Computer Vision I _2018

Homework assignment #8

R07522717 機械所製造組碩一 林温雅

#使用 python
#import 套件
-*- coding: utf-8 -*-

```
# -*- coding: utf-8 -*-

import cv2
import numpy as np

#  讀取原始影像
original_img = cv2.imread('lena.bmp', 0)

def gaussianNoise(img, amp):

    noise = np.random.normal(loc = 0, scale = 1, size = img.shape)
    noisy_img = img + amp*noise
    return   noisy_img

def saltpepperNoise(img, prob):
    #rows, columns = img.shape
    noisy_img = img.copy()
    noise = np.random.uniform(low=0, high=1, size = img.shape)
    #for i in range(rows):
    for i, j in np.ndindex(noise.shape):
        if noise[i,j]<prob:
            noisy_img[i,j] = 0
        elif noise[i,j]> 1-prob:
            noisy_img[i, j] = 255
        else:
            pass
```

```
    return    noisy_img

def boxFiltering(img, boxsize):
    #  獲得輸入圖檔之行列數
    img_rows, img_columns = img.shape
    #  獲得 kernel 之行列數
    ker_rows = ker_columns = boxsize
    box = np.full((boxsize, boxsize), 1, dtype=int)

    #  計算 kernel 中心距離邊界有多遠，為的是擴大原始圖檔，方便後續迴圈處理
    row_dist, column_dist = int((ker_rows - 1) / 2), int((ker_columns - 1) / 2)

    #nan?
    temp_img = np.full((img_rows + 2 * row_dist, img_columns + 2 * column_dist), -1)
    #temp_img = np.zeros((img_rows + 2 * row_dist, img_columns + 2 * column_dist), np.int)
    temp_img[row_dist:img_rows + row_dist, column_dist:img_columns + column_dist] = img.copy()

    new_img = np.zeros((img_rows, img_columns), np.int)

    for i in range(row_dist, img_rows + row_dist):
        for j in range(column_dist, img_columns + column_dist):
            #dict(zip(unique, counts))
            temp = temp_img[i - row_dist: i + row_dist + 1, j - column_dist: j + column_dist + 1]

            unique, counts = np.unique(temp, return_counts=True)
            dict4den = dict(zip(unique, counts))
            if -1 in dict4den:
                temp2 = temp.copy()
                for i2 in range(boxsize):
```

```
                        for j2 in range(boxsize):
                            if temp[i2, j2] == -1:
                                temp2[i2, j2] = 0
                    num = np.sum(np.multiply(box, temp2))
                    den = boxsize ** 2 - dict4den[-1]
                else:
                    num = np.sum(np.multiply(box, temp))
                    den = boxsize ** 2
                new_img[i - row_dist, j - column_dist] = num / den
    return new_img


def medianFiltering(img, boxsize):
    # 獲得輸入圖檔之行列數
    img_rows, img_columns = img.shape
    # 獲得 kernel 之行列數
    ker_rows = ker_columns = boxsize
    #box = np.full((boxsize, boxsize), 1, dtype=int)
    # 計算 kernel 中心距離邊界有多遠，為的是擴大原始圖檔，方便後續迴圈處理
    row_dist, column_dist = int((ker_rows - 1) / 2), int((ker_columns - 1) / 2)
