案例一：车牌图像预处理

1. 案例目的

• 掌握车牌图像预处理的基本操作

2. 案例内容

本案例主要进行车牌图像的预处理操作，包括彩色图像灰度化，计算车牌灰度图像直方图，车牌图像高斯平滑滤波去除噪点。所有的功能模块均自行编写代码，不借助OpenCV的API函数。

3. 案例知识点

• 彩色图像灰度化；

• 图像直方图；

• 空间滤波

4. 案例时长

共2学时，具体安排如下：

• 彩色图像灰度化（0.5学时）

• 计算灰度图像直方图（1学时）

• 灰度图像高斯平滑滤波（0.5学时）

5. 案例实验环境

**•软件环境：**

1）Python 3

2）opencv 3.x

**•开发环境与工具：**

1）Jupyter Notebook

6. 案例分析

本案例主要分为以下3个步骤：

1） 彩色图像灰度化；

2） 计算灰度图像直方图；

3） 采用高斯平滑滤波去除车牌图像的噪点。

• 对输入图像进行Zero padding处理

• 求解高斯滤波模板

• 采用高斯平滑模板对待处理图像进行卷积操作

7. 案例实验过程

7.1开发准备

7.1.1 获取数据

本案例需要1个数据集，是plate.jpg。

点击右侧菜单栏中的【资源下载】按钮，复制提供的URL前缀，如https://xxx.eec-cn.com/。

将URL前缀和数据集地址拼接到一起，如https://xxx.eec-cn.com/dataSet/systemLib/2f9a51190eaa42ecba2056fc09f6535b.zip下载数据集。将下载后的数据集解压得到名为DIP实验数据集的文件夹。文件夹下存储着本案例用到的数据plate.jpg。

7.1.2 新建案例目录

在桌面上新建一个目录，名字叫SY02。在该目录下新建input目录用于存放待处理的数据。将图像plate.jpg复制到input目录下。

打开Jupyter Notebook，此时input目录的样式如图1所示。

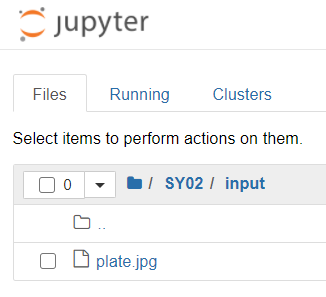


图1 input目录

7.1.3 新建代码文件

使用Jupyter Notebook新建plate\_img\_process.ipynb文件。

当前文件目录结构如图2所示。

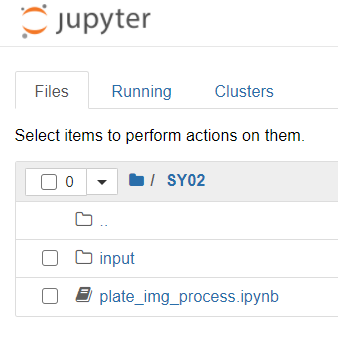


图2 案例目录结构

7.2实现RGB彩色图像转化为灰度图像

在理论课中介绍过加权平均值法灰度化：

分别为RGB图像三个通道的单通道图像。效果图如下图所示。

|  |
| --- |
|  |
| (a)车牌彩色图 |
|  |
| (b)车牌灰度图 |

图3 车牌图像灰度化效果图

打开plate\_img\_process.ipynb文件。按如下四个步骤逐步实现。

7.2.1导入相关模块

主要用到cv2，numpy和matplotlib.pyplot三个库。cv2主要用于读取图像，numpy用于构建矩阵，matplotlib.pyplot主要用于图像显示。

【代码7-2-1】plate\_img\_process.ipynb

# -\*- coding: utf-8 -\*-

'''

车牌图像预处理

'''

# 导入相关包

import cv2

import numpy as np

import matplotlib.pyplot as plt

执行该代码块。

7.2.2读取车牌彩色图像

从input目录中读取彩色车牌图像plate.jpg。

【代码7-2-2】plate\_img\_process.ipynb

# 读入车牌图片

srcImg = cv2.imread('input/plate.jpg', 1)

