卷积

**一 实验原理**

一副图像本身相当于一个二维的离散的函数，在连续函数中我们可以通过检测函数的微分和二阶微分来检测图像的特征并对这些特征进行操作，在离散的图像中使用差分来进行这些操作，其具体的实现方式就是利用卷积核。这些卷积核是一个n×n的矩阵，常见有3×3、5×5等。很多时候卷积核也称为算子并以他们的发现者命名，例如检测二阶导的拉普拉斯算子和检测一阶导的sobel算子、prewitt算子。拉普拉斯算子如图一。卷积核有时也称为滤波器，以图像平滑处理为例，卷积核的工作是将高频的早点滤去。



图一：八邻域拉普拉斯算子

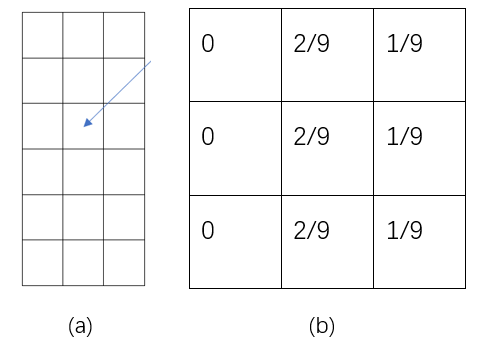
通常情况下在计算卷积时，将卷积核压在图像上，将卷积核与图像重叠的部分进行相乘再总体相加，得到新图像的像素值。在这个过程中有几个需要注意的地方。

第一是卷积核的形状。一般而言卷积核在压到图像上时以中心为对齐点，在tensorflow、pytorch、opencv提供的卷积网络或卷积操作中默认遵循这种方法，这也要求卷积核的边长是奇数，这样才能出现一个中心。不过在本次实验种使用了2×2的卷积核，所以对齐方式变为了以右上角对齐。

第二是卷积核操作后图像的取值。在卷积操作中由于有累加的步骤，可能出现像素值大于255的情况，同时如果使用带有负值的卷积核，也可能出现像素值小于0的情况。我们将大于255的值置为255，小于0的值置为0。

第三是卷积核中元素的值。通常而言卷积核的和等于1，这样图像在处理之后亮度不会发生显著变化，如果滤波器矩阵所有元素之和大于1，那么滤波后的图像就会比原图像更亮，反之如果小于1，那么得到的图像就会变暗。不过观察拉普拉斯算子等就会发现他们的和等于零，其原因是为了突出边缘，这些边缘检测算子处理过的图像会变的整体非常暗，只有在边缘部分保持高亮。

第四也是最重要的部分是边缘处理。当卷积核压在图像边缘时，会有一部分外侧卷积核无法与图像重叠，所以需要一些特判来应对这些情况。在作业要求中给出了三种处理方法。第一种是将没有重叠的部分置为零，第二种是将图像镜像对称，第三种是暂时重新分配卷积核的权值，使卷积核全部落在图像上。第一种情况会出现显著的黑边，因为原本卷积核的和为1，能够保持图像的亮度不变，而现在有部分像素置零，导致亮度降低产生黑边。第二三种情况不会导致黑边。第二种情况没有充分考虑像素之间的加权关系，会导致镜像之后最外层的像素的值向次外层入侵，见图二，对于(a)中箭头指向的点，此时卷积核与(b)等价。第三种情况考虑了加权的比例，我认为相对更为合理。



图二：第二种情况的解释

**二 结果展示**

**原图：**图片来源：https://cn.bing.com/images/search?q=%e5%99%aa%e7%82%b9%e5%9b%be%e7%89%87&id=A0AC3AFAF33F4AB2ECA0472C70D7FA1710E4E9A4&FORM=IQFRBA



