**第三次实验：图像卷积**

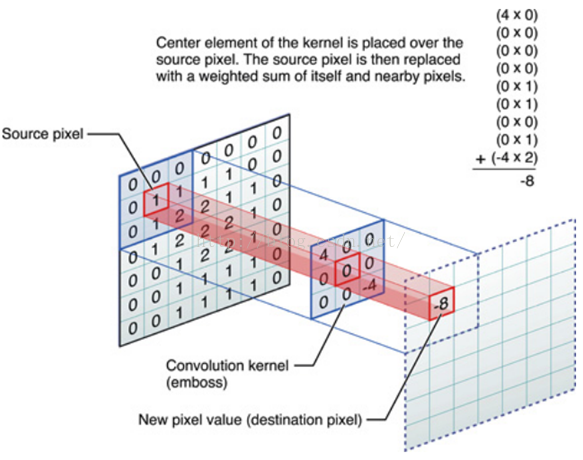
电计1704 孙羽萱 201765130

**一、实验原理**

一幅数字图像可以看作一个二维空间的离散函数，它可以表示为f(x, y)。 假设有对于二维卷积操作函数（可称为**卷积核**）C(u,v)，则会产生输出图像:

**g(x,y) = f(x,y） \* C(u,v)**

对于图像的每一个像素点，计算它的邻域像素和滤波器矩阵的对应元素的乘积，然后加起来，作为该像素位置的值，这个操作就叫卷积。通过设置特定的算子矩阵进行卷积操作,我们可以完成对图像的滤波过程，从而可以提取出图像的特殊特征。



图一：卷积操作图示

我们把图像卷积的计算方法归纳如下：

1.将卷积核的锚点放在该特定位置的像素上（本次实验我们选择右上角作为锚点），同时，核内的其他值与该像素邻域的各像素重合；

2.将卷积核内各值与相应像素值相乘，并将乘积相加；

3.将所得结果放到与锚点对应的像素上；

4.对图像所有像素重复上述过程。

因此，计算的关键是要**根据锚点找到图像像素点与卷积核的对应关系**，编程中采用**四重循环**。在卷积核与图像像素点均有对应时，操作如下所示：

for i in range(ih):

for j in range(iw):

**if i+fh<=ih and j+1-fw>=0: #边界条件**

for k in range(fh):

for m in range(fw):

**out[i][j] += I[i+k][j-m]\*F[k][fw-m-1]**

当对图像四个边的像素进行处理时，必然会出现没有图像的像素与滤波器模板值（即卷积核）对应的情况发生。对于本次实验来说，这种情况发生在图像的左侧和下侧（因为我们的锚点选定在右上角）。

我们采用以下方法**处理边缘像素**：将图像外部滤波器的权重设置为零，同时，为了保留图像的能量，重新**按比例调整图像内部的滤波器的权重，使其总和为1**。

具体可分为三种情况讨论，并根据不同情况设置边界循环条件。

1. 卷积核矩阵下侧没有对应像素点相乘。

**elif i+fh>ih and j+1-fw>=0:**

**Sum=0**

**for k in range(ih-i)**:

for m in range(fw):

**Sum += F[k][fw-m-1]**

out[i][j] += I[i+k][j-m]\*F[k][fw-m-1]

**out[i][j]/= Sum**

（2）卷积核矩阵左侧没有对应像素点相乘。

**elif i+fh<=ih and j+1-fw<0:**

**Sum=0**

for k in range(fh):

**for m in range(j+1):**

**Sum += F[k][fw-m-1]**

out[i][j] += I[i+k][j-m]\*F[k][fw-m-1]

**out[i][j]/= Sum**

（3）卷积核矩阵下侧和左侧均没有对应像素点相乘.

