《数字图像处理》实验方案
(Python 版本)

教 学 院 （ 部 ） 自动化工程学院

实 验 教 室 _____

授 课 班 级 _____

授 课 教 师 李玉霞

职 称 职 务 副教授

教 材 名 称 数字图像处理

电子科技大学 自动化工程学院

2022-4-7

Python 实验环境搭建

以下实验内容是基于以下软件版本进行的，请大家参考：

1. Python 版本： Python3.7

   参考安装教程：https://blog.csdn.net/weixin_45468278/article/details/102518569

2. 编辑器：推荐 Pycharm

   参考安装教程：https://cloud.tencent.com/developer/article/1482705

3. Opencv 版本：3.x 版本应该都可以

   参考安装教程：https://blog.csdn.net/qq_38328871/article/details/84842381

实验一 Python 中数字图像处理的基本操作

一、实验目的

（1）熟悉及掌握在 python 中能够处理哪些格式图像；
（2）熟练掌握在 python 中如何用 OpenCV 读取图像；
（3）掌握如何利用 python 来获取图像的大小、颜色、高度、宽度等等相关信息；
（4）掌握如何在 python 中用 OpenCV 按照指定要求存储一幅图像的方法；
（5）图像间如何转化。

二、实验主要仪器设备

(1)计算机；
(2)python 软件；
(3)典型的灰度、彩色图像文件。

三、实验原理

1.数字图像的表示和类别

一幅图像可以被定义为一个二维函数 f(x,y),其中 x 和 y 是空间(平面)坐标，f 在任何坐标(x,y)处的振幅称为图像在该点的亮度。灰度是用来表示黑白图像亮度的一个术语，而彩色图像是由单个二维图像组合形成的。例如，在 RGB 彩色系统中，一幅彩色图像是由三幅独立的分量图像(红、绿、蓝)组成的。因此，许多为黑白图像处理开发的技术适用于彩色图像处理，方法是分别处理三副独立的分量图像即可。

要将这样的一幅图像转化为数字形式，就要求数字化坐标和振幅。将坐标值数字化成为取样；将振幅数字化成为量化。采样和量化的过程如图 1 所示。因此，当 f 的 x、y 分量和振幅都是有限且离散的量时，称该图像为数字图像。

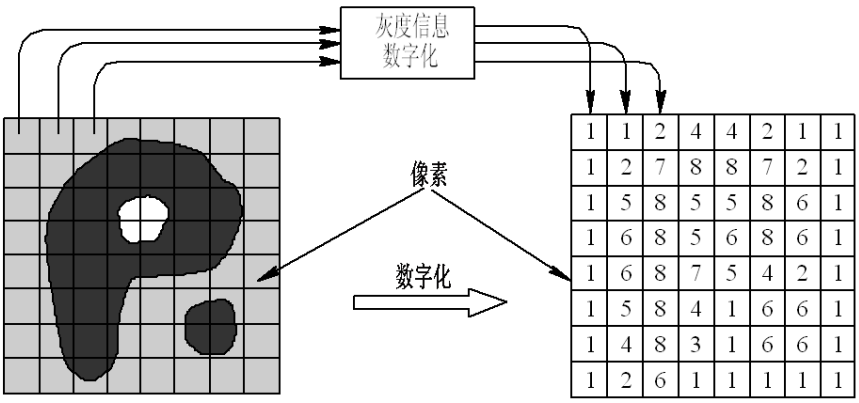


图 1 图像的采样和量化

在 python 中，一幅图像可能包含一个数据矩阵，也可能有一个颜色映射表矩阵。opencv 支持四种图像类型，其区别在于数据矩阵元素的不同含义。它们是:

- 亮度图像（Intensity images）
- 二值图像（Binary images）
- 索引图像（Indexed images）
- RGB 图像（RGB images）

(1)亮度图像

也称灰度图像。一幅亮度图像是一个数据矩阵，其归一化的取值表示亮度。若亮度图像的像素都是 uint8 类或 uint16 类，则它们的整数值范围分别是[0，255]和[0，65536]。若图像是 double 类，则像素取值就是浮点数。规定双精度型归一化亮度图像的取值范围是[0，1]。

(2)二值图像

二值图像是指在图像中，每个像素的灰度等级只有两种。即全黑或者全白，在 python 种，一幅二值图像是一个取值只有 0 和 255 的 numpy 数组。

(3)索引图像

索引颜色通常也称为映射颜色，在这种模式下，颜色都是预先定义的，并且可供选用的一组颜色也很有限，索引颜色的图像最多只能显示 256 种颜色。

一幅索引颜色图像在图像文件里定义，当打开该文件时，构成该图像具体颜色的索引值就被读入程序里，然后根据索引值找到最终的颜色。

(4) RGB 图像

一幅 RGB 图像就是彩色像素的一个 M×N×3 数组，其中每一个彩色相似点都是在特定空间位置的彩色图像相对应的红、绿、蓝三个分量。按照惯例，形成一幅 RGB 彩色图像的三个图像常称为红、绿或蓝分量图像。

值得注意的是，在 OpenCV 中，加载图像通道的顺序是 BGR（实验五中您将用到此特性）。但是 Matplotlib（python 的一个绘图库）以 RGB 模式显示。因此，如果使用 OpenCV 读取彩色图像，则 Matplotlib 中将无法正确显示彩色图像。

2.OpenCV 图像文件格式

OpenCV 支持处理以下几种图像文件格式：

（1）PCX（Windows Paintbrush）格式。可处理 1，4，8，16，24 位等图像数据。文件内容包括：文件头（128 字节），图像数据、扩展颜色映射表数据。

（2）BMP（Windows Bitmap）格式。有 1，4，8，24 位非压缩图像，8 位 RLE（Run-length Encoded）图像。文件内容包括：文件头（一个 BITMAP FILEHEADER 数据结构），位图信息数据块（位图信息头 BITMAP INFOHEADER 和一个颜色表）和图像数据。

（3）HDF（Hierarchical Data Format）格式。有 8 位，24 位光栅数据集。

（4）JPEG(Joint Photographic Experts Group)格式，是一种成为联合图像专家组的图像压缩格式。

（5）TIFF（Tagged Image File Format）格式。处理 1，4，8，24 位非压缩图像，1，4，8，24 位 packbit 压缩图像，一位 CCITT 压缩图像等。文件内容包括：文件头，参数指针表与参数域，参数数据表和图像数据四部分。

（6）XWD(X Windows Dump)格式。1，8 位 Zpixmaps,XYbitmaps,1 位 XYpixmaps。

（7）PNG（Portable Network Graphics）格式。

四、实验内容

（1）利用 OpenCV 读取一幅彩色图像，并读取图像的基本信息；

（2）利用 OpenCV 显示图像；

（3）对彩色图像进行灰度化处理；

（4）对灰度图像进行二值化处理；

（5）对图像进行几何变换（缩放，平移，翻转）：

（6）储存处理后的图像

五、实验报告要求

（1） 给出使用 opencv-python 进行图像读取、显示、翻转、裁剪，存储的完整代码。
（2） 写出实验的心得与体会。

六、预习要求

（1）了解 python 基本语法以及图像处理 API--OpenCV。
（2）了解 opencv 图像基础操作函数。

七、实验指导

1.下面给出示例程序

```
import cv2 as cv              #引入 OpenCV 库
img = cv.imread('1.jpg')      #使用 imread 函数读取图像，并以 numpy 数组形式储存
print(img.shape)      #查看图像的大小。返回的元组（touple）中的三个数依次表示高度、宽度和通道数
print(img.dtype)      #查看图片的类型
cv.imshow('img',img)      #使用imshow 函数显示图像，第一个参数是窗口名称（可不写），第二个参数是要显示的图像的名称，一定要写
cv.waitKey(0)     #可以让窗口一直显示图像直到按下任意按键
img_GRAY = cv.cvtColor(img,cv.COLOR_BGR2GRAY)     #使用 cv.cvtColor 函数转换色彩空间，参数‘cv.COLOR_BGR2GRAY’表示从 RGB 空间转换到灰度空间
cv.imshow('gray',img_GRAY)
cv.waitKey(0)
ret,thresh = cv.threshold(img_GRAY,127,255,cv.THRESH_BINARY)      #使用 cv.threshold 函数进行图像阈值处理，参数‘cv.THRESH_BINARY’代表了阈值的类型，127 为阈值
cv.imshow('threshold',thresh)
cv.waitKey(0)
res = cv.resize(img,None,fx=2,fy=2,interpolation=cv.INTER_CUBIC)      #使用 cv.resize 函数进行图像缩放
cv.imshow('resize',res)
cv.waitKey(0)
cv.imwrite('result.jpg',res)      #保存图像
```

