实验五 空间域平滑滤波

1. 滤波基础

1.1 添加图像边框

进行卷积运算的时候，会面临一个问题，即边缘像素的值如何处理。常用的方法有两种：一是保留边缘处的像素值，不做任何的操作；二是将图像扩充一个边缘，卷积运算后再将边缘去除。本实验选择第二种方法。

在Python中为图像添加边框就要用到cv2.copyMakeBorder()函数，其函数原型为：

copyMakeBorder(src, top, bottom, left, right, borderType[, dst[, value]]) -> dst

其中：

(1) src 输入图像；

(2) dst 输出图像；

(3) top、bottom、left、right 为图像中在上、下、左和右边框各添加的像元数量；

(4) borderType 扩充边框的类型。边框的类型有：

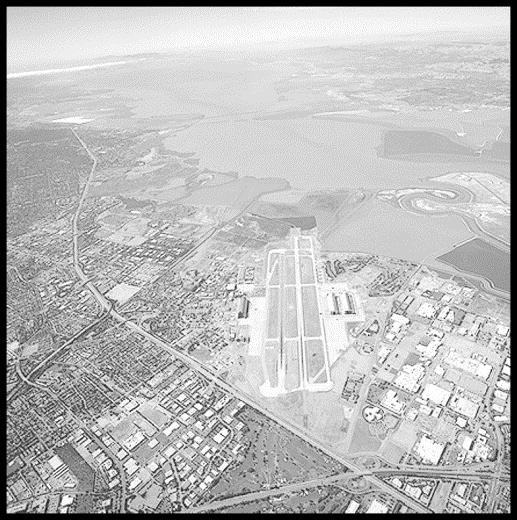
cv2.BORDER\_CONSTANT，添加一个常数值作为边框，此时还需要指定value的值；

cv2.BORDER\_REFLECT，添加的边框数值是边缘像素的镜像，如123|321；

cv2.BORDER\_REFLECT\_101，添加的边框数值是边缘像素的镜像，但不包含最边缘的像素，如123|21；

cv2.BORDER\_DEFAULT，效果与cv2.BORDER\_REFLECT\_101相同；

cv2.BORDER\_REPLICATE，复制最后一个像元的像素值。

(a) (b)

(c) (d)

(e) (f)

1. 原图像和不同类型边框扩充，(a)原图像，(b) cv2.BORDER\_CONSTANT，

(c) cv2.BORDER\_REFLECT，(d) cv2.BORDER\_REFLECT\_101，

(e) cv2.BORDER\_DEFAULT，(f) cv2.BORDER\_REPLICATE

代码：

import cv2

img = cv2.imread('city.tif', cv2.IMREAD\_GRAYSCALE)

cv2.imshow('input', img)

# 添加不同类型边框，并显示结果图像

img1 = cv2.copyMakeBorder(img, 10, 10, 10, 10, cv2.BORDER\_CONSTANT)

cv2.imshow('cv2.BORDER\_CONSTANT value=10', img1)

img2 = cv2.copyMakeBorder(img, 10, 10, 10, 10, cv2.BORDER\_REFLECT)

cv2.imshow('cv2.BORDER\_REFLECT', img2)

img3 = cv2.copyMakeBorder(img, 10, 10, 10, 10, cv2.BORDER\_REFLECT\_101)

cv2.imshow('cv2.BORDER\_REFLECT\_101', img3)

img4 = cv2.copyMakeBorder(img, 10, 10, 10, 10, cv2.BORDER\_DEFAULT)

cv2.imshow('cv2.BORDER\_DEFAULT', img4)

img5 = cv2.copyMakeBorder(img, 10, 10, 10, 10, cv2.BORDER\_REPLICATE)

cv2.imshow('cv2.BORDER\_REPLICATE', img5)

cv2.waitKey(0)

cv2.destroyAllWindows()

**卷积运算结束后，只需要用获取子图像的方法即可去除图像的边框。**

1.2 定义模板

在Python和Numpy中，数据的排列结构有两种形式：一个是阵列类型，用numpy.array()来定义；另一个是矩阵类型，用numpy.mat()来定义。两种数据形式的主要区别是运算的规则，矩阵类型的数据是按线性代数中矩阵的运算规则来计算，而阵列类型数据是将参与运算的阵列按相应位置进行四则运算。在数字图像处理中，一般用的是阵列，其函数原型为：

numpy.array(object, dtype=None, copy=True, order=None, subok=False, ndmin=0)

其中：

(1) object 所要定义的数据；

(2) dtype 所定义数据的类型。

(3) order 指定数据存储形式。

‘C’: row major （默认）

‘F’: column major

‘A’: unchanged

‘K’: unchanged

常用的阵列有全0阵列和全1阵列，分别由numpy.zeros()和numpy.ones()函数得到。其原型分别为：

numpy.zeros(shape, dtype=float, order='C')

numpy.ones(shape, dtype=None, order='C')

