COMP9517 Computer Vision

2020 Term 2 Assignment 1

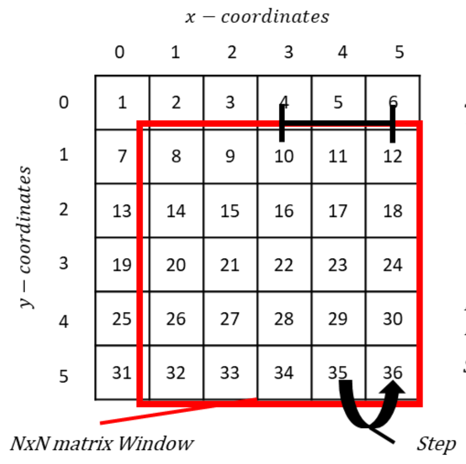
Z3417618 Kan-Lin Lu

Submitted on: 17-06-2020

Task 1 – Background Estimation

Methodology:

Two different approaches were taken in sliding the NxN window across given image. Illustration is shown below with pseudo code of the functions:



```
Steps is the number of step from center of the window to boundary of the window

Window Sliding loop:
    For y in range(0, height):
        For x in range(0,width):
            Max/Min Operation

Indicator = 'max' / 'min'
N = (Odd integers >=3)
Steps = $int\left(\frac{N}{2}\right)$  $e.g. N = 5, steps = 2$
```

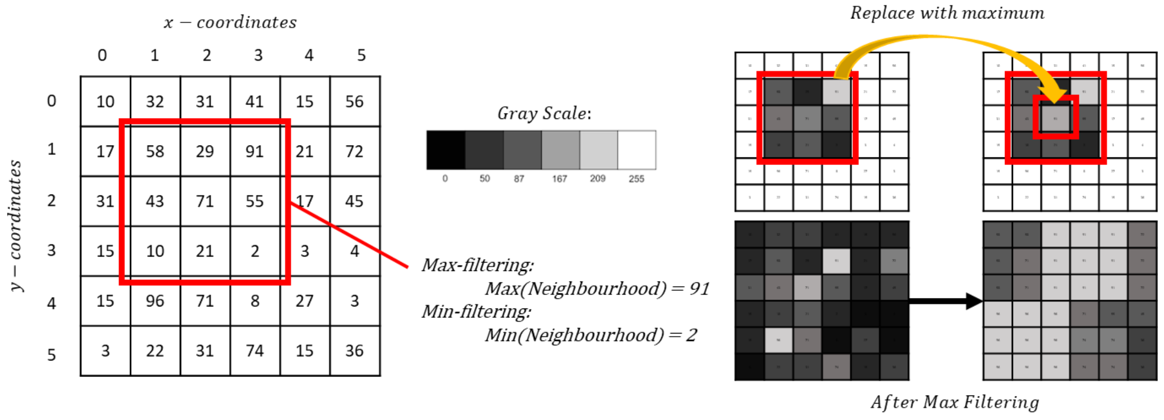
```
Brute Force Method:
    Input: Input_Image, N, Indicator
    Output: Output_image
    $For\ (x,y)\ in\ image:$
        $For\ step\ in\ range(-steps, steps):$
                $Neighbourhood.append(image[y + step][x + step]$
        $Neighbourhood = [8,9,10,11 \ldots \ldots 32,33,34,35,36]$
        $Image[y][x] = \max(Neighbourhood)\ OR\ \min(Neighbourhood)$

Numpy Slicing Method:
    Input: Input_Image, N, Indicator
    Output: Output_image
    $For\ (x,y)\ in\ image:$
        $For\ step\ in\ range(-steps, steps):$
            $Find\ X_{upper\ Bound}, X_{lower\ Bound}\ of\ NxN\ window$
            $Find\ Y_{upper\ Bound}, Y_{lower\ Bound}\ of\ NxN\ window$
        $Neighbourhood = image[Y_{lower\ Bound}:Y_{upper\ Bound} + 1][X_{lower\ Bound}:X_{upper\ Bound} + 1]$
        $Image[y][x] = numpy.amax(Neighbourhood)\ OR\ numpy.amin(Neighbourhood)$
```

