Computer Vision Homework 7   D07922015   謝銘峰

Write a program to generate thinning after samplimg(using kernel 3 3 3):

Original

Thinning 1

Thinning 2

Thinning 3

Thinning 4

Thinning 5

(a) Binary 128

```python
def getBinaryImage(originalImage, threshold):
    """
    :type originalImage: Image (from PIL)
    :type threshold: int
    :return type: Image (from PIL)
    """
    from PIL import Image
    # New image with the same size and 'binary' format.
    binaryImage = Image.new('1', originalImage.size)
    # Scan each column in original image.
    for c in range(originalImage.size[0]):
        # Scan each row in original image.
        for r in range(originalImage.size[1]):
            # Get pixel value in original image at (c, r).
            originalPixel = originalImage.getpixel((c, r))
            if (originalPixel >= threshold):
                # Put pixel value '1'  to binary image.
                binaryImage.putpixel((c, r), 1)
            else:
                # Put pixel value '0'  to binary image.
                binaryImage.putpixel((c, r), 0)
    # Return binary image.
    return binaryImage
```

```python
if __name__ == '__main__':
    from PIL import Image
    import numpy as np

    # Load image from file.
    originalImage = Image.open('lena.bmp')
    # Get binary image.
    binaryImage = getBinaryImage(originalImage, 128)
    # Save binary image fo file.
    binaryImage.save('binary.bmp')
```

(b) Downsampling (128x128 divided by 8x8)

```python
def downsampling(originalImage, sampleFactor):
    """
    :type originalImage: Image (from PIL)
    :type sampleFactor: int
    :return type: Image (from PIL)
    """
    from PIL import Image
    # Calculate the width and height of downsampling image.
    downsamplingWidth = int(originalImage.size[0] / sampleFactor)
    downsamplingHeight = int(originalImage.size[1] / sampleFactor)
    # New image with the downsampling size and 'binary' format.
    downsamplingImage = Image.new('1', (downsamplingWidth, downsamplingHeight))
    # Scan each column in downsampling image.
    for c in range(downsamplingImage.size[0]):
        # Scan each row in downsampling image.
        for r in range(downsamplingImage.size[1]):
            # Get pixel value in original image at (c * sampleFactor, r * sampleFactor).
            originalPixel = originalImage.getpixel((c * sampleFactor, r * sampleFactor))
            # Put pixel to downsampling image.
            downsamplingImage.putpixel((c, r), originalPixel)
    # Return downsampling image.
    return downsamplingImage
```

```python
if __name__ == '__main__':
    from PIL import Image
    import numpy as np

    # Load image from file.
    originalImage = Image.open('lena.bmp')
    # Get binary image.
    binaryImage = getBinaryImage(originalImage, 128)
    # Save binary image fo file.
    binaryImage.save('binary.bmp')

    # Get downsampling image.
    downsamplingImage = downsampling(binaryImage, 8)
    # Save downsampling image fo file.
    downsamplingImage.save('downsampling.bmp')
```

(c) Thinning

a. 4 connected neighborhood detection

```python
def getNeighborhoodPixels(originalImage, position):
    """
    :type originalImage: Image (from PIL)
    :type position: tuple
    :return type: numpy array
    """
    # Allocate memory space of neighborhoodPixels.
    neighborhoodPixels = np.zeros(9)
    # Get x and y of position.
    x, y = position
    # Scan dx from -1 to 1.
    for dx in range(3):
        # Scan dy from -1 to 1.
        for dy in range(3):
            # Calculate destination x, y position.
            destX = x + (dx - 1)
            destY = y + (dy - 1)
            # Avoid out of image range.
            if ((0 <= destX < originalImage.size[0]) and \
                (0 <= destY < originalImage.size[1])):
                # Get neighborhood pixel values.
                neighborhoodPixels[3 * dy + dx] = originalImage.getpixel((destX, destY))
            # It is out of image range.
            else:
                # Padding zeros when it is out of image range.
                neighborhoodPixels[3 * dy + dx] = 0
    # Original order:   [[x0, x1, x2], [x3, x4, x5], [x6, x7, x8]]
    # Sort pixels in    [[x7, x2, x6], [x3, x0, x1], [x8, x4, x5]] order.
    neighborhoodPixels = [
        neighborhoodPixels[4], neighborhoodPixels[5], neighborhoodPixels[1],
        neighborhoodPixels[3], neighborhoodPixels[7], neighborhoodPixels[8],
        neighborhoodPixels[2], neighborhoodPixels[0], neighborhoodPixels[6]]
    # Return Neighborhood Pixels.
    return neighborhoodPixels
```