**二、一 2×2矩阵：**

**Opencv的处理结果：**



1.第一种边缘处理方法，左侧下侧边缘有明显黑边，噪点有减少



2.第二种边缘处理方法，没有黑边，噪点有减少



3.第三种边缘处理方法，无黑边，噪点有减少



**二、二 3×3矩阵：**

**Opencv的处理结果：噪点更少了，也更模糊了**



1.第一种，同样有黑边，黑边有扩大现象，原因是第二个卷积核更大，让更多的点受到了黑边的影响



2.第二种，无黑边，少噪点，模糊



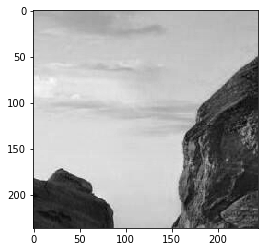
3.第三种，同上



**二、三 番外，测试拉普拉斯算子：**

使用图一所示的拉普拉斯算子，采用第三种边缘检测方法。 这里将原图转化为了灰度图以方便显示。

**原图和原图的灰度图：**

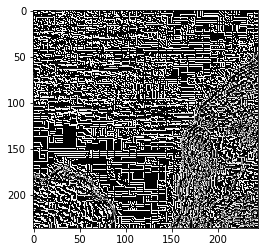
**Opencv的处理结果：**



采用第三种边缘处理方法的结果：

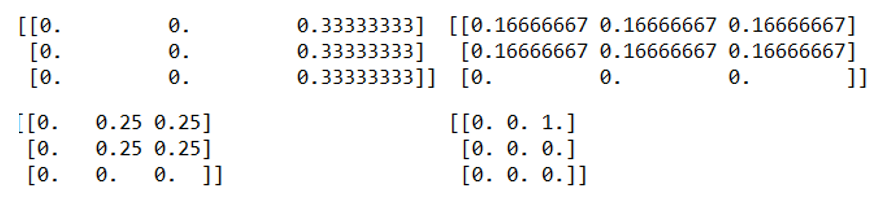


两者的差可以看到不同的地方比较明显，虽然仅就观察而言两者效果相差不大。我猜测这是因为opencv使用了中心对齐而我以右上角为准对齐。



**三 分析与测试**

**首先是对卷积核变换的说明**。根据作业要求中的演示，我编写了如下算法。一，检测是否有边界现象。二，传参，计算获得行、列与图像不重叠位置的下标。三，先按每行进行，计算行的每个值的和，计算每行与图像重叠部分的元素的和，当前值=原始值/重叠和\*行和，更新卷积核。四，按每列进行，步骤与行一致。对于第二个矩阵，图三分别展示了三种情况。



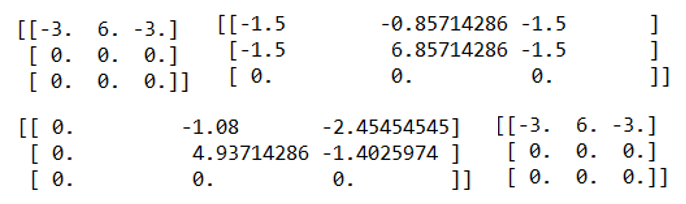
图三：四种卷积核变换后的情况

不过当矩阵中的值出现负的值时，需要进行一些处理。我的第一个处理思路是在求和时求每行、列的绝对值之和。但是在实现之后发现这种处理方式会导致整体的和与原矩阵发生变化。以图一举例，若需要保留第一行的三个数字中右侧的两个，由于这两个数字都是负数会发生如图四所示的现象，这显然是我不希望看到的。



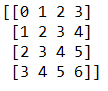
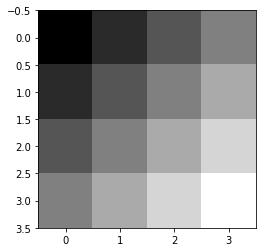
图四：不理想的变换

很遗憾的是我在多次尝试之后选择了最初的处理方法，这种方法优势有三，首先可以维持总体和不变，其次是可以大致保持行、列之间的总体比例，再次是保证行、列之内的各个数字绝对值之间的大小关系。如图五，可以看到在保证和不变的情况下保留了各行列的大小关系、中间值较大的特征，各个值的符号。

