**Computer Vision HW4**

R08922a27 資工系 人工智慧碩士班 李吉昌

I use python 3.7 to implement all image processing requirements. Reading .bmp file by **PIL**, and then processing through **NumPy** array.

* **(a) Dilation**

1. **Results**



1. **Code fragment**

def dilation(bin\_img, kernel):

img\_dil = np.zeros(bin\_img.shape).astype(int)

for i in range(bin\_img.shape[0]):

for j in range(bin\_img.shape[1]):

if bin\_img[i, j] > 0:

for (p, q) in kernel:

i\_dil, j\_dil = i + p, j + q

if i\_dil >= 0 and j\_dil >= 0 and \

i\_dil <= (bin\_img.shape[0] - 1) and j\_dil <= (bin\_img.shape[1] - 1):

img\_dil[i\_dil, j\_dil] = 255

return img\_dil

img\_dil = dilation(bin\_img, kernel)

PIL\_image = Image.fromarray(img\_dil.astype('uint8'))

PIL\_image.save('results/Dilation.bmp')

1. **Brief description**

The dilation function is defined. Those grey level values of pixels in which neighbors in the kernel range are not 0 would be assigned to 255.

* **(b) Erosion**

1. **Results**



1. **Code fragment**

def erosion(bin\_img, kernel):

img\_ero = np.zeros(bin\_img.shape).astype(int)

for i in range(bin\_img.shape[0]):

for j in range(bin\_img.shape[1]):

Isdraw = True

for (p, q) in kernel:

i\_dil, j\_dil = i + p, j + q

if not(i\_dil >= 0 and j\_dil >= 0 and

i\_dil <= (

bin\_img.shape[0] - 1) and j\_dil <= (bin\_img.shape[1] - 1)

and bin\_img[i\_dil, j\_dil] > 0):

Isdraw = False

break

if Isdraw:

img\_ero[i, j] = 255

return img\_ero

img\_ero = erosion(bin\_img, kernel)

PIL\_image = Image.fromarray(img\_ero.astype('uint8'))

PIL\_image.save('results/Erosion.bmp')

1. **Brief description**

The erosion function is defined. Those grey level values of pixels in which neighbors in the kernel range are all 255 would be assigned to 255.

* **(c) Opening**

1. **Results**



1. **Code fragment**

img\_opn = dilation(erosion(bin\_img, kernel), kernel)

PIL\_image = Image.fromarray(img\_opn.astype('uint8'))

PIL\_image.save('results/Opening.bmp')

1. **Brief description**

The previously defined dilation and erosion functions are used. Starting with the erosion, and then the dilation is conducted.

* **(d) Closing**

1. **Results**



1. **Code fragment**

img\_cls = erosion(dilation(bin\_img, kernel), kernel)

PIL\_image = Image.fromarray(img\_cls.astype('uint8'))

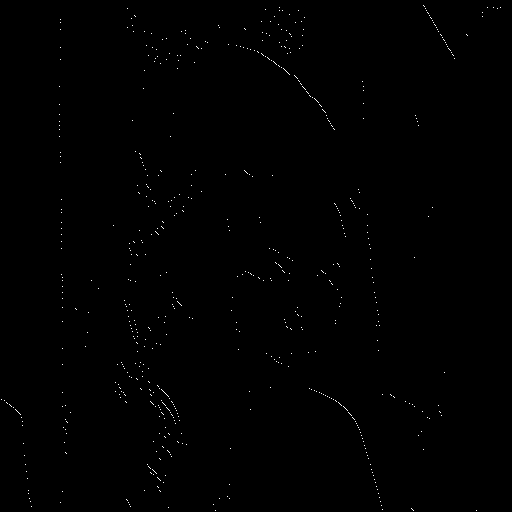
PIL\_image.save('results/Closing.bmp')

1. **Brief description**

The previously defined dilation and erosion functions are used. Starting with the dilation, and then the erosion is conducted.

* **(e) Hit-and-miss transform**

1. **Results**



1. **Code fragment**

J\_kernel = [[0, -1], [0, 0], [1, 0]]

K\_kernel = [[-1, 0], [-1, 1], [0, 1]]

def hit\_and\_miss(bin\_img, J\_kernel, K\_kernel):

img\_ham = np.ones(bin\_img.shape).astype(int) \* 255

img\_ham[np.logical\_or(erosion(bin\_img, J\_kernel) < 128,

erosion(-bin\_img + 255, K\_kernel) < 128)] = 0

return img\_ham

img\_ham = hit\_and\_miss(bin\_img, J\_kernel, K\_kernel)

PIL\_image = Image.fromarray(img\_ham.astype('uint8'))

PIL\_image.save('results/HitAndMiss.bmp')

1. **Brief description**

The original binary image and the component one are conducted the erosion with J kernel and K kernel, respectively. Those pixels which are both 255 in the above results would be assigned to 255.