**elif i+fh>ih and j+1-fw<0:**

**Sum=0**

**for k in range(ih-i):**

**for m in range(j+1):**

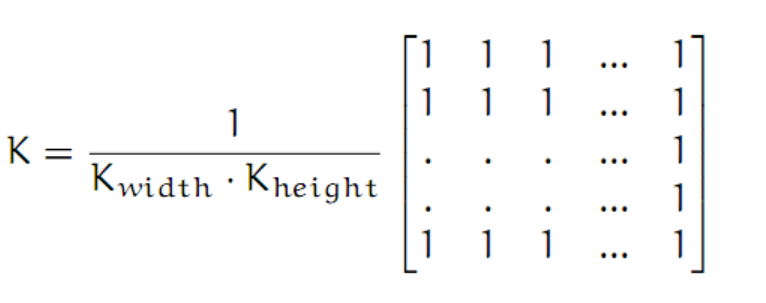
**Sum += F[k][fw-m-1]**

out[i][j] += I[i+k][j-m]\*F[k][fw-m-1]

**out[i][j]/= Sum**

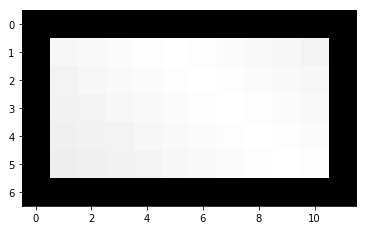
每种边界情况都要在循环中计算有效位置的滤波器值的和Sum，并在退出该像素点的循环之前将像素值除以Sum，从而达到按比例调整滤波器权重的效果。

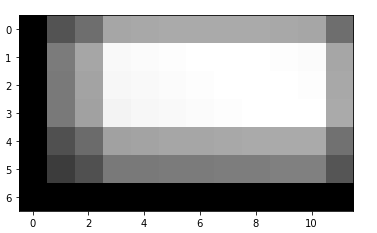
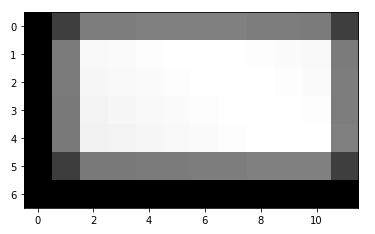
最后我们讨论一下实验中给定的两种滤波器，它们都属于**均值滤波**，滤波后的具体效果我们会在结果分析中进行讨论。给出均值滤波通式如下：



**二、程序输入与输出图像对比**

本次实验首先利用一个测试图像来检验给定的两种滤波器的计算是否正确，然后处理一张带有噪声点的图片，检验实验中给定的均值滤波器的效果。最后利用自己编写的卷积函数对实际图像进行边缘检测，并与opencv自带的卷积函数效果进行对比。



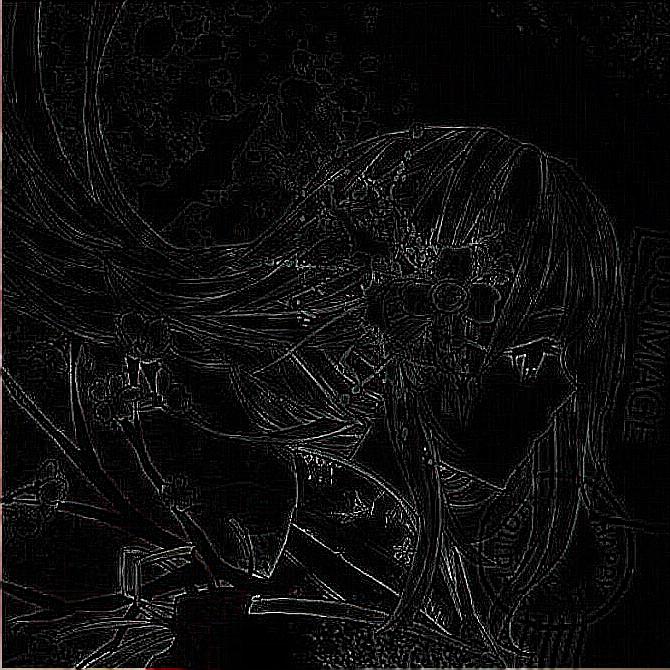


图二：第一组实验——检验滤波器计算结果的正确性



图三：第二组实验——均值滤波（原图、2\*2滤波、3\*3滤波、5\*5滤波）



图四：第三组实验——边缘检测（左图为实验函数，右图为opencv函数）

**三、结果分析**

**1.创建一个测试矩阵（7\*12）来检验给定的两种滤波器的计算是否正确。**

测试矩阵中像素点的值如下：

pic = np.array([[0,0,0,0,0,0,0,0,0,0,0,0],

[0,111,112,113,114,115,114,113,112,111,110,0],

[0,110,111,112,113,114,115,114,113,112,111,0],

[0,109,110,111,112,113,114,115,114,113,112,0],

[0,108,109,110,111,112,113,114,115,114,113,0],

[0,107,108,109,110,111,112,113,114,115,114,0],

[0,0,0,0,0,0,0,0,0,0,0,0]])

