv2.threshold (grayImage, 127, 255, cv2. THRESH_TOZEROINV)
```

12.2 基于阈值的图像分割

参数	参数作用效果
cv2.THRESH_BINARY 二值阈值化	像素点的灰度值大于阈值，设其灰度值为最大值（255）；小于阈值，设其灰度值为最小值（0）
cv2.THRESH_BINARY_INV （反二进制阈值化）	像素点的灰度值大于阈值，设其灰度值为最小值（0）；小于阈值，设其灰度值为最大值（255）
cv2.THRESH_TRUNC （截断阈值化，大于阈值设为阈值）	像素点的灰度值小于阈值，其灰度值不变；大于阈值，设其灰度值为阈值
cv2.THRESH_TOZERO （阈值化为 0 （高于阈值））	像素点的灰度值小于阈值，其灰度值不变；大于阈值，设其灰度值为最小值（0）
cv2.THRESH_TOZERO_INV （反阈值化为 0 （低于阈值））	像素点的灰度值大于阈值，其灰度值不变；小于阈值，设其灰度值为最小值（0）

12.2 基于阈值的图像分割

5 种阈值化算法的图像分割结果对比



(a)Gray Image　(b)BINARY　(c)BINARY_INV

(d)TRUNC　(e)TOZERO　(f)TOZERO_INV

3

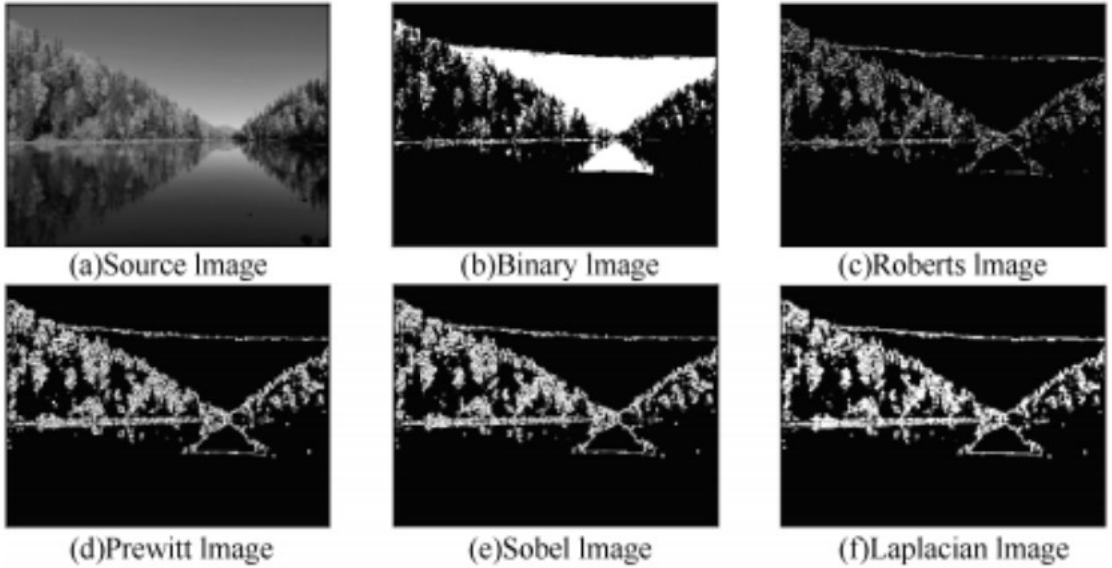
基于边缘检测的图像分割

12.3 基于边缘检测的图像分割

- 基于边缘检测的图像分割方法是通过确定图像中的边缘轮廓像素，然后将这些（边缘轮廓）像素连接起来构建区域边界的过程。
- 由于沿着图像边缘走向的像素值变化比较平缓，而沿着垂直于边缘走向的像素值变化比较大，所以通常会采用一阶导数和二阶导数来描述与检测边缘。
- 第 11 章详细介绍了 Python 边缘检测的方法，下面的代码是对比常用的微分算子，如 Roberts. Prewitt. Sobel. Laplacian 等。

12.3 基于边缘检测的图像分割

- 第 11 章详细介绍了 Python 边缘检测的方法，下面的代码是对比常用的微分算子，如 Roberts. Prewitt. Sobel. Laplacia



4

基于纹理背景的图像分割

12.4 基于纹理背景的图像分割

- 基于纹理背景的图像分割是指基于图像纹理信息（颜色）、边界信息（反差）和背景信息的图像分割算法。
- 在 OpenCV 中， GrabCut 算法能够有效地利用纹理信息和边界信息分割背景，提取图像目标物体。该算法是微软研究院基于图像分割和抠图的课题，它能有效地将目标图像分割提取，如图 14-6 所示。

