实验五 滤波处理

一、 实验目的

进一步了解 Python 软件/语言，学会使用 Python 对图像作滤波处理，掌握滤波算法，体会滤波效果。

了解几种不同滤波方式的使用和使用的场合，培养处理实际图像的能力，并为课堂教学提供配套的实践机会。

二、实验内容

1.在均值滤波核值下，对图像进行卷积操作，利用 cv.filter 方法进行。

2.使用均值滤波，中值滤波，高斯滤波对带噪声图像（lena.bmp）进行处理，比较图像处理效果。

3.使用空域滤波方法查找出图像边缘，Sobel 算子和 Laplacian 算子

4.对图像进行傅立叶变换，在 Numpy 中的 FFT 包可以帮助我们实现快速傅里叶变换，使用函数 np.fft.fft2()可以对信号进行频率转换,再用函数 np.fft.ifftshift()将结果沿两个方向平移 N/2。然后频率变换之后，我们构建振幅谱。然后在进行频域变换，高通滤波和重建图像（DFT 的逆变换）。首先进行掩膜操作出去低频分量，然后再使用函数 np.fft.ifftshift() 进行逆平移操作，所以现在直流分量又回到左上角了，然后使用函数 np.ifft2() 进行 FFT 逆变换，然后输出傅立叶变换处理后的图像。

5.利用 OpenCV 为我们提供了函数：cv2.matchTemplate()进行图片匹配。

三、实验原理与步骤

（一）cv.filter2D 函数进行滤波

可以采用 2 维卷积对 2D 图像实施平滑与锐化等操作等。平滑处理帮助我们去除噪声，模糊图像。锐化处理帮助我们找到图像的边缘。可以利用 OpenCV 提供的函数 cv.filter2D() 对一幅图像进行卷积操作。下面采用一个 5x5 平均滤波器对图像进行卷积处理。

$$K = \frac{1}{25}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

操作如下：将核放在图像的一个像素 A 上，求与核对应的图像上 25（5x5）个像素的和，在取平均数，用这个平均数替代像素 A 的值。重复以上操作直到将图像的每一个像素值都更新一遍。

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
img = cv2.imread('lena.bmp')
kernel = np.ones((5,5),np.float32)/25
#cv.filter2D(src, dst, kernel, anchor=(-1, -1))
#ddepth  – desired depth of the destination image;
#if it is negative, it will be the same as src.depth();
#the following combinations of src.depth() and ddepth are supported:
#src.depth() = CV_8U, ddepth = -1/CV_16S/CV_32F/CV_64F
#src.depth() = CV_16U/CV_16S, ddepth = -1/CV_32F/CV_64F
#src.depth() = CV_32F, ddepth = -1/CV_32F/CV_64F
#src.depth() = CV_64F, ddepth = -1/CV_64F
#when ddepth=-1, the output image will have the same depth as the source.
dst = cv2.filter2D(img,-1,kernel)
plt.subplot(121),plt.imshow(img),plt.title('Original')
plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(dst),plt.title('Averaging')
plt.xticks([]), plt.yticks([])
plt.show()
```

结果显示：

（2）cv2.blur 函数进行均值滤波

可以使用函数 cv2.blur() 和 cv2.boxFilter() 来完均值滤波任务。与 filter2D 不同的是，我们需要设定卷积窗口的尺寸（宽和高），即可进行卷积处理。

进行滤波的代码实现如下：

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
img = cv2.imread('lena.bmp')
blur3 = cv2.blur(img,(3,3))
blur5 = cv2.blur(img,(5,5))
blur7 = cv2.blur(img,(7,7))
plt.subplot(221),plt.imshow(img),plt.title('Original')
plt.xticks([]), plt.yticks([])
plt.subplot(222),plt.imshow(blur3),plt.title('Blurred3*3')
plt.xticks([]), plt.yticks([])
plt.subplot(223),plt.imshow(blur5),plt.title('Blurred 5*5')
plt.xticks([]), plt.yticks([])
```

plt.subplot(224),plt.imshow(blur7),plt.title('Blurred 7*7')

plt.xticks([]), plt.yticks([])

plt.show()