    # nan?
    temp_img = np.full((img_rows + 2 * row_dist, img_columns + 2 * column_dist), -1)
    # temp_img = np.zeros((img_rows + 2 * row_dist, img_columns + 2 * column_dist), np.int)
    temp_img[row_dist:img_rows + row_dist, column_dist:img_columns + column_dist] = img.copy()
    new_img = np.zeros((img_rows, img_columns), np.int)
    for i in range(row_dist, img_rows + row_dist):
        for j in range(column_dist, img_columns + column_dist):
            temp = temp_img[i - row_dist: i + row_dist + 1, j - column_dist: j + column_dist + 1]
            unique, counts = np.unique(temp, return_counts=True)
```

```
            dict4den = dict(zip(unique, counts))
            if -1 in dict4den:
                temp2 = np.array([])
                for i2 in range(boxsize):
                    for j2 in range(boxsize):
                        if temp[i2, j2] == -1:
                            pass
                        else:
                            temp2 = np.append(temp2, temp[i2, j2])
                m = np.sort(temp2, axis=None)
                new_img[i - row_dist, j - column_dist] = m[ int((m.size - 1)
/ 2)]
            else:
                #m = np.median(np.ravel(temp))
                m = np.sort(temp, axis=None)
                new_img[i - row_dist, j - column_dist] = m[ int((m.size-1)
/ 2)]
    return new_img


def GrayScale_Dilation(img, ker):
    # 獲得輸入圖檔之行列數
    img_rows, img_columns = img.shape
    # 獲得 kernel 之行列數
    ker_rows, ker_columns = ker.shape
    # 計算 kernel 中心距離邊界有多遠，為的是擴大原始圖檔，方便後續迴圈
處理
    row_dist, column_dist = int((ker_rows - 1) / 2), int((ker_columns - 1) /
2)
    # 根據上述計算，製作一個比原始圖檔大的暫存圖檔，以 img 為 512*512,
kernel 為 5*5 來說，暫存圖檔為 516*516，暫存圖檔為往上、往下、往左、往
右分別外擴兩列/行，外擴新增的 pixel 值另為 0，中間則就是原本輸入圖檔的值

    # dilation 要找最大的，所以外擴的填 0
    temp_img = np.zeros((img_rows + 2 * row_dist, img_columns + 2 *
```

```
column_dist), np.int)
    temp_img[row_dist:img_rows + row_dist, column_dist:img_columns
+ column_dist] = img
    # 製作一個新圖檔準備接受 dilation 後的圖
    # 為了 for 迴圈裡面 index 好寫，這邊一樣把 new_img 改成擴大後的，之
後再來裁，和 hw4 做法有一點點不一樣
    new_img = np.zeros((img_rows + 2 * row_dist, img_columns + 2 *
column_dist), np.int)

    # 為了矩陣相乘，先 flip kernel，erosion 不用這樣
    kernel_flip = np.flip(ker)

    # 進行 dilation 計算
    for i in range(row_dist, img_rows + row_dist):
        for j in range(column_dist, img_columns + column_dist):
            new_img[i, j] = np.nanmax(
                temp_img[i - row_dist: i + row_dist + 1, j - column_dist: j +
column_dist + 1] + kernel_flip)
    new_img = new_img[row_dist:img_rows + row_dist,
column_dist:img_columns + column_dist]

    return new_img


def GrayScale_Erosion(img, ker):
    # 獲得輸入圖檔之行列數
    img_rows, img_columns = img.shape
    # 獲得 kernel 之行列數
    ker_rows, ker_columns = ker.shape
    # 計算 kernel 中心距離邊界有多遠，為的是擴大原始圖檔，方便後續迴圈
處理
    row_dist, column_dist = int((ker_rows - 1) / 2), int((ker_columns - 1) /
2)
    # 根據上述計算，製作一個比原始圖檔大的暫存圖檔，以 img 為 512*512,
kernel 為 5*5 來說，暫存圖檔為 516*516，暫存圖檔為往上、往下、往左、往
```

右分別外擴兩列/行，外擴新增的 pixel 值另為 0，中間則就是原本輸入圖檔的值

```
    # erosion 要找最小的，所以外擴的填 255
    temp_img = 255 * np.ones((img_rows + 2 * row_dist, img_columns + 2 * column_dist), np.int)
    temp_img[row_dist:img_rows + row_dist, column_dist:img_columns + column_dist] = img
