COMP9517 Computer Vision 20T2
Assignment Report

Student Name: Raymond Lu
Student Number: z5277884

Task 1: Background Estimation

In task 1, *Particles.png* is processed. The algorithm in task 1 only fits in images in grayscale mode with white background. The algorithm in task 1 could be separated into three two parts: (1) Remove dark particles; (2) Get the background that without any dark particles.

(1) Remove Particles by Max-Filter

This is realized by finding the maximum gray value in an $N \times N$ neighborhood for each pixel in the original input image O and putting the maximum value to the corresponding position (x, y) in a new image A. This process is called max-filter. Considering each pixel is the center of the $N \times N$ neighborhood in its iteration, N would be an odd number, otherwise the pixel number on the left-hand side would be different from the right-hand side, which requires further manual tuning.

For the choice of N's value, it should be the smallest value that makes no dark particles appear in image A. The reason why the smallest N value is chosen is that when the value of N increases, more details of the original image will be lost. This could be represented as some pixels whose gray values are higher than the dark color (the color of particles) and lower than the white background, will be set to bright color, losing more information in the background. Hence, we should stop when the value of N starts to satisfy that all dark particles are gone.

The output of image A with different sizes of neighborhoods is given below:

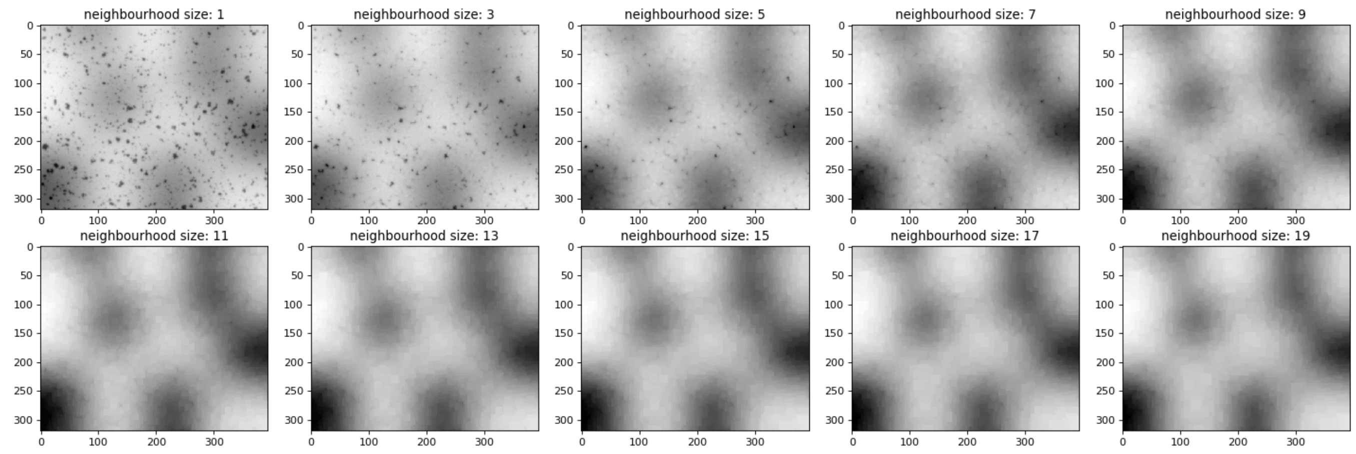

Figure 1. The output of image A in different neighborhood sizes

It could be seen that, when $N = 13$, all dark particles become vanished. This is the smallest number that all dark particles are gone. Therefore, $N = 13$ is chosen.

(2) Get the Background that Without Any Dark Particles by Min-Filter

In this step, the algorithm iterates through pixels in image A one by one to find out the minimum gray value in the current pixel's $N \times N$ neighborhood. Then, the minimum gray value is put to the corresponding position in a new image B.

In the min-filtered process, it should be noticed that the size of the neighborhood is the same as the max-filter process. This stage is to remedy the loss of low-gray-value pixels. The reason why the N value is identical as max-filter here is that we are trying to offset the effects of over-brightening and do not want to make some of the pixels too dark.