经2\*2均值滤波器计算后，矩阵的值如下：

[[ 0 28 56 56 57 57 57 57 56 56 55 28]

[ 0 55 111 112 113 114 114 114 113 112 111 55]

[ 0 55 110 111 112 113 114 114 114 113 112 56]

[ 0 54 109 110 111 112 113 114 114 114 113 56]

[ 0 54 108 109 110 111 112 113 114 114 114 57]

[ 0 27 54 54 55 55 56 56 57 57 57 28]

[ 0 0 0 0 0 0 0 0 0 0 0 0]]

经3\*3均值滤波器计算后，矩阵的值如下：

[[ 0 37 49 74 75 76 76 76 76 75 74 49]

[ 0 55 74 111 112 113 114 114 114 113 112 74]

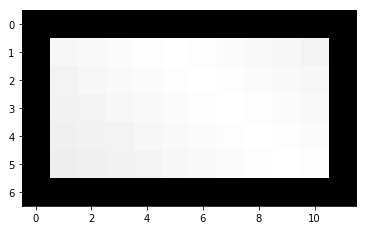
[ 0 54 73 110 111 112 113 114 114 114 113 75]

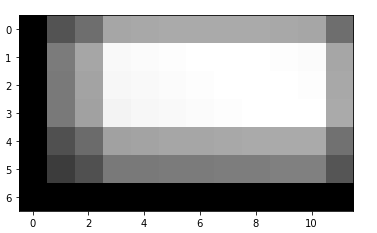
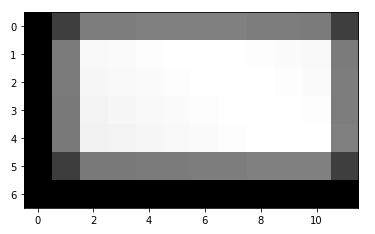
[ 0 54 72 109 110 111 112 113 114 114 114 76]

[ 0 36 48 72 73 74 74 75 76 76 76 51]

[ 0 27 36 54 54 55 55 56 56 57 57 38]

[ 0 0 0 0 0 0 0 0 0 0 0 0]]





图二：第一组实验——检验滤波器计算结果的正确性

经计算器手工检验，卷积结果正确，且从图像直观观察可以感受到以右上角为锚点的处理特点，左侧和下测边缘像素点也没有能量损失。

**2.处理一张带有噪声点的图片，检验实验中给定的均值滤波器的效果。**



图三：第二组实验——均值滤波（原图、2\*2滤波、3\*3滤波、5\*5滤波）

原图像（左上）有很明显的黑色噪声点，之后依次为用2\*2均值滤波器、3\*3滤波器、5\*5滤波器处理后的结果。不难看出，均值滤波后，图像中噪声虽然有所减弱，但是图像变模糊了。并且随着均值滤波器阶数的增大，虽然去噪能力不断增强，但图像也变得越来越模糊。这是因为均值滤波器过滤掉了图像中的高频分量，所以图像的边缘都变模糊了。