12.4 基于纹理背景的图像分割

- GrabCut 算法原型如下所示。
- mask, bgdModel, fgdModel =grabCut (img, mask, rect, bgdModel, fgdModel, iterCount [, mode])

img: 输入图像，必须是 8 位 3 通道图像，在处理过程中不会被修改。

mask: 掩码图像，用来确定哪些区域是背景，前景，可能是背景，可能是前景等。 GCD_BGD (=0), 明确属于背景的像素； GCD_FGD (=1)，明确属于前景的像素 ;GCD_PR_BGD (=2)，可能是背景的像素 ;GCD_PR_FGD(=3)，可能是前景的像素。如果没有手工标记 GCD_BGD 或者 GCD_FGD, 那么结果只会有 GCD_PR_BGD 和 GCD_PR_FGD

rect: 包含前景的矩形，格式为（x, y, w, h）

bdgModel,fgdModel: 算法内部使用的数组，只需要创建两个大小为（1，65），数据类型为 np.float64 的数组

iterCount: 算法迭代的次数

12.4 基于纹理背景的图像分割

```
import cv2
import numpy as np
# Step1. 加载图像
img = cv2. imread('5. jpg')
# Step2. 创建掩模、背景图和前景图
mask = np. zeros( img. shape[:2], np. uint8)# 创建大小相同的掩模
bgdModel = np. zeros((1,65) ,  np.float64)#  创建背景图像
fgdModel = np.zeros((1,65), np. float64)# 创建前景图像
```

12.4 基于纹理背景的图像分割

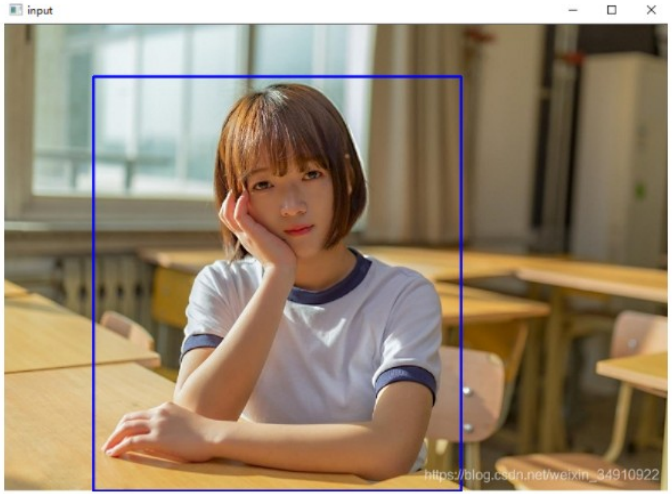
```
# Step3. 初始化矩形区域
# 这个矩形必须完全包含前景
rect = (50,0,450,333) # 格式为 (x, y, W, h)
# Step4. GrubCut 算法，  迭代 5 次
# mask 的取值为 0,1,2,3
cv2. grabCut(img, mask, rect, bgdModel, fgdModel, 5, cv2.GC_ INIT_ WITH_
RECT)
# 迭代 5 次
# Step5. mask 中，值为 2 和 0 的统一转化  为 0 ，  1 和 3 转化为 1
mask2 = np.where((mask == 2)| (mask == 0) ，  0, 1) .astype( 'uint8')