The output of image B is:

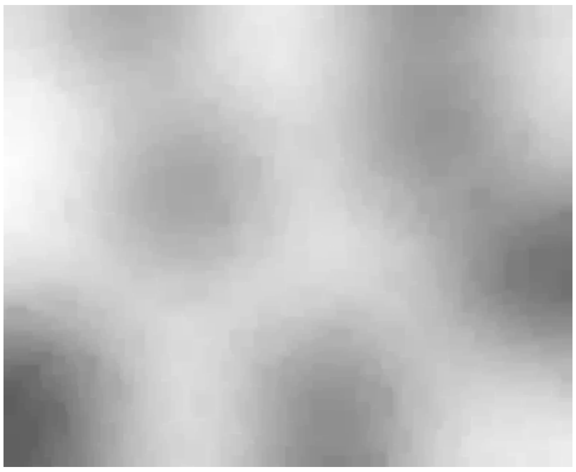

Figure 2. The output of image B

A glance at the image B reveals that there are no dark particles on the surface. This is an ideal background we want.

Task 2: Background Subtraction

In this task, we subtract Image B in task 1 from the original image I to get the output image O. Then, we add a white image in the same size of the input image and image O to make the background pure white. Eventually, we normalize the gray values in the image O.

Initially, the output of subtraction is given in Figure 3.

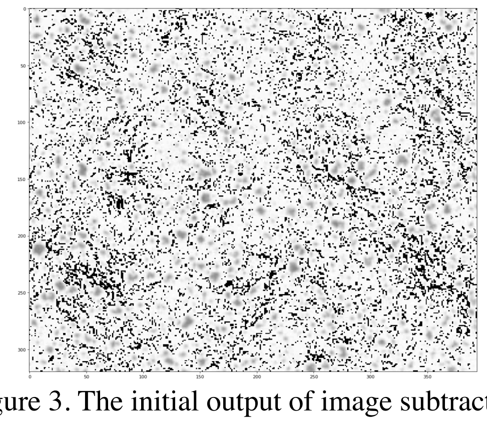

Figure 3. The initial output of image subtraction

It could be seen that the background subtraction produces black pixels and white pixels.

In the process of background subtraction, we know that after applying min-filter, the brightness of image B will be slightly higher or equal to the gray values of the background pixels in image I. As a pixel's value ranges from 0 to 255 in *OPENCV* package's setting, it should be pointed out that in background subtraction, when the result is below 0, there is an automatic adjustment that $-m$ would be compensated with 255 to be $255 - m$. Because m is a small number (the gray values of the image I's background is close to the ones of image B), $255 - m$ would be a bright color, which is close to the original white background. Nevertheless, if the image I's gray value is equal to B's, then the result will be 0, namely a black color. On account that the background color of the original image is white, we should try to turn the black parts in the background to be white. Adding a white image in the size of the input image to the image O could solve this problem because 255 (white gray value) + 0 (black gray value) = 255 (white gray value). The side effect of doing this would be any non-black(gray value is not zero) pixels will get their gray values subtracted by 1, which just could be reckoned as trivial changes.

After adding a white image in the same size, and normalizing the output, we get the output image O (Figure 4).

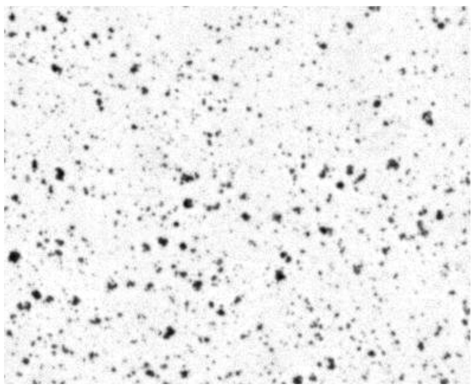

Figure 4. The normalized output image of *Particles.png*

Task 3: Algorithm Generalization

Part 1. Remove Shading in *Cells.png*

In the first part, we apply min-filter then max-filter to *Cells.png* to get an image B. Then, we subtract image B from *Cells.png* to get a temporary result image O. At last, we output the normalized image O as a result.