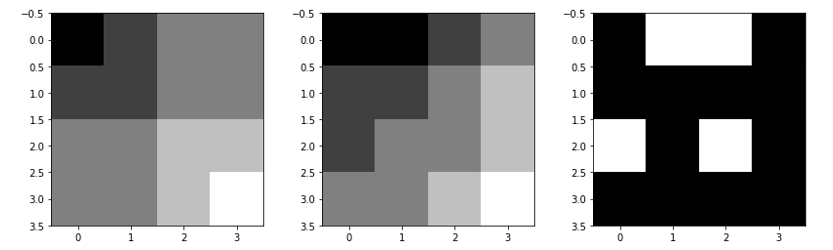


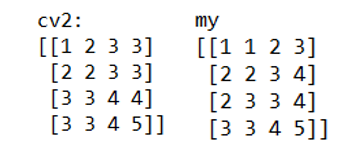
图五：拉普拉斯算子在边缘时的变换

**其次是对卷积核的测试。**我构造了一个小的图像，如图六，图七是使用第二个矩阵，第三种方法下我的图像和opencv的图像，以及两者差值。图八是各自矩阵。

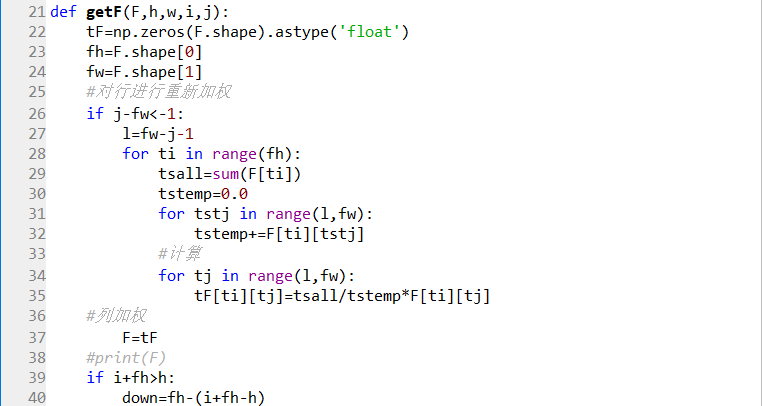
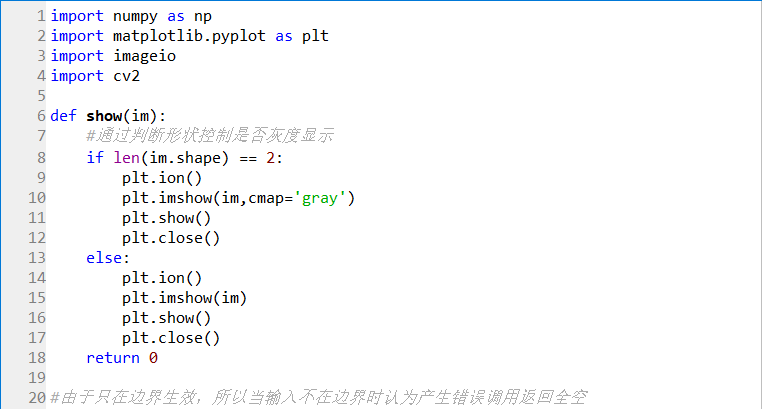
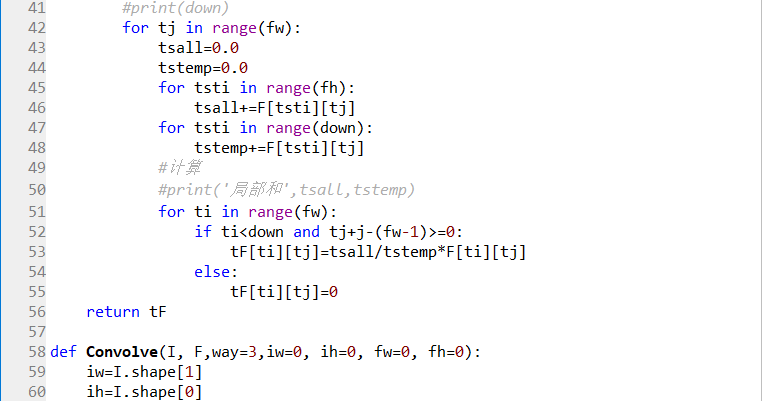
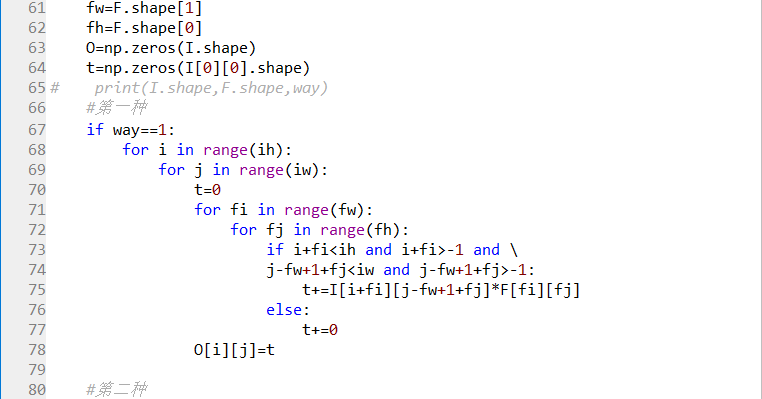
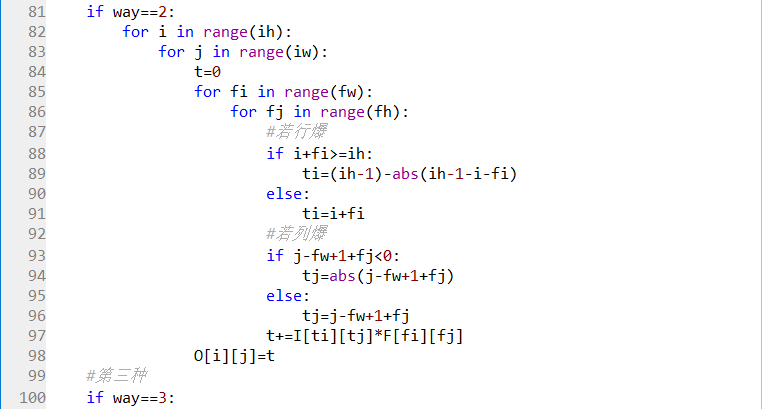
图六：测试图像