img = img * mask2[:,: ,np.newaxis] # np.newaxis  插入一个新维度，相当于将
二  维矩阵扩充为三维
cv2. imshow("dst", img)
cv2. waitKey(0)
```

12.4 基于纹理背景的图像分割

结果如上图，左边是 grabCut 结果，右边是原图。
这个算法的关键就是，在开始用户画一个矩形方块把前景图圈起来，前景区域应该完全在矩形内，然后算法反复进行分割以达到最好效果。

12.4 基于纹理背景的图像分割

```
import cv2
import numpy as np
img = cv2. imread( img_ path)
r=cv2.selectROI('input' ,     img, False) # 返回 (x_min, y_min,w, h)
print("input:" ,   r)
```

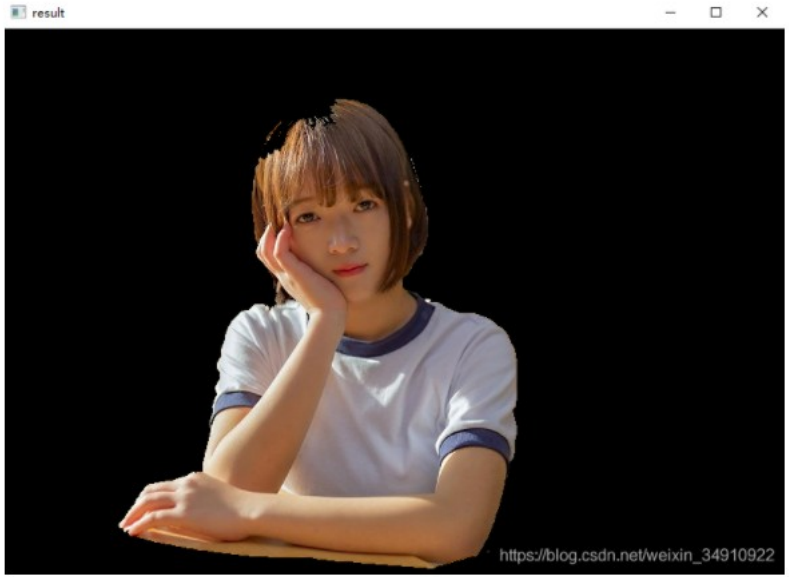


12.4 基于纹理背景的图像分割

```
# roi 区域
roi = img[int(r[1]):int(r[1] + r[3]) ,     int(r[0]):int(r[0] + r[2])]
# 原图 mask
mask = np.zeros ( img. shape[:2], dtype=np. uint8)
# 矩形 roi
rect = (int(r[0]), int(r[1]) ,   int(r[2]) ,     int(r[3])) # 包括前景的矩形，格式为(x,y,w,h)
bgdmodel = np.zeros((1, 65) ,   np.float64) # bg 模型的临时数组
fgdmodel = np.zeros((1, 65) ,   np.float64) # fg 模型的临时数组
cv2.grabCut(img, mask, rect, bgdmodel, fgdmodel, 11 ,     mode=cv2.GC_ INIT_ WITH_ RECT)
# 提取前景和可能的前景区域
mask2 = np.where((mask == 1) + (mask == 3) ,   255 ,     0).astype('uint8')
print (mask2. shape)
result = cv2.bitwise_ and(img ,   img, mask=mask2)
