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:

(a) 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:

(b) 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:

(c) 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:

(d) 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

(e) 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