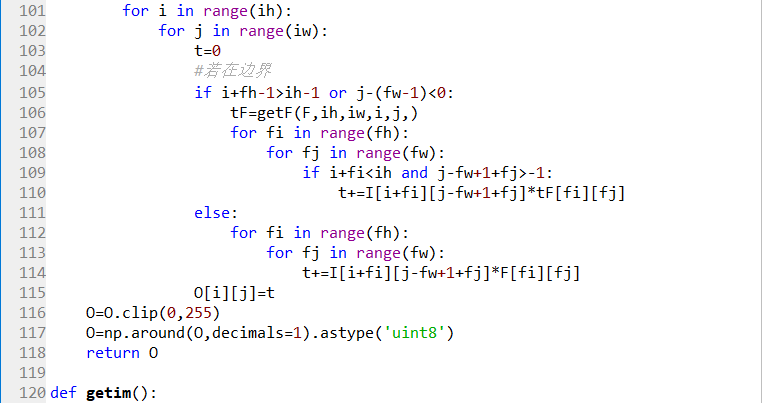
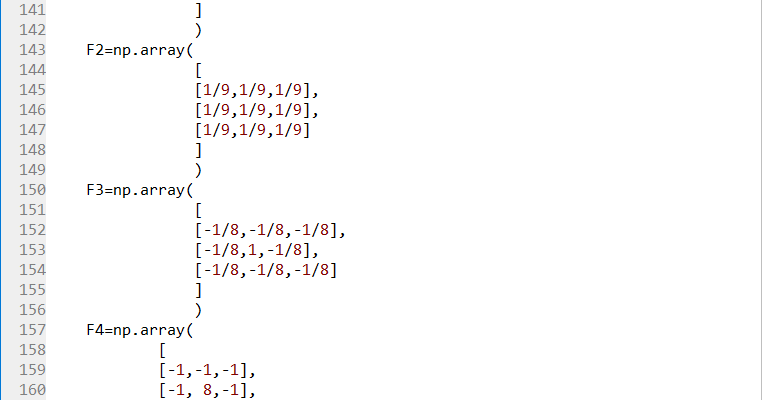
图七：自左至右为：opencv，我的图像，差值



图八：各自矩阵

**四 代码**

import numpy as np

import matplotlib.pyplot as plt

import imageio

import cv2

def show(im):

#通过判断形状控制是否灰度显示

if len(im.shape) == 2:

plt.ion()

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

plt.show()

plt.close()

else:

plt.ion()

plt.imshow(im)

plt.show()

plt.close()

return 0

#由于只在边界生效，所以当输入不在边界时认为产生错误调用返回全空

def getF(F,h,w,i,j):

tF=np.zeros(F.shape).astype('float')

fh=F.shape[0]

fw=F.shape[1]