八、部分实验结果

（1）原图像:

（2）灰度图像:

（3）二值图像:

（4）放大两倍后的图像:


resize

实验二 数字图像增强实验

一、实验目的

1.熟悉并学会 opencv-python 中图像增强的相关函数;
2.了解图像增强的方法、去噪的方法和效果。

二、实验主要仪器设备

(1)计算机;
(2)python 软件;
(3)典型的灰度、彩色图像文件。

三、实验原理

图像增强是指按特定的需要突出一幅图像中的某些信息，同时，消弱或去除某些不需要的信息的处理方法。其主要目的是处理后的图像对某些特定的应用比原来的图像更加有效。图像增强技术主要有直方图修改处理、图像平滑化处理、图像尖锐化处理和彩色处理技术等。本实验以直方图均衡化增强图像对比度的方法为主要内容,其他方法同学们可以在课后自行练习。

1.直方图

直方图是多种空间域处理技术的基础。直方图操作能有效地用于图像增强。除了提供有用的图像统计资料外，直方图固有的信息在其他图像处理应用中也是非常有用的，如图像压缩与分割。直方图在软件中易于计算，也适用于商用硬件设备，因此，它们成为了实时图像处理的一个流行工具。

直方图是图像的最基本的统计特征，它反映的是图像的灰度值的分布情况。直方图均衡化的目的是使图像在整个灰度值动态变化范围内的分布均匀化，改善图像的亮度分布状态，增强图像的视觉效果。灰度直方图是图像预处理中涉及最广泛的基本概念之一。

图像的直方图事实上就是图像的亮度分布的概率密度函数，是一幅图像的所有象素集合的最基本的统计规律。直方图反映了图像的明暗分布规律，可以通过图像变换进行直方图调整，获得较好的视觉效果。

直方图均衡化是通过灰度变换将一幅图像转换为另一幅具有均衡直方图，即在每个灰度级上都具有相同的象素点数的过程。

2.图像锐化

图像锐化(image sharpening)是补偿图像的轮廓，增强图像的边缘及灰度跳变的部分，使图像变得清晰，分为空域处理和频域处理两类。

3.图像平滑

图像平滑是对图像作低通滤波，可在空间域或频率域实现。

四、实验内容

(1) 绘制灰度图像直方图;

（2）对直方图均衡化；

（3）利用模版进行空域滤波；

（4）分别利用常见低通（平滑）滤波器与高通（锐化）滤波器进行频域滤波（滤波器公式见实验指导）。

五、实验报告要求

(1)说明利用 opencv-python 图像处理工具包实现灰度修正、图像平滑、锐化的方法；
(2)列出上述图像处理的程序；
(3)记录灰度修正、图像平滑、图像锐化的图像，回答思考题；
(4)心得和体会。

六、预习要求

(1)了解 opencv-python 图像处理包关于图像增强的有关功能；
(2)列出上述图像处理的流程。

七、思考题

(1)如何针对图像过暗、过亮、对比度不足设计灰度变换函数?
(2)比较同一种去噪方法对不同噪声处理的效果。
(3)讨论梯度法锐度图像的 4 种不同方法的应用范围。

八、实验指导

1.绘制灰度直方图

OpenCV 利用 calcHist() 函数来绘制直方图，calcHist()函数原型为：

hist = cv2.calcHist(img, channels, mask, histSize, ranges, accumulate)

参数	描述
hist	直方图，返回的是一个二维数组；
img	原始图像；
channels	指定通道，通道编号需要用中括号括起，输入图像是灰度图像时，它的值为[0]，彩色图像则为[0]、[1]、[2]，分别表示 B、G、R；
mask	掩码图像，统计整副图像的直方图，设为 None，统计图像的某一部分直方图时，需要掩码图像；
histSize	BINS 的数量，参数子集的数目，如下图当 bins=3 表示三个灰度级；
ranges	像素值范围，例如[0, 255]；
accumulate	ccumulate 表示累计叠加标识，默认为 false，如果被设置为 true，则直方图在开始分配时不会被清零。

由于 calcHist() 函数返回的是二维数组，可利用 matplotlib 库绘制图像。

部分代码示例如下：

```
import cv2
```

```
import numpy as np
import matplotlib.pyplot as plt
#读取图片
img = cv2.imread('images/buffalos.png',0)
hist = cv2.calcHist([img], [0], None, [256], [0, 255])
plt.plot(hist)
plt.show()
```

2.直方图均衡

OpenCV 中的直方图均衡化函数为 cv2.equalizeHist()。该函数的输入为灰度图像，输出结果为直方图均衡化后的图像。

```
equ = cv2.equalizeHist(img)
```

3.空域滤波(空域平滑或锐化)

（1）平滑滤波

平滑滤波是低频增强的空间域滤波技术。它的目的有两类：一类是模糊；另一类是消除噪音。空间域的平滑滤波一般采用简单平均法进行，就是求邻近像元点的平均亮度值。邻域的大小与平滑的效果直接相关，邻域越大平滑的效果越好，但邻域过大，平滑会使边缘信息损失的越大，从而使输出的图像变得模糊，因此需合理选择邻域的大小。

- **均值滤波**

OpenCV 中均值模板可以用 cv2.blur 和 cv2.boxFilter 函数实现。其基本用法如下：

```
blur = cv2.blur(img, (3, 5))  # 模板大小为 3*5，模板的大小是可以设定的
box = cv2.boxFilter(img, -1, (3, 5))
```

- **高斯模糊滤波**

Opencv 中使用 cv2.GaussianBlur()函数实现高斯模糊。其基本用法如下：

```
blur = cv2.GaussianBlur(img,(5,5),0)  # （5,5）表示的是卷积模板的大小，0 表示的是沿 x 与 y 方向上的标准差
```

- **中值滤波**

Opencv 用 cv2.medianBlur()函数实现中值滤波。其基本用法如下：

```
blur = cv2.medianBlur(img, 5)
```

（2）锐化滤波

（1） Roberts 算子

Roberts 算法简介

Roberts 算法又称为交叉微分算法，它是基于交叉差分的梯度算法，通过局部差分计算检测边缘线条。常用来处理具有陡峭的低噪声图像，当图像边缘接近于正 45 度或负 45 度时，该算法处理效果更理想。

Roberts 算子的模板分为水平方向和垂直方向，如下所示：

$$\begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix} \quad \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$$

OpenCV 中 Roberts 算法库函数使用

在 OpenCV 官方的库中，也提供了 Roberts 算法库函数，调用 OpenCV 的 filter2D()函数实现边缘提取。该函数主要是利用内核实现对图像的卷积运算，然后通过 addWeighted()函数来进行 x 方向与 y 方向上的结合。

- filter2D()函数原型：result=cv2.filter2D(img, cv2.CV_16S, kernel)

参数	描述
img	图像
dst	目标图像，与原图像尺寸和通过数相同
ddepth	目标图像的所需深度
kernel	卷积核（或相当于相关核），单通道浮点矩阵;如果要将不同的内核应用于不同的通道，请使用拆分将图像拆分为单独的颜色平面，然后单独处理它们。

其中 ddepth 表示目标图像的所需深度，它包含有关图像中存储的数据类型的信息，可以是 unsigned char（CV_8U），signed char（CV_8S），unsigned short（CV_16U）等等...

