Computer Vision I _2018

Homework assignment #8

R07522717 機械所製造組碩一 林温雅

Part1（此次作業僅 one part）

Description:

Write the following programs

1. Generate additive white Gaussian noise
2. Generate salt-and-pepper noise
3. Run box filter *(3X3, 5X5)* on all noisy images
4. Run median filter *(3X3, 5X5)* on all noisy images
5. Run opening followed by closing and closing followed by opening

Algorithm:

1. gaussianNoise:
   使用 ppt 提供的公式，做 random normal 的 noise 乘上 noise，並加到原始圖檔上
2. saltpepperNoise:
   使用 ppt 提供的公式，將 I-x 的值切三段賦予值
3. boxfiltering:
   用 k*k 的 kernel 去掃圖，取平均值
4. medianFiltering:
   用 k*k 的 kernel 去掃圖，取 median
5. Dilation:
   獲得 kernel 與輸入 image 的行列數以後，製作一個『外擴』圖檔，為的是處理 kernel 在尋訪時超出邊界的情況（以 image 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 值為 0，才不會影響後續取 max 的計算，中間則就是原本輸入圖檔的值）。
   接著再進行計算，kernel 設計為[[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]]。在計算時會忽略遇到 nan 的運算，因此計算結果即為 3553 形狀 kernel 中的最大值，將最大值填入 kernel 中心覆蓋的某 p 點。

特別的是，在 dilation 進行 convolution，因此 kernel 須先 flip 180 度，但作業指定的是點對稱 kernel，有沒有 flip 不對結果造成影響。

6. Erosion:

獲得 kernel 與輸入 image 的行列數以後，製作一個『外擴』圖檔，為的是處理 kernel 在尋訪時超出邊界的情況（以 image 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 值為 255，才不會影響後續取 min 的計算，中間則就是原本輸入圖檔的值。特別注意這邊外擴 pixel 的值為 255，而在 dilation function 內是 0）。

接著再進行計算，kernel 設計為[[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]]。在計算時會忽略遇到 nan 的運算，因此計算結果即為 3553 形狀 kernel 中的最小值，將最小值填入 kernel 中心覆蓋的某 p 點。

7. Opening:

對影像進行 GrayScale_Erosion 後再進行 GrayScale_Dilation

8. Closing:

對影像進行 GrayScale_Dilation 後再 GrayScale_Erosion

9. vsCalc:

使用助教提供之公式計算 VS

10. vnCalc:

使用助教提供之公式計算 VN

Parameters:

1. In function “gaussianNoise”:

noise	#噪音
noisy_img	#有 noise 的新圖
amp	#noise 倍率

2. In function “saltpepperNoise”:

noise	#噪音
noisy_img	#有 noise 的新圖
prob	#salt&pepper 的參數

3. In function “boxFiltering”:
   img_rows, img_columns #輸入圖檔的行列數
   ker_rows, ker_columns #kernel 的行列數
   row_dist, column_dist #計算 kernel 中心距離邊界有多遠，主要目的是看原始圖檔要擴大多少
   temp_img #原始圖檔擴大後的暫存圖檔
   new_img #新圖檔準備接受 dilation 後的圖
   i,j,i2,j2 #迴圈計數用參數
   unique, counts, dict4den#輔助邊界處理用參數

4. In function “medianFiltering”:
   img_rows, img_columns #輸入圖檔的行列數
   ker_rows, ker_columns #kernel 的行列數
   row_dist, column_dist #計算 kernel 中心距離邊界有多遠，主要目的是看原始圖檔要擴大多少
   temp_img #原始圖檔擴大後的暫存圖檔
   new_img #新圖檔準備接受 dilation 後的圖
   i,j,i2,j2 #迴圈計數用參數
   unique, counts, dict4den#輔助邊界處理用參數

5. In function “GrayScale_Dilation”:
   img_rows, img_columns #輸入圖檔的行列數
   ker_rows, ker_columns #kernel 的行列數
   row_dist, column_dist #計算 kernel 中心距離邊界有多遠，主要目的是看原始圖檔要擴大多少
   temp_img #原始圖檔擴大後的暫存圖檔
   kernel_flip #flip 180 度後的 kernel
   new_img #新圖檔準備接受 dilation 後的圖
   i,j #迴圈計數用參數

6. In function “GrayScale_Erosion”:
   img_rows, img_columns #輸入圖檔的行列數
   ker_rows, ker_columns #kernel 的行列數
   row_dist, column_dist #計算 kernel 中心距離邊界有多遠，主要目的是看原始圖檔要擴大多少

temp_img　　#原始圖檔擴大後的暫存圖檔
new_img　　#新圖檔準備接受 Erosion 後的圖
i,j　　#迴圈計數用參數

Principal code fragment:

```
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
```

