CS270 Homework 3 Report

任怡静 2018533144

Question 1: Graph Cut for Image Segmentation (50 points)

- Please describe your algorithms in words or flowcharts. (15’)
  - **The Seed Collecting**
    - This part used the OpenCv built-in functions to catch mouse operations to append coordinates to seed lists (**foreground_seeds** and **background_seeds**) and illustrate seeds on image
    - The image size is adjusted for drawboards so that user can draw accurately, the saved results will change the size back
  - **The Cut Graph Algorithm**
    - I first calculate a graph from seeds obtained in the previous step, which assign 0 to background_seeds' corresponding coordinates in **graph**, 1 to foreground_seeds' and 0.5 for the rest.
    - Then according to the **graph**, I can construct the node list and edge list for **maxflow graph**. I append **node** (node_flatcoord, capacity_towards_source, capacity_towards_sink) into the **nodeList**, and **edge** (curr_index, neighbor_index, capacity) into the **edgeList**. It follows the rule that if the point in the **graph** is 1 then I append (node_flatcoord, 0, MAXIMUM), 0 append (node_flatcoord, MAXIMUM, 0) and (node_flatcoord, 0, 0) to the rest. I calculate and append two edges into **edgeList** for each points in the **graph**, whose capacity using $\frac{1}{1+Euclidean(I(x,y),I(x+1,y))}$ and $\frac{1}{1+Euclidean(I(x,y),I(x,y+1))}$
    - Then I use built-in maxflow package to construct the maxflow graph **g**. First connect all nodes to source and sink, then for nodes which has edges to each other (have edge in **edgeList**) add edges between them. Finally do the **g.maxflow()** to get the maxflow result.
    - Then I obtain the mask that the foreground is valued 1 and background 0, so that I can generate the mask and overlays based on **g.get_segment(index)**
  - **The multi-segments division**
    - I reuse the cut graph algorithm. T
      - To divide multiple parts, I create 4 lists to collect four kinds of seeds, select one to be the foreground and others combined to be the background
      - Loop this procedure for all four lists as they all become once the foreground. Then I check the mask before filling in color to see if it is occupied by other colors before thus only fill in the non-occupied parts.
    - I modified the GUI to fit for four kinds of seeds by pressing '1', '2', '3', '4' in the keyboard
- As shown in figure 1 you need to perform graph cut method to segment the foreground and background of the given image q1_1.jpeg. (15’)
  - First original image, second seed record image, third mask image, fourth mask-origin overlayed image

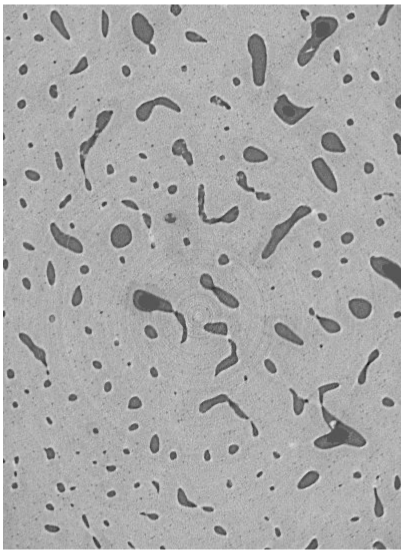
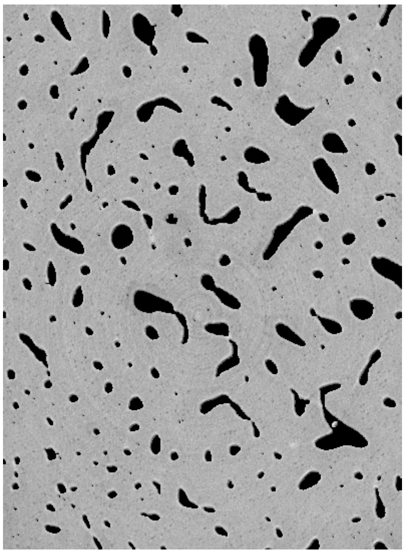
- Modify your graph cut program for multi-class segmentation. You need to segment the given image q1_2.jpeg to four parts, and show the segmentation results via transparent painting overlayed with the origin image as shown in Figure 2. (20')
  - First original image, second seed record image, third masks image, fourth masks-origin overlayed image

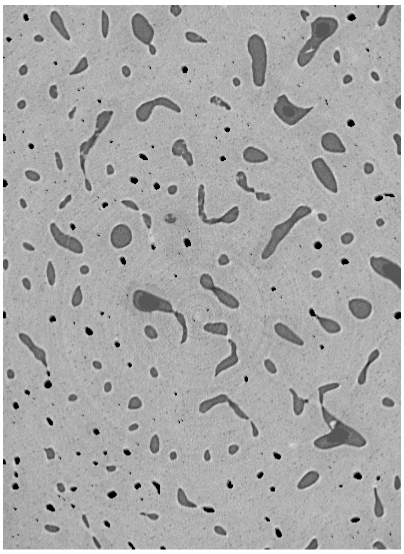
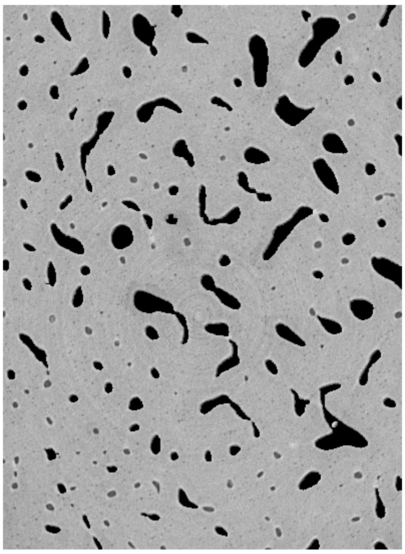
Question 2: Canal classification (50 points)