Input depth (src.depth())	Output depth (ddepth)
CV_8U	-1/CV_16S/CV_32F/CV_64F
CV_16U/CV_16S	-1/CV_32F/CV_64F
CV_32F	-1/CV_32F/CV_64F
CV_64F	-1/CV_64F

Note：当 ddepth=-1 时，表示输出图像与原图像有相同的深度。

- addWeighted()函数原型：
  result=cv2.addWeighted(absX,alphax,absY,alphay,num)

参数	描述
absX	X 方向的处理结果
alphax	X 方向上的权重
absY	Y 方向的处理结果
alphay	Y 方向上的权重
num	X 与 Y 方向上求和后添加的偏移量，不能设置太大，否则容易溢出(总和超过 255 就算溢出)

代码示例如下:

```
import cv2
import numpy as np
'''
一般来说，对图像轮廓提取都会经过如下步骤，灰度-滤波去噪-阈值化处理-(形态学处理，前面如果达标，这步骤可以省略)-轮廓提取
'''
#读取图像
img=cv2.imread("images/building.jpg")
#图像高斯滤波去噪
blur=cv2.GaussianBlur(img,(3,3),1,1) #核尺寸通过对图像的调节自行定义
#图像阈值化处理
ret,thresh1=cv2.threshold(blur,127,255,cv2.THRESH_BINARY)  #二进制阈值化
#调用 Roberts 算法的 OpenCV 库函数进行图像轮廓提取
```

```
kernelx = np.array([[-1,0],[0,1]], dtype=int)
kernely = np.array([[0,-1],[1,0]], dtype=int)
x = cv2.filter2D(thresh1, cv2.CV_16S, kernelx)
y = cv2.filter2D(thresh1, cv2.CV_16S, kernely)
#转 uint8
absX = cv2.convertScaleAbs(x)
absY = cv2.convertScaleAbs(y)
Roberts = cv2.addWeighted(absX,0.5,absY,0.5,0)

cv2.imshow("Roberts",Roberts)
cv2.waitKey()
```

（2） Prewitt 算子

Prewitt 算法简介

Prewitt 算子是一种一阶微分算子的边缘检测，利用像素点上下、左右邻点的灰度差，在边缘处达到极值检测边缘，去掉部分伪边缘，对噪声具有平滑作用 。其原理是在图像空间利用两个方向模板与图像进行邻域卷积来完成的，这两个方向模板一个检测水平边缘，一个检测垂直边缘。Prewitt 算法适合用来识别噪声较多、灰度渐变的图像；水平和竖直方向上的卷积模板如下所示:

$$\begin{pmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix} \qquad \begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix}$$

OpenCV 中 Prewitt 算法库函数使用

OpenCV 官方同样对 Prewitt 算法有对应的库函数,与 Roberts 算法算法的库函数一样，只是传递的卷积核有变化；通过 Numpy 定义模板，再调用 OpenCV 的 filter2D()函数实现对图像的卷积运算，最终通过 convertScaleAbs()和 addWeighted()函数实现边缘提取，函数原型参考 Roberts 算法介绍的函数原型。

部分代码示例如下:

```
#调用 Prewitt 算法的 OpenCV 库函数进行图像轮廓提取
kernelx = np.array([[1,1,1],[0,0,0],[-1,-1,-1]],dtype=int)
kernely = np.array([[-1,0,1],[-1,0,1],[-1,0,1]],dtype=int)
x = cv2.filter2D(thresh1, -1, kernelx)
y = cv2.filter2D(thresh1, -1, kernely)
```

（3） Sobel 算子

Sobel 算法简介

Sobel 算法(索贝尔算子)是一种用于边缘检测的离散微分算子，它结合了高斯平滑和微分求导。该算子用于计算图像明暗程度近似值，根据图像边缘旁边明暗程度把该区域内超过某个数的特定点记为边缘。Sobel 算子在 Prewitt 算子的基础上增加了权重的概念，认为相邻点的距离远近对当前像素点的影响是不同的，距离越近的像素点对应当前像素的影响越大，从而实现图像锐化并突出边缘轮廓。当对精度要求不是很高时，Sobel 算子是一种较为常用的边缘检测方法，其卷积模板如下所示:

$$\begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix} \quad \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$$

OpenCV 中 Roberts 算法库函数使用

OpenCV 中提供了专门的 Soble 算法的库函数，函数原型：

result=cv2.Sobel(img, ddepth, dx, dy, ksize)

参数	描述
img	需要轮廓提取的图像
ddepth	目标图像深度
dx	x 方向上的差分阶数，取值 1 或 0
dy	y 方向上的差分阶数，取值 1 或 0
ksize	Sobel 算子的大小，其值必须是正数和奇数

部分代码示例如下：

```
#调用 Sobel 算法的 OpenCV 库函数进行图像轮廓提取
x = cv2.Sobel(thresh1, cv2.CV_16S, 1, 0) #对 x 求一阶导
y = cv2.Sobel(thresh1, cv2.CV_16S, 0, 1) #对 y 求一阶导
absX = cv2.convertScaleAbs(x)   #对 x 取绝对值，并将图像转换为 8 位图
absY = cv2.convertScaleAbs(y)    #对 y 取绝对值，并将图像转换为 8 位图
Sobel = cv2.addWeighted(absX, 0.5, absY, 0.5, 0)
```

4、频域滤波(频域平滑或锐化)

（1）常见低通（平滑）滤波器

理想：$\mathrm{H}(u,v)=\begin{cases}1, D(u,v)\le D_0\\ 0, D(u,v)\ge D_0\end{cases}$

巴特沃斯：$\mathrm{H}(\mathrm{u},\mathrm{v})=\frac{1}{1+[D(u,v)/D_0]^{2n}}$

高斯：$H(u,v)=e^{-D^2(u,v)/2D_0^2}$

其中，D(u,v)为点（u,v）到滤波器中心的距离，D0 是截至频率，n 是巴特沃斯滤波器的阶。

（2）常见高通（锐化）滤波器

理想：$\mathrm{H}(u,v)=\begin{cases}0, D(u,v)\le D_0\\ 1, D(u,v)\ge D_0\end{cases}$

巴特沃斯：$$H(u,v)=\frac{1}{1+[D_0/D(u,v)]^{2n}}$$

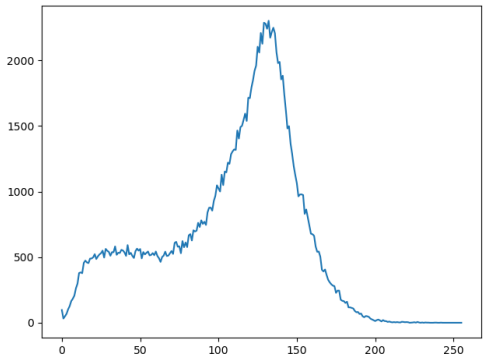
高斯：$H(u,v)=1-e^{-D^2(u,v)/2D_0^2}$

其中，D(u,v)为点（u,v）到滤波器中心的距离，D0 是截至频率，n 是巴特沃斯滤波器的阶。

九、实验结果

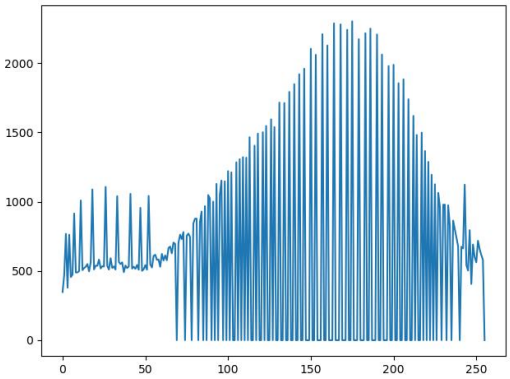
(1)直方图处理:

原图像



灰度直方图

直方图均衡后的图像



直方图均衡后的灰度直方图

(2)平滑与锐化

原始图像

Sobel 算子

实验三 形态学图像处理实验

一、实验目的

1.使用形态学滤波对图像进行腐蚀、膨胀运算;
2.使用形态学滤波对图像进行开闭运算;
3.利用 opencv-python 对图像进行形态学运算。

二、实验主要仪器设备

(1)计算机;
(2)python 软件;
(3)典型的灰度、彩色图像文件。

三、实验原理

数学形态学是以形态结构元素为基础对图像进行分析的数学工具。它的基本思想是用具有一定形态的结构元素去度量和提取图像中的对应形状以达到对图像分析和识别的目的。数学形态学的应用可以简化图像数据，保持它们基本的形状特征，并除去不相干的结构。数学形态学的基本运算有 4 个：膨胀、腐蚀、开启和闭合。它们在二值图像中和灰度图像中各有特点。基于这些基本运算还可以推导和组合成各种数学形态学实用算法。

基本的形态运算是腐蚀和膨胀。

在形态学中，结构元素是最重要最基本的概念。结构元素在形态变换中的作用相当于信号处理中的"滤波窗口"。用 B（x）代表结构元素，对工作空间 E 中的每一点 x，腐蚀和膨胀的定义为：

腐蚀：$X = E \odot B(x)$;

膨胀：$Y = E \otimes B(y)$。

用 B（x）对 E 进行膨胀的结果就是把结构元素 B 平移后使 B 与 E 的交集非空的点构成的集合。先腐蚀后膨胀的过程称为开运算。它具有消除细小物体，在纤细处分离物体和平滑较大物体边界的作用。先膨胀后腐蚀的过程称为闭运算。它具有填充物体内细小空洞，连接邻近物体和平滑边界的作用。