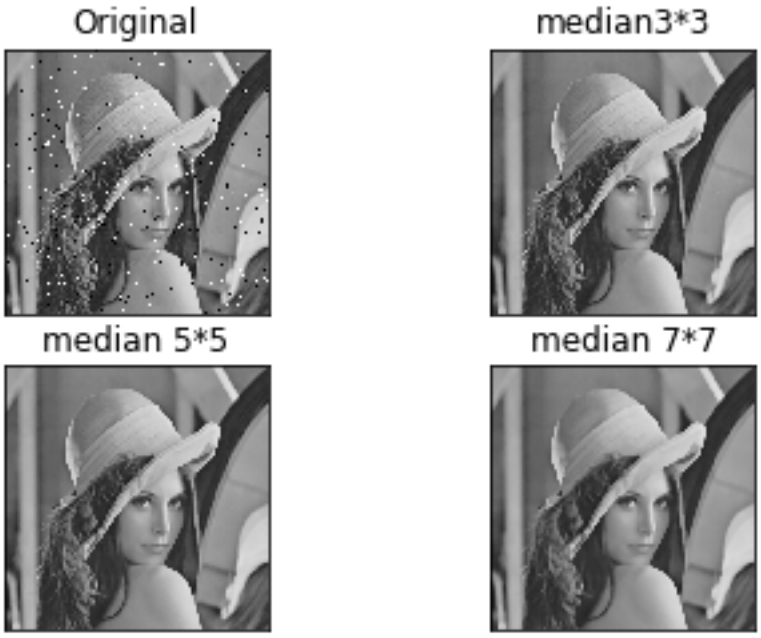
结果显示：

（3）中值滤波

前面的滤波器都是用计算得到的一个平均值来取代中心像素的值，而中值滤波一种统计滤波器，采用中位数来取代模板中原点对应位置的值。

median = cv2.medianBlur(img,5)

结果显示：



（4）高斯模糊：

高斯模式本质上是一种算术均值滤波，模板中的系数是对某个高斯函数离散化的结果。在 OpenCV 中可以用 cv2.GaussianBlur()实现。指定高斯核的宽和高（必须是奇数）。以及高斯函数沿 X，Y 方向的标准差。如果只指定了 X 方向的标准差，Y 方向也会取相同值。如果两个标准差都是 0，那么函数会根据核函数的大小自己计算。高斯滤波可以有效的从图像中去除高斯噪声。

blur = cv2.GaussianBlur(img,(5,5),0)

结果示意：

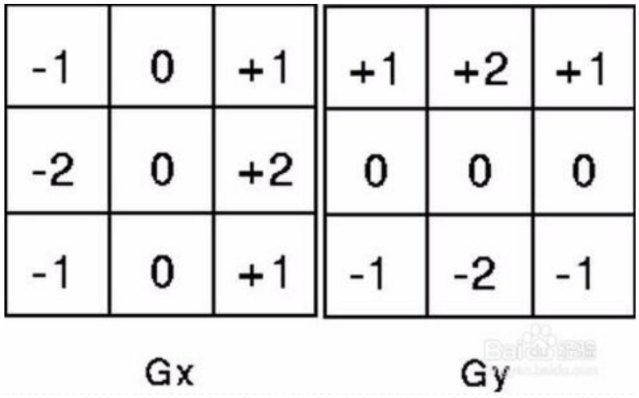
（若原图是高斯噪声，可以比较一下中止滤波和高斯滤波效果的好坏。）

（二）空域图像锐化

Sobel 和 Laplacian 是常见的处理方法锐化方法， 其实就是求一阶或二阶导数。

Sobel 算子

该算子都可以设定求导的方向（xorder 或 yorder）和使用的卷积核的大小（ksize）。



Laplacian 算子

拉普拉斯算子可以使用二阶导数的形式定义，可假设其离散实现类似于二阶 Sobel 导数。计算公式如下：

$$\triangle src = \frac{\partial^2 src}{\partial x^2} + \frac{\partial^2 src}{\partial y^2}$$

