实验三 OpenCV 3在图像处理中的使用

1. 实验目的
2. 掌握OpenCV 3在图像处理中常用函数的使用。
3. 熟练操作Anaconda环境中Jupyter Notebook的使用。
4. 熟悉OpenCV中傅里叶变化的原理。
5. 实验要求
6. 熟练掌握Anaconda环境的操作。
7. 理解OpenCV 3在图像处理中的工作原理。
8. 将上机程序调试通过，并能独立完成课堂练习题目。
9. 实验内容

安装完成OpenCV3在图像处理中常用的函数，并完成相关配置操作；掌握在Python编程中调用相应的库；掌握在Jupyter Notebook中的基本操作;掌握基本OpenCV python程序的编写。

1. 实验内容

**一 不同色彩空间的转换**

OpenCV中有数百种关于在不同色彩空间之间转换的方法。当前，在计算机中有三种常用的色彩空间：灰度，BGR以及HSV(Hue,Saturation,Value)。

灰度色彩空间是通过去除色彩信息来将其转换成灰阶，灰度色彩空间对中间处理特别有效，比如人脸检测。

BGR，即蓝-绿-红色彩空间，每一个像素点都由一个三元数组来表示，分别代表蓝、绿、红三种颜色。网页开发者可能熟悉另一个与之相似的色彩空间：RGB，他们只是在颜色顺序上不同。

HSV,H(Hue)是色调，S(Saturation)是饱和度，V(Value)表示黑暗的程度(或光谱另一端的命令程度)。

在第一次处理BGR色彩空间的时候，可以不要其中的一个色彩分量，比如像素值[0 255 255](没有蓝色，绿色分量取最大值，红色分量取最大值)表示黄色。如果读者有艺术背景，会发现绿色和红色混合产生浑浊的褐色，这是因为计算所使用的颜色模型具有可加性并且处理的是光照，而绘画不是这样的(它遵从减色模型 subtractive color model)。计算机使用显示器发光来做颜色的媒介，因此运行在计算机上的软件所使用的色彩模型是加色模型。

**二 傅里叶变换**

在OpenCV中，对图像和视频的大多数处理或多或少都会涉及到傅里叶变换的概念。Joseph Fourier(约瑟夫.傅里叶)是以为18世纪的法国数学家，他发现并推广了很多数学概念，主要研究热学规律，在数学上，他认为一切都可以用波形来描述。具体而言，他观察到所有的波形都是由一系列简单且频率不同的正弦曲线叠加得到。

也就是说人们看到的波形都是由其他波形叠加得到的。这个概念对操作图像非常有帮助，因为这样我们可以区分图像哪些区域的信号变化特别强，哪些区域的信号变化不那么强，从而可以任意地标记噪声区域，感兴趣区域，前景和背景等。原始图像由许多频率组成，人们能够分离这些频率来处理图像和提取感兴趣的数据。

下面通过傅里叶变换来介绍图像的幅度谱(magnitude specturm)。图像的幅度谱是另一种图像，幅度谱图像呈现了原始图像在变化方面的一种表示：把一张图像中最明亮的像素放到图像中央，然后逐渐变暗，在边缘上像素最暗。这样可以发现图像中有多少亮的像素和暗的像素，以及它们分布的比例。

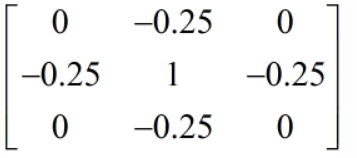
傅里叶变换的概念是许多常见的图像处理操作的基础，比如边缘检测或线段和形状检测。

下面介绍两个概念：高通滤波器和低通滤波器。

1.高通滤波器

高通滤波器(HPF)是检测图像的某个区域，然后根据像素与周围像素的亮度差值来提升该像素的亮度的滤波器。

 以如下的核(kernel)，即滤波器矩阵为例：



注：核是指一组权重的集合，它会应用在源图像的一个区域，并由此生成目标图像的一个像素。比如，大小为7的核意味着每49(7x7)个源图像的像素会产生目标图像的一个像素。  
可把核看做一块覆盖在源图像上可移动的毛玻璃片，玻璃片覆盖区域的光线会按某种方式进行扩散混合后透过去。