    #  製作一個新圖檔準備接受 dilation 後的圖
    #  為了 for 迴圈裡面 index 好寫，這邊一樣把 new_img 改成擴大後的，之後再來裁，和 hw4 做法有一點點不一樣

    new_img = 255 * np.ones((img_rows + 2 * row_dist, img_columns + 2 * column_dist), np.int)

    #  進行 erosion 計算
    for i in range(row_dist, img_rows + row_dist):
        for j in range(column_dist, img_columns + column_dist):
            new_img[i, j] = np.nanmin(
                temp_img[i - row_dist: i + row_dist + 1, j - column_dist: j + column_dist + 1] - ker)

    new_img = new_img[row_dist:img_rows + row_dist, column_dist:img_columns + column_dist]
    return new_img


def GrayScale_Opening(img, ker):
    return GrayScale_Dilation(GrayScale_Erosion(img, ker), ker)

def GrayScale_Closing(img, ker):
    return GrayScale_Erosion(GrayScale_Dilation(img, ker), ker)

def op_cl_Filtering(img, ker):
    return GrayScale_Closing(GrayScale_Opening(img, ker), ker)
```

```
def cl_op_Filtering(img, ker):
    return GrayScale_Opening(GrayScale_Closing(img, ker), ker)


#輸入 original image
def vsCalc(img):
    rows, cols = img.shape
    sum1=sum2=0

    for i in range(rows):
        for j in range(cols):
            sum1 += img[i, j]
    mu = sum1 / (rows*cols)

    for i in range(rows):
        for j in range(cols):
            sum2 += ((img[i, j]-mu) ** 2)
    vs = sum2 / (rows*cols)
    return vs

#輸入 original image + 一張處理前/後 noise image
def vnCalc(o_img, n_img):
    sum1 = sum2 = 0
    rows, cols = o_img.shape
    for i in range(rows):
        for j in range(cols):
            sum1 += (n_img[i, j]-o_img[i, j])
    mun = sum1 / (rows*cols)
    for i in range(rows):
        for j in range(cols):
            sum2 += ((n_img[i, j]-o_img[i, j]-mun) ** 2)
    vn = sum2 / (rows*cols)
    return vn
```

```
gaussian10 = gaussianNoise(original_img, 10)
gaussian30 = gaussianNoise(original_img, 30)
saltpepper005 = saltpepperNoise(original_img, 0.05)
saltpepper01 = saltpepperNoise(original_img, 0.1)

#op, cl 要用的
kernel = np.array([[np.nan, 0, 0, 0, np.nan], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [np.nan, 0, 0, 0, np.nan]])


# 處理前 noise image（4 個）
cv2.imwrite('gaussian10.bmp', gaussian10)
cv2.imwrite('gaussian30.bmp', gaussian30)
cv2.imwrite('saltpepper005.bmp', saltpepper005)
cv2.imwrite('saltpepper01.bmp', saltpepper01)

#處理後（4*6 個）

gaussian10_33box = boxFiltering(gaussian10, 3)
gaussian10_55box = boxFiltering(gaussian10, 5)
gaussian10_33median = medianFiltering(gaussian10, 3)
gaussian10_55median = medianFiltering(gaussian10, 5)
gaussian10_op_cl = op_cl_Filtering(gaussian10, kernel)
gaussian10_cl_op = cl_op_Filtering(gaussian10, kernel)
cv2.imwrite('gaussian10_33box.bmp', gaussian10_33box)
cv2.imwrite('gaussian10_55box.bmp', gaussian10_55box)
cv2.imwrite('gaussian10_33median.bmp', gaussian10_33median)
cv2.imwrite('gaussian10_55median.bmp', gaussian10_55median)
cv2.imwrite('gaussian10_op_cl.bmp', gaussian10_op_cl)
cv2.imwrite('gaussian10_cl_op.bmp', gaussian10_cl_op)

gaussian30_33box = boxFiltering(gaussian30, 3)
gaussian30_55box = boxFiltering(gaussian30, 5)
gaussian30_33median = medianFiltering(gaussian30, 3)
gaussian30_55median = medianFiltering(gaussian30, 5)
```

```
gaussian30_op_cl = op_cl_Filtering(gaussian30, kernel)
