**Computer Vision HW4: Mathematical Morphology – Binary Morphology**

**R10741015 鄭傑鴻**

**Oct. 08, 2022**

**Original Binary Image:**

****

**Mutual Parameters:**

* **Binary Image: 0 for intensity<128, 1 for intensity>=128**
* **Kernel: (3,5,5,5,3): Disk shaped kernel for dilation/erosion/closing/opening**
* **Kernel\_hit, Kernel\_miss: L-shaped kernel for hit & miss**

**Write programs which do binary morphology on a binary image:**

1. Dilation

* Description:

Using the (3,5,5,5,3) kernel to perform dilation on the binary image, spreading the white points outward.

* Algorithm:

*For every pixel in the binary image with Boolean value 1(White pixel):*

*Spread white pixels using the kernel*

* Code:

def spread(self, array, row, col):  
 for point in self.kernel:  
 if self.inRange(row+point[0], col+point[1]):  
 array[row+point[0]][col+point[1]] = 1  
  
  
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]):  
 if tgtpic[i][j] == True:  
 self.spread(dilation\_pic, i, j)  
 return dilation\_pic

def sequential(self):  
 # Dilation  
 dilation\_pic = self.dilation(self.binary)  
 cv2.imwrite('lena\_dilation.bmp', dilation\_pic \* 255)

……

* Resulting Image:



1. Erosion

* Description

Using the (3,5,5,5,3) kernel to perform erosion on the binary image, checking whether the kernel can fit in each pixel location.

* Algorithm

*For every pixel in the binary image with Boolean value 1(White pixel):*

*If the kernel can fit:*

*Assign 1 to the corresponding pixel in the erosion picture*

*Else if the kernel cannot fit:*

*Assign 0 to the corresponding pixel in the erosion picture*

* Code:

def checkFit(self, array, refer, row, col):  
 ret\_bool = True  
 for point in self.kernel:  
 if self.inRange(row+point[0], col+point[1]):  
 if not refer[row+point[0]][col+point[1]] == True:  
 ret\_bool = False  
 break  
 array[row][col] = ret\_bool  
  
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]):  
 if tgtpic[i][j] == True:  
 self.checkFit(erosion\_pic, tgtpic, i, j)  
 return erosion\_pic

def sequential(self):

……

# Erosion  
erosion\_pic = self.erosion(self.binary)  
cv2.imwrite('lena\_erosion.bmp', erosion\_pic \* 255)

……

* 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.binary)  
cv2.imwrite('lena\_opening.bmp', opening\_pic \* 255)

……

* 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.binary)  
cv2.imwrite('lena\_closing.bmp', closing\_pic \* 255)

……

* Resulting Image



1. Hit-and-miss Transformation

* Description:

Using the L shaped kernel to detect upper-right edges in the binary picture.

* Algorithm

*For every pixel in the binary image with Boolean value 1(White pixel):*

*If the kernel\_hit cannot fully fit in this pixel:*

*Assign 0 to the corresponding pixel in the hit&miss picture*

*Else if the kernel\_miss can fit in any position:*

*Assign 0 to the corresponding pixel in the hit&miss picture*

*Else:*

*Assign 1 to the corresponding pixel in the hit&miss picture*

* Code:

def match(self, row, col):  
 ret\_bool = True  
 for point in self.kernel\_hit:  
 if self.inRange(row + point[0], col + point[1]):  
 if not self.binary[row + point[0]][col + point[1]] == True:  
 ret\_bool = False  
 break  
 for point in self.kernel\_miss:  
 if self.inRange(row + point[0], col + point[1]):  
 if self.binary[row + point[0]][col + point[1]] == True:  
 ret\_bool = False  
 break  
 return ret\_bool  
  
def hit\_miss(self):  
 hitmiss\_pic = np.copy(self.binary)  
 for i in range(self.binary.shape[0]):  
 for j in range(self.binary.shape[1]):  
 if self.binary[i][j] == True:  
 hitmiss\_pic[i][j] = self.match(i, j)  
 return hitmiss\_pic

def sequential(self):

……

# Hit & Miss  
hitmiss\_pic = self.hit\_miss()  
cv2.imwrite('lena\_hitmiss.bmp', hitmiss\_pic \* 255)

* Resulting Image