执行该代码块。

7.2.3自定义彩色图像灰度化函数

这里，将彩色图像灰度化封装成函数形式。函数名定义为rgb2gray，输入参数是彩色图像，输出是灰度图像。

【代码7-2-3-1】plate\_img\_process.ipynb

# 自定义函数实现彩色图转灰度图

def rgb2gray(rgbImg):

在转换之前，需要加一个判断条件，判断输入图像是否是彩色图。依据是获取输入的维度信息ndim，当ndim为3时，表明这是一幅彩色图像。

【代码7-2-3-2】plate\_img\_process.ipynb

# 加一个预判，只处理彩色图

assert rgbImg.ndim == 3, "must input a color image"

获取图像的宽高信息。

【代码7-2-3-3】plate\_img\_process.ipynb

height = rgbImg.shape[0] # 获取图像的高

width = rgbImg.shape[1] # 获取图像的宽

根据获取到的宽高信息，定义一个零值二维矩阵，宽高跟输入图像保持一致，用于存储灰度图像。

【代码7-2-3-4】plate\_img\_process.ipynb

# 定义一个灰度图像，初始像素值为0

grayImg = np.zeros([height, width], np.uint8)

构造一个2层for循环，按照行列的顺序，遍历图像像素值。获取每个像素三通道的值b\_value,g\_value和g\_value。根据灰度转换公式，得到灰度值p\_value，并将其赋值给灰度图像对应位置处的像素。最后，将灰度图grayImg作为函数返回值输出。

【代码7-2-3-5】plate\_img\_process.ipynb

for row in range(height): # 遍历高

for col in range(width): # 遍历宽

b\_value = rgbImg[row, col, 0] # 取b通道的像素值

g\_value = rgbImg[row, col, 1] # 取g通道的像素值

r\_value = rgbImg[row, col, 2] # 取r通道的像素值

p\_value = (b\_value\*0.11 + g\_value\*0.59 + r\_value\*0.30) #加权平均值

grayImg[row, col] = int(p\_value)

return grayImg

函数rgb2gray完整代码参见代码7-2-3。

【代码7-2-3】plate\_img\_process.ipynb

# 自定义函数实现彩色图转灰度图

def rgb2gray(rgbImg):

# 加一个预判，只处理彩色图

assert rgbImg.ndim == 3, "must input a color image"

height = rgbImg.shape[0] # 获取图像的高

width = rgbImg.shape[1] # 获取图像的宽

# 定义一个灰度图像，初始像素值为0

grayImg = np.zeros([height, width], np.uint8)

for row in range(height): # 遍历高

for col in range(width): # 遍历宽

b\_value = rgbImg[row, col, 0] # 取b通道的像素值

g\_value = rgbImg[row, col, 1] # 取g通道的像素值

r\_value = rgbImg[row, col, 2] # 取r通道的像素值

p\_value = (b\_value\*0.11 + g\_value\*0.59 + r\_value\*0.30) #加权平均值

grayImg[row, col] = int(p\_value)

return grayImg

执行该代码块。

7.2.4执行灰度化函数并显示灰度化结果

调用rgb2gray函数，显示结果。

【代码7-2-4】plate\_img\_process.ipynb

#调用彩色转灰度函数

grayImg = rgb2gray(srcImg)

#显示灰度化结果图

plt.imshow(grayImg,cmap ='gray')

plt.show()

执行该代码块。

彩色图像灰度化后的结果如图4所示。

|  |
| --- |
|  |
| 图4灰度化后的车牌图像 |

7.3计算灰度图像直方图

理论课介绍过，直方图是图像的一个重要特征，反映了图像灰度分布的情况。除此之外，直方图在车牌字符分割中也经常会用到，在后面的几个案例中会有具体的介绍。在OpenCV中有专门的接口函数cv2.calcHist()用于计算图像的直方图，本案例，为了加深对直方图概念的理解，我们自行编写代码实现图像直方图的计算。

7.3.1自定义灰度图像直方图计算函数

定义一个灰度图像直方图计算函数，函数名为myHist，输入是灰度图，输出是一个含有256个元素的数组hist，分别对应着像素值0~255的像素的个数。对应关系为hist[0]是灰度值为0的像素的个数，hist[1]是灰度值为1的像素的个数，…，依次类推。