可见，二值形态膨胀与腐蚀可转化为集合的逻辑运算，算法简单，适于并行处理，且易于硬件实现，适于对二值图像进行图像分割、细化、抽取骨架、边缘提取、形状分析。但是，在不同的应用场合，结构元素的选择及其相应的处理算法是不一样的，对不同的目标图像需设计不同的结构元素和不同的处理算法。结构元素的大小、形状选择合适与否，将直接影响图像的形态运算结果。因此，很多学者结合自己的应用实际，提出了一系列的改进算法。如梁勇提出的用多方位形态学结构元素进行边缘检测算法既具有较好的边缘定位能力，又具有很好的噪声平滑能力。许超提出的以最短线段结构元素构造准圆结构元素或序列结构元素生成准圆结构元素相结合的设计方法，用于骨架的提取，可大大减少形态运算的计算量，并可同时满足尺度、平移及旋转相容性，适于对形状进行分析和描述。

四、实验内容

(1)对原始图像进行灰度化、二值化处理;

(2)对所得二值图像进行腐蚀运算;
(3)对二值图像进行膨胀运算;
(4)对二值图像进行开运算;
(5)对二值图像进行闭运算。

五、实验报告要求

(1)说明对图像进行形态学运算的理论方法;
(2)列出上述图像处理程序;
(3)记录结果图像;
(4)心得和体会。

六、预习要求

(1)了解图像形态学运算的方法;
(2)列出上述图像处理方法的流程。

七、实验指导

（1） 腐蚀：

腐蚀主要就是调用 cv2.erode(img,kernel,iterations)，其中第一个参数：img 指需要腐蚀的图；第二个参数：kernel 指腐蚀操作的内核，默认是一个简单的 3X3 矩阵；第三个参数：iterations 指的是腐蚀次数，省略是默认为 1。

使用 3X3 的卷积核，对二值图进行腐蚀操作的示例如下：

```
import cv2
import numpy as np

img = cv2.imread("./images/binary.jpg",0)
kernel = np.ones((3,3),np.uint8)
erosion = cv2.erode(img,kernel)
cv2.imshow('eroded image',erosion)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

（2） 膨胀：

```
dilation = cv2.dilate(img,kernel,iterations = 1)
```

（3） 开运算：先腐蚀再膨胀

openvc 中 morphologyEx()函数是一种形态学变化函数。开运算是调用 cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)，其中第一个参数：img 指需要开运算的图；第二个参数：指的是开运算；第三个参数：kernel 指开运算的内核。

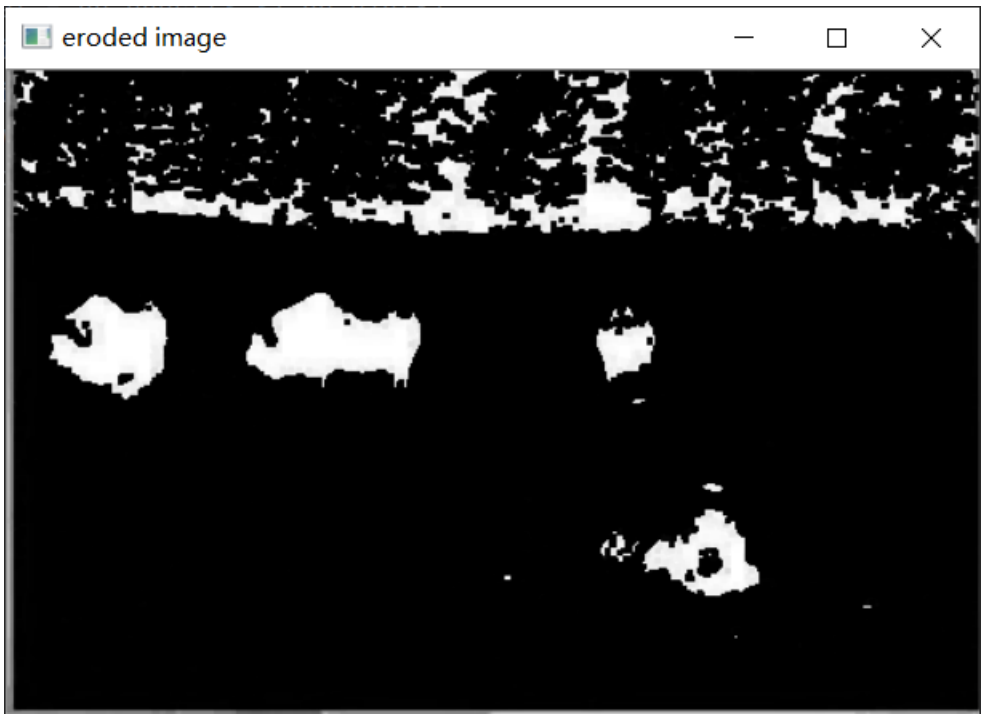
（4） 闭运算：先膨胀再腐蚀

```
closing = cv2.morphologyEx(img, cv2.MORPH_CLOSE, kernel)
```

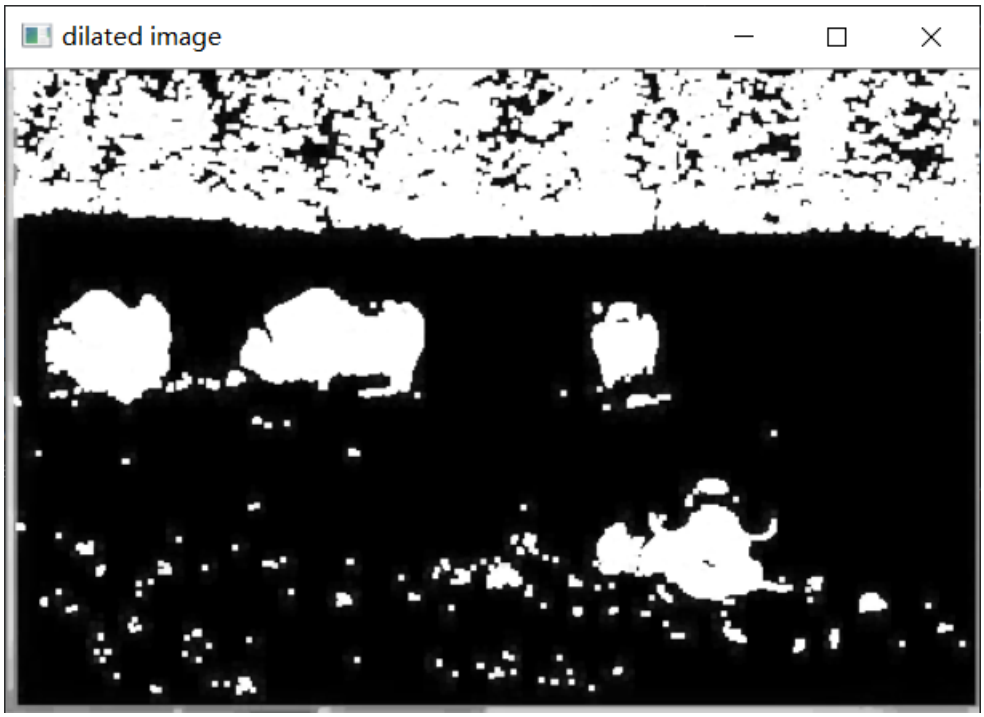
八、部分实验结果

输入的二值图像:

腐蚀图像:



膨胀图像:



实验四 数字图像的边缘检测实验

一、实验目的

1、掌握数字图像的空间域滤波原理;
2、掌握数字图像的边缘检测原理及常用的边缘检测算子;
3、掌握阈值分割及图像的二值化。

二、实验主要仪器设备

(1)计算机;
(2)python 软件;
(3)典型的灰度、彩色图像文件。

三、实验原理

(1) 图像空间滤波原理: 空间滤波是一种采用滤波处理的影像增强方法。其理论基础是空间卷积和空间相关。目的是改善影像质量,包括去除高频噪声与干扰,及影像边缘增强、线性增强以及去模糊等。分为低通滤波（平滑化)、高通滤波（锐化）和带通滤波。

(2) 边缘检测原理: 边缘检测是图像处理和计算机视觉中的基本问题, 边缘检测的目的是标识数字图像中亮度变化明显的点。图像属性中的显著变化通常反映了属性的重要事件和变化。这些包括(i)深度上的不连续、(ii)表面方向不连续、(iii)物质属性变化和(iv)场景照明变化。边缘检测是图像处理和计算机视觉中, 尤其是特征提取中的一个研究领域。

四、实验内容

(1) 读取图像;
(2) 使用 opencv-python 对六种边缘检测器进行图片边缘检测, 并对三种算子得出的结果进行比较。

五、实验报告要求

(1)熟悉边缘检测的原理;
(2)列出实验程序及结果, 并比较各边缘提取算子的特点;
(3)写出心得、体会。

六、预习要求

(1) 了解边缘检测原理;
(2) 熟悉边缘检测程序流程。

七、实验指导

(1) LoG 检测器

实现 LOG 算法要对图像先进行高斯滤波降噪处理，然后通过 Laplacian 算法进行轮廓提取。

OpenCV 官方将 Laplacian 算法封装到 cv2.Laplacian()函数中，函数原型:

result=cv2.Laplacian(img, ddepth, ksize)

参数	描述
img	需要进行 Laplacian 算法的图像
ddepth	目标图像深度
ksize	二阶导数的滤波器的孔径大小，其值必须是正数和奇数，且默认值为 1(4 邻域模板)

部分代码示例如下:

```
#图像高斯滤波去噪
blur=cv2.GaussianBlur(gray,(3,3),1,1)#核尺寸通过对图像的调节自行定义
#调用 Laplacian 算法的 OpenCV 库函数进行图像轮廓提取
result = cv2.Laplacian(blur,cv2.CV_16S,ksize=1)
LOG = cv2.convertScaleAbs(result)#得到 LOG 算法处理结果
```

(2) Scharr 算子

Scharr 算子又称为 Scharr 滤波器，也是计算 x 或 y 方向上的图像差分，在 OpenCV 中主要是配合 Sobel 算子的运算而存在的。scharr 算子的卷积模板如下所示:

$$dx = \begin{bmatrix} -3 & 0 & 3 \\ -10 & 0 & 10 \\ -3 & 0 & 3 \end{bmatrix}, dy = \begin{bmatrix} -3 & -10 & -3 \\ 0 & 0 & 0 \\ 3 & 10 & 3 \end{bmatrix}$$

OpenCV 官方给出了明确的 Scharr 算法的库函数，函数原型:

result=cv2.Scharr(img, ddepth, dx, dy)

参数	描述
img	需要轮廓提取的图像
ddepth	目标图像深度
dx	dx 方向上的差分阶数，取值 1 或 0
dy	y 方向上的差分阶数，取值 1 或 0

Scharr 算子的代码基本实现类似于 Sobel 算子。

(3) Canny 边缘检测器

Canny 边缘检测器是函数 edge 中最强大的边缘检测器。方法总结如下:

1.使用具有指定标准差 δ 的一个高斯滤波器来平滑图像，以减少噪声。

2.在每个点处计算局部梯度$\left[g_x^2+g_y^2\right]^{1/2}$和边缘方向$\arctan(g_x/g_y)$。边缘点定义为梯度方向强度局部最大的点。

3.步骤 2 中确定的边缘点产生梯度中的脊线。然后，算法沿这些脊线的顶部进行追踪，并将实际上不在脊线顶部的像素设置为零，从而在输出中给出一条细线，该过程称为非最大值抑制。然后使用称为滞后阈值处理的方法来对这些脊线像素进行阈值处理，这一处理方法

使用两个阈值T_1和T_2其中$T_1 < T_2$。其值大于T_2的脊线像素称为“强”边缘像素，值在T_1和T_2之间的脊线像素称为“弱”边缘像素。

在 OpenCV 中，Canny()库函数原型为：

Canny = cv2.Canny(img, threshold1, threshold2, apertureSize)

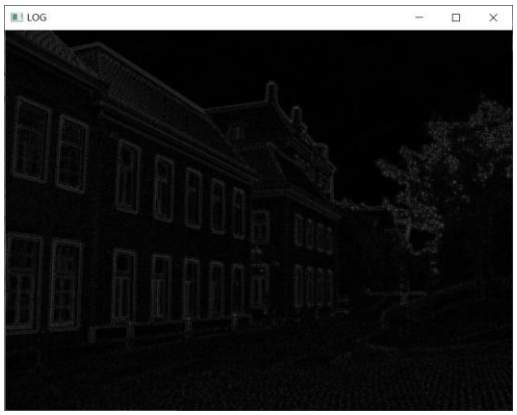
参数	描述
img	需要轮廓提取的图像
threshold1	第一个滞后性阈值
threshold2	第二个滞后性阈值
apertureSize	Sobel 算法的孔径大小，其默认值为 3

部分代码示例如下

```
#图像高斯滤波去噪
blur=cv2.GaussianBlur(gray,(7,7),1,1)
#Canny 算子进行边缘提取
Canny = cv2.Canny(blur, 50, 150)
```

八、部分实验结果

原始图像



Laplacian-Gauss 算子

实验五 基于图像分割的车牌定位识别

一、实验目的

1.掌握车牌阈值分割;
2.掌握基于形态学计算的图像分割;
3.掌握图像的二值化;
4.掌握基于像素投影的字符分割;
5.掌握字符识别原理。

二、实验主要仪器设备

(1)计算机;
(2)python 3.x;
(3)需进行车牌识别的图片。

三、实验原理

（1）图像灰度化

灰度数字图像是每个像素只有一个采样颜色的图像。这类图像通常显示为从最暗黑色到最亮的白色的灰度，尽管理论上这个采样可以任何颜色的不同深浅，甚至可以是不同亮度上的不同颜色。灰度图像与黑白图像不同，在计算机图像领域中黑白图像只有黑白两种颜色，灰度图像在黑色与白色之间还有许多级的颜色深度。

（2）图像二值化

图像二值化就是将图像上的像素点的灰度值设置为 0 或 255，也就是将整个图像呈现出明显的黑白效果。

（3）图像形态学运算

(见实验三)

（4）阈值分割原理

阈值分割算法是图形分割中应用场景最多的算法之一。简单地说，对灰度图像进行阈值分割就是先确定一个处于图像灰度取值范围内的阈值，然后将图像中各个像素的灰度值与这个阈值比较，并根据比较的结果将对应的像素划分为两类：像素灰度大于阈值的一类和像素值小于阈值的另一类，灰度值等于阈值的像素可以归入这两类之一。分割后的两类像素一般分属图像的两个不同区域，所以对像素根据阈值分类达到了区域分割的目的。

（5）字符分割原理

二值化后的图像,在没有字符的区域，y 方向上像素灰度和为 0，在有字符的区域为灰度和非 0。

四、实验内容

（1）定位车牌区域(可基于灰度阈值或形态学运算等)；
（2）车牌图像预处理(灰度化、二值化)；
（3）字符分割(可基于灰度垂直投影等)；
（4）字符识别(可使用模版匹配等方法)。

五、实验报告

（1）说明车牌区域识别、车牌图像预处理、字符分割和识别的原理；
（2）列出实验程序；
（3）记录实验过程中每一步的图像；
（4）心得和体会。

六、预习要求

（1）总结和掌握车牌识别的基本流程和常见方法；
（2）熟悉 opencv 图像基本处理的相关函数。

七、实验指导

程序流程：

（1）车牌定位

按照下面给出的阈值遍历图片，选取适当区域进行分割。遍历图像可利用 for 循环遍历图片上所有点，遍历方法为：

