# COMP9517

Assignment 1-Report

Haowei Lou
Z5258575

Task1:

The aim for this task is to create a max/min filter to t find the maximum gray value in neighbourhood for every pixel in an image. The size of neighbourhood is defined by a free parameter called n.

My approach for this task is as follow:

1. Open the image by cv2.imread().
2. Find the dimension of the image by Image.shape and then find width and height.
3. Get free parameter n from user input.
4. Define the start point of the max/min filter, the method to define start point is as follow.
   a. In default, the x,y value of start point should be the current pixel's x,y value minus half of n, round down to integer.
   b. If the x or y value of start point is smaller than 0 after subtraction, then make it to be zero to avoid index error.
   c. If x+n or y+n is greater than the height or width of input image, then make it to equal height/width -n to avoid program go over the limit during the process of finding maximum value.
5. Create a numpy 2d-array with same shape of the Image, initialise with zero .
6. Loop all pixels inside image.
   a. For each pixel, loop from start point(x,y) as I defined in step 4).
   b. End until current pixel reaches point(x+n,y+n).
   c. Return the maximum/min value of for the pixels between start point and end point.
   d. Set the 2d-array that I created in step 5) equals to the return max\min value in c).
   e. The time complexity for finding the max\min value for each pixel in brute force is $O(n^2)$.
7. When the step 6 is finished, the numpy 2d-array is the output image, using cv2.imwrite() to output it in jpg format.

The table below shows the output image of different n value:

N=1	N=2	N=3	N=4	N=5
N=6	N=7	N=8	N=9	N=10
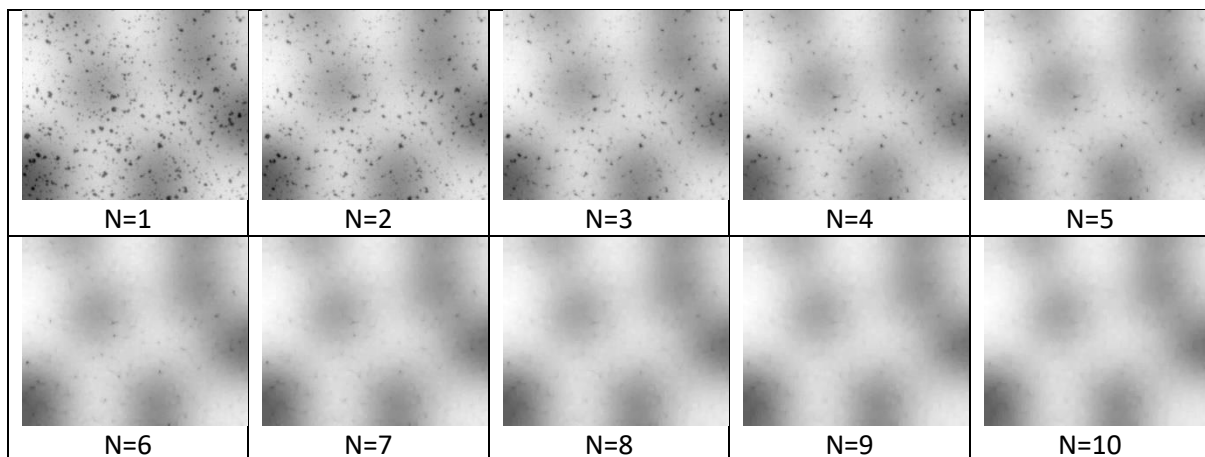
It has shown from the table that particles inside this image start becoming invisible as n value increase, and when N value reach 10, the particles inside this image become disappear. The reason for this to happen is because the max filter will find the maximum gray value inside a $n \times n$ square that in neighbour of each pixel. As the value for n increasing, this filter will become bigger and bigger hence a pixel with great gray value(let's say it is greatest value inside its filter) will influence a greater range of pixels and equalise pixels around it and the pixel will become lighter as gray value

increase. Thus, the image will become lighter because small black particles inside this image is get rid of increased gray value.