 在计算完中央像素与周围邻近像素的亮度差值之和以后，如果亮度变化很大，中央像素的亮度会增加，反之则不会。换句话说，如果一个像素比它周围的像素更突出，就会提升它的亮度。

这在边缘检测上尤为有效，它采用一种称为高频提升滤波器(high boost filter)的高通滤波器。

高通和低通滤波器都有一个半径(radius)的属性，它决定了多大面积的临近像素参与滤波运算。

下面是一个高通滤波器的例子，代码如下：

import cv2

import numpy as np

from scipy import ndimage

'''

1.傅里叶变换

'''

'''

(1)高通滤波器

'''

kernel\_3x3 = np.array([[-1,-1,-1],[-1,8,-1],[-1,-1,-1]])

kernel\_5x5 = np.array([[-1,-1,-1,-1,-1],

[-1,-1, 2, 1,-1],

[-1, 2, 4, 2,-2],

[-1, 1, 2, 2,-1],

[-1,-1,-1,-1,-1]])

#读取图像，指定格式为灰度图像

img = cv2.imread('./image/img6.jpg',cv2.IMREAD\_GRAYSCALE)

#进行卷积运算

k3 = ndimage.convolve(img,kernel\_3x3)

k5 = ndimage.convolve(img,kernel\_5x5)

#模糊滤波

blurred = cv2.GaussianBlur(img,(11,11),0)

#作差

g\_hpf = img - blurred

#显示图像

cv2.imshow('original',img)

cv2.imshow('3x3',k3)

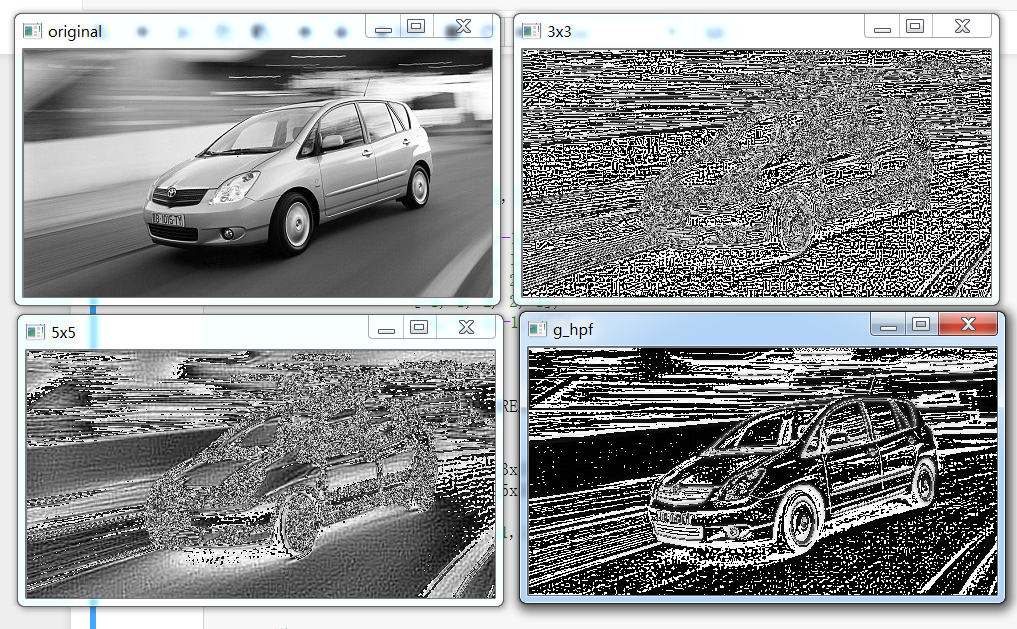
cv2.imshow('5x5',k5)

cv2.imshow('g\_hpf',g\_hpf)

cv2.waitKey()

cv2.destroyAllWindows()