【代码7-3-1-1】plate\_img\_process.ipynb

# 自定义函数计算灰度图直方图

def myHist(grayImg):

添加判断语句，保证输入图像是灰度图。

【代码7-3-1-2】plate\_img\_process.ipynb

# 添加预判断，只处理灰度图像

assert grayImg.ndim == 2, "must input a gray image"

定义一个数组hist，统计灰度直方图。

【代码7-3-1-3】plate\_img\_process.ipynb

hist = np.zeros(256, np.uint32) # 定义一个数组，用于存储像数值计数

获取图像的宽高信息。

【代码7-3-1-4】plate\_img\_process.ipynb

height = grayImg.shape[0] # 获取图像的高

width = grayImg.shape[1] # 获取图像的宽

构造2层for循环，依次遍历图像中的每一个像素。得到其像素值p\_value，对应的hist[p\_value]进行累加。最后函数返回直方图信息即数组hist。

【代码7-3-1-5】plate\_img\_process.ipynb

for row in range(height): # 遍历高

for col in range(width): #遍 历宽

p\_value = grayImg[row, col] # 获取像素值

hist[p\_value] = hist[p\_value] + 1 # 累加统计对应像素值像素数

return hist

函数myHist完整的代码参见代码7-3-1。

【代码7-3-1】plate\_img\_process.ipynb

# 自定义函数计算灰度图灰度直方图

def myHist(grayImg):

# 添加预判断，只处理灰度图像

assert grayImg.ndim == 2, "must input a gray image"

hist = np.zeros(256, np.uint32) # 定义一个数组，用于存储像数值计数

height = grayImg.shape[0] # 获取图像的高

width = grayImg.shape[1] # 获取图像的宽

for row in range(height): # 遍历高

for col in range(width): #遍 历宽

p\_value = grayImg[row, col] # 获取像素值

hist[p\_value] = hist[p\_value] + 1 # 累加统计对应像素值像素数

return hist

执行该代码块。

7.3.2执行直方图计算函数，显示车牌直方图

调用上面自定义的直方图计算函数myHist，并显示结果。

【代码7-3-2】plate\_img\_process.ipynb

# 调用直方图函数

hist = myHist(grayImg)

# 显示直方图，用于观察灰度图像的直方图特性

X = [i for i in range(256)]

fig = plt.figure()

plt.bar(X, hist, color="green")

plt.title(r'Histogram')

plt.show()

plt.show()