拉普拉斯滤波器使用的卷积核：

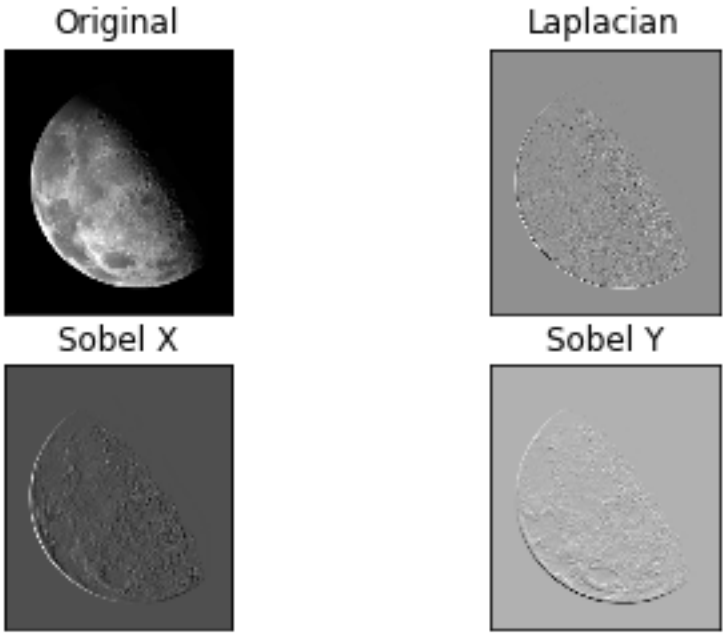
$$kernel = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

下面的代码分别使用以上 2 中种滤波器对同一幅图进行操作。使用的卷积核都是 5x5 的。

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
img=cv2.imread('lena.jpg',0)
#cv2.CV_64F 输出图像的深度（数据类型），可以使用-1, 与原图像保持一致 np.uint8
laplacian=cv2.Laplacian(img,cv2.CV_64F)
# 参数 1,0 为只在 x 方向求一阶导数，最大可以求 2 阶导数。
sobelx=cv2.Sobel(img,cv2.CV_64F,1,0,ksize=5)
# 参数 0,1 为只在 y 方向求一阶导数，最大可以求 2 阶导数。
```