The minimum filter is similar to max filter except the value this function seeks is minimum value other than maximum. Image B computed for different N value is as follow:

| N=1 | N=2 | N=3 | N=4 | N=5 |
| N=6 | N=7 | N=8 | N=9 | N=10 |
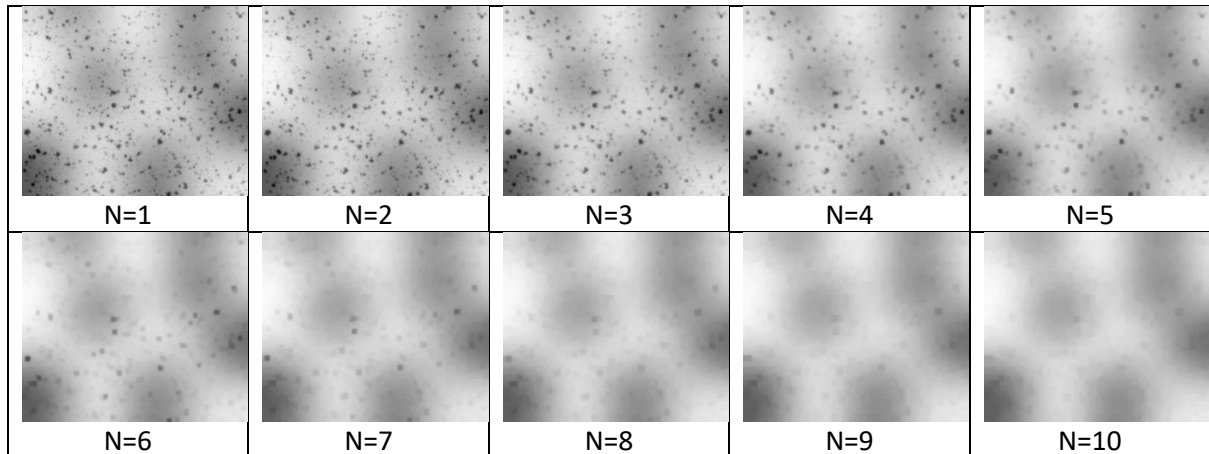
Task2:

The aim for this task is to remove the shading artifacts from I by subtract B form I pixel by pixel. My approach for this task is simply write two loops that loop through the 2d-array row by row and operating subtract operation. The formula for subtract shade operation is:

$$O[x][y] = Image[x][y] - B[x][y] + 255$$

The reason for 255 is added after subtraction is because the subtraction can only find the difference between Image's pixel and B's pixel, that cannot be used for represent the gray value. The meaning for this difference is , if it is positive it means the image's pixel is lighter otherwise it will be darker.

Gray value is representing in a range [0,255], where small number represent a dark pixel and big number is for light pixel. In order to find the gray value correctly, 255 + difference is required. If difference is negative, the pixel will go darker so the particles in image can be shown (as 255 – some number). If the difference is positive or relative small in negative(used to remove shading because image B has a similar background with Image and the difference will become very small for these background pixels), then the pixel will remain in 255 and be light in output image.
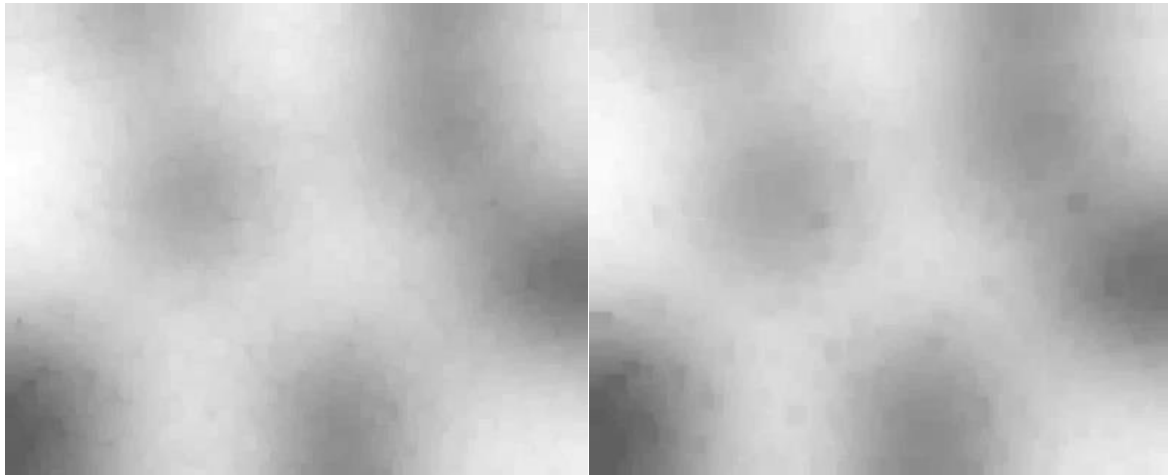
In the end, all O[x][y] will be store in a new 2d-array called O and print it out in image by cv2.imgwrite().