- N value has to be an odd integer that is greater than or equal to 3, this is to centralise each pixel when sliding NxN window
  - For Even number – (x,y) pixel can't be centralised within NxN window, unless bias anchoring is performed
  - For N < 3 – the only odd number is 1, which resulted in 1x1 window, meaning no filtering.
- If the Window goes outside the image boundary, in other words Padding, *Brute Force method* force ignores and goes to the next valid pixel value. Whereas *Numpy Slicing method*, iterate through 'steps' list finding the right value for boundary, if above or below image boundary range, go to the next valid boundary value. This is similar to Zero/255 Padding approach, where for max filter padding value is set to 0 and min filter padding value is set to 255, to avoid distorting image and choosing wrong value.
- The outcome of the 2 methods is pixel-wise compared (using comparison matrix) over a range of N values, and no difference was found.
- As observed, *Brute Force method* complexity is extremely high comparing to *Numpy Slicing method*, where the difference in running time under **Cells.png** with M = 1, N = 27, is:
  - *Brute Force Method* (Store and compare): $\approx 610\ seconds$
  - *Numpy Slicing Method* (Slice out Sub Matrix then compare): $\approx 10\ seconds$
- Hence *Numpy Slicing method* is selected due to overall efficiency.

Particles.png Output:

Under **Particles.png**, *image_A* is a max-filtered image that aims to filter out the dark particles, to explain relationship between N value and dark particle disappearance see the following:

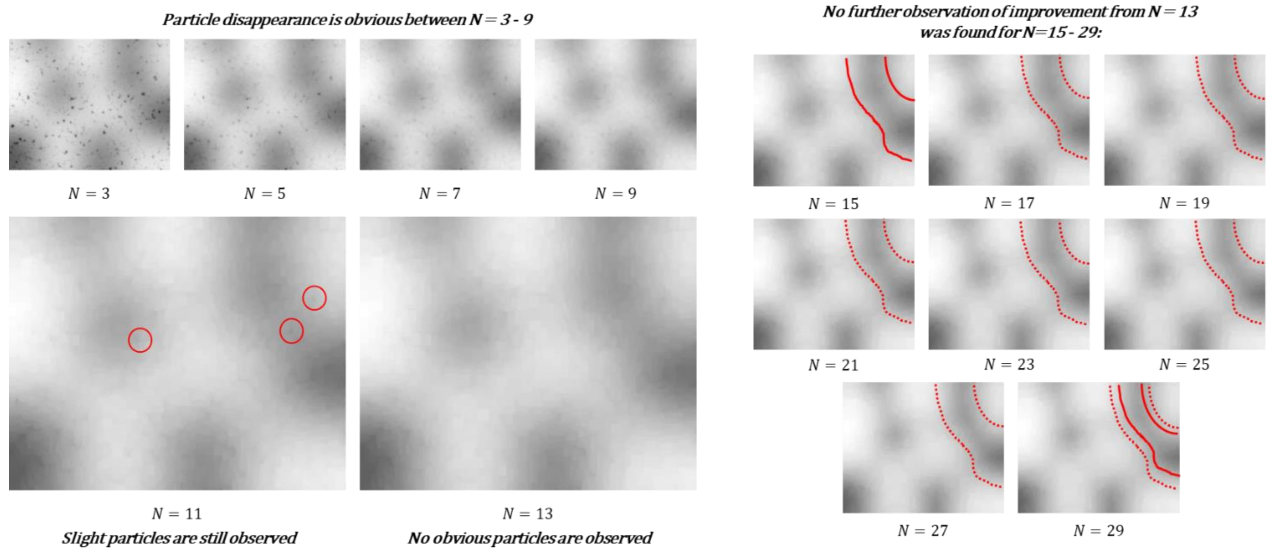
- Let's consider a simple matrix filtered using 3x3 window, along with the grey scale intensity illustration in the following:



	0	1	2	3	4	5
0	10	32	31	41	15	56
1	17	58	29	91	21	72
2	31	43	71	55	17	45
3	15	10	21	2	3	4
4	15	96	71	8	27	3
5	3	22	31	74	15	36