执行该代码块。

车牌图像灰度化后的直方图如图5所示。

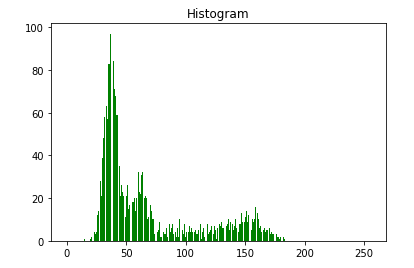


图5 车牌灰度化后的直方图

7.4灰度图像高斯平滑滤波

通过观察图4发现车牌灰度图像有颗粒感，原因是存在噪点，可以通过高斯平滑进行一定程度的改善。

预计改善后的效果如图6所示，图像整体颗粒感有所减轻。

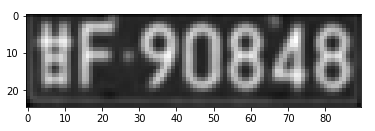


图6 车牌高斯平滑后的期望效果

7.4.1基本原理介绍

进行高斯平滑滤波的基本思路是采用一个高斯平滑模板对图像进行卷积操作。

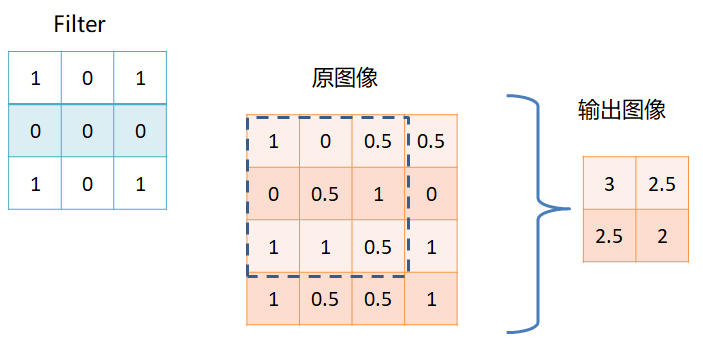


图7 滤波器模板、卷积计算示意图

如图7所示，如果直接应用模板对输入图像进行卷积操作，输出图像尺寸会有一定的缩减。假设原图像的尺寸为W\*H，模板尺寸为S\*S，输出图像的尺寸变为（W-S+1）\*（H-S+1）。为了保证输出图像跟输入图像尺寸保持一致，一般选择先将输入图像进行padding操作，即在图像的四周填充像素。常用的有zero padding，即在原图像的四周用0值像素填充。

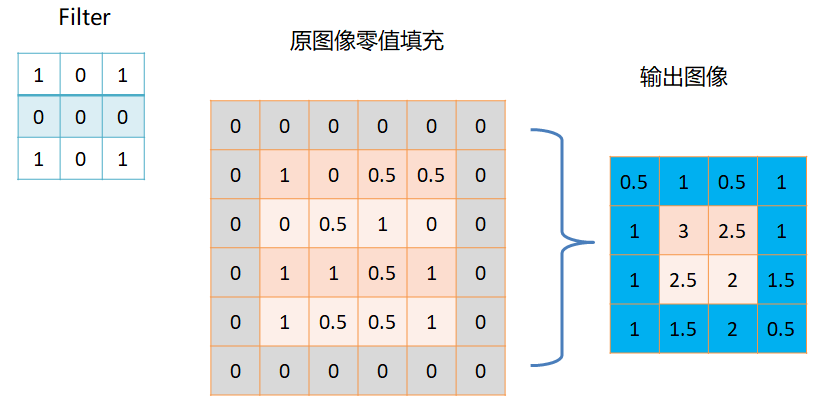


图8 对原图进行填充后的卷积

如图8所示，填充后再做卷积操作，输出图像的尺寸跟输入图像保持一致。

7.4.2自定义高斯平滑滤波函数

首先，定义一个高斯平滑函数gaussian\_filter，输入参数有灰度图srcImg，卷积核尺寸K\_size和高斯核参数sigma。

【代码7-4-2-1】plate\_img\_process.ipynb

# 自定义高斯平滑滤波函数

def gaussian\_filter(srcImg, K\_size=3, sigma=1.3):

先添加输入图像合法性判断。要求输入图像必须是灰度图，也即输入图像的ndim应该为2。

【代码7-4-2-2】plate\_img\_process.ipynb

assert srcImg.ndim == 2, "must input a gray image"

获取图像的宽高信息H，W，同时根据输入参数卷积核K\_size的值计算padding的尺寸pad，计算公式为K\_size的值除以2向下取整。例如，K\_size的值为3，则pad的值为1。

【代码7-4-2-3】plate\_img\_process.ipynb

H,W = srcImg.shape

pad = K\_size // 2

定义平滑滤波后的输出图像dstImg。dstImg是在原图srcImg的四周进行补零得到。

【代码7-4-2-4】plate\_img\_process.ipynb

# 这里采用Zero padding：将图像四周加一圈0值像素

dstImg = np.zeros((H + pad \* 2, W + pad \* 2), dtype=np.float)

dstImg[pad: pad + H, pad: pad + W] = srcImg.copy().astype(np.float)

计算卷积核模板K。先构造一个K\_size\*K\_size大小的矩阵。在通过一个2层for循环，计算卷积核模板内的元素值，计算公式：。x,y是跟像素位置坐标相关的变量。是函数输入参数sigma。

【代码7-4-2-5】plate\_img\_process.ipynb

# 计算滤波核模板 Kernel

K = np.zeros((K\_size, K\_size), dtype=np.float)

for x in range(-pad, -pad + K\_size):

for y in range(-pad, -pad + K\_size):

