COMP 9517

S1, 2018

Assignment 1: Specification

Maximum marks achievable: 10 marks

This assignment is worth 10% of the total course marks. The optional parts of the assignment will attract additional marks as specified and will count towards the final course mark.

Preliminaries: The assessment environment

Please ensure that your code compiles and runs on the generic CSE (linux) vlab environment. You can find details about access to vlab remotely here:
https://www.cse.unsw.edu.au/~cs1511/17s2/home-computing/vlab/

The following software is available to you in the testing environment:
Python 2.7.13
gcc (Debian 4.9.2-10+deb8u1) 4.9.2 (supports C++11)

The following libraries are also available:
OpenCV 2.4.9.1
Numpy
(don't assume other libraries are available!)

If you are using the most recent release of OpenCV to develop with, note there are a variety of differences between old and new summarized here:
https://docs.opencv.org/master/db/dfa/tutorial_transition_guide.html

You **must** use either C++ or Python for this assignment.

You should submit 2 (optional 3) files:
otsu_threshold.{py, cpp}
count_nodules4.{py,cpp}
count_nodules8.{py,cpp}
You shouldn't use any of the following OpenCV library functions:
`threshold`
`adaptiveThreshold`
`watershed`
`findContours`

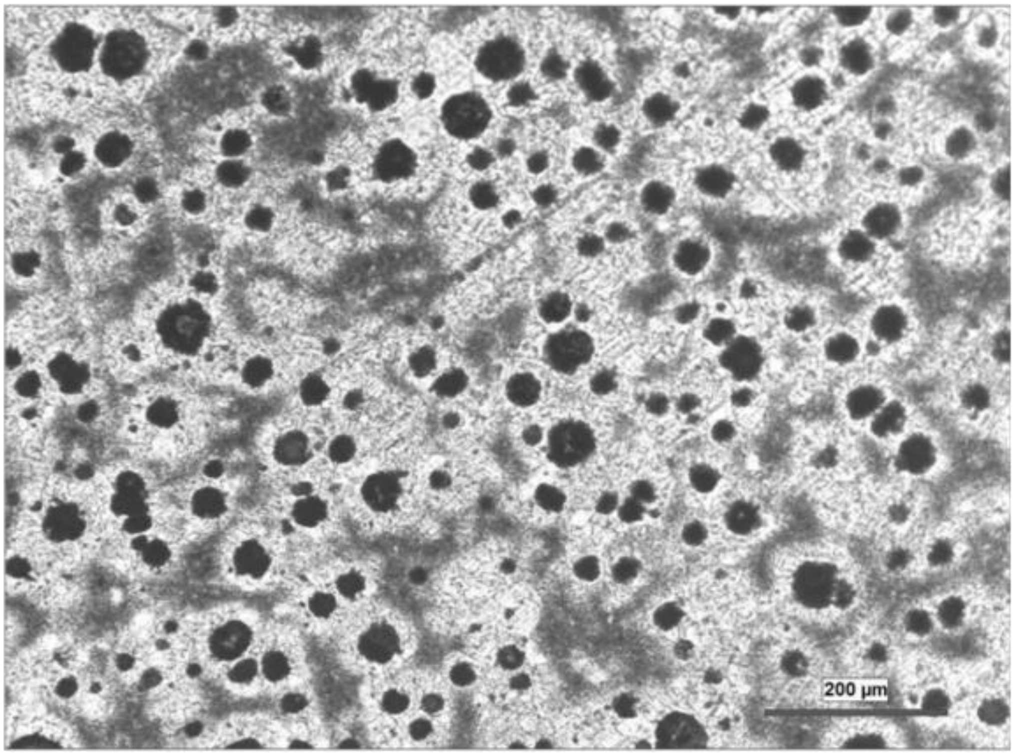
```
contourArea
drawContours
connectedComponents
```

Each subsection of the code should be able to be run as a command line run as follows

```
python step1 --input my_input.jpg --output my_output.jpg --
optional_argument a
```
or
```
./step1 --input my_input.jpg --output my_output.jpg --
optional_argument a
```

The problem: *Segmentation by thresholding*

Computer vision techniques are fruitfully applied in microscopy with applications to areas such as microbiology, electronics, and material science. Micrography represents a simplified domain where we can automate parts of the quality control process. Cast iron is a widely used but brittle material, and this brittleness can be reduced by adding a large amount of carbon to the molten iron. Our aim for this part of the assignment is to develop an automated method for counting graphite (carbon) flakes in micrographs of ductile iron. The graphite flakes appear as black circles in the image below.



You can download the images from the class website.

This task of isolating groups of pixels of interest in an image is called segmentation and is very challenging to perform in full generality. In this assignment we shall combine a method for thresholding greyscale images (Step 1) with a simple contour finding algorithm (Step 2).

Step 1: Adaptive Thresholding (6 marks)

Otsu's algorithm is a simple way to determine a threshold to separate pixels in a greyscale image. It assumes that he image contains two classes of pixels - *foreground* and *background* and has a *bi-modal histogram*. It then attempts to minimize their combined spread (intra-class variance) by finding the threshold (in the range [0, 255]) that best minimizes the weighted sum of the within-class variances. That is, we wish to minimize the quantity:

$$\sigma^2{}_W(t) = \omega_0(t)\,\sigma^2{}_0(t) + \omega_1(t)\,\sigma^2{}_1(t)$$

where t is the proposed threshold, $\omega_i(t)$is the sum of probabilities in class i, and $\sigma^2{}_i(t)$ is the variance of class i.