cv2. imwrite('result.jpg', result)
cv2. imwrite('roi.jpg', roi)
```

12.4 基于纹理背景的图像分割

```
cv2. imshow( 'roi' ,  roi)
cv2. imshow("result" ,  result)
cv2. waitKey(0)
cv2. destroyAl lWindows ( )
```



5

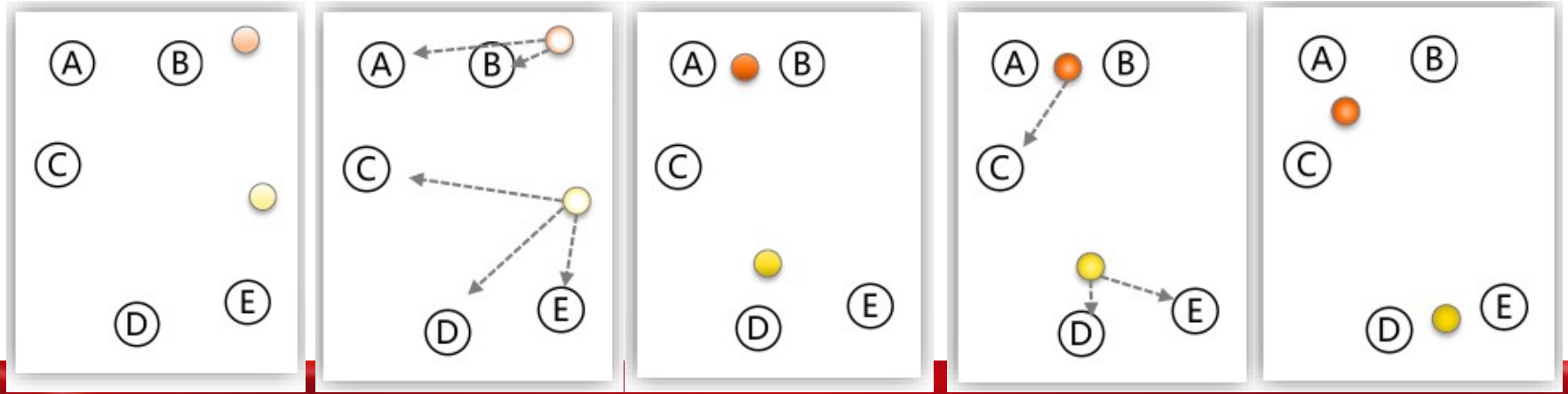
基于 K-Means 聚类的图像分割

12.5 基于 K-Means 聚类的图像分

K-means 聚类原理

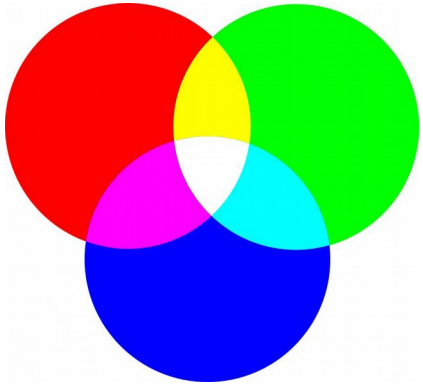
K-means 算法以 k 为参数，把 n 个对象分成 k 个簇。处理过程如下：

1. 随机选择 k 个点作为初始的聚类中心；
2. 对于剩下的点，根据其与聚类中心的距离，将其 归入最近的簇；
3. 对每个簇，计算所有点的均值作为新的聚类中心；
4. 重复 2 、 3 直到聚类中心不再发生改变。

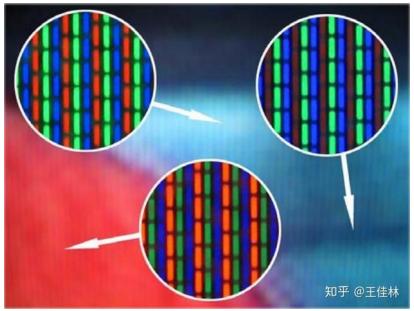


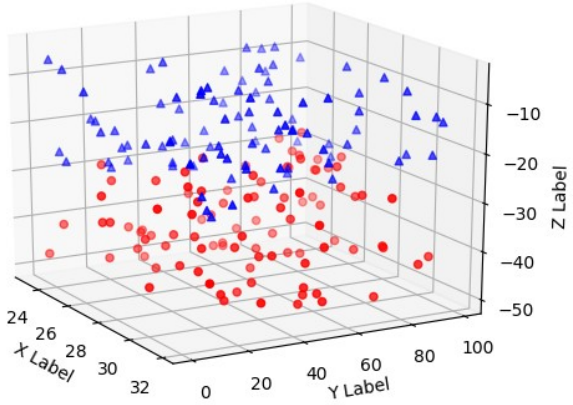
12.5 基于 K-Means 聚类的图像分

K-means 聚类原理



像素点	红	绿	蓝
1	0	0	0
2	125	125	125
…			
n	50	50	50





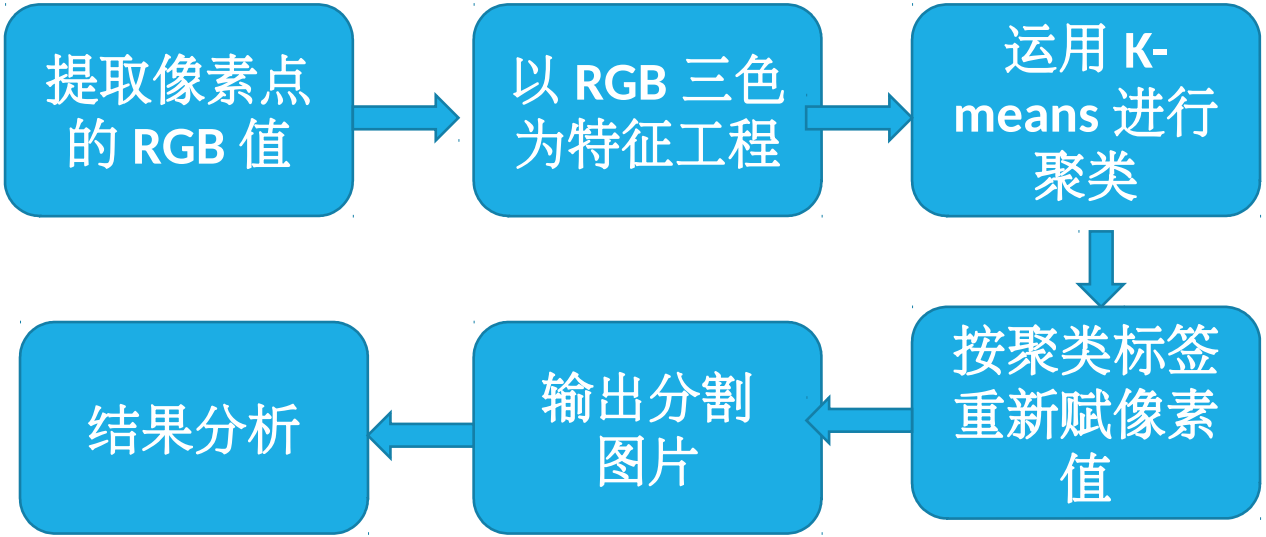
12.5 基于 K-Means 聚类的图像分

K-means 聚类原理

0 1 2 3

12.5 基于 K-Means 聚类的图像分

K-means 聚类原理（问答）



12.5 基于 K-Means 聚类的图像分割

基于 K-means 聚类的图像分割程序

(1) 准备工作（导入相关库）

①Numpy （科学计算）

②PIL(Python Imaging Libary)

③Sklearn （机器学习算法）

程序实现

```
import numpy as np
import PIL.Image as image
from sklearn.cluster import KMeans
```

12.5 基于 K-Means 聚类的图像分

基于 K-means 聚类的图像分割程序

(2) 加载图片及预处理

① 读入图片

image.open()

② 数据预处理

a. 获取各像像素点的 RGB 值