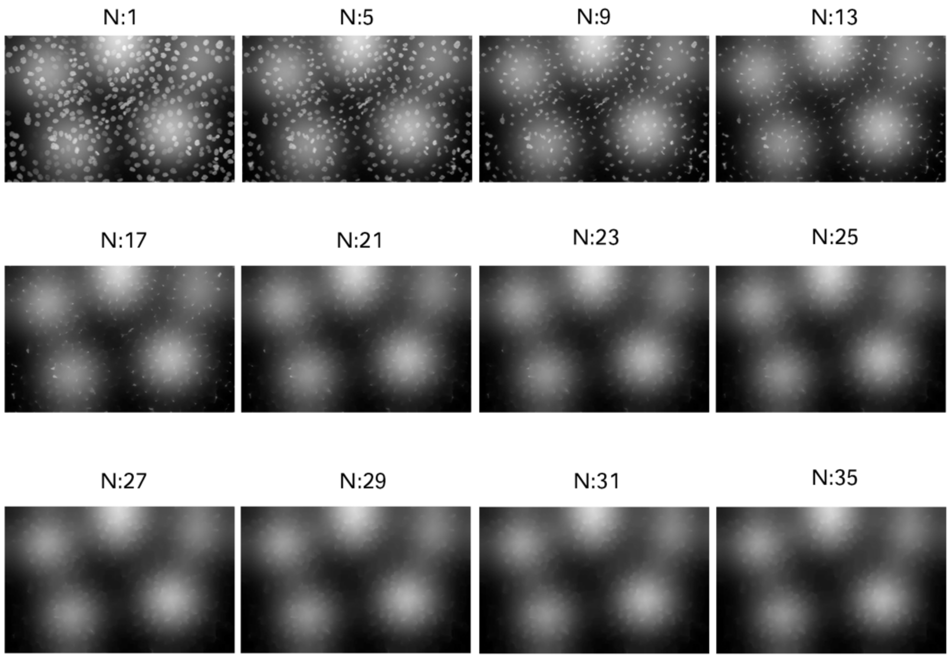

Figure 5. Min-filtered Cells.png with different neighborhood values

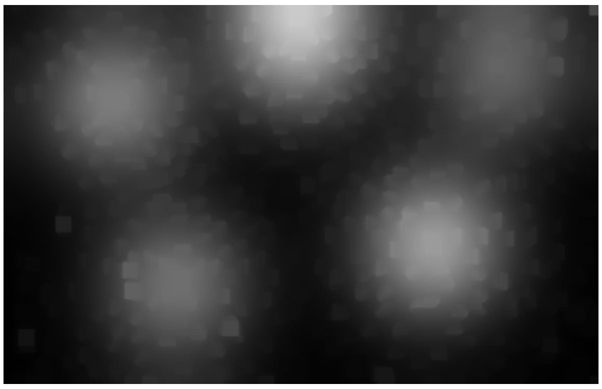
Figure 6. Background image B with $N = 27$

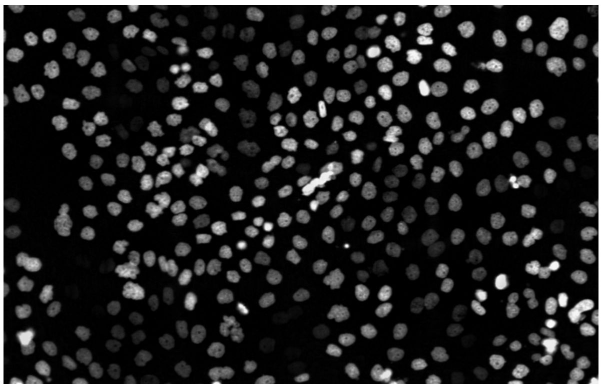
Figure 7. Cells.png after shading removal