运行结果如下：



导入 模块后，我们定义了一个3x3和一个5x5的核，然后将读入的图像转换成灰度格式。通常大多数图像处理都会用Numpy模块来完成，但是这里的情况比较特殊，因为需要用一个给定核与图像进行'卷积'，但是Numpy碰巧只接受一维数组。

上面代码用了两个自定义卷积核来实现两个高通滤波器。最后又用一种不同的方法来实现高通滤波器：通过对图像应用低通滤波器之后，与原始图像计算差值。这样得到的效果会更好。

 这里注意有一点需要注意：使用卷积进行运算，并不能保证的每个像素值都在0~255之间。对于在区间外的像素点会导致灰度图无法显示，所以还需要做归一化，然后每个值乘以255，再将所有的值映射到这个区间内。归一化算法:x=(x-Min)(Max-Min)，这样x的范围就在[0,1]之间了。我们在上面调用的函数ndimage.convolve()在内部已经做了这些处理，所以我们就不需要自己写归一化的处理过程了。

**2 低通滤波器**

高通滤波器是根据像素与邻近像素的亮度差值来提升该像素的亮度。低通滤波器(LPF)则是在像素与周围像素的亮度差值小于一定特征值，平滑该像素的亮度。它主要用于去噪和模糊化，比如说，高斯模糊是最常用的模糊滤波器(平滑滤波器)之一，它是削弱高频信号强度的低通滤波器。

**三 边缘检测**

边缘在人类视觉和计算机视觉中均起着重要的作用。人类能够仅凭一张背景剪影或一个草图就能识别出物体的类型和姿态。

Open CV提供了许多边缘检测滤波函数，包括以下：

Laplacian() #作为边缘检测函数，他会产生明显的边缘线条，灰度图像更是如此。

Sobel()

Scharr()

这些滤波函数都会将非边缘区域转换为黑色，边缘区域转换成白色或其他饱和的颜色。但是这些函数都容易将噪声错误的识别为边缘。缓解这个问题的方法就是在找到边缘之前对图像进行模糊处理，去除噪声。

Open CV也提供了需要模糊滤波函数，包括以下：

blur()

medianBlur() #它对去除数字化的视频噪声特别有效，特别是去除彩色图像的噪声

GaussianBlur()

边缘检测和模糊滤波的函数的参数有很多，但总会有一个ksize参数，它是一个奇数，表示滤波核的宽和高(以像素为单位)。

[cv2.blur(src,ksize[,dst[,anchor[,borderType]]])](https://blog.csdn.net/keith_bb/article/details/54382426)函数，均值滤波。

均值滤波是一种典型的线性滤波算法，主要是利用像素点邻域的像素值来计算像素点的值。其具体方法是首先给出一个滤波kernel，该核将覆盖像素点周围的其他邻域像素点，去掉像素本身，将其邻域像素点相加然后取平均值即为该像素点的新的像素值，这就是均值滤波的本质。

src: 输入图像，图像深度是cv2.CV\_8U、cv2.CV\_16U、cv2.CV\_16S、cv2.CV\_32F以及cv2.CV\_64F其中的某一个。

dst: 输出图像，深度和类型与输入图像一致。

ksize: 滤波kernel的尺寸,元组类型。

anchor: 字面意思是锚点，也就是处理的像素位于kernel的什么位置，默认值为(-1, -1)即位于kernel中心点，如果没有特殊需要则不需要更改。

borderType 用于推断图像外部像素的某种边界模式，有默认值cv2.BORDER\_DEFAULT。

[cv2.medianBlur(src,ksize[,dst])](https://blog.csdn.net/keith_bb/article/details/54426920)中值滤波函数。