```
sobely=cv2.Sobel(img,cv2.CV_64F,0,1,ksize=5)
plt.subplot(2,2,1),plt.imshow(img,cmap = 'gray')
plt.title('Original'), plt.xticks([]), plt.yticks([])
plt.subplot(2,2,2),plt.imshow(laplacian,cmap = 'gray')
plt.title('Laplacian'), plt.xticks([]), plt.yticks([])
plt.subplot(2,2,3),plt.imshow(sobelx,cmap = 'gray')
plt.title('Sobel X'), plt.xticks([]), plt.yticks([])
plt.subplot(2,2,4),plt.imshow(sobely,cmap = 'gray')
plt.title('Sobel Y'), plt.xticks([]), plt.yticks([])
plt.show()
```

示例结果：



（提示：想不想看看从原图减去 laplacian 操作结果后的图像？如何完成？）

（三）图像的傅立叶变换

1.应用傅立叶变换进行图像处理

傅里叶变换是线性系统分析的有力工具，它能够定量地分析诸如数字化系统、采样点、电子放大器、卷积滤波器、噪声和显示点等的作用。2 傅立叶（Fourier）变换的定义

对于二维信号，二维 Fourier 变换定义为：

$$F(u,v)=\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}f(x,y)e^{-j2\pi(ux+uy)}dxdy$$

逆变换：

$$f(x,y)=\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}F(u,v)e^{j2\pi(ux+uy)}dudv$$

二维离散傅立叶变换为：

$$F(m,n)=\frac{1}{N}\sum_{i=0}^{N-1}\sum_{k=0}^{N-1}f(i,k)e^{-j2\pi(m\frac{i}{N}+n\frac{k}{N})}$$

逆变换：

$$f(i,k)=\frac{1}{N}\sum_{m=0}^{N-1}\sum_{n=0}^{N-1}F(m,n)e^{j2\pi(m\frac{i}{N}+n\frac{k}{N})}$$

图像的傅立叶变换与一维信号的傅立叶变换变换一样，有快速算法，具体参见参考书目，有关傅立叶变换的快速算法的程序不难找到。实际上，现在有实现傅立叶变换的芯片，可以实时实现傅立叶变换。

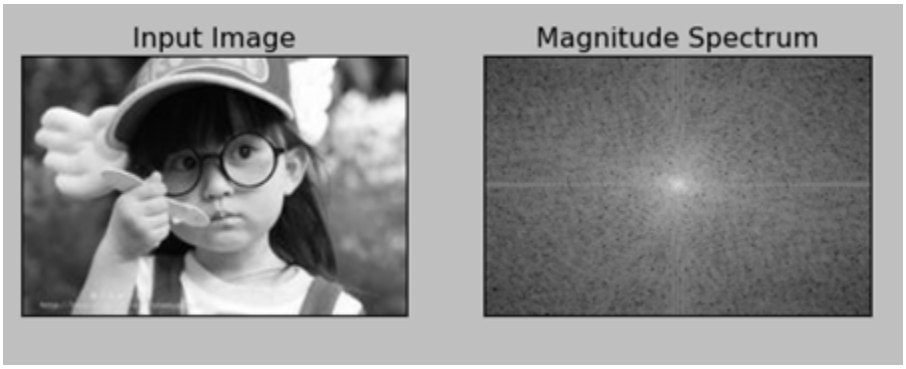
Numpy 中的傅里叶变换

首先我们看看如何使用 Numpy 进行傅里叶变换。Numpy 中的 FFT 包可以帮助我们实现快速傅里叶变换。函数 np.fft.fft2()可以对信号进行频率转换，输出结果是一个复数数组。本函数的第一个参数是输入图像，要求是灰度格式。第二个参数是可选的，决定输出数组的大小。如果输出结果比输入图像大，输入图像就需要在进行 FFT 前补 0。如果输出结果比输入图像小的话，输入图像就会被切割。现在我们得到了结果，频率为 0 的部分（直流分量）在输出图像的左上角。如果想让它（直流分量）在输出图像的中心，我们还需要将结果沿两个方向平移 N/2。函数 np.fft.ifftshift() 可以帮助我们实现这一步。进行完频率变换之后，就可以构建振幅谱了。

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
img = cv2.imread('messi5.jpg',0)
f = np.fft.fft2(img)
fshift = np.fft.fftshift(f)
#构建振幅
```

```
magnitude_spectrum = 20*np.log(np.abs(fshift))
plt.subplot(121),plt.imshow(img, cmap = 'gray')
plt.title('Input Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(magnitude_spectrum, cmap = 'gray')
plt.title('Magnitude Spectrum'), plt.xticks([]), plt.yticks([])
plt.show()
```

结果显示：

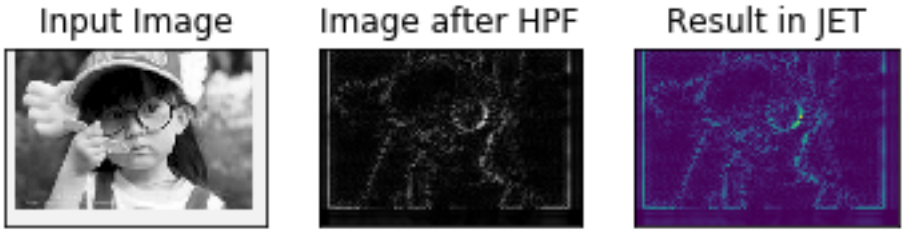


现在我们可以进行频域变换了，我们就可以在频域对图像进行一些操作了，例如高通滤波和重建图像（DFT 的逆变换）。比如我们可以使用一个 60x60 的矩形窗口对图像进行掩模操作从而去除低频分量。然后再使用函数 np.fft.ifftshift() 进行逆平移操作，所以现在直流分量又回到左上角了，左后使用函数 np.ifft2() 进行 FFT 逆变换。