image.getpixel(i,j)

b. 对 RGB 值归一化，存列表

[x/256.0,y/256.0,z/256.0]

c. 将数据转换成矩阵形式

np.mat()

程序实现

```
img=image.open('bull.jpg')
row,col=img.size
data=[ ]
 for i in range(row):
     for j in range(col):
             x,y,z=img.getpixel((i,j))

      data.append([x/256.0,      y/
256.0,z/256.0])
imgData=np.mat(data)
```

12.5 基于 K-Means 聚类的图像分割

基于 K-means 聚类的图像分割程序

(3) 运用 K-means 进行聚类

	程序实现
① 加载 Kmeans 聚类算法 KMeans(n_clusters=4)	km=KMeans(n_clusters=4)
② 对像素点进行聚类并输出 fit_predict()	label=km.fit_predict(imgData)
③ 变换聚类结果形式 label.reshape()	label=label.reshape([row,col])

12.5 基于 K-Means 聚类的图像分

基于 K-means 聚类的图像分割程序

(4) 按聚类标签重新赋像素值

① 创建一张新的灰度图 (同行列)

image.new('L',(row,col),255)

② 根据像素点的聚类标签向图片中添加灰度值

putpixel((i,j), 像素值)

像素值 =256/(label+1)

③ 保存图片

pic_new.save()

程序实现

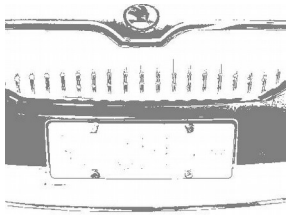
```
pic_new=image.new('L',
(row,col),255)
for i in range(row):
    for j in range(col):
        pic_new.putpixel((i,j),
int(256/(label[i][j]+1)))
pic_new.save("result-
bull.jpg","JPEG")
```

12.5 基于 K-Means 聚类的图像分

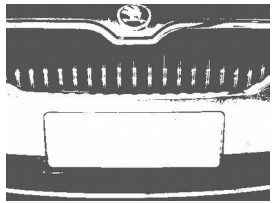
基于 K-means 聚类的图像分割程序

(5) 分割结果显示

0



1



2

3