The output image O for different N value is as follow:

| N=1 | N=2 | N=3 | N=4 | N=5 |

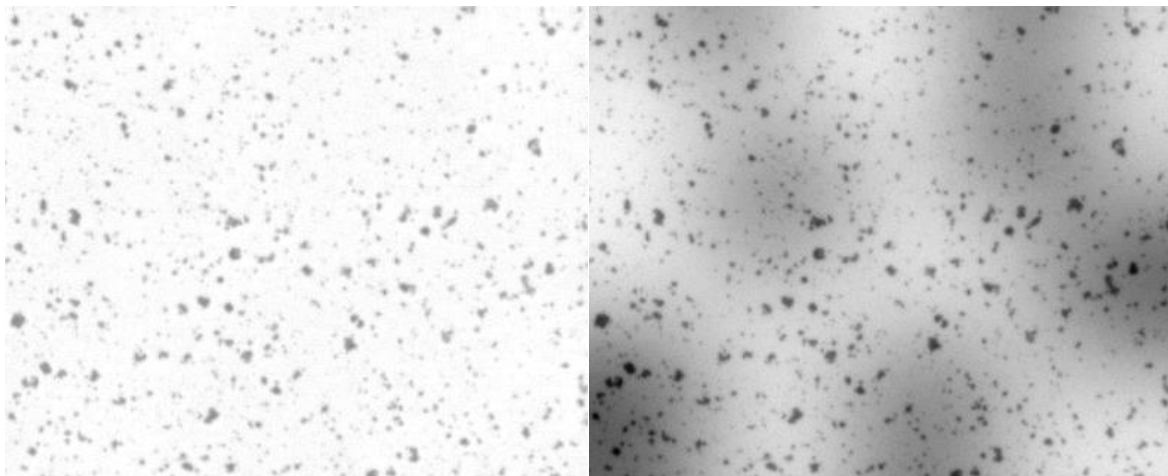|				
N=6	N=7	N=8	N=9	N=10
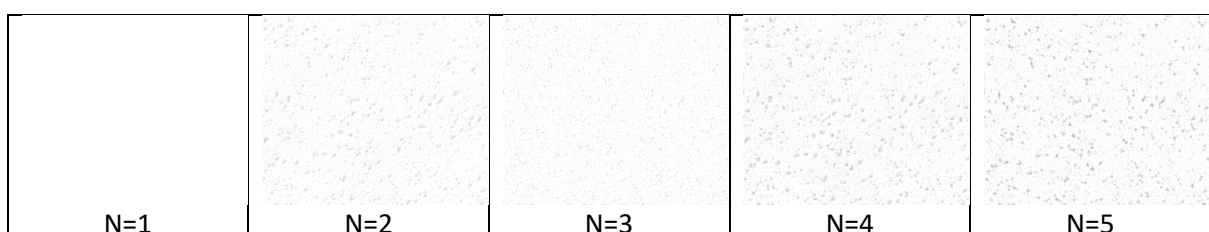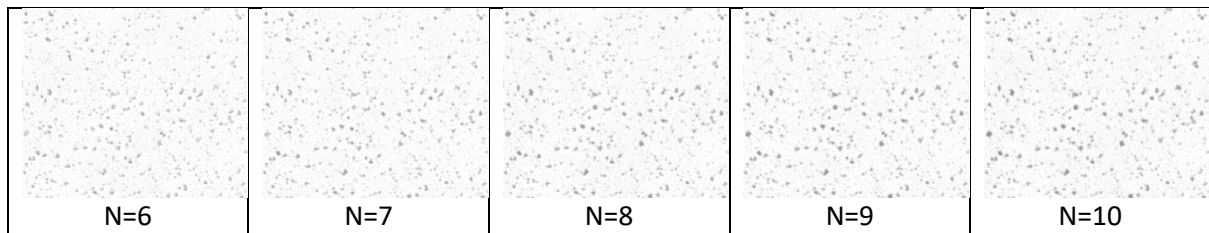
More clear image of A,B,O and Particles at n = 10:

A

B

O

Particles.png

Task 3:

The aim for this task is to find an algorithm to remove background shading of **Cells.png**. Comparing to **Particles.png**, the new image's size and size of object are relatively bigger, and the color theme transit from light background/dark object to dark background/ light object. Which means the object in **Cells.png** will have a higher gray value and background have lower. It results in the algorithm that has been developed in task 1 & 2 becomes invalid for this task.