**3.利用自己编写的卷积函数对实际图像进行边缘检测，并与opencv自带的卷积函数效果进行对比。**

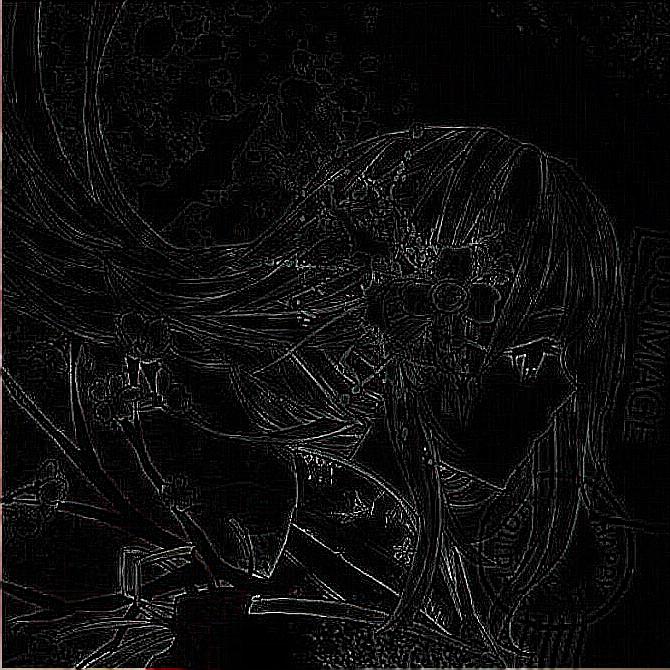
边缘检测所用的滤波器如下：

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

[-1,8,-1],

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

将该滤波器作为参数分别传入本次实验的卷积函数和opencv提供的卷积函数中，得到如下结果：

图四：第三组实验——边缘检测（左图为实验函数，右图为opencv函数）

可以看出两者边缘检测的效果几乎相同，区别是本次实验函数以右上角为卷积核锚点，而opencv函数的卷积核以中心点为锚点，因此opencv所检测的图像边缘比本次实验函数的结果会向右上方有一定程度的偏移。

综上所述，本次图像卷积实验图像处理结果正确、均值滤波效果符合其特点、边缘检测效果与opencv自带函数几乎没有区别，实验较为成功，为完成最终的大作业打下了坚实的基础。

**四、编程代码（python）**

import numpy as np

import math

import imageio #用来做图像读写

from PIL import Image

import matplotlib.pyplot as plt #用来显示图像

import cv2

def Convolve(I, F, iw, ih, fw, fh):

out = np.zeros([ih,iw],dtype=np.float)

for i in range(ih):

for j in range(iw):

if i+fh<=ih and j+1-fw>=0:

for k in range(fh):

for m in range(fw):

out[i][j] += I[i+k][j-m]\*F[k][fw-m-1]

elif i+fh>ih and j+1-fw>=0:

Sum=0

for k in range(ih-i):

for m in range(fw):

Sum += F[k][fw-m-1]

out[i][j] += I[i+k][j-m]\*F[k][fw-m-1]

out[i][j]/= Sum

elif i+fh<=ih and j+1-fw<0:

Sum=0

for k in range(fh):

for m in range(j+1):

Sum += F[k][fw-m-1]

out[i][j] += I[i+k][j-m]\*F[k][fw-m-1]

out[i][j]/= Sum

elif i+fh>ih and j+1-fw<0:

Sum=0

for k in range(ih-i):

for m in range(j+1):

Sum += F[k][fw-m-1]

out[i][j] += I[i+k][j-m]\*F[k][fw-m-1]

out[i][j]/= Sum

#去掉卷积操作后小于0的和大于255的原值,重置为0和255

out = out.clip(0, 255)

out = np.rint(out).astype(np.uint8)

return out

def show(im):

'''

显示RGB图像函数

'''

plt.imshow(im)

plt.show()

plt.close()

def main():

#图像路径信息

path\_work = 'C:/Users/Feather/Desktop/Image\_Processing/Work3/'

file\_in1 = path\_work+'a.jpg'

file\_in2 = path\_work+'girl.jpg'

file\_out1 = path\_work+'a\_2.jpg'

file\_out2 = path\_work+'a\_3.jpg'

file\_out3 = path\_work+'a\_5.jpg'

file\_out = path\_work+'girl\_edges.jpg'

file\_model = path\_work+'girl\_model.jpg'

test = path\_work+'test.jpg'

test1 = path\_work+'test1.jpg'

test2 = path\_work+'test2.jpg'