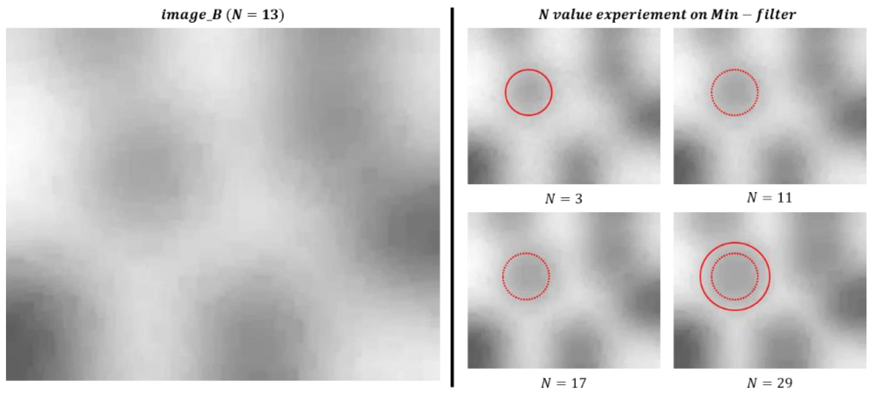
- As the N value increases, the window gets larger and larger, which in other words occupy more pixels.
- Max-Filtering picks the largest pixel value within the neighbourhood and replace pixel value at centre (x,y) with that maximum number, as observe from grey scale, higher pixel intensity refers to brighter/whiter color.
- In other words, when NxN is larger, there is higher possibilities in selecting "whiter cell" intensity from neighbourhood, which causes "darker cell" to disappear, see above.

Various N value is examined for **Particles.png**, as shown below different output for *image_A*, another observation was found between N=15 – 19, where the black shading become smaller due to max filtering mentioned above:



Particle disappearance is obvious between N = 3 - 9
N = 3
N = 5
N = 7
N = 9
N = 11
Slight particles are still observed
N = 13
No obvious particles are observed
No further observation of improvement from N = 13 was found for N=15 - 29:
N = 15
N = 17
N = 19
N = 21
N = 23
N = 25
N = 27
N = 29

Hence N = 13 is selected, therefore by directly implementing min-filter (counter effect - 'whitening' of background shading due to max-filter) on *image_A* with same size of N, *image_B* is obtained:



For experiment, different N value is examined and it is observed that as N value increases the 'black shading' circle begins to increase, shown on RHS of the image on the left. *Image_B* matrix is as below:

```
[[227 227 227 ... 225 225 225]
 [227 227 227 ... 225 225 225]
 [227 227 227 ... 225 225 225]
 ...
 [ 98  98  99 ... 238 238 238]
 [ 98  98  99 ... 238 238 238]
 [ 98  98  99 ... 238 238 238]]
```

Task 2 – Background Subtraction

In order to understand approach taken, first examine the matrix of *image_I*, *image_A* and *image_B* side by side:

image_I

```
[[217 222 220 ... 212 219 221]
 [217 220 217 ... 214 220 221]
 [216 217 217 ... 212 217 218]
 ...
 [ 90  88  90 ... 234 235 235]
 [ 91  90  91 ... 231 232 233]
 [ 93  91  87 ... 228 229 232]]
```

image_A

```
[[223 223 223 ... 221 221 221]
 [223 223 223 ... 222 221 221]
 [226 226 226 ... 222 221 221]
 ...
 [ 97  97  97 ... 238 238 236]
 [ 97  97  97 ... 238 238 236]
 [ 97  97  97 ... 238 238 236]]
```

image_B

```
[[227 227 227 ... 225 225 225]
 [227 227 227 ... 225 225 225]
 [227 227 227 ... 225 225 225]
 ...
 [ 98  98  99 ... 238 238 238]
 [ 98  98  99 ... 238 238 238]
 [ 98  98  99 ... 238 238 238]]
```

As mentioned earlier, *image_A* causes the particles to disappear, and *image_B* enhance the background shading in order to counter balance the effect done by max-filter. As a result, pixel values in background, *image_B*, is actually higher than *image_I*, this is due to the input picture having white/grey background, and dark particles – considering scale: 0 = black, 255 = white.

In order to do operations on the image matrix, first converting to a larger data type in avoiding overflow, this is done by simply changing to numpy.float32, using *.astype()* function.

As observed from matrix below, when subtracting *image_B* from *image_I*, a negative value is resulted, this matrix represents the 'filtered image', however it cannot be displayed using cv2 as pixel value is outside of grey scale range.

Now in order to fit within [0,255] scale, simply add the maximum value of the scale, which is 255, this puts the 'filtered image' matrix, back within the range of [0, 255].

The side-effect of this addition, results in 'whitening' of dark particles, in other words, increasing dark particle's pixel intensity (whiter). A contrast stretch is then required to balance it out.