b. Detect the edge and counting in Yokoi

```python
def hFunctionYokoi(b, c, d, e):
    """
    :type b: int
    :type c: int
    :type d: int
    :type e: int
    :return type: str
    """
    if ((b == c) and (b != d or b != e)):
        return 'q'
    if ((b == c) and (b == d and b == e)):
        return 'r'
    if (b != c):
        return 's'

def fFunctionYokoi(a1, a2, a3, a4):
    """
    :type a1: str
    :type a2: str
    :type a3: str
    :type a4: str
    :return type: str
    """
    # a1 == a2 == a3 == a4 == r
    if ([a1, a2, a3, a4].count('r') == 4):
        # Return label 5 (interior).
        return 5
    else:
        # Return count of 'q'.
        # 0: Isolated, 1: Edge, 2: Connecting, 3: Branching, 4: Crossing.
        return [a1, a2, a3, a4].count('q')
```

c. Mapping Yokoi number to image

```python
def YokoiConnectivityNumber(originalImage):
    """
    :type originalImage: Image (from PIL)
    :return type: numpy array
    """
    # Allocate memory space of Yokoi Connectivity Number.
    YokoiConnectivityNumber = np.full(downsamplingImage.size, ' ')

    # Scan each column in original image.
    for c in range(originalImage.size[0]):
        # Scan each row in original image.
        for r in range(originalImage.size[1]):
            # This point is object.
            if (originalImage.getpixel((c, r)) != 0):
                # Get neighborhood pixel values.
                neighborhoodPixels = getNeighborhoodPixels(originalImage, (c, r))
                YokoiConnectivityNumber[c, r] = fFunctionYokoi(
                    hFunctionYokoi(neighborhoodPixels[0], neighborhoodPixels[1], neighborhoodPixels[6],
                    hFunctionYokoi(neighborhoodPixels[0], neighborhoodPixels[2], neighborhoodPixels[7],
                    hFunctionYokoi(neighborhoodPixels[0], neighborhoodPixels[3], neighborhoodPixels[8],
                    hFunctionYokoi(neighborhoodPixels[0], neighborhoodPixels[4], neighborhoodPixels[5],
            # This point is background.
            else:
                YokoiConnectivityNumber[c, r] = ' '

    # Return Yokoi Connectivity Number.
    return YokoiConnectivityNumber
```

d. Interior and putpixel 0 to Yokoi connected number 1-4

```python
def getInteriorImage(YokoiConnectivityNumber):
    """
    :type YokoiConnectivityNumber: numpy array
    :return type: Image (from PIL)
    """
    from PIL import Image
    # New image with the same size as Yokoi connectivity number and 'binary' format.
    interiorImage = Image.new('1', YokoiConnectivityNumber.shape)
    # Scan each column in interior image.
    for c in range(interiorImage.size[0]):
        # Scan each row in interior image.
        for r in range(interiorImage.size[1]):
            # If this pixel is interior(label = 5).
            if (YokoiConnectivityNumber[c, r] == '5'):
                # Put white pixel to interior image.
                interiorImage.putpixel((c, r), 1)
            # If this pixel is not interior.
            else:
                # Put black pixel to interior image.
                interiorImage.putpixel((c, r), 0)
    # Return interior image.
    return interiorImage
```