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
img = cv2.imread('test2.jpg',0)
f = np.fft.fft2(img)
fshift = np.fft.fftshift(f)
 rows, cols = img.shape
 crow,ccol = rows/2 , cols/2
 fshift[crow-30:crow+30, ccol-30:ccol+30] = 0
 f_ishift = np.fft.ifftshift(fshift)
 img_back = np.fft.ifft2(f_ishift)
 # 取绝对值
 img_back = np.abs(img_back)
```

```
plt.subplot(131),plt.imshow(img, cmap = 'gray')
plt.title('Input Image'), plt.xticks([]), plt.yticks([])
plt.subplot(132),plt.imshow(img_back, cmap = 'gray')
plt.title('Image after HPF'), plt.xticks([]), plt.yticks([])
plt.subplot(133),plt.imshow(img_back)
plt.title('Result in JET'), plt.xticks([]), plt.yticks([])
plt.show()
```

结果：



（四）图像匹配

（1）目标

1. 使用模板匹配在一幅图像中查找目标,且会使用多对象的模版匹配。
2. 函数：cv2.matchTemplate()，cv2.minMaxLoc()，cv.imMaxLoc()

（2）原理

模板匹配是用来在一副大图中搜寻查找模版图像位置的方法。OpenCV 为我们提供了函数：cv2.matchTemplate()。和 2D 卷积一样，它也是用模板图像在输入图像（大图）上滑动，并在每一个位置对模板图像和与其对应的输入图像的子区域进行比较。OpenCV 提供了几种不同的比较方法（细节请看文档）。返回的结果是一个灰度图像，每一个像素值表示了此区域与模板的匹配程度。

代码实现：

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
img = cv2.imread('girl.jpg',0)
img2 = img.copy()
```

```
template = cv2.imread('eye.png',0)
w, h = template.shape[::-1]
# All the 6 methods for comparison in a list
methods = ['cv2.TM_CCOEFF', 'cv2.TM_CCOEFF_NORMED', 'cv2.TM_CCORR',
'cv2.TM_CCORR_NORMED', 'cv2.TM_SQDIFF', 'cv2.TM_SQDIFF_NORMED']
for meth in methods:
img = img2.copy()
#exec 语句用来执行储存在字符串或文件中的 Python 语句。
# 例如，我们可以在运行时生成一个包含 Python 代码的字符串，然后使用 exec 语句执
行这些语句。
#eval 语句用来计算存储在字符串中的有效 Python 表达式
method = eval(meth)
# Apply template Matching
res = cv2.matchTemplate(img,template,method)
min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(res)
# 使用不同的比较方法，对结果的解释不同
# If the method is TM_SQDIFF or TM_SQDIFF_NORMED, take minimum
if method in [cv2.TM_SQDIFF, cv2.TM_SQDIFF_NORMED]:
    top_left = min_loc
else:
    top_left = max_loc
bottom_right = (top_left[0] + w, top_left[1] + h)
cv2.rectangle(img,top_left, bottom_right, 255, 2)
plt.subplot(121),plt.imshow(res,cmap = 'gray')
plt.title('Matching Result'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(img,cmap = 'gray')
plt.title('Detected Point'), plt.xticks([]), plt.yticks([])
plt.suptitle(meth)
plt.show()
```

结果显示：

思考题：

1.请自行完成在不同均值滤波核值下的操作（以上是在 5x5 模版下实行），另外可尝试自编程完成卷积运算函数用于滤波。

2.LPF（低通滤波）将高频部分去除实现傅立叶变换.

3. 对 child.jpg 图像的灰度图像进行高通、带通、低通滤波后的结果分别表示彩色图像 B、G、R 分量。根据颜色分析观察图像中空间频率变化快慢的区域。

 child.jpg  eys.png

lena.jpg

lenawithnoise.bmp