

图像处理基础

（作业3）



学 院（系）： 电子信息与电气工程学部

专 业： 计算机科学与技术

班 级： 电计1704

学 生 姓 名： 谢玉宁

学 号： 201792260

完 成 日 期： 2020年3月28日

实验目标：编写卷积函数

实现原理:

卷积 的计算方法：

1. 将核的锚点放在该特定位置的像素上，同时，核内的其他值与该像素邻域的各像素重合；
2. 将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。

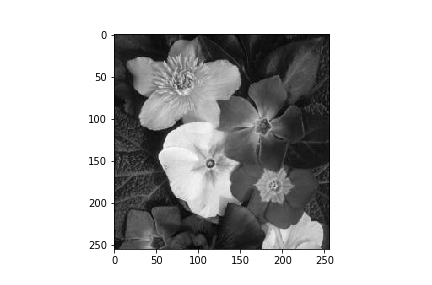
本实验中锚点为卷积核的右上角。

实验中为了保证卷积后输出图像与原图像大小相同并保留边缘像素点信息，需要将原图像边缘进行填充（padding）。如果没有加padding的话，输入图片最边缘的像素点信息只会被卷积核操作一次，但是图像中间的像素点会被扫描到很多遍，那么就会在一定程度上降低边界信息的参考程度，但是在加入padding之后，在实际处理过程中就会从新的边界进行操作，就从一定程度上解决了这个问题。

实验给出三种填充方式：置零法，图像沿边沿作镜像对称，调整滤波器的值。

输入与输出对比:

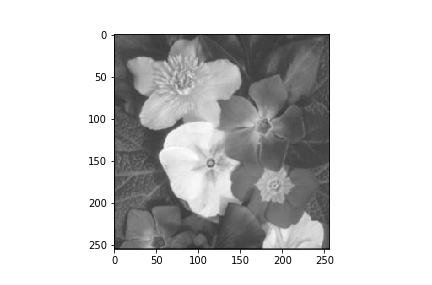
输入图像：



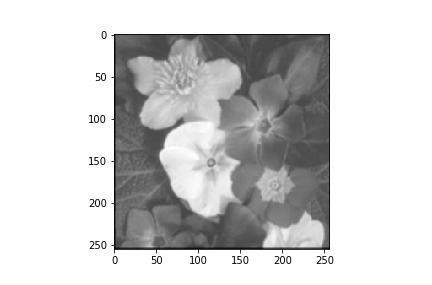
图表 1 输入图像

输出图像：

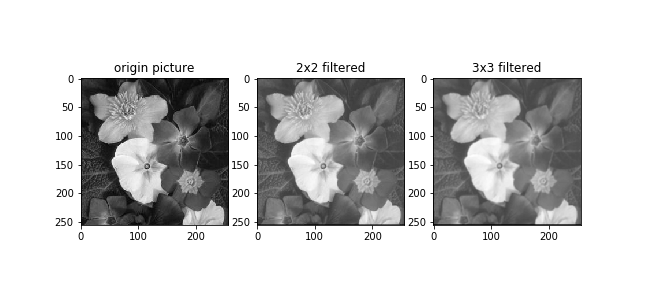
滤波器卷积后图像（置0法padding）：



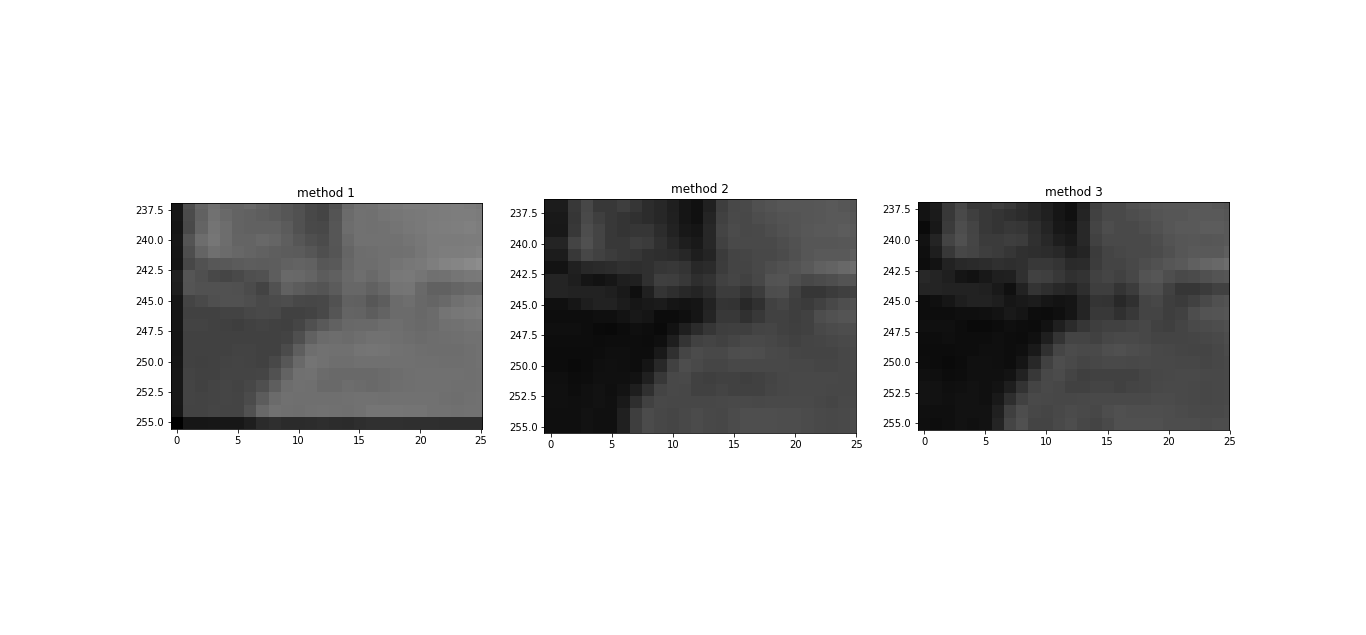