It was shown by Otsu that this is equivalent to maximizing the between-class variances, thus the overall algorithm can be described concisely in the following way.

```
For each potential threshold T in [0,255] do:
   1. Separate the pixels into two clusters according to the threshold.
   2. Find the mean of each cluster.
   3. Square the difference between the means.
   4. Multiply the number of pixels in one cluster by the number in the
      other.
Return the threshold T corresponding to the largest value of the inter-
class variance by application of steps 1 to 4.
```

(i) Implement Otsu's method to find a good threshold to generate a binary image.

For convenience of displaying the image, ensure the nodules are encoded as [0,0,0] (i.e. black) and the background is encoded as [255, 255, 255] (i.e. white).

- Input: a greyscale image
- Output: a binary image png image
- Include optional argument which prints the threshold found
- Example:
  ```
  python otsu_threshold.py --input input.png --output binary.png --
  threshold
  ```

(ii) *split the image into a* ***grid of cells*** of given size and then apply Otsu's method on each cell treating it as a separate image (and presuming a bi-modal histogram). If a sub-image cannot be thresholded well, then the threshold from one of the neighbouring cells can be used. Use the same encoding of 0/255 for nodules/background as before.

- Input: a greyscale image, grid size
- Output: a binary image png image
- Example:
```
python grid_otsu_threshold.py --input input.png n --output binary.png
```

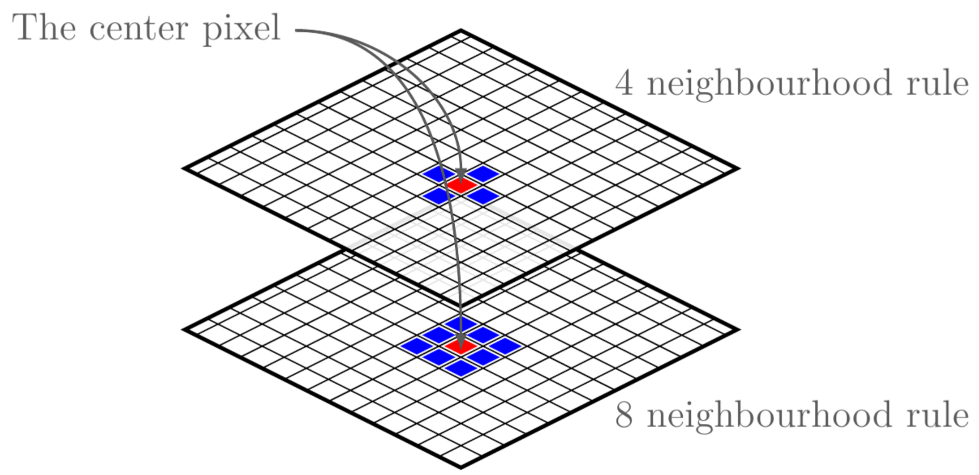
(iii) It may be necessary to apply some morphological operators or filtering to get good results in Step 2. Without such preprocessing, noisy pixels may affect the segment quality and quantity. You are encouraged to try different operators in OpenCV for this purpose.

- Input: a binary image png image
- Output: a binary image png image
- Example:

  -
    ```
    python filter.py –-input input.png n –-output output.pn
    ```

Step 2: Nodule counting using 4-neighbourhood (4 marks)

In this section we shall count the number of graphite nodules in each image by counting the number of connected components in the underlying graph. For the purposes of nodule counting, consider two pixels to be adjacent if they share the same colour/intensity and they belong to the four-neighbourhood as illustrated below:



Find the connected components in the graph generated by this procedure to isolate the nodules in the original image. Thus, count the number of graphite nodules in the original image.

You should apply the two-pass connected component algorithm:

On the first pass:

```
Iterate through each pixel in the image matrix:
     If the pixel is not the background:
          Get the neighbouring elements of the current pixel
     If there are no neighbours:
          uniquely label the current pixel and continue
     else:
          find the neighbour with the smallest label and assign it to
     the current pixel
     Store the equivalence between neighbouring labels
```

On the second pass:

```
Iterate through each element of the data by column, then by row
     If the element is not the background:
          Relabel the element with the lowest equivalent label
```

(source: https://en.wikipedia.org/wiki/Connected-component_labeling)

- Input: a binary image, minimum area n
- Output: the number of nodules with area larger than n pixels in the image, written to stdout
- Include an optional argument to print each nodule to screen with a different colour.
- Example:
  ```
  python count_nodules4.py --input binary_image.png --size n --
  optional_output nodules.png
  ```

Step 3 (OPTIONAL): Nodule counting using 8-neighbourhood (2 marks)

Modify Step 2 to use 8-neighbours instead of 4-neighbours.

- Input: a binary image, minimum area n
- Output: the number of nodules with area larger than n pixels in the image, written to stdout
- Include an optional argument to print each nodule to screen with a different colour.
- Example:
  ```
  python count_nodules8.py --input binary_image.png --size n --
  optional_output nodules.png
  ```

08 March 2018