K[y + pad, x + pad] = np.exp( -(x \*\* 2 + y \*\* 2) / (2 \* (sigma \*\* 2)))

K /= (2 \* np.pi \* sigma \* sigma)

对卷积核进行归一化，保证卷积核元素值在0~1之间。

【代码7-4-2-6】plate\_img\_process.ipynb

K /= K.sum() # 归一化

复制一份dstImg数据tmp用于中间计算。

【代码7-4-2-7】plate\_img\_process.ipynb

tmp = dstImg.copy()

构造2层for循环，用前面计算得到卷积核和图像tmp进行卷积运算。得到的结果保存到dstImg中。

【代码7-4-2-8】plate\_img\_process.ipynb

# 采用模板对原图进行卷积滤波

for y in range(H): #遍历高

for x in range(W): # 遍历宽

#计算卷积后的值更新

dstImg[pad + y, pad + x] = np.sum(K \* tmp[y: y + K\_size, x: x + K\_size])

对卷积后的图像dstImg元素值进行截断，保证元素值在0~255之间。

【代码7-4-2-9】plate\_img\_process.ipynb

#对卷积后的图像做像素值截断，保证值都在0~255。超过255的赋值255，低于0的赋值0

dstImg = np.clip(dstImg, 0, 255)

图像数据类型进行一次转换，转换之前是float型，将其转为int型。同时，图像的尺寸进行调整。最后，将最终滤波后的图像dstImg返回。

【代码7-4-2-10】plate\_img\_process.ipynb

#图像数值类型转换为int型，转换之前是float类型

dstImg = dstImg[pad: pad + H, pad: pad + W].astype(np.uint8)

return dstImg

gaussian\_filter函数完整的代码参见代码7-4-2。

【代码7-4-2】plate\_img\_process.ipynb

# 自定义高斯平滑滤波函数

def gaussian\_filter(srcImg, K\_size=3, sigma=1.3):

assert srcImg.ndim == 2, "must input a gray image"

H,W = srcImg.shape

pad = K\_size // 2

# 这里采用Zero padding：将图像四周加一圈0值像素

dstImg = np.zeros((H + pad \* 2, W + pad \* 2), dtype=np.float)

dstImg[pad: pad + H, pad: pad + W] = srcImg.copy().astype(np.float)

# 计算滤波核模板 Kernel

K = np.zeros((K\_size, K\_size), dtype=np.float)

for x in range(-pad, -pad + K\_size):

for y in range(-pad, -pad + K\_size):

K[y + pad, x + pad] = np.exp( -(x \*\* 2 + y \*\* 2) / (2 \* (sigma \*\* 2)))

K /= (2 \* np.pi \* sigma \* sigma)

K /= K.sum() # 归一化

tmp = dstImg.copy()

# 采用模板对原图进行卷积滤波

for y in range(H): # 遍历高

for x in range(W): # 遍历宽

# 计算卷积后的值更新

dstImg[pad + y, pad + x] = np.sum(K \* tmp[y: y + K\_size, x: x + K\_size])

# 对卷积后的图像做像素值截断，保证值都在0~255。超过255的赋值255，低于0的赋值0

dstImg = np.clip(dstImg, 0, 255)

# 图像数值类型转换为int型，转换之前是float类型

dstImg = dstImg[pad: pad + H, pad: pad + W].astype(np.uint8)

return dstImg

执行该代码块。

7.4.3调用高斯平滑滤波函数并显示结果

调用前面自定义的高斯平滑滤波函数，并显示平滑结果。

【代码7-4-3】plate\_img\_process.ipynb

# 调用高斯平滑滤波函数

gaussImg = gaussian\_filter(grayImg, 3, 1.1)

#显示高斯滤波结果

plt.imshow(gaussImg,cmap ='gray')

plt.show()

执行该代码块。

运行效果如图9所示。

|  |
| --- |
| 图9高斯平滑滤波运行效果 |
| 把高斯平滑滤波处理后的灰度图和车牌原灰度图做对比，发现图像整体颗粒感有所减轻，如图10所示。 |
|  |
| （a）车牌原灰度图像 |
|  |
| （b）车牌灰度图高斯平滑滤波效果图 |

