**Computer Vision HW5: Mathematical Morphology – Gray Scaled Morphology**

**R10741015 鄭傑鴻**

**Oct. 08, 2022**

**Original Binary Image:**

****

**Mutual Parameters:**

* **pic: original Lena.bmp image**
* **Kernel: (3,5,5,5,3): Disk shaped kernel for dilation/erosion/closing/opening**

**Write programs which do gray-scale morphology on a gray-scale image:**

1. Dilation

* Description:

Using the (3,5,5,5,3) kernel to perform dilation on the binary image, making each pixel the maximum value in the kernel.

* Algorithm:

*For every pixel in the gray-scale image:*

*Apply the kernel and record the values within*

*Assign the maximum value to the pixel*

* Code:

def getValInKernel(self, refer, row, col):  
 intensity\_lst = []  
 for point in self.kernel:  
 if self.inRange(row+point[0], col+point[1]):  
 intensity\_lst.append(refer[row+point[0]][col+point[1]])  
 return intensity\_lst  
  
def dilation(self, tgtpic):  
 dilation\_pic = np.copy(tgtpic)  
 for i in range(dilation\_pic.shape[0]):  
 for j in range(dilation\_pic.shape[1]):  
 dilation\_pic[i][j] = max(self.getValInKernel(tgtpic, i, j))  
 return dilation\_pic

def sequential(self):  
 # Dilation  
 dilation\_pic = self.dilation(self.pic)  
 cv2.imwrite('lena\_dilation.bmp', dilation\_pic)

……

* Resulting Image:



1. Erosion

* Description

Using the (3,5,5,5,3) kernel to perform erosion on the gray-scale image, making each pixel the minimum value in the kernel.

* Algorithm

*For every pixel in the gray-scale image:*

*Apply the kernel and record the values within*

*Assign the minimum value to the pixel*

* Code:

def getValInKernel(self, refer, row, col):  
 intensity\_lst = []  
 for point in self.kernel:  
 if self.inRange(row+point[0], col+point[1]):  
 intensity\_lst.append(refer[row+point[0]][col+point[1]])  
 return intensity\_lst

def erosion(self, tgtpic):  
 erosion\_pic = np.copy(tgtpic)  
 for i in range(erosion\_pic.shape[0]):  
 for j in range(erosion\_pic.shape[1]):  
 erosion\_pic[i][j] = min(self.getValInKernel(tgtpic, i, j))  
 return erosion\_pic

def sequential(self):

……

# Erosion  
erosion\_pic = self.erosion(self.pic)  
cv2.imwrite('lena\_erosion.bmp', erosion\_pic)

……

* Resulting Image:



1. Opening

* Description/Algorithm:

Perform erosion first, then apply dilation. In practice, call the erosion function in part (b), then call the dilation function in part (a)

* Code:

def opening(self, tgtpic):  
 opening\_pic = self.erosion(tgtpic)  
 opening\_pic = self.dilation(opening\_pic)  
 return opening\_pic

def sequential(self):

……

# Opening  
opening\_pic = self.opening(self.pic)  
cv2.imwrite('lena\_opening.bmp', opening\_pic)

……

* Resulting Image:



1. Closing

* Description/Algorithm:

Perform dilation first, then apply erosion. In practice, call the dilation function in part (a), then call the erosion function in part (b)

* Code:

def closing(self, tgtpic):  
 # Dilation -> Erosion  
 closing\_pic = self.dilation(tgtpic)  
 closing\_pic = self.erosion(closing\_pic)  
 return closing\_pic

def sequential(self):

……

# Closing  
closing\_pic = self.closing(self.pic)  
cv2.imwrite('lena\_closing.bmp', closing\_pic)

……

* Resulting Image