e. Dilation kernel 3 3 3 to image

```python
def dilation(originalImage, kernel):
    """
    :type originalImage: Image (from PIL)
    :type kernel: numpy array
    :return type: Image (from PIL)
    """
    from PIL import Image
    # Get center position of kernel.
    centerKernel = tuple([x // 2 for x in kernel.shape])
    # New image with the same size and 'binary' format.
    dilationImage = Image.new('1', originalImage.size)
    # Scan each column in original image.
    for c in range(originalImage.size[0]):
        # Scan each row in original image.
        for r in range(originalImage.size[1]):
            # Get pixel value in original image at (c, r).
            originalPixel = originalImage.getpixel((c, r))
            # If this pixel is object (1, white).
            if (originalPixel != 0):
                # Paste kernel on original image at (c, r).
                # Scan each column in kernel.
                for x in range(kernel.shape[0]):
                    # Scan each row in kernel.
                    for y in range(kernel.shape[1]):
                        # Only paste '1' value from kernel.
                        if (kernel[x, y] == 1):
                            # Calculate destination x, y position.
                            destX = c + (x - centerKernel[0])
                            destY = r + (y - centerKernel[1])
                            # Avoid out of image range.
                            if ((0 <= destX < originalImage.size[0]) and \
                                (0 <= destY < originalImage.size[1])):
                                # Paste '1' value on original image.
                                dilationImage.putpixel((destX, destY), 1)
    # Return dilation image.
    return dilationImage
```

f. Using Yokoi connected number to thinning image

```python
def getThinningImage(originalImage, YokoiConnectivityNumber, markedImage):
    """
    :type originalImage: Image (from PIL)
    :type YokoiConnectivityNumber: numpy array
    :type markedImage: Image (from PIL)
    :return type: Image (from PIL)
    """
    from PIL import Image
    # New image with the same size as original image and 'binary' format.
    thinningImage = Image.new('1', originalImage.size)
    # Scan each column in thinning image.
    for c in range(thinningImage.size[0]):
        # Scan each row in thinning image.
        for r in range(thinningImage.size[1]):
            # If this point is removable(label = 1) and is in marked image.
            if (YokoiConnectivityNumber[c, r] == '1' and markedImage.getpixel((c, r)) != 0):
                # Remove this pixel from original image.
                thinningImage.putpixel((c, r), 0)
            else :
                thinningImage.putpixel((c, r), originalImage.getpixel((c, r)))
    # Return thinning image.
    return thinningImage
```

g. Check the image pixels value is equal or not

```python
def isEqualImage(image1, image2):
    """
    :type image1: Image (from PIL)
    :type image2: Image (from PIL)
    :return type: bool
    """
    from PIL import ImageChops
    return ImageChops.difference(image1, image2).getbbox() is None
```

Execute thinning process

```python
if __name__ == '__main__':
    from PIL import Image
    import numpy as np

    # Load image from file.
    originalImage = Image.open('lena.bmp')
    # Get binary image.
    binaryImage = getBinaryImage(originalImage, 128)
    # Save binary image fo file.
    binaryImage.save('binary.bmp')

    # Define kernel for dilation.
    kernel = np.array([
        [1, 1, 1],
        [1, 1, 1],
        [1, 1, 1]])

    # Get downsampling image.
    downsamplingImage = downsampling(binaryImage, 8)

    # Clear iteration counter.
    i = 0
    # Pass binary image to thinning image.
    thinningImage = downsamplingImage
    while True:
        # Get Yokoi Connectivity Number.
        YokoiConnectivityNumber = getYokoiConnectivityNumber(thinningImage)
        # Get interior image from Yokoi Connectivity Number.
        interiorImage = getInteriorImage(YokoiConnectivityNumber)
        # Get marked image by dilation of interoir image..
        markedImage = dilation(interiorImage, kernel)
        # Get thinning image.
        tempImage = getThinningImage(thinningImage, YokoiConnectivityNumber, markedImage)
        # If this iteration doesn't change image.
        if (isEqualImage(tempImage, thinningImage)):
            # Jump out this while loop.
            break
        # Update thinning image.
        thinningImage = tempImage
        # Increase iteration counter.
        i = i + 1
        # Show iteration counter on console.
        print ('Iteraion: ', i)
        # Save thinning image fo file.
        thinningImage.save('thinning' + str(i) + '.bmp')

    # Save Yokoi Connectivity Number to file.
    np.savetxt('YokoiConnectivityNumber.txt',
        YokoiConnectivityNumber.T,
        delimiter='', fmt='%s')
```