gaussian30_cl_op = cl_op_Filtering(gaussian30, kernel)
cv2.imwrite('gaussian30_33box.bmp', gaussian30_33box)
cv2.imwrite('gaussian30_55box.bmp', gaussian30_55box)
cv2.imwrite('gaussian30_33median.bmp', gaussian30_33median)
cv2.imwrite('gaussian30_55median.bmp', gaussian30_55median)
cv2.imwrite('gaussian30_op_cl.bmp', gaussian30_op_cl)
cv2.imwrite('gaussian30_cl_op.bmp', gaussian30_cl_op)

saltpepper005_33box = boxFiltering(saltpepper005, 3)
saltpepper005_55box = boxFiltering(saltpepper005, 5)
saltpepper005_33median = medianFiltering(saltpepper005, 3)
saltpepper005_55median = medianFiltering(saltpepper005, 5)
saltpepper005_op_cl = op_cl_Filtering(saltpepper005, kernel)
saltpepper005_cl_op = cl_op_Filtering(saltpepper005, kernel)
cv2.imwrite('saltpepper005_33box.bmp', saltpepper005_33box)
cv2.imwrite('saltpepper005_55box.bmp', saltpepper005_55box)
cv2.imwrite('saltpepper005_33median.bmp', saltpepper005_33median)
cv2.imwrite('saltpepper005_55median.bmp', saltpepper005_55median)
cv2.imwrite('saltpepper005_op_cl.bmp', saltpepper005_op_cl)
cv2.imwrite('saltpepper005_cl_op.bmp', saltpepper005_cl_op)


saltpepper01_33box = boxFiltering(saltpepper01, 3)
saltpepper01_55box = boxFiltering(saltpepper01, 5)
saltpepper01_33median = medianFiltering(saltpepper01, 3)
saltpepper01_55median = medianFiltering(saltpepper01, 5)
saltpepper01_op_cl = op_cl_Filtering(saltpepper01, kernel)
saltpepper01_cl_op = cl_op_Filtering(saltpepper01, kernel)
cv2.imwrite('saltpepper01_33box.bmp', saltpepper01_33box)
cv2.imwrite('saltpepper01_55box.bmp', saltpepper01_55box)
cv2.imwrite('saltpepper01_33median.bmp', saltpepper01_33median)
cv2.imwrite('saltpepper01_55median.bmp', saltpepper01_55median)
cv2.imwrite('saltpepper01_op_cl.bmp', saltpepper01_op_cl)
cv2.imwrite('saltpepper01_cl_op.bmp', saltpepper01_cl_op)
```

```
# SNR
# VS
#VS_orginal_img    = 20 * np.log10()
VS_orginal_img    = vsCalc(original_img)

#SNR
list_pre = [gaussian10, gaussian30, saltpepper005, saltpepper01]
print ('處理前')
for noise_img in list_pre:
      print (20 * np.log10(np.sqrt(VS_orginal_img / vnCalc(original_img,
noise_img))))

list_gaussain10 = [gaussian10_33box, gaussian10_55box,
gaussian10_33median,    gaussian10_55median, gaussian10_op_cl,
gaussian10_cl_op]
print ('gaussian10')
for noise_img in list_gaussain10:
      print (20 * np.log10(np.sqrt(VS_orginal_img / vnCalc(original_img,
noise_img))))

list_gaussain30 = [gaussian30_33box, gaussian30_55box,
gaussian30_33median,    gaussian30_55median, gaussian30_op_cl,
gaussian30_cl_op]
print ('gaussian30')
for noise_img in list_gaussain30:
      print (20 * np.log10(np.sqrt(VS_orginal_img / vnCalc(original_img,
noise_img))))

list_saltpepper005 = [saltpepper005_33box, saltpepper005_55box,
saltpepper005_33median, saltpepper005_55median,
saltpepper005_op_cl, saltpepper005_cl_op]
print ('saltpepper005')
for noise_img in list_saltpepper005:
```

```
    print (20 * np.log10(np.sqrt(VS_orginal_img / vnCalc(original_img,
noise_img))))

list_saltpepper01 = [saltpepper01_33box, saltpepper01_55box,
saltpepper01_33median, saltpepper01_55median, saltpepper01_op_cl,
saltpepper01_cl_op]
print ('saltpepper01')
for noise_img in list_saltpepper01:
    print (20 * np.log10(np.sqrt(VS_orginal_img / vnCalc(original_img,
noise_img))))
```