#对行进行重新加权

if j-fw<-1:

l=fw-j-1

for ti in range(fh):

tsall=sum(F[ti])

tstemp=0.0

for tstj in range(l,fw):

tstemp+=F[ti][tstj]

#计算

for tj in range(l,fw):

tF[ti][tj]=tsall/tstemp\*F[ti][tj]

#列加权

F=tF

#print(F)

if i+fh>h:

down=fh-(i+fh-h)

#print(down)

for tj in range(fw):

tsall=0.0

tstemp=0.0

for tsti in range(fh):

tsall+=F[tsti][tj]

for tsti in range(down):

tstemp+=F[tsti][tj]

#计算

#print('局部和',tsall,tstemp)

for ti in range(fw):

if ti<down and tj+j-(fw-1)>=0:

tF[ti][tj]=tsall/tstemp\*F[ti][tj]

else:

tF[ti][tj]=0

return tF

def Convolve(I, F,way=3,iw=0, ih=0, fw=0, fh=0):

iw=I.shape[1]

ih=I.shape[0]

fw=F.shape[1]

fh=F.shape[0]

O=np.zeros(I.shape)

t=np.zeros(I[0][0].shape)

# print(I.shape,F.shape,way)

#第一种

if way==1:

for i in range(ih):

for j in range(iw):

t=0

for fi in range(fw):

for fj in range(fh):

if i+fi<ih and i+fi>-1 and \

j-fw+1+fj<iw and j-fw+1+fj>-1:

t+=I[i+fi][j-fw+1+fj]\*F[fi][fj]

else:

t+=0

O[i][j]=t

#第二种

if way==2:

for i in range(ih):

for j in range(iw):

t=0

for fi in range(fw):

for fj in range(fh):

#若行爆

if i+fi>=ih:

ti=(ih-1)-abs(ih-1-i-fi)

else:

ti=i+fi

#若列爆

if j-fw+1+fj<0:

tj=abs(j-fw+1+fj)

else:

tj=j-fw+1+fj

t+=I[ti][tj]\*F[fi][fj]

O[i][j]=t

#第三种

if way==3:

for i in range(ih):

for j in range(iw):

t=0

#若在边界

if i+fh-1>ih-1 or j-(fw-1)<0:

tF=getF(F,ih,iw,i,j,)

for fi in range(fh):

for fj in range(fw):

if i+fi<ih and j-fw+1+fj>-1:

t+=I[i+fi][j-fw+1+fj]\*tF[fi][fj]

else:

for fi in range(fh):

for fj in range(fw):

t+=I[i+fi][j-fw+1+fj]\*F[fi][fj]

O[i][j]=t

O=O.clip(0,255)

O=np.around(O,decimals=1).astype('uint8')

return O

def getim():

m=4

n=4

t=np.zeros((m,n)).astype('uint8')

for i in range(m):

for j in range(n):

t[i][j]=i+j

# if i == int(m/2):

# t[i][j]=127

# else:

# t[i][j]=30

print(t)

return t

if \_\_name\_\_=='\_\_main\_\_':

path='C:/Users/19041/Desktop/图像处理/图像处理作业/图像处理练习3/'

im=imageio.imread(path+'1.jpg')

F1=np.array(

[

[1/4,1/4],

[1/4,1/4]

]

)

F2=np.array(

[

[1/9,1/9,1/9],

[1/9,1/9,1/9],

[1/9,1/9,1/9]

]

)

F3=np.array(

[

[-1/8,-1/8,-1/8],

[-1/8,1,-1/8],

[-1/8,-1/8,-1/8]

]

)

F4=np.array(

[

[-1,-1,-1],

[-1, 8,-1],

[-1,-1,-1]

]

)

F=F4#调整这里选择核

im = cv2.cvtColor(im,cv2.COLOR\_BGR2GRAY)#控制转变为灰度图

#im=getim()#调整这里获得构造图像

show(im)

#opencv

tcv=cv2.filter2D(im,-1,F)

print('cv2:')

#print(tcv)

show(tcv)

#我的程序

way=3#调整这里选择边缘处理方式，默认为第三种

t=Convolve(im,F,way)

print(t.shape)

print('my')

#print(t)

show(t)

print('相减')

show(t-tcv)

#保存

iname='1\_'+str(way)+'.jpg'

imageio.imwrite(path+iname,t)

inamecv='1\_'+'cv.jpg'

imageio.imwrite(path+inamecv,tcv)