#创建测试用的滤波器

f1 = np.array([[1/4,1/4],

[1/4,1/4]])

fh1 = f1.shape[0]

fw1 = f1.shape[1]

f2 = np.array([[1/9,1/9,1/9],

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

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

fh2 = f2.shape[0]

fw2 = f2.shape[1]

f3 = np.array([[1/25,1/25,1/25,1/25,1/25],

[1/25,1/25,1/25,1/25,1/25],

[1/25,1/25,1/25,1/25,1/25],

[1/25,1/25,1/25,1/25,1/25],

[1/25,1/25,1/25,1/25,1/25]])

fh3 = f3.shape[0]

fw3 = f3.shape[1]

#创建测试图像矩阵

pic = np.array([[0,0,0,0,0,0,0,0,0,0,0,0],

[0,111,112,113,114,115,114,113,112,111,110,0],

[0,110,111,112,113,114,115,114,113,112,111,0],

[0,109,110,111,112,113,114,115,114,113,112,0],

[0,108,109,110,111,112,113,114,115,114,113,0],

[0,107,108,109,110,111,112,113,114,115,114,0],

[0,0,0,0,0,0,0,0,0,0,0,0]])

ih0 = pic.shape[0]

iw0 = pic.shape[1]

imageio.imwrite(test, pic)

show(pic)

#测试卷积函数

out1=Convolve(pic, f1, iw0, ih0, fw1, fh1)

imageio.imwrite(test1, out1)

show(out1)

out2=Convolve(pic, f2, iw0, ih0, fw2, fh2)

imageio.imwrite(test2, out2)

show(out2)

#读入原始图像

im = Image.open(file\_in1)

im = np.array(im,dtype=np.uint8)

show(im)

ih = im.shape[0]

iw = im.shape[1]

#均值滤波实验

#调用卷积函数分别处理图像的RGB通道

B1 = Convolve(im[:,:,0], f1, iw, ih ,fw1, fh1)

G1 = Convolve(im[:,:,1], f1, iw, ih ,fw1, fh1)

R1 = Convolve(im[:,:,2], f1, iw, ih ,fw1, fh1)

out1 = np.zeros([ih,iw,3],dtype=np.uint8)

out1[:, :, 2] = R1

out1[:, :, 1] = G1

out1[:, :, 0] = B1

imageio.imwrite(file\_out1, out1)

show(out1)

B2 = Convolve(im[:,:,0], f2, iw, ih ,fw2, fh2)

G2 = Convolve(im[:,:,1], f2, iw, ih ,fw2, fh2)

R2 = Convolve(im[:,:,2], f2, iw, ih ,fw2, fh2)

out2 = np.zeros([ih,iw,3],dtype=np.uint8)

out2[:, :, 2] = R2

out2[:, :, 1] = G2

out2[:, :, 0] = B2

imageio.imwrite(file\_out2, out2)

show(out2)

B3 = Convolve(im[:,:,0], f3, iw, ih ,fw3, fh3)

G3 = Convolve(im[:,:,1], f3, iw, ih ,fw3, fh3)

R3 = Convolve(im[:,:,2], f3, iw, ih ,fw3, fh3)

out3 = np.zeros([ih,iw,3],dtype=np.uint8)

out3[:, :, 2] = R3

out3[:, :, 1] = G3

out3[:, :, 0] = B3

imageio.imwrite(file\_out3, out3)

show(out3)

#读入原始图像

im = Image.open(file\_in2)

im = np.array(im,dtype=np.uint8)

show(im)

ih = im.shape[0]

iw = im.shape[1]

#边缘检测滤波实验

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

[-1,8,-1],

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

fh = kernel.shape[0]

fw = kernel.shape[1]

#调用卷积函数分别处理图像的RGB通道

B = Convolve(im[:,:,0], kernel, iw, ih ,fw, fh)

G = Convolve(im[:,:,1], kernel, iw, ih ,fw, fh)

R = Convolve(im[:,:,2], kernel, iw, ih ,fw, fh)

out = np.zeros([ih,iw,3],dtype=np.uint8)

out[:, :, 2] = R

out[:, :, 1] = G

out[:, :, 0] = B

show(out)

imageio.imwrite(file\_out, out)

#调用opencv进行边缘检测滤波，对比效果

edges=cv2.filter2D(im,-1,kernel)

imageio.imwrite(file\_model, edges)

show(edges)

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

main()