中值滤波是一种典型的非线性滤波，是基于排序统计理论的一种能够有效抑制噪声的非线性信号处理技术，基本思想是用像素点邻域灰度值的中值来代替该像素点的灰度值，让周围的像素值接近真实的值从而消除孤立的噪声点。该方法在取出脉冲噪声、椒盐噪声的同时能保留图像的边缘细节。这些优良特性是线性滤波所不具备的。   
中值滤波首先也得生成一个滤波核，将该核内的各像素值进行排序，生成单调上升或单调下降的二维数据序列，二维中值滤波输出为g(x, y)=medf{f(x-k, y-1),(k, l∈w)}，其中f(x,y)和g(x,y)分别是原图像和处理后图像, w为输入的二维模板，能够在整幅图像上滑动，通常尺寸为3\*3或5\*5区域，也可以是不同的形状如线状、圆形、十字形、圆环形等。通过从图像中的二维模板取出奇数个数据进行排序，用排序后的中值取代要处理的数据即可。   
中值滤波对消除椒盐噪声非常有效，能够克服线性滤波器带来的图像细节模糊等弊端，能够有效保护图像边缘信息，是非常经典的平滑噪声处理方法。在光学测量条纹图像的相位分析处理方法中有特殊作用，但在条纹中心分析方法中作用不大。

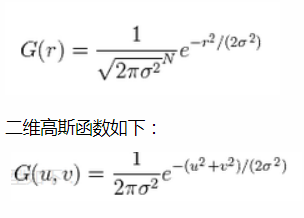
src: 输入图像，图像为1、3、4通道的图像，当核尺寸为3或5时，图像深度只能为cv2.CV\_8U、cv2.CV\_16U、cv2.CV\_32F中的一个，如而对于较大孔径尺寸的图片，图像深度只能是cv2.CV\_8U。

dst: 输出图像，尺寸和类型与输入图像一致。

 ksize: 滤波核的尺寸大小，必须是大于1的奇数，如3、5、7……

cv2.GaussianBlur(src,ksize,sigmaX[,sigmaxY[,borderType]]])高斯滤波函数。

高斯滤波是一种线性平滑滤波，对于除去高斯噪声有很好的效果。在其官方文档中形容高斯滤波为”Probably the most useful filter”，同时也指出高斯滤波并不是效率最高的滤波算法。高斯算法在官方文档给出的解释是高斯滤波是通过对输入数组的每个点与输入的高斯滤波核执行卷积计算然后将这些结果一块组成了滤波后的输出数组，通俗的讲就是高斯滤波是对整幅图像进行加权平均的过程，每一个像素点的值都由其本身和邻域内的其他像素值经过加权平均后得到。高斯滤波的具体操作是：用一个核（或称卷积、掩模）扫描图像中的每一个像素，用模板确定的邻域内像素的加权平均灰度值去替代模板中心像素点的值。   
在图像处理中高斯滤波一般有两种实现方式：一种是用离散化窗口滑窗卷积，另一种是通过傅里叶变换。最常见的就是第一种滑窗实现，只有当离散化的窗口非常大，用滑窗计算量非常大的情况下会考虑基于傅里叶变换的方法。   
我们在参考其他文章的时候可能会出现高斯模糊和高斯滤波两种说法，其实这两种说法是有一定区别的。我们知道滤波器分为高通、低通、带通等类型，高斯滤波和高斯模糊就是依据滤波器是低通滤波器还是高通滤波器来区分的。比如低通滤波器，像素能量低的通过，而对于像素能量高的部分将会采取加权平均的方法重新计算像素的值，将能量像素的值编程能量较低的值，我们知道对于图像而言其高频部分展现图像细节，所以经过低通滤波器之后整幅图像变成低频造成图像模糊，这就被称为高斯模糊；相反高通滤波是允许高频通过而过滤掉低频，这样将低频像素进行锐化操作，图像变的更加清晰，被称为高斯滤波。说白了很简单就是：高斯滤波是指用高斯函数作为滤波函数的滤波操作而高斯模糊是用高斯低通滤波器。   
高斯滤波在图像处理中常用来对图像进行预处理操作，虽然耗时但是数字图像用于后期应用但是其噪声是最大的问题，噪声会造成很大的误差而误差在不同的处理操作中会累积传递，为了能够得到较好的图像，对图像进行预处理去除噪声也是针对数字图像处理的无奈之举。   
高斯滤波器是一类根据高斯函数的形状来选择权值的线性平滑滤波器，高斯滤波器对于服从正太分布的噪声非常有效，一维高斯函数如下：