滤波器卷积后的图像（置0法padding）：



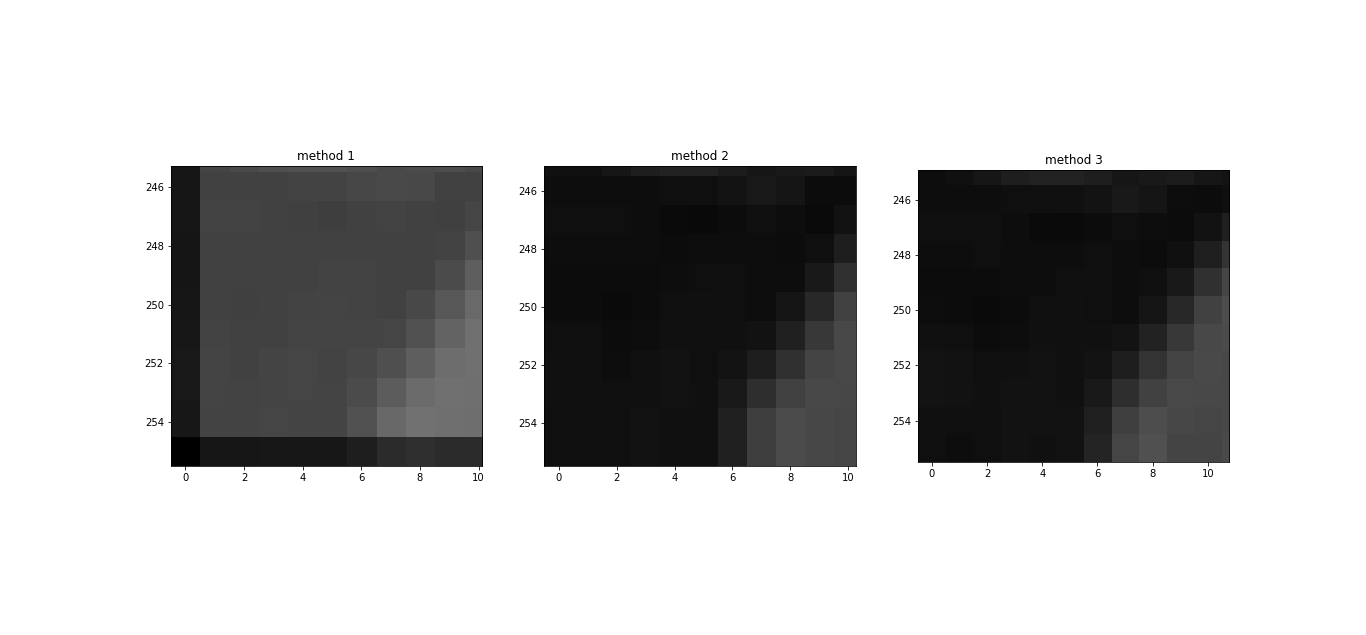
对比图：



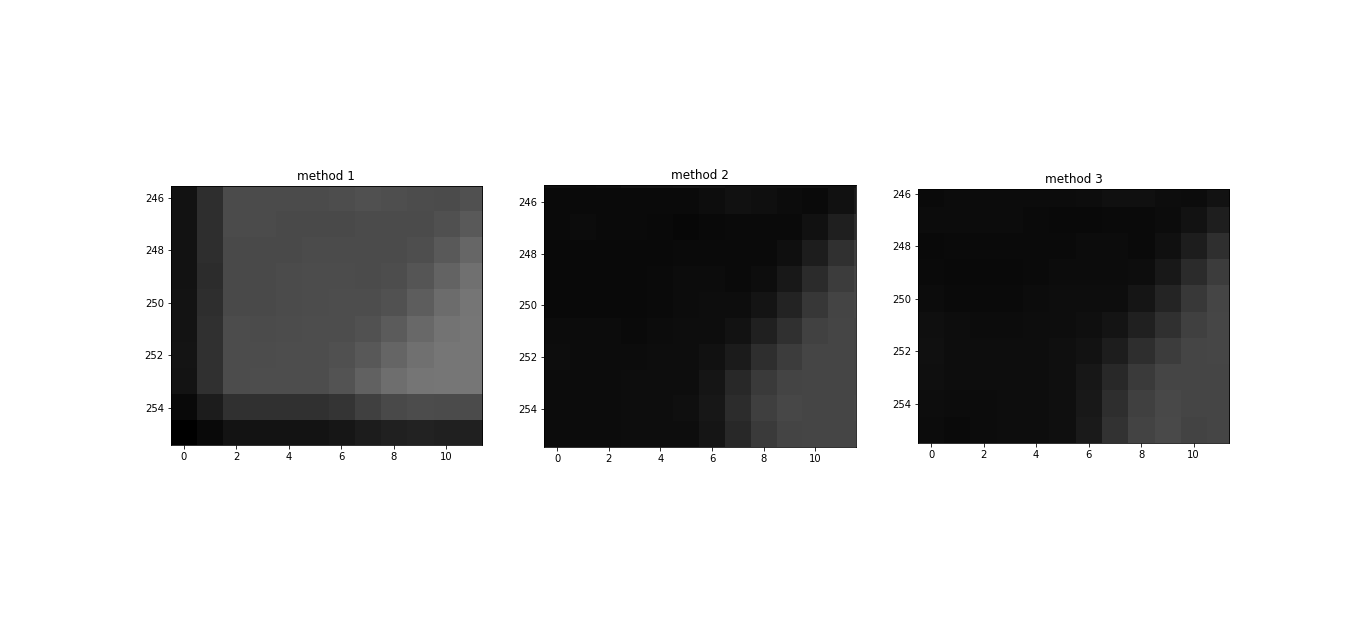
下面以展示三种padding方法（以2x2filter为例）：



图表 2 3种padding效果对比图1



图表 3 3种padding效果对比图2



图表 4 3种padding效果对比图3

结果分析:

实验最终成功编写滤波器函数并实现图片卷积操作。通过用给定的2\*2卷积核与3\*3卷积核对滤波器函数进行测试，两个滤波器均为低通滤波（均值滤波器），这个滤波是一个平滑图像的滤波器，它用一个点邻域内像素的平均灰度值来代替该点的灰度，由卷积出来的图像可见，两个滤波器作用均是使原图像变模糊，结果符合预期。其中3\*3滤波器较2\*2滤波器效果更明显，是因为3\*3滤波器每个像素点灰度计算使用了较2\*2滤波器更多的相邻像素点灰度信息，因此图像更加平滑模糊。

在卷积操作中，为了使新图像与原图像大小相同并增加边缘信息的参考程度，需要对原图像边缘进行填充（padding），实验给出了三种padding方法：

1. 图像边缘置0

这种方法会让卷积出来的边缘产生失真，由上述3张卷积图像边缘放大对比图可见（method 1），置0法padding卷积出来的图像边缘严重失真，出现了黑色条带，原因是0参与了平均，使得边缘灰度变低，因此较图片内部黑。

1. 图像沿边沿作镜像对称

这种方法能较好的避免图像边缘的失真问题，很好的保留了原图像边缘的信息，见上述3张卷积图像边缘放大对比图（method 2），当卷积核较大而图像较小时，就会导致原图像边缘信息冗余，但是一般图像远大于卷积核，所以不需要考虑冗余的少量边缘信息。

1. 调整滤波器模板的值

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。这样做的好处是，能够很大程度的保留原图像边缘像素点灰度信息而不产生冗余，效果见上述3张卷积图像边缘放大对比图（method 3）

结论：方法1填充后卷积的图像效果最差，图像边缘失真严重，方法2与方法3填充后卷积的图像效果较好。

代码：

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

"""

Created on Wed Mar 25 15:47:48 2020

@author: Lenovo

"""

import numpy as np

import matplotlib.pyplot as plt

import matplotlib.image as pli

im1\_path = r'C:\Users\Lenovo\Desktop\flowergray.jpg'

im1 = pli.imread(im1\_path)

# mode=0

#method1,2,3分别是实验中三种padding方式

def Convolve(I,F,iw,ih,fw,fh,Kernel\_mode = 0):

Conv\_I = np.zeros\_like(I)

if Kernel\_mode == 0:

Image\_pad = np.pad(I, ((0,fh-1),(fw-1,0) ), 'constant', constant\_values=0)

func = lambda x,y:np.sum(Image\_pad[x:x+fh,y:y+fw]\*F)

for i in range(ih):

for j in range(iw):

Conv\_I[i,j] = func(i,j)

elif Kernel\_mode == 1:

Image\_pad = np.pad(I, ((0,fh-1),(fw-1,0) ), 'reflect')

func = lambda x,y:np.sum(Image\_pad[x:x+fh,y:y+fw]\*F)

for i in range(ih):

for j in range(iw):

Conv\_I[i,j] = func(i,j)

elif Kernel\_mode == 2:

Image\_pad = np.pad(I, ((0,fh-1),(fw-1,0) ), 'constant', constant\_values=0)

func = lambda x,y:np.sum(Image\_pad[x:x+fh,y:y+fw]\*F)

for i in range(ih):

for j in range(iw):

bottom = np.clip(ih-i,0,3)

left = np.clip(fw-j-1,0,3)

s = np.sum(F[:bottom,left:])

Conv\_I[i,j] = int(np.round(func(i,j)/s))

return Conv\_I

F = np.array([[0,-0.25,0],

[-0.25,1,-0.25],

[0,-0.25,0]])

FF = np.array([[1/4]\*4]).reshape(2,2)

FFF = np.array([[1/9]\*9]).reshape(3,3)

ih,iw = im1.shape

fh,fw = FF.shape

O = Convolve(im1,FF,iw,ih,fw,fh,Kernel\_mode = 0)/255

fh,fw = FFF.shape

OO = Convolve(im1,FFF,iw,ih,fw,fh,Kernel\_mode = 0)/255

plt.subplot(1,3,1) #原图

plt.imshow(im1,cmap='gray')

plt.title('origin picture')

plt.subplot(1,3,2) #filter2\*2

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

plt.title('2x2 filtered')

plt.subplot(1,3,3)#filter3\*3

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

plt.title('3x3 filtered')