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

1. saltpepperNoise:

使用ppt提供的公式，將I-x的值切三段賦予值

1. boxfiltering:

用k\*k的kernel去掃圖，取平均值

1. medianFiltering:

用k\*k的kernel去掃圖，取median

1. 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。

1. 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。

1. Opening:

對影像進行GrayScale\_Erosion後再進行GrayScale\_Dilation

1. Closing:

對影像進行GrayScale\_Dilation後再GrayScale\_Erosion

1. vsCalc:

使用助教提供之公式計算VS

1. 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

1. In function “saltpepperNoise”:

noise #噪音

noisy\_img #有noise的新圖

prob #salt&pepper的參數

1. 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

1. 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

1. 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

1. 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\_cl\_op



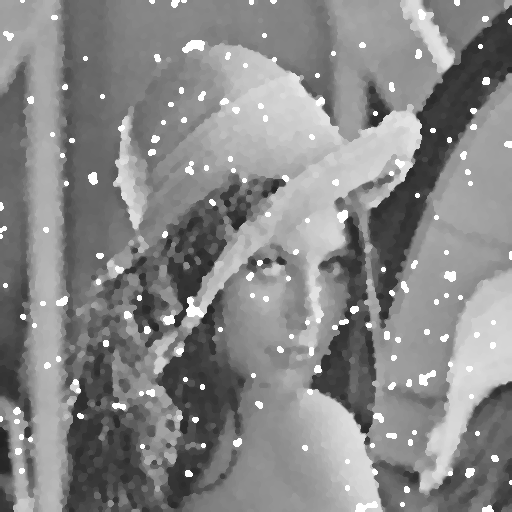
saltpepper01\_cl\_op



saltpepper01\_op\_cl



saltpepper005\_cl\_op



saltpepper005\_op\_cl