When we try to apply min-filter to *Cells.png*, the output indicates that no matter how big the value N is, the white cells cannot vanish. Because these white cells are large and scattering in a comparatively high intensity, applying $N \times N$ neighborhood min-filter cannot completely remove these cells. What we can do here is attempting to remove bright parts in the original image, so that less brightness detailed will be lost in the subtraction process. Based on this, we choose N to be 27, the smallest number to make all bright parts nonexistent. After confirming the value of N, we apply this value to max-filter to get image B (Figure 6) for *Cells.png*. Afterward, we subtract image B from *Cells.png* and normalize the result as image O (Figure 7).

Part 2. Generalization Approach

In the second part, two parameters are introduced: the size of neighborhood N and the input image background color indicator M. The reason of the introduction of M is that the execution order of max-filter and min-filter is different for images with different backgrounds. For an input image with white background and black particles, M is equal to 0, which means that max-filter is applied first, then comes with min-filter. And the order is reversed when the input image has a black background and white particles, and we set M to be 1 here. The explanation of why the execution order matters to different images is that max-filter replaces a pixel's gray value with the maximum gray value in its $N \times N$ neighborhood, namely, filling that pixel with a brighter color. By doing this, black particles in a white background image could be removed and we are able to get the background image. Max-filter could be regarded as a filter that only lets bright colors pass through. As for min-filter, it has an opposite functionality that it only let dark colors pass through, which means bright colors are blocked.

With the parameter M, this algorithm can cope with both white and black ground images. For white background images, the algorithm sets M to 0 and applies max-filter to let dark elements on the background vanish, and then uses min-filter to make compensation to the background to get it darkened, in case that dark details in the background are lost in the max-filter phase. When it comes to black background images, the algorithm sets M to 1 and applies min-filter to let bright elements on the background disappear, and then utilizes max-filter to compensate the brightness loss in the previous step.

Another merit of this general algorithm is that it is not constraint in the size of the input image. Any arbitrary size images in grayscale can be processed in this algorithm.

In the algorithm testing phase, we test the code with *Particles.png* and *Cells.png*. Because *Particles.png* is in white background, we set $M = 0$ and vice versa for *Cells.png*. Since we have already known the best N values for these two images respectively, we just directly apply them here again.

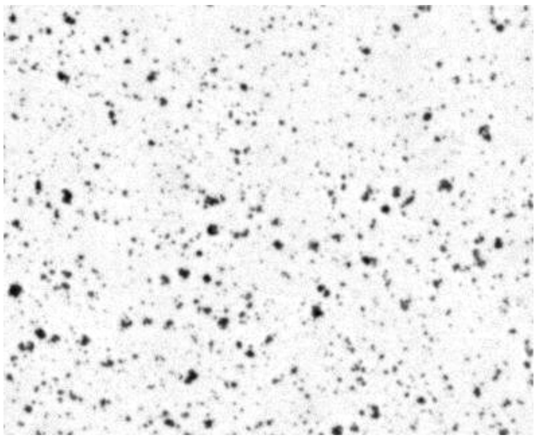
Figure 8. *Particles.png* after general shading removal algorithm applied

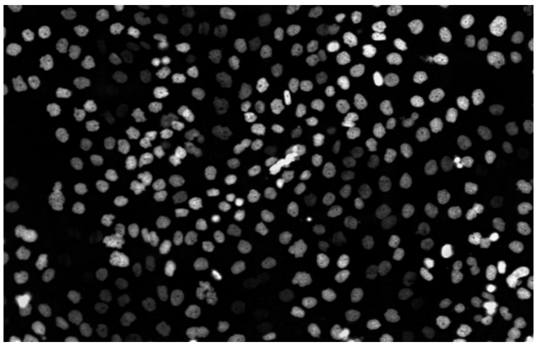
Figure 9. *Cells.png* after general shading removal algorithm applied

From the test results above, it could be seen that the outputs are the same as previous works. Hence, we can assert that the algorithm works as expected.