```
for i in range(h)
    for j in range(w)
        Rij=I[i,j,2];
        Gij=I[i,j,1];
        Bij=I[i,j,0];
```

其中 I 是 h*w 的 numpy 矩阵，Rij、Gij、Bij 分别为(i,j)点像素的 R、G、B 值，将三个值与下方给出的阈值比较，可得出像素是否属于车牌区域。

定位车牌区域时可以分别从行和列的角度进行遍历，即若某行符合要求的像素点数量大于等于某阈值时则认为该行属于车牌区域；遍历列时亦然，即若某列符合要求的像素点数量大于等于某阈值时则认为该列属于车牌区域。

车牌分割参考阈值：

RGB 图像可使用阈值：

- 若 Rij、Gij、Bi 分别为(i,j)点的 RGB 值，则
  Rij/Bij<0.35, Gij/Bij<0.9, Bij>90 或 Gij/Bi< 0.35, Rij/Bij<0.9, Bi<90；

- 也可将 RGB 图像转化为 HSV 图像进行阈值比较，HSV 图像可使用阈值：

H：190~245, S：0.35~1, V：0.3~1；

提示：使用 numpy 切片特性可以简化代码

（2）分割区域灰度化、二值化

将（1）中获得的图像进行处理方便下一步分割。可能用到的 opencv 函数：

图像灰度化函数：cv.cvtColor(img,cv.COLOR_BGR2GRAY)，将 RGB 图像 img 转化为

灰度图像;

图像二值化函数：cv.threshold(img, 0, 255, cv.THRESH_BINARY + cv.THRESH_OTSU) , 将灰度图像 img 转化为二值图像(otsu 二值化);

（3）车牌分割

二值化后的图像,在没有字符的区域，y 方向上像素灰度和为 0，在有字符的区域为灰度和非 0，因此可根据灰度值在纵轴的投影对车牌二值图像进行分割。利用 numpy 数组的切片性质可以轻松实现车牌分割

（4）车牌识别

本实验依照模版匹配进行识别, 将分割结果 I 分别与模版 I'进行比对, 得出其差值$|I - I'|$, 则所得差值最小的模版即为识别结果。其中 I 为分割后的字符图像，I'为模版图像。
由于 opencv 中图像就是以 numpy 数组形式存储的，所以$|I - I'|$相当于直接将两矩阵相减取绝对值即可，取绝对值的函数为 numpy.abs( )。

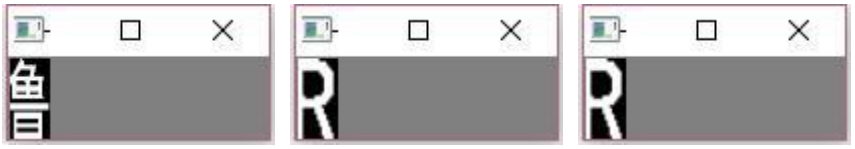
八、部分实验结果

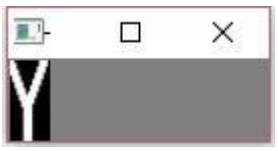
原图像:

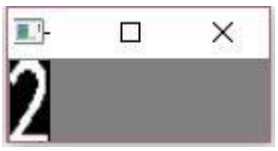
车牌分割结果:

灰度化、二值化、去边框结果:

字符分割结果:





模板匹配结果:

```
Python 3.7.7 (default, Mar 23 2020, 23:19:08) [MSC v.1916 64 bit (AMD64)] on win32
In[2]: runfile('D:/A Personal/LB/Projects/DFPexperiment/exp5.2.py', wdir='D:/A Personal/LB/Projects/DFPexperiment')
[4, 33, 51, 65, 79, 94, 108] [29, 45, 63, 77, 91, 106, 120]
鲁RRY902
```

实验六 基于彩色图像特征融合的水果分割识别

一、实验目的

实现对所给图像中的水果（水蜜桃、苹果、梨、香蕉、芒果、荔枝、草莓）的分割识别，主要包括以下几点：

（1）掌握图像的二值化；

（2）掌握图像的标签化；

（3）掌握图像的分割技术；

（4）掌握图像的特征提取、特征参数的计算；

（5）掌握图像的掩膜操作。

二、实验内容

（1）水果图像预处理(去噪、灰度化、二值化)；

（3）图像的标签化；

（4）图像特征提取、特征参数的计算**（至少使用三种特征参数）**；

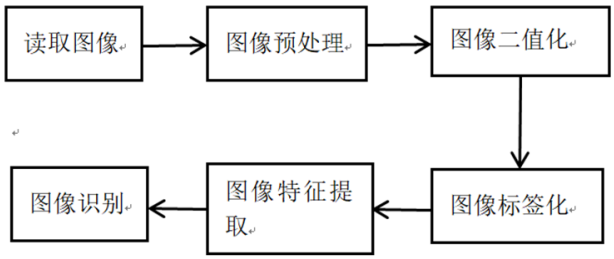
（5）水果图像的分割识别。

三、实验原理

数字图像处理的最终目的是用计算机代替人去认识图像以及找出一幅图像中人们感兴趣的目标。数字图像识别的主要方法可分为统计模式识别、结构模式识别、模糊模式识别与智能模式识别。为了对图像进行识别，首先要进行图像处理，并且有时候处理和识别是同时进行的。一般来说，图像处理包括图像编码、图像增强、图像压缩、图像复原、图像分割等内容。对图像处理环节来说，输入是图像，输出也是图像；而图像识别以研究图像的分类与描述为主要内容，找出图像各个部分的形状、颜色以及纹理特征，亦即特征提取，并对图像作结构上的分析。因此，对图像识别环节来说，输入时图像，输出是类别和图像的结构分析。

四、实验指导

参考的步骤如下：



1.图像的二值化

经过去噪的图像，我们就可以对其进行二值化处理。二值即 0 和 1,其中 0 代表黑色，1 代表白色。设置一个阈值 T，然后根据这个阈值可将原灰度图转换成只有 0 和 1 的两种图像。阈值 T 通过多次实验取得，当原像素值大于等于 T 时，设置这个像素为纯白色，反之为纯黑色。

2.图像边缘检测处理

在拍摄的水果原图中由于光照及水果自身特点的不同，使得图像二值化后达不到图像处理的要求，比如二值图像中水果体中会出现一些影响之后标记连通闭区域的空洞，边缘处还存在一些断裂情况，需要对图像进行边缘检测从而确定目标水果的边界范围，然后再基于数学形态学算子进行去除断边、图像填充等必要的后续处理。

Opencv 中的 connectedComponents 函数可以用于计算二值图像中的连通区域，使用方法如下：

[num,L] = cv.connectedComponents (image, labels=None, connectivity=None, ltype=None)

num：最大的连通数量

L：标记图，标记每个连通像素属于第几个连通区域

Image： 输入图像，二值图

Connectivity：连接类型，四或者八

其他参数可以忽略

除了 connectedComponents 函数外，还有其他函数可供选择，此处仅举例说明。其余相关内容可参考《数字图像处理》第九章内容）

3.图像的标签化

为了能够把每个物体都相互区分开，就要检查有关像素是连接着的还是分离开的，这种处理称为标签化。所谓的图像的标签化，是指对图像中互相连通的所有像素赋予同样的标号，而对于不同的连接成分则给予不同的标号处理的过程。经过标签化处理就能把各个连接成分进行分离，从而研究他们各自的特征。

4.图像特征提取、特征参数计算

本实验中可以用于分类识别的图像特征颜色特征、形状特征和纹理特征等。

颜色特征是一种全局特征，描述了图像区域所对应的景物的表面性质。颜色特征的描述是建立在颜色空间的基础上并反映各个像素点的信息。当前较常见的颜色空间主要有：RGB、HSI、HSV、Lab、YIQ、YCgCr、YcbCr、CIE 等。

形状特征是一种局部特征，其对象是目标区域内像素点的数目及位置。常用的形状特征有：面积、周长、直径、曲率、圆形度、伸长度、离散度、圆度、圆方差等。

在图像识别的研究中，纹理一般是指图像像素灰度呈现出的空间分布特性，纹理特征则是从图像中计算出一个对这一特性进行量化描述的值。纹理对区域内的像素点进行统计计算，其中包含有大量微观和宏观信息。目前，常用的纹理特征提取方法有矩分析法、基于灰度共生矩阵方法、基于小波变换方法等。不同的提取方法可以获得不同的纹理特征参数。例如，通过矩分析法获得的均值、方差、峰值、扭曲度、熵等参数；通过灰度共生矩阵方法定义的纹理特征主要有对比度、能量、逆差分矩、距离中心趋势、相关性等。

此处仅以香蕉的识别为例。本实验中，香蕉和芒果虽然颜色相似，但它们的形状差别很大，可以通过计算弧度来区分彼此。弧度是形状特征中常用的特征参数，其定义如下：

$$C \overset{\text{def}}{=} 4\pi \frac{S}{L^2}$$

式中，S 为闭区域的面积，L 为闭区域的周长。

五、实验报告

1) 说明图像预处理方法、图像特征提取、水果目标分割和识别的原理；

2) 列出实验程序；

3) 记录实验过程中每一步的图像；

4) 心得和体会。

实验七 复杂场景的车牌识别实验

一、 实验目的

1）掌握图像增强预处理方法和车牌阈值分割；

2）掌握基于形态学计算的图像分割；

3）掌握图像的二值化；

4）掌握基于像素投影的字符分割；

5）掌握字符识别原理。

二、 实验主要仪器设备

1）计算机；

2）MATLAB 软件；

3）需进行车牌识别的复杂场景车辆图片（见图片 7-1 至 7-3）。

三、 实验内容

（1）复杂场景图像预处理（此步骤根据需要选择）

（2）定位车牌区域(可基于灰度阈值或形态学运算等)；

（3）车牌图像预处理(灰度化、二值化、去边界)；

（4）字符分割(可基于灰度垂直投影等)；

（5）字符识别(可使用模版匹配等方法)。

四、 实验原理

(1) 图像增强预处理和灰度化

针对复杂场景图像进行图像增强预处理。灰度数字图像是每个像素只有一个采样颜色的图像。这类图像通常显示为从最暗黑色到最亮的白色的灰度，尽管理论上这个采样可以任何颜色的不同深浅，甚至可以是不同亮度上的不同颜色。灰度图像与黑白图像不同，在计算机图像领域中黑白图像只有黑白两种颜色，灰度图像在黑色与白色之间还有许多级的颜色深度。

（2）图像二值化

图像二值化就是将图像上的像素点的灰度值设置为 0 或 255，也就是将整个图像呈现出明显的黑白效果。

（3）图像形态学运算

参考实验三对应实验内容。

（4）阈值分割原理

阈值分割算法是图形分割中应用场景最多的算法之一。简单地说，对灰度图像进行阈值分割就是先确定一个处于图像灰度取值范围内的阈值，然后将图像中各个像素的灰度值与这个阈值比较，并根据比较的结果将对应的像素划分为两类：像素灰度大于阈值的一类和像素值小于阈值的另一类，灰度值等于阈值的像素可以归入这两类之一。分割后的两类像素一般分属图像的两个不同区域，所以对像素根据阈值分类达到了区域分割的目的。

（5）字符分割原理

二值化后的图像,在没有字符的区域，y 方向上像素灰度和为 0，在有字符的区域为灰度和非 0。

五、 实验指导

参考实验二和实验五的实验指导内容

六、 实验报告

1) 说明图像预处理方法、复杂场景下车牌区域检测、车牌图像预处理、字符分割和识别的原理；

2) 列出实验程序；

3) 记录实验过程中每一步的图像；

4) 心得和体会。

实验八 基于图像特征的道路提取

一、实验目的

实现对所给的图像中的道路提取，主要包括以下几点：

（1）掌握图像的特征提取、形状特征分类识别；

（2）掌握图像的分割技术。

二、实验内容

（1）图像特征提取、形状特征分类识别；

（2）道路图像分割；

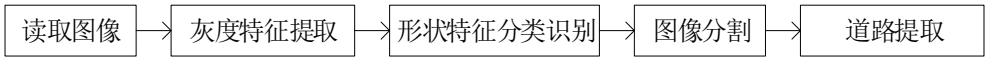
（3）道路提取。

三、实验原理

道路是地图绘制、路径分析和应急处理的重要基础地理数据。道路具有一些光谱、几何和空间特征。道路的光谱特征与混凝土、柏油或沙土等材料的光谱特征有关。道路一般都是平直的线，其局部曲率有一上限，另外，道路很少单独存在，总是相互连接成道路网。基于特征的道路提取的整个处理过程分为灰度特征特征提取、进行形状特征分类识别、图像分割、道路提取四个部分。

四、实验指导

参考步骤如下：



1. 图像特征选择和提取

常用的图像特征有颜色特征、纹理特征、形状特征、空间关系特征。

1）颜色特征

颜色特征是一种全局特征,描述了图像或图像区域所对应的景物的表面性质。一般颜色特征是基于像素点的特征，此时所有属于图像或图像区域的像素都有各自的贡献。由于颜色对图像或图像区域的方向、大小等变化不敏感，所以颜色特征不能很好地捕捉图像中对象的局部特征。另外，仅使用颜色特征查询时，如果数据库很大，常会将许多不需要的图像也检索出来。颜色直方图是最常用的表达颜色特征的方法，其优点是不受图像旋转和平移变化的影响，进一步借助归一化还可不受图像尺度变化的影响，基缺点是没有表达出颜

色空间分布的信息。

常用的颜色特征提取方法有颜色直方图、颜色集等。

2）纹理特征

纹理特征也是一种全局特征，它也描述了图像或图像区域所对应景物的表面性质。但由于纹理只是一种物体表面的特性，并不能完全反映出物体的本质属性， 所以仅仅利用纹理特征是无法获得高层次图像内容的。与颜色特征不同，纹理特征不是基于像素点的特征，它需要在包含多个像素点的区域中进行统计计算。在模式匹配中，这种区域性的特征具有较大的优越性，不会由于局部的偏差而无法匹配成功。作为一种统计特征，纹理特征常具有旋转不变性，并且对于噪声有较强的抵抗能力。但是，纹理特征也有其缺点，一个很明显的缺点是当图像的分辨率变化的时候，所计算出来的纹理可能会有较大偏差。另外，由于有可能受到光照、反射情况 的影响，从 2D 图像中反映出来的纹理不一定是 3D 物体表面真实的纹理。例如，水中的倒影，光滑的金属面互相反射造成的影响等都会导致纹理的变化。由于这些不是物体本身的特性，因而将纹理信息应用于检索时，有时这些虚假的纹理会对检索造成“误导”。

在检索具有粗细、疏密等方面较大差别的纹理图像时，利用纹理特征是一种有效的方法。但当纹理之间的粗细、疏密等易于分辨的信息之间相差不大的时候，通常的纹理特征很难准确地反映出人的视觉感觉不同的纹理之间的差别。

常用的纹理特征提取方法有灰度共生矩阵、Tamura 纹理特征、自回归纹理模型、小波变换等。

3）形状特征

各种基于形状特征的检索方法都可以比较有效地利用图像中感兴趣的目标来进行检索，但它们也有一些共同的问题，包括：①目前基于形状的检索方法还缺乏 比较完善的数学模型；②如果目标有变形时检索结果往往不太可靠；③许多形状特征仅描述了目标局部的性质，要全面描述目标常对计算时间和存储量有较高的要 求；④许多形状特征所反映的目标形状信息与人的直观感觉不完全一致，或者说，特征空间的相似性与人视觉系统感受到的相似性有差别。另外，从 2-D 图像中表现的 3-D 物体实际上只是物体在空间某一平面的投影，从 2-D 图像中反映出来的形状常不是 3-D 物体真实的形状，由于视点的变化，可能会产生各种失真。

几种典型的形状特征提取方法有边界特征法、傅里叶形状描述符法、几何参数法、形状不变矩法等。

4）空间关系特征

所谓空间关系，是指图像中分割出来的多个目标之间的相互的空间位置或相对方向关系，这些关系也可分为连接/邻接关系、交叠/重叠关系和包含/包 容关系等。通常空间位置信息可以分为两类：相对空间位置信息和绝对空间位置信息。前一种关系强调的是目标之间的相对情况，如上下左右关系等，后一种关系强调的是目标之间的距离大小以及方位。显而易见，由绝对空间位置可推出相对空间位置，但表达相对空间位置信息常比较简单。

空间关系特征的使用可加强对图像内容的描述区分能力，但空间关系特征常对图像或目标的旋转、反转、尺度变化等比较敏感。另外，实际应用中，仅仅利用空间信息往往是不够的，不能有效准确地表达场景信息。为了检索，除使用空间关系特征外，还需要其它特征来配合。

提取图像空间关系特征可以有两种方法：一种方法是首先对图像进行自动分割，划分出图像中所包含的对象或颜色区域，然后根据这些区域提取图像特征，并建立索引；另一种方法则简单地将图像均匀地划分为若干规则子块，然后对每个图像子块提取特征，并建立索引。

2.图像分割

传统分割方法一般有基于区域和基于边缘的两大类分割方法。基于边缘特征的道路理解算法一般利用道路轮廓比较规则这一特点，即利用道路的几何特征，从道路图像的边缘图中提取路边信息。然而，这种方法会受许多因素的影响，如路边建筑物阴影的边缘一般也很规则，这就容易和真实的道路边缘混淆，且路边植被的阴影也会使路边的边缘模糊难辩。基于灰度一致性的区域的分割方法则可以将灰度均匀的大目标物体较好地分割开来，形成目标区域。边缘信息由于灰度突变，从而被认为是背景信息。为达到较好的分割效果，处理之前需先进行平滑滤波预处理。

3.道路提取

进行道路信息提取后，道路的主体信息已经被提取出来，再利用形态学等方法在不改变道路整体信息的情况下，对道路信息进一步完善，包括填补空洞和连接中断的道路等。数学形态学的基本思想是利用一定形态的结构元量测和提取图像中对应的形状，以达到对图像分析和识别的目的。它主要用于图像的边缘检测、形态分析、骨架化和图像恢复重建等。本功能使用的形态学方法有开闭运算、细化、裁剪和形态学重建。

五、实验报告

5) 说明图像预处理方法、图像特征提取、道路分割和识别的原理；

6) 列出实验程序；

7) 记录实验过程中每一步的图像；

8) 心得和体会。

实验九 交通道路车道线的提取

一、实验目的

熟练掌握数字图像处理中常用的一些算法，理解道路车道线提取的流程。

（1）掌握图像的特征提取、形状特征分类识别；

（2）掌握图像的分割技术。

（3）掌握边缘检测，霍夫变换等技术。

二、实验内容和方法步骤

1. 加载图片，将其变为由 RGB 表示的数组
2. 对图片进行灰度处理，并去除噪点
3. 通过 opencv 自带函数实现边缘检测（sobel 算子，canny 算子，prewitt 算子）
4. 提取感兴趣的区域
5. 对检测后的图像实施霍夫变换，提取直线段
6. 对检测到的直线进行筛选，分类，再拟合出左右两根车道线
7. 将检测到的车道线加载到原图像中

三、实验报告

（1）说明对霍夫变换出来的直线段，是怎么筛选，拟合成左右直线的？

（2）列出实验程序；

（3）记录实验过程中每一步的图像；

（4）心得和体会。

四、预习要求

（1）总结和掌握霍夫变换原理

（2）熟悉 python opencv 中的相关函数。

三、实验原理和指导

1. 霍夫变换的原理

我们知道直线可以表示为$y = kx + q$，这个是以 x,y 为坐标轴，如图 9.1:

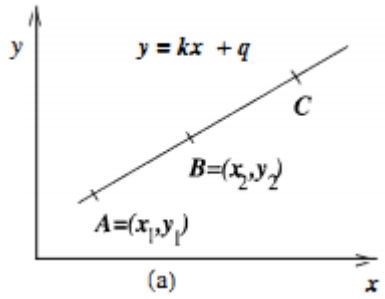


图 9.1

在 x,y 坐标系里有 A，B 两点，分别为（x1,y1）和(x2,y2),现在考虑以 k,q 为坐标轴的坐标系，由$y = kx + q$可知，k,q 的关系也是线性关系，也就是说固定一个 x,y 值，可以获得 k,q 坐标系里的一条直线，对于 x,y 坐标系的 A，B 两点，就可以在 k,q 坐标系获得两条直线，如图 9.2 所示：

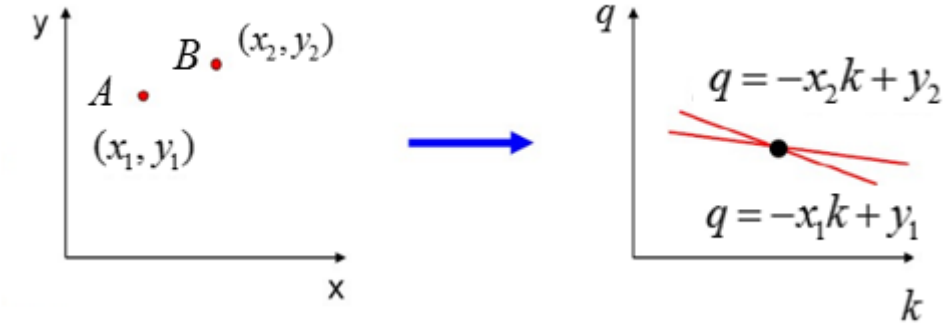


图 9.2

在 k,q 坐标系的交点处，两条直线具有相同的 k,q 值，映射到 x,y 坐标系，指的就是穿过 A，B 两点直线的斜率 k 和截距 q。

霍夫变换，就是将黑白图片中的每个白色像素映射到 k,q 坐标系（也可以是极坐标系，原理一样）中，然后统计 q，k 坐标系中每个交点处直线数量，最后进行排序，选择前 n 个 k,q 值形成 n 条直线。

2. 为何提取 ROI 区域

如下图所示，车道线一般存在于正前方，所以可以用一个梯形框将车道线存在的区域提取出来，这样可以去除额外的很多噪声，比如后面霍夫变换求取直线的时候，就可以忽略左右两边山，树木的影响。

3. 霍夫变换检测到的很多段直线，怎么转换为左右两侧，两条道路线？

首先根据每条线段的斜率分为左右直线，因为左侧跟右侧的直线，一个斜率为正，一个斜率为负，然后再去除一些偏离斜率均值较远的直线，最后再将左右侧的直线分别拟合成一条直线即可。

四、实验结果（代码见附件）

附件：车道线提取的程序