src: 输入图像，图像深度为cv2.CV\_8U、cv2.CV\_16U、cv2.CV\_16S、cv2.CV\_32F、cv2.CV\_64F。

dst: 输出图像，与输入图像有相同的类型和尺寸。

ksize: 高斯内核大小,元组类型

sigmaX: 高斯核函数在X方向上的标准偏差

sigmaY: 高斯核函数在Y方向上的标准偏差，如果sigmaY是0，则函数会自动将sigmaY的值设置为与sigmaX相同的值，如果sigmaX和sigmaY都是0，这两个值将由ksize[0]和ksize[1]计算而来。具体可以参考[getGaussianKernel()](http://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html?highlight=gaussianblur#Mat%20getGaussianKernel%28int%20ksize,%20double%20sigma,%20int%20ktype%29)函数查看具体细节。建议将size、sigmaX和sigmaY都指定出来。

borderType: 推断图像外部像素的某种便捷模式，有默认值cv2.BORDER\_DEFAULT，如果没有特殊需要不用更改，具体可以参考[borderInterpolate()](http://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html?highlight=gaussianblur#int%20borderInterpolate%28int%20p,%20int%20len,%20int%20borderType%29)函数。

[cv2. bilateralFilter()](https://blog.csdn.net/keith_bb/article/details/54427779)双边滤波函数。

这里使用medianBlur()作为模糊函数，使用Laplacian()作为边缘检测函数。在使用medianBlur()之后，需要将图像从BGR色彩空间转换为灰度色彩空间。在得到Laplacian()函数结果之后，需要将图像转换为黑色边缘和白色背景(之前是白色边缘黑色背景)。然后将其归一化，并乘以源图像以便能将边缘变黑。在filters.py文件中实现这个函数：

def strokeEdges(src,blurKsize=7,edgeKsize=5):

'''

该函数实现性能更好的边缘检测

这里使用medianBlur()作为模糊函数，使用Laplacian()作为边缘检测函数。在使用medianBlur()之后，

需要将图像从BGR色彩空间转换为灰度色彩空间。在得到Laplacian()函数结果之后，需要将图像转换为

黑色边缘和白色背景(之前是白色边缘黑色背景)。然后将其归一化，并乘以源图像以便能将边缘变黑。

args:

src：源图像数据 BGR色彩空间

blurKsize：模糊滤波卷积核的宽和高 小于3，不进行模糊处理

edgeKsize：边缘检测卷积核的宽和高

　　return:

　　　　 dst：目标图像数据 灰度色彩空间

'''

if blurKsize >= 3:

#先模糊处理

blurredSrc = cv2.medianBlur(src,blurKsize)

cv2.imshow('blurredSrc',blurredSrc)

#BGR格式转化为灰度格式

graySrc = cv2.cvtColor(blurredSrc,cv2.COLOR\_BGR2GRAY)

else:

graySrc = cv2.cvtColor(src,cv2.COLOR\_BGR2GRAY)

cv2.imshow('graySrc',graySrc)

#边缘检测 对灰度图像检测效果更好

cv2.Laplacian(graySrc,cv2.CV\_8U,graySrc,ksize = edgeKsize)

cv2.imshow('laplacian',graySrc)

#颜色反向处理 并归一化

normalizedInverseAlpha = (1.0/255)\*(255 - graySrc)

cv2.imshow('normalizedInverseAlpha',normalizedInverseAlpha)

#通道分离 B,G,R 单通道图像

channels = cv2.split(src)

cv2.imshow('B',channels[0])

#计算后的结果分别与每个通道相乘

for channel in channels:

#这里是点乘，即对应元素相乘

channel[:] = channel \* normalizedInverseAlpha

cv2.imshow('B1',channels[0])

#通道合并(只能合并多个单通道成为多通道)

return cv2.merge(channels)

img = cv2.imread('car.jpg',cv2.IMREAD\_COLOR)

dst = strokeEdges(img)

cv2.imshow('dst',dst)

cv2.waitKey()

cv2.destroyAllWindows()

完成上面代码的调试，看看运行效果。

五、课堂练习

1. 参照chapter3.ipynb文件，把相关操作执行一遍。