图10高斯平滑滤波处理后的灰度图和车牌原灰度图的对比

8. 案例代码

【案例代码】plate\_img\_process.ipynb

# -\*- coding: utf-8 -\*-

'''

车牌图像预处理

'''

# 导入相关包

import cv2

import numpy as np

import matplotlib.pyplot as plt

# 读入车牌图片

srcImg = cv2.imread('input/plate.jpg', 1)

# 自定义函数实现彩色图转灰度图

def rgb2gray(rgbImg):

# 加一个预判，只处理彩色图

assert rgbImg.ndim == 3, "must input a color image"

height = rgbImg.shape[0] # 获取图像的高

width = rgbImg.shape[1] # 获取图像的宽

# 定义一个灰度图像，初始像素值为0

grayImg = np.zeros([height, width], np.uint8)

for row in range(height): # 遍历高

for col in range(width): # 遍历宽

b\_value = rgbImg[row, col, 0] # 取b通道的像素值

g\_value = rgbImg[row, col, 1] # 取g通道的像素值

r\_value = rgbImg[row, col, 2] # 取r通道的像素值

p\_value = (b\_value\*0.11 + g\_value\*0.59 + r\_value\*0.30) #加权平均值

grayImg[row, col] = int(p\_value)

return grayImg

#调用彩色转灰度函数

grayImg = rgb2gray(srcImg)

#显示灰度化结果图

plt.imshow(grayImg,cmap ='gray')

plt.show()

# 自定义函数计算灰度图灰度直方图

def myHist(grayImg):

# 添加预判断，只处理灰度图像

assert grayImg.ndim == 2, "must input a gray image"

hist = np.zeros(256, np.uint32) # 定义一个数组，用于存储像数值计数

height = grayImg.shape[0] # 获取图像的高

width = grayImg.shape[1] # 获取图像的宽

for row in range(height): # 遍历高

for col in range(width): #遍 历宽

p\_value = grayImg[row, col] # 获取像素值

hist[p\_value] = hist[p\_value] + 1 # 累加统计对应像素值像素数

return hist

# 调用直方图函数

hist = myHist(grayImg)

# 显示直方图，用于观察灰度图像的直方图特性

X = [i for i in range(256)]

fig = plt.figure()

plt.bar(X, hist, color="green")

plt.title(r'Histogram')

plt.show()

# 自定义高斯平滑滤波函数

def gaussian\_filter(srcImg, K\_size=3, sigma=1.3):

assert srcImg.ndim == 2, "must input a gray image"

H,W = srcImg.shape

pad = K\_size // 2

# 这里采用Zero padding：将图像四周加一圈0值像素

dstImg = np.zeros((H + pad \* 2, W + pad \* 2), dtype=np.float)

dstImg[pad: pad + H, pad: pad + W] = srcImg.copy().astype(np.float)

# 计算滤波核模板 Kernel

K = np.zeros((K\_size, K\_size), dtype=np.float)

for x in range(-pad, -pad + K\_size):

for y in range(-pad, -pad + K\_size):

K[y + pad, x + pad] = np.exp( -(x \*\* 2 + y \*\* 2) / (2 \* (sigma \*\* 2)))

K /= (2 \* np.pi \* sigma \* sigma)

K /= K.sum() # 归一化

tmp = dstImg.copy()

# 采用模板对原图进行卷积滤波

for y in range(H): # 遍历高

for x in range(W): # 遍历宽

# 计算卷积后的值更新

dstImg[pad + y, pad + x] = np.sum(K \* tmp[y: y + K\_size, x: x + K\_size])

# 对卷积后的图像做像素值截断，保证值都在0~255。超过255的赋值255，低于0的赋值0

dstImg = np.clip(dstImg, 0, 255)

# 图像数值类型转换为int型，转换之前是float类型

dstImg = dstImg[pad: pad + H, pad: pad + W].astype(np.uint8)

return dstImg

# 调用高斯平滑滤波函数

gaussImg = gaussian\_filter(grayImg, 3, 1.1)

# 显示高斯滤波结果

plt.imshow(gaussImg,cmap ='gray')

plt.show()