其中：

(1) shape 阵列的行和列(形状)；

(2) dtype 阵列中数据的类型。

代码：定义模板

import numpy as np

template1 = np.ones((3, 3), dtype = np.float32) / 9.0

template2 = np.array([[1, 1, 1], [1, 2, 1], [1, 1, 1]], dtype = np.float32) / 10.0

template3 = np.array([[1, 2, 1], [2, 4, 2], [1, 2, 1]], dtype = np.float32) / 16.0

print(template1)

print(template2)

print(template3)

输出：

[[ 0.11111111 0.11111111 0.11111111]

[ 0.11111111 0.11111111 0.11111111]

[ 0.11111111 0.11111111 0.11111111]]

[[ 0.1 0.1 0.1]

[ 0.1 0.2 0.1]

[ 0.1 0.1 0.1]]

[[ 0.0625 0.125 0.0625]

[ 0.125 0.25 0.125 ]

[ 0.0625 0.125 0.0625]]

1.3 卷积运算

卷积、叠积、摺积或旋积，是通过两个函数f和g生成第三个函数的一种数学算子，表征函数f与经过翻转和平移的g的重叠部分的面积。如果将参加卷积的一个函数看作区间的指示函数，卷积还可以被看作是“移动平均”的推广。

代码：滑动平均

import numpy as np

import matplotlib.pyplot as plt

# 定义原始数据

ary = np.array([183, 183, 175, 159, 159, 160, 151, 155, 166, 159, 166, 164, 159, 164, 167, 163, 156, 163, 157, 155], dtype = np.float64)

template = np.array([1, 1, 1], np.float64) / 3.0 # 定义模板

ans = ary.copy() # 定义数组，存储运算结果

# 卷积运算

for i in range(1, len(ary)-1, 1):

temp = ary[i-1: i+2]

ans[i] = np.sum(temp \* template)

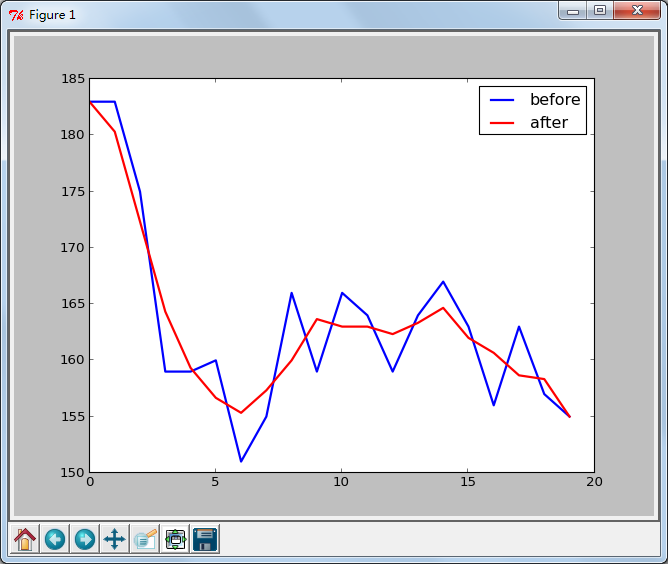
# 绘制原始数据与结果

plt.plot(range(20), ary, linewidth = 2, color = 'b')

plt.plot(range(20), ans, linewidth = 2, color = 'r')

plt.legend(('before', 'after'), loc = 'best')

plt.show()



1. 原数据(蓝)与滑动平均数据(红)

1.4 添加椒盐噪声

椒盐噪声在图像上表现为白色和黑色的污染点，添加椒盐噪声的步骤为：

(1) 利用随机函数，生成两个数值作为当前噪声点的坐标位置；

(2) 利用随机函数，生成一个数值，决定当前的噪声是椒噪声还是盐噪声；

(3) 将当前噪声坐标位置上的像元值替换成噪声的DN值；

1. 添加椒盐噪声图像

代码：添加椒盐噪声

def addPepperSaltNoise(src, num):

dst = src.copy() # 定义数组，存储结果

# 添加椒盐噪声

rows, cols = src.shape[0: 2]

for k in range(num):

# 生成随机数对(i, j)，作为噪声的位置

i = np.random.randint(0, rows)

j = np.random.randint(0, cols)

# 生成随机数0和1，决定是椒噪声还是盐噪声

judge = np.random.randint(0, 2)

# 灰度图像

if src.ndim == 2:

if judge == 0:

dst[i, j] = 0

else:

dst[i, j] = 255

# 彩色图像

elif src.ndim == 3:

if judge == 0:

dst[i, j, :] = 0

else:

dst[i, j, :] = 255

return dst

2 平滑滤波

平滑滤波是低频增强的空域滤波技术，常用于模糊处理，或用于减少噪声。当图像上的噪声点过多时，可用平滑滤波抑制噪声，改善图像质量。

2.1 均值滤波器

均值滤波器的输出是包含在滤波器模板邻域内的像素的简单平均值，是一种低通滤波器。一幅的图像经过一个大小为(和是奇数)的加权均值滤波器的过程可由下式给出：

其中，为输入图像，为输出图像，为滤波器。这一等式对所有位移变量和求值，以便的所有元素访问每一个像素。和为位移量，，，并假设和是奇整数。

1. 椒盐噪声图像(左)与均值滤波后图像(右)

