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))))*