Resulting 1 : SNR value of each image

image	SNR
加入 noise 後，noise removal 處理前	
gaussian10	13.592766509939194
gaussian30	4.060167602001724
saltpepper005	1.0759904938600393
saltpepper01	-1.4900778457635253
取 gaussian10 做處理	
gaussian30_33box	17.741838799921968
gaussian30_55box	14.866794900408959
gaussian30_33median	17.66192630194827
gaussian30_55median	16.011435250514133
gaussian30_op_cl	13.26475053974471
gaussian30_cl_op	13.597751665350819
取 gaussian30 做處理	
gaussian30_33box	12.579020151433191
gaussian30_55box	13.350486154632994
gaussian30_33median	11.123587728636915
gaussian30_55median	12.95568487018745
gaussian30_op_cl	11.137887006033228
gaussian30_cl_op	11.1560540598589
取 saltpepper005 做處理	
saltpepper005_33box	9.47780968130954
saltpepper005_55box	11.184618312421936
saltpepper005_33median	19.319157425618712
saltpepper005_55median	16.38871799722966
saltpepper005_op_cl	5.591209210482802
saltpepper005_cl_op	5.282347580853731
取 saltpepper01 做處理	
saltpepper005_33box	6.3453398170185515
saltpepper005_55box	8.512992109209153
saltpepper005_33median	14.924560614990018
saltpepper005_55median	15.744199798051891
saltpepper005_op_cl	-2.083428144867428
saltpepper005_cl_op	-2.468259917453715

Resulting Image:

gaussian10

gaussian30

saltpepper01

saltpepper005

gaussian10_33box

gaussian10_55box

gaussian30_33box

gaussian30_55box

saltpepper01_33box

saltpepper01_55box

saltpepper005_33box

saltpepper005_55box

gaussian10_33median

gaussian10_55median

gaussian30_33median

gaussian30_55median

saltpepper01_33median

saltpepper01_55median

saltpepper005_33median

saltpepper005_55median

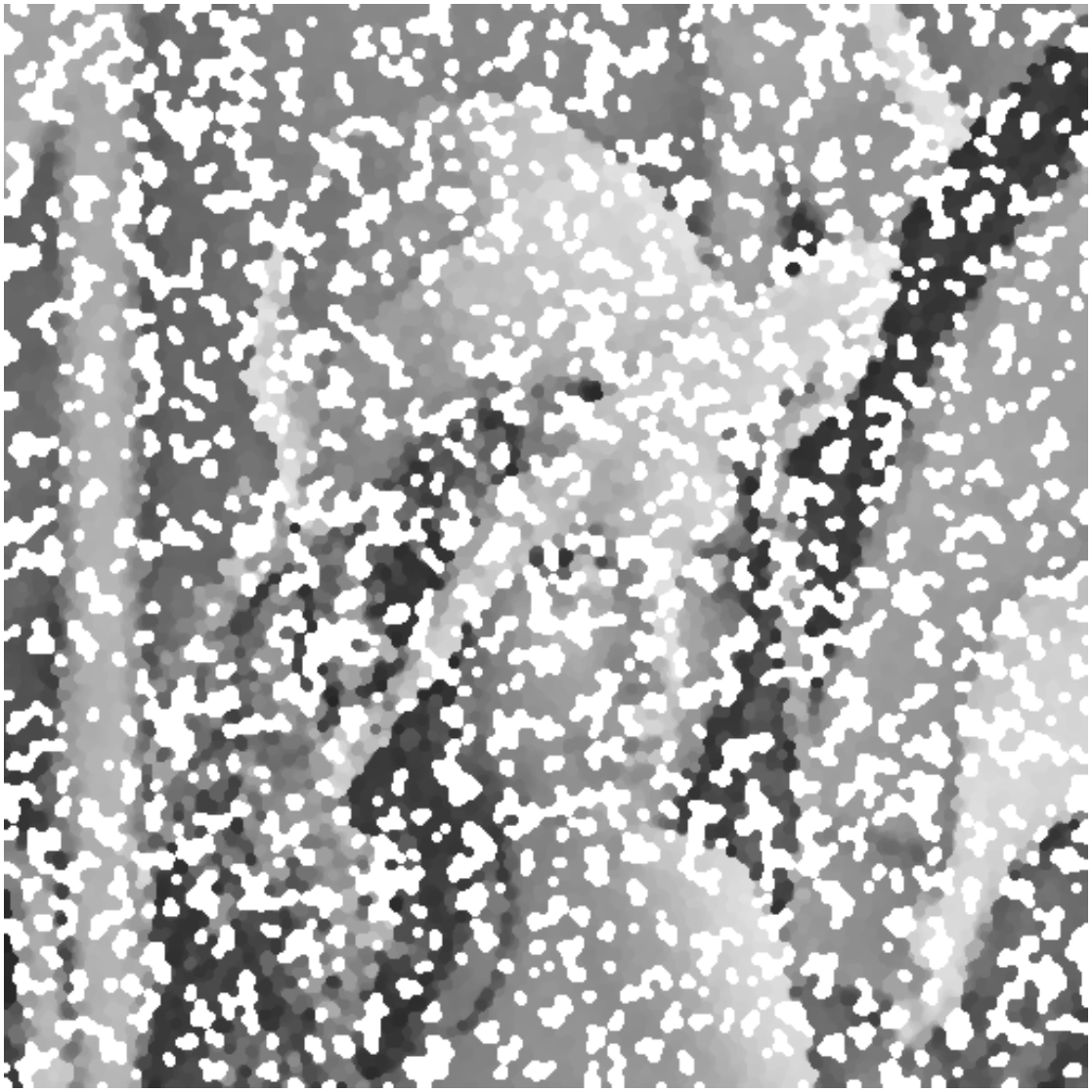
gaussian10_cl_op

gaussian10_op_cl

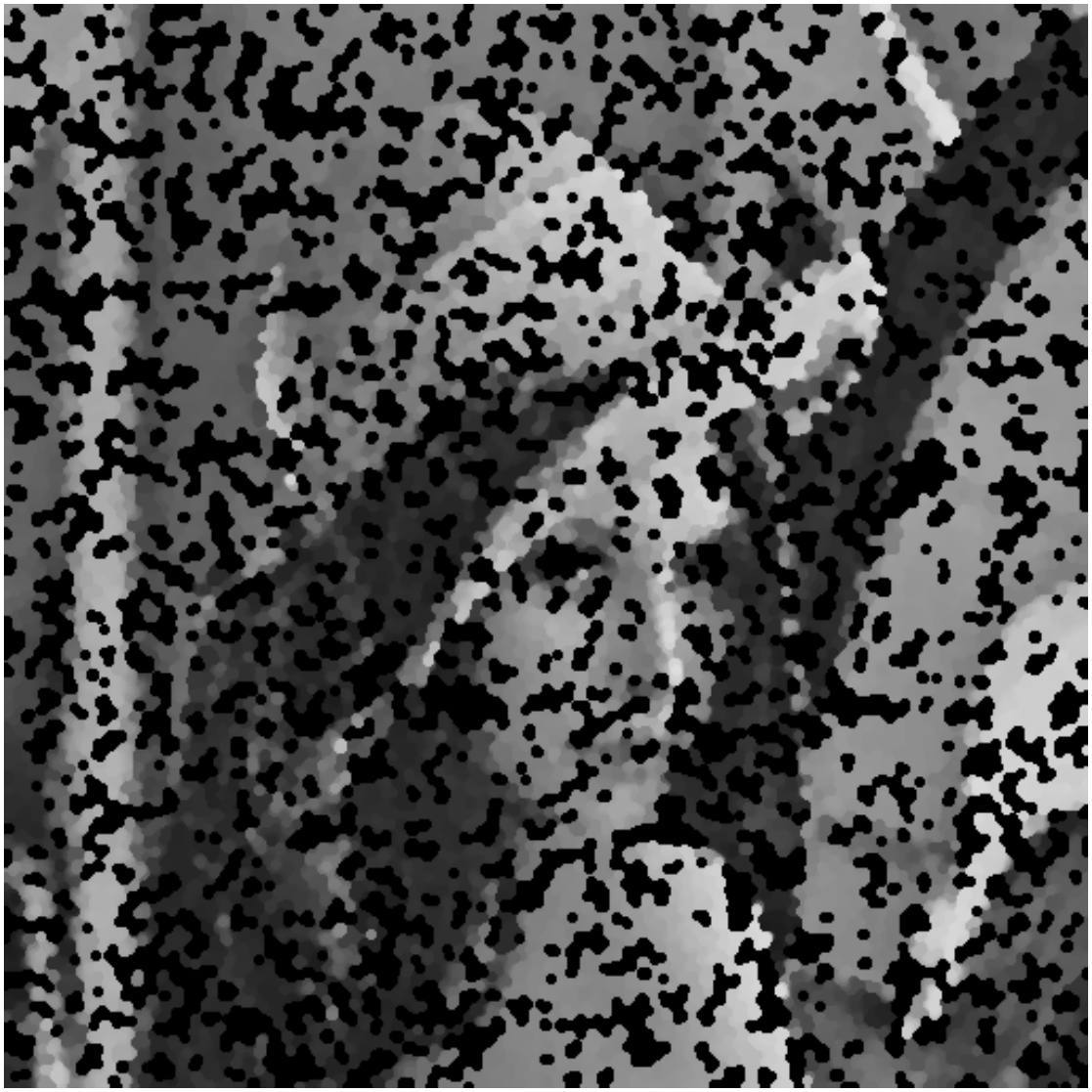
gaussian30_cl_op

gaussian30_op_cl

saltpepper01_cl_op



saltpepper01_op_cl



saltpepper005_cl_op

saltpepper005_op_cl