代码：均值滤波

def meanFilter(src):

# 扩充一个边缘，宽度为1

src = cv2.copyMakeBorder(src, 1, 1, 1, 1, cv2.BORDER\_REPLICATE)

dst = np.zeros(src.shape, dtype = np.float32) # 均值滤波结果

# 均值滤波

rows, cols, dim = src.shape

template = 1.0/9 \* np.ones((3, 3), dtype = np.float32) # 生成均值滤波模板

for i in range(1, rows-1):

for j in range(1, cols-1):

for k in range(dim):

temp = src[i-1: i+2, j-1: j+2, k] # 取出参与运算的像元值

dst[i, j, k] = np.sum(temp \* template) # 当前像元和邻域与模板相乘，再求和

dst = np.uint8(dst + 0.5)

# 裁剪出最终结果

dst = dst[1: rows-1, 1: cols-1,:]

return dst

在Python程序中，有内置的均值滤波函数cv2.blur()，原型如下：

cv2.blur(src, ksize[, dst[, anchor[, borderType]]]) -> dst

其中：

src 输入图像；

ksize 模板大小。

比较两种均值滤波器的差异。

2.2 中值滤波器

中值滤波是将像元邻域内的所有像元按亮度值大小排序，用中值作为该像元滤波后的亮度值，以达到去除噪声、平滑图像的目的。中值滤波对椒盐噪声及脉冲干扰的抑制效果较好，在抑制噪声的同时能有效保护边缘少受模糊。

1. 椒盐噪声图像(左)与中值滤波后图像(右)

代码：中值滤波

#中值滤波

def medianFilter(src, fsize = 3):

border = np.uint8((fsize - 1) / 2)

src = cv2.copyMakeBorder(src, border, border, border, border, cv2.BORDER\_REPLICATE)

dst = np.zeros(src.shape, dtype = np.uint8)

rows, cols, dim = src.shape

for i in range(border, rows - border):

for j in range(border, cols - border):

for k in range(dim):

temp = src[i - border : i + border + 1, j - border : j + border + 1, k]

dst[i, j ,k] = np.median(temp)

dst = dst[border : rows - border, border : cols – border, :]

return dst

2.3 最大值滤波器

最大值滤波是用窗口像元排序后的最大值来代替中心像元的亮度值，可以发现图像中的亮点，并消除图像中的“胡椒”噪声，可由下面公式给出：

式中参数的意义与本讲2.1节公式相同。

1. 椒噪声图像(左)与最大值滤波后图像(右)

代码：

# 最大值滤波(待滤波图像，滤波窗口大小)

def maxFilter(src, fsize = 3):

# 为图像添加边缘，宽度为滤波窗口大小减一的一半

border = np.uint8((fsize - 1) / 2)

src = cv2.copyMakeBorder(src, border, border, border, border, cv2.BORDER\_REPLICATE)

# 最大值滤波

dst = np.zeros(src.shape, dtype = np.uint8) # 滤波结果变量

rows, cols , dim = src.shape

for i in range(border, rows-border):

for j in range(border, cols-border):

for k in range(dim):

temp = src[i-border: i+1+border, j-border: j+1+border, k]

# 取出滤波窗口对应的像元值

dst[i, j, k] = np.max(temp) # 取出最大值，并赋值给结果像元

# 裁剪出结果，并返回结果

dst = dst[border: rows-border, border: cols-border, :]

return dst

2.4 最小值滤波器

最小值滤波是用窗口像元排序后的最小值来代替中心像元的亮度值，可以发现图像中的暗点，并消除图像中的“盐”噪声，可由下面公式给出：

式中参数的意义与本讲2.1节公式相同。

1. 盐噪声图像(左)与最小值滤波后图像(右)

代码：

# 最小值滤波(待滤波图像，滤波窗口大小)

def minFilter(src, fsize = 3):

# 为图像添加边缘，宽度为滤波窗口大小减一的一半

border = np.uint8((fsize - 1) / 2)

src = cv2.copyMakeBorder(src, border, border, border, border, cv2.BORDER\_REPLICATE)

# 最小值滤波

dst = np.zeros(src.shape, dtype = np.uint8) # 滤波结果变量

rows, cols, dim = src.shape

for i in range(border, rows-border):

for j in range(border, cols-border):

for k in range(dim):

temp = src[i-border: i+1+border, j-border: j+1+border, k] # 取出滤波窗口对应的像元值

dst[i, j, k] = np.min(temp) # 取出最小值，并赋值给结果像元

# 裁剪出结果，并返回结果

dst = dst[border: rows-border, border: cols-border, :]

return dst