To remove shading of cells.png, the algorithm should do mini-filtering first and then max-filtering. It is because **Cells.png**'s object is in light(high gray value) and backgroung is in dark(low gray value). Doing min-filering first can make every pixels equal to the minium value in neighbours to get rid of

object with high gray value but surround by dark background with low gray value. Hence, shading areas in this image have been succesfully estimated in image A.

Next, doing max-filtering will remove the negative affect that cause by min-filtering(gray values in A to be lower than the actual background). Max-filtering will be used for correction in this step, and store output in image B. In the end, applying O=I-B to get to output of image that has succcesfully removed shading areas.

I have added a new varaible called M to let users decide the algorithm they want. When M = 0, the algorithem will do max-filtering first, then min-filtering. If M = 1, it will do min-filtering then max-filtering. In order to remove shading areas in image, Particles.png requires M=0(max-filtering first then min-filtering) and Cells.png requires M=1(min-filtering first then max-filtering).

Refering to the table in below that contains different N value's output image. A good N value for remove shading area of **Cells.png** is N=30 because cells in image after operations become more clear when N = 30 and there has no significant changes for output image after N>30:

N=1	N=5	N=10	N=15	N=20
N=25	N=30	N=35	N=40	N=45
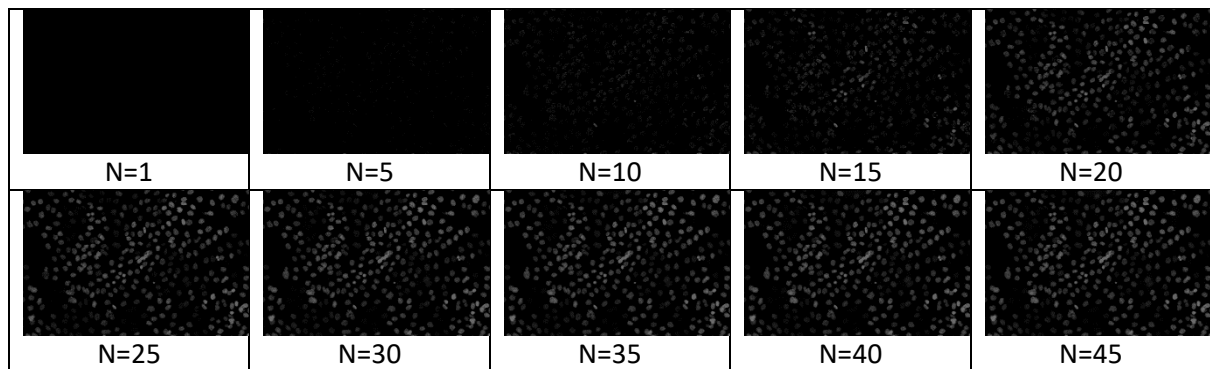
The reason for a greater N value is required for **Cells.png** is because its size is bigger than **Particles.png's** size. And the size of cells in new image is also bigger than particles in old image. A greater N value is required to differ object, shading, and background in image. Otherwise it cannot estimate the shading area properly. Hence, it makes object is also been classified as shading area in image B. In the end, these cells been subtracted from original table and make the final output become black (Refer to N=1 to 10 in upon table).
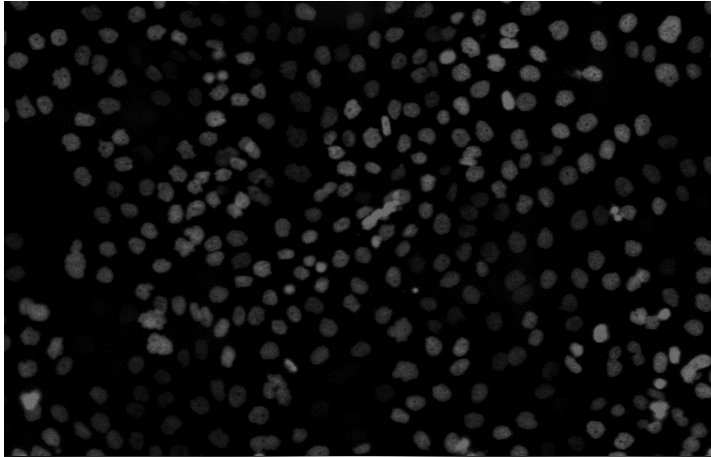
Image B and O computed for Cells.png are:

Image O

Image B