Computer Vision Homework 7

R08922079 資工所一 洪浩翔

Part 0

```python
import cv2
import numpy as np

def readImg(filename='lena.bmp'):
    #read img
    image = cv2.imread(filename, cv2.IMREAD_GRAYSCALE)
    #print('shape:', image.shape)
    #binarize
    index = np.where(image >= 128)
    binary = np.zeros(image.shape)
    binary[index] = 255
    cv2.imwrite('binary.jpg', binary)
    return image, binary

def downSample(image):
    #down sample to 64*64
    w,h = image.shape
    result = np.zeros((int(w/8), int(h/8)))
    for i in range(0,w,8):
        for j in range(0,h,8):
            result[int(i/8),int(j/8)] = image[i,j]
    cv2.imwrite('downSample.jpg', result)
    return result
```

Pre-processing and down-sampling code fragment

Part 1

(a) Write a program which does thinning on a down-sampled image (lena.bmp).

Result of thinning operator

```
class yokoiConnect(object):
    def __init__(self):
        super(yokoiConnect, self).__init__()

    def hFunction(self, b, c, d, e):
        if b == c and (d != b or e != b):
            return "q"
        elif b == c and (d == b and e == b):
            return "r"
        else:
            return "s"

    def counter(self, record):
        countQ = 0
        countR = 0
        for i in record:
            if i == "q":
                countQ += 1
            elif i == "r":
                countR += 1
        if countR == 4:
            return 5
        else:
            return countQ

    def yokoi(self, image, counter):
        result = np.zeros(image.shape)
        result.fill(-1)
        image = np.pad(image, ((1,1),(1,1)), 'constant', constant_values=0)
        #print('padding shape:', image.shape)
        w, h = image.shape
        with open('yokoi_'+(str(counter))+'.txt', "w") as f:
            for i in range(1, w-1):
                for j in range(1, h-1):
                    if image[i,j] == 255:
                        record = []
                        record.append(self.hFunction(image[i,j], image[i+1,j], image[i+1,j-1], image[i,j-1]))
                        record.append(self.hFunction(image[i,j], image[i,j-1], image[i-1,j-1], image[i-1,j]))
                        record.append(self.hFunction(image[i,j], image[i-1,j], image[i-1,j+1], image[i,j+1]))
                        record.append(self.hFunction(image[i,j], image[i,j+1], image[i+1,j+1], image[i+1,j]))
                        num = self.counter(record)
                        f.write(str(num))
                        result[i-1,j-1] = num
                    else:
                        f.write(' ')
                f.write('\n')
        return result
```

Code for Yokoi

```python
class pairRelation(object):
    def __init__(self):
        super(pairRelation, self).__init__()
    def hFunction(self, a, m=1):
        if a == m:
            return 1
        else:
            return 0
    def assign(self, Xs, m=1):
        #Xs = [X0, X1, X2, X3, X4]
        counter = 0
        for i in range(1, len(Xs)):
            counter += self.hFunction(Xs[i], m)
        if counter < 1 or Xs[0] != m:
            return 0 # 0 == "q"
        else:
            return 1 # 1 == "p"
    def pair(self, image, counter):
        result = np.zeros(image.shape)
        result.fill(-1)
        image = np.pad(image, ((1,1),(1,1)), 'constant', constant_values=0)
        #print('padding shape:', image.shape)
        w, h = image.shape
        with open('pair_'+(str(counter))+'.txt', "w") as f:
            for i in range(1, w-1):
                for j in range(1, h-1):
                    if image[i,j] != -1:
                        num = self.assign([image[i,j], image[i+1,j], image[i,j-1], image[i-1,j], image[i,j+1]])
                        if num == 0:
                            f.write("q")
                        else:
                            f.write("p")
                        result[i-1,j-1] = num
                    else:
                        f.write(' ')
                f.write('\n')
        return result
```

Code for pair relation mark

```python
class connectShrink(object):
    def __init__(self):
        super(connectShrink, self).__init__()
    def hFunction(self, b, c, d, e):
        if b == c and (d != b or e != b):
            return 1
        else:
            return 0
    def fFunction(self, record, x):
        if sum(record) == 1:
            return "g"
        else:
            return x
    def connShrink(self, image, markedImg, counter):
        result = np.zeros(image.shape)
        result.fill(-1)
        image = np.pad(image, ((1,1),(1,1)), 'constant', constant_values=0)
        #print('padding shape:', image.shape)
        w, h = image.shape
        with open('connShk_'+(str(counter))+'.txt', "w") as f:
            for i in range(1, w-1):
                for j in range(1, h-1):
                    if image[i,j] == 255:
                        record = []
                        record.append(self.hFunction(image[i,j], image[i+1,j], image[i+1,j-1], image[i,j-1]))
                        record.append(self.hFunction(image[i,j], image[i,j-1], image[i-1,j-1], image[i-1,j]))
                        record.append(self.hFunction(image[i,j], image[i-1,j], image[i-1,j+1], image[i,j+1]))
                        record.append(self.hFunction(image[i,j], image[i,j+1], image[i+1,j+1], image[i+1,j]))
                        num = self.fFunction(record, image[i,j])
                        if num == 'g' and markedImg[i-1,j-1] == 1:
                            f.write(num)
                            result[i-1,j-1] = 1
                            image[i,j] = 0
                        else:
                            f.write('*')
                    else:
                        f.write(' ')
                f.write('\n')
        return result
```

Code for connective shrink operator

```python
def cmp(image, markedImg, shrinkImg):
    #cmp two result and determine deleting pixel or not
    w,h = markedImg.shape
    result = image.copy()
    flag = 0
    for i in range(w):
        for j in range(h):
            if markedImg[i,j] == 1 and shrinkImg[i,j] == 1:
                result[i,j] = 0
                flag = 1
    return result, flag
```

Code for comparing marked and shrunk images and determining whether to stop

```python
def main():
    image, binary = readImg()
    down = downSample(binary)

    yokoiConn = yokoiConnect()
    pairRelat = pairRelation()
    connShrk = connectShrink()
    counter = 0
    while True:
        print(counter)
        #step 1
        yokoiResult = yokoiConn.yokoi(down.copy(), counter)
        #step 2
        markedImg = pairRelat.pair(yokoiResult, counter)
        #step 3
        connResult = connShrk.connShrink(down.copy(), markedImg, counter)
        #check
        result, flag = cmp(down, markedImg, connResult)
        cv2.imwrite("thinning_"+str(counter)+".jpg", result)
        counter += 1
        if flag == 1:
            down = result.copy()
        else:
            cv2.imwrite("thinning.jpg", result)
            break

if __name__ == "__main__":
    main()
```

Main function for thinning

Thinning result in each iteration