- Overall illustration of the process you designed. (5')
  - **Image binarize and closing, sort out background**
    - In this step I use threshold() and morphologyEx() to extract the canals from the background, representing canals as white and background as black
    - Extract black areas in the binarized image in the original image to form the background image **layer_background** (I_b)
  - **8-connection white pixel finding and sort large and small canals**

- **Calculate distortion** $\sum_{i \in I_\epsilon} I_\epsilon(i)$

- Your metric of dividing canals into large and small canals. You may describe it in mathematical and natural languages. (10')
  - First I define **BFS()** and **EightNeighbors()** to find any white pixel fragments that inside pixels are connected by 8 connection rule in the binarized image , save their coordinates tuple into a list **area** for each fragment
  - Collect all the white fragment lists and put them into another list **segments**
  - **Sort large and small canals**
    - Iterate through **segments**, find element list **s** whose length is larger than 80 (contains more than 80 pixels), iterate through **s** and extract all pixels indicated by element tuples in list **s** and extract corresponding coordinates into a graph **layer_large** from the original image to gather the large canals. Vice versa, collect all small canals in another graph **layer_small**
    - **layer_large** = **layer_large** + **layer_background**
    - **layer_small** = **layer_small** + **layer_background**
- Classification results, I_b, I_l and I_s. (24', 7' for each)
  - First original image, second I_b, third I_l, fourth I_s

- Result of element-wised summation of $I_\epsilon, i.e., \sum_{i \in I_\epsilon} I_\epsilon(i)$(6')

- $\sum_{i \in I_\epsilon} I_\epsilon(i) \ = \ 0$

```
Windows PowerShell
版权所有 (C) Microsoft Corporation。保留所有权利。

尝试新的跨平台 PowerShell https://aka.ms/pscore6

PS D:\Rigin_Rain\Classes\CS270\ShanghaitechCS270-DIPHws\hw3-任怡静\hw3_任怡静_2018533144> conda activate base
PS D:\Rigin_Rain\Classes\CS270\ShanghaitechCS270-DIPHws\hw3-任怡静\hw3_任怡静_2018533144> cd code
PS D:\Rigin_Rain\Classes\CS270\ShanghaitechCS270-DIPHws\hw3-任怡静\hw3_任怡静_2018533144\code> python Q2.py "..\material\images\q2.jpeg" "..\result\Q2\"
the summation of I_epsilon is:  0
PS D:\Rigin_Rain\Classes\CS270\ShanghaitechCS270-DIPHws\hw3-任怡静\hw3_任怡静_2018533144\code>
```

- Code and result illustrations, including filenames and variable names of the results above. (5')
  - For function **EightNeighbors()**:
    - It takes two parameters **layer** and **coordinate**, which **layer** implies the binarized 2D image, and **coordinate** is a tuple that means the 2D coordinate of the pixel that you want to find its 8-connected neighbors
    - The function iterate in list **directions** to get the pixels around, and append pixel that are legal (within the origin image) coordinates in tuple into the scope list **eight_neighbors** and return it
  - For function **BFSConnected()**:
    - It takes a parameter **connected_list**, which means the 8 connected neighbor list for the previous pixel, and initial input should be the found white pixel alone in a list.
    - The function perform a recursive BFS search from one pixel. It returns when no 8-connected neighbors are found
    - else it will create an empty list **neigbor_list** iterate the coordinates tuples **coord** in **connected_list**:
      - Mark the pixel in the global variable **layer** (a copy of binarized 2D image (**th3**) in binarize operation) black
      - Append **coord** into list **area**, which will be new each time calls **BFSConnected**() in main function search loop (see the main function description)
      - Use **EightNeighbors()** to find **coord**'s 8-connected neighbors and save them in list **neighbors**, iterate in **neighbors**, if the element is not in **neibor_list** created before, add it to **neigbor_list**, this step is to prevent redundant recursive calls since pixels in picture has more connections than a traditional tree.
      - Finally call **BFSConnected()** on the newly obtained **neigbor_list**
  - For the main function:
    - It first read in the image, transfer it to gray image **image_gray** using **cv2.cvtColor()**, then use **cv2.threshold()** to binarize **image_gray**, save the binarized result in **th3**, then create two kinds of rectangle kernels of size [3,3] and [4,4], use first 3 then 4 kernel to perform **cv2.morphologyEx(th3, cv2.MORPH_CLOSE, kernel)** so that the noises are cleared in **th3**
    - Allocate two empty lists: **segments** and **area**, do a copy of **th3** into **layer**, allocate an image-size all zero matrix **layer_background**, then iterate in **layer**:
      - If **layer[i,j] > 0**, meaning I find a start of an area of canal:
        - I clear up the area list, setup the input **connected_list** to [(i,j)] for **BFSConnected(connected_list)**
        - When **BFSConnected()** finished, it will append **area** to **segments**, which will save all the canal segments' coordinates lists
      - If **th3[i,j] == 0**, meaning I find a pixel of background

        - Extract the original pixel **image[i,j]** into **layer_background**
- Allocate two image-size all zero matrix **layer_large** and **layer_small** ,Iterate in **segments**:
    - For element list **s** in segments, if its length is larger than 80, iterate **s** and each element is (x,y), extract the original pixel **image[x,y]** into **layer_large**
    - Vice Versa, extract the original pixel **image[x,y]** into **layer_small**
- Adjust **layer_large** and **layer_small** into final results by adding **layer_background** into them
- Save **layer_background**, **layer_large** and **layer_small**
- Calculate **layer_epsilon** by **layer_epsilon** = **layer_large** + **layer_small** - **layer_background** - **image**, then calculate the sum of layer_epsilon by using **np.sum()**, print the sum result