```
import cv2
import numpy as np
from moviepy.editor import VideoFileClip

blur_ksize = 5
canny_lthreshold = 50
canny_hthreshold = 150

# Hough transform parameters
rho = 1
theta = np.pi / 180
threshold = 15
min_line_length = 40
max_line_gap = 20


def roi_mask(img, vertices):  # 提取 ROI 区域
    mask = np.zeros_like(img)
    if len(img.shape) > 2:
        channel_count = img.shape[2]
        mask_color = (255,) * channel_count
    else:
        mask_color = 255

    cv2.fillPoly(mask, vertices, mask_color)
    masked_img = cv2.bitwise_and(img, mask)
    return masked_img
```

```
def hough_lines(img, rho, theta, threshold, min_line_len,
max_line_gap):  # 霍夫变换，返回画好车道线的图片
    lines = cv2.HoughLinesP(img, rho, theta, threshold, np.array([]),
minLineLength=min_line_len,
                            maxLineGap=max_line_gap)
    line_img = np.zeros((img.shape[0], img.shape[1], 3),
dtype=np.uint8)
    draw_lanes(line_img, lines)
    return line_img


def draw_lanes(img, lines, color=None, thickness=8):  # 首先将霍夫变换获
得的直线进行分类，然后分别再拟合出左右两条直线
    if color is None:
        color = [255, 0, 0]
    left_lines, right_lines = [], []
    for line in lines:
        for x1, y1, x2, y2 in line:
            k = (y2 - y1) / (x2 - x1)
            if k < 0:
                left_lines.append(line)
            else:
                right_lines.append(line)

    if (len(left_lines) <= 0 or len(right_lines) <= 0):
        return img

    clean_lines(left_lines, 0.1)
    clean_lines(right_lines, 0.1)
    left_points = [(x1, y1) for line in left_lines for x1, y1, x2, y2
in line]
    left_points = left_points + [(x2, y2) for line in left_lines for
x1, y1, x2, y2 in line]
    right_points = [(x1, y1) for line in right_lines for x1, y1, x2,
y2 in line]
    right_points = right_points + [(x2, y2) for line in right_lines
for x1, y1, x2, y2 in line]

    left_vtx = calc_lane_vertices(left_points, 325, img.shape[0])
    right_vtx = calc_lane_vertices(right_points, 325, img.shape[0])

    cv2.line(img, left_vtx[0], left_vtx[1], color, thickness)
    cv2.line(img, right_vtx[0], right_vtx[1], color, thickness)
```

```
def clean_lines(lines, threshold):  # 去掉偏离均值较远的直线，这里的值是指斜率
    slope = [(y2 - y1) / (x2 - x1) for line in lines for x1, y1, x2, y2 in line]
    while len(lines) > 0:
        mean = np.mean(slope)
        diff = [abs(s - mean) for s in slope]
        idx = np.argmax(diff)
        if diff[idx] > threshold:
            slope.pop(idx)
            lines.pop(idx)
        else:
            break


def calc_lane_vertices(point_list, ymin, ymax):  # 将左右两侧的直线拟合，返回左右车道线的顶点
    x = [p[0] for p in point_list]
    y = [p[1] for p in point_list]
    fit = np.polyfit(y, x, 1)
    fit_fn = np.poly1d(fit)

    xmin = int(fit_fn(ymin))
    xmax = int(fit_fn(ymax))

    return [(xmin, ymin), (xmax, ymax)]


def process_an_image(img):  # 总的处理函数，输入一张图片，返回一张标记车道线的图片
    roi_vtx = np.array([[(0, img.shape[0]), (460, 325), (520, 325), (img.shape[1], img.shape[0])]])
    gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
    blur_gray = cv2.GaussianBlur(gray, (blur_ksize, blur_ksize), 0, 0)
    edges = cv2.Canny(blur_gray, canny_lthreshold, canny_hthreshold)
    roi_edges = roi_mask(edges, roi_vtx)
    line_img = hough_lines(roi_edges, rho, theta, threshold, min_line_length, max_line_gap)
    res_img = cv2.addWeighted(img, 0.8, line_img, 1, 0)
    return res_img
```

```
#mp4_path = 'video_1.mp4'
#out_path = 'video_1_out.mp4'
#clip = VideoFileClip(mp4_path)
#out_clip = clip.fl_image(process_an_image)
#out_clip.write_videofile(out_path, audio=False)
```