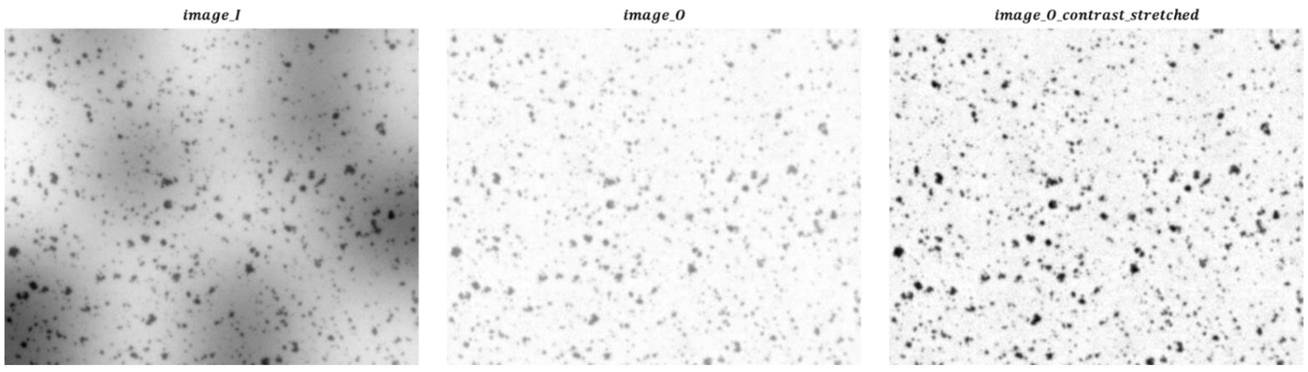
$$\boldsymbol{image\_I} - \boldsymbol{image\_B}$$

```
[[-10.  -5.  -7. ... -13.  -6.  -4.]
 [-10.  -7. -10. ... -11.  -5.  -4.]
 [-11. -10. -10. ... -13.  -8.  -7.]
 ...
 [ -8. -10.  -9. ...  -4.  -3.  -3.]
 [ -7.  -8.  -8. ...  -7.  -6.  -5.]
 [ -5.  -7. -12. ... -10.  -9.  -6.]]
```

$$\boldsymbol{image\_O} = (\boldsymbol{image\_I} - \boldsymbol{image\_B}) + \mathbf{255}$$

```
[[245. 250. 248. ... 242. 249. 251.]
 [245. 248. 245. ... 244. 250. 251.]
 [244. 245. 245. ... 242. 247. 248.]
 ...
 [247. 245. 246. ... 251. 252. 252.]
 [248. 247. 247. ... 248. 249. 250.]
 [250. 248. 243. ... 245. 246. 249.]]
```

The output of Task 2, max-min filtered **Particles.png,** with N =13 and M = 0, is as the following:



The overall process is shown below:

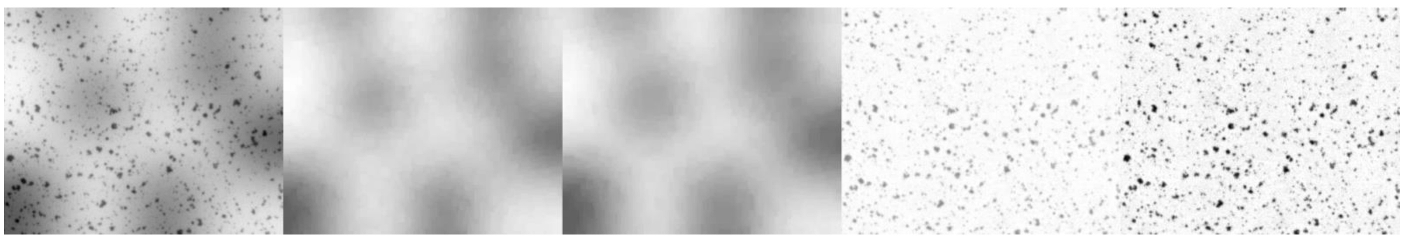


Figure 1: (1) - Input Image, (2) - Image_A, (3) - Image_B, (4) - Image_O, (5) - Contrast_streteched_O

Task 3 – Algorithm generalisation

Methodology:

Generalized Algorithm is structure accordingly to the following Pseudo code:

```
Generalized_algorithm (input_image, N, M)
        Input: Input_Image, N, M
        Output: None
        If M == 0:
                image_A = numpy_slicing_method(input_image, N, 'max')
                image_B = numpy_slicing_method(image_A, N, 'min')
                image_O = Subtracting image_B from input_image + 255
        Elif M == 1:
                image_A = numpy_slicing_method(input_image, N, 'min')
                image_B = numpy_slicing_method(image_A, N, 'max')
                image_O = Subtracting image_B from input_image
        Display images
        return
```

```
Numpy Slicing Method:
        Input: Input_Image, N, Indicator
        Output: Output_image
```

M = 1 Example (Cells):

image_I
```
[[ 18  17  17 ...  82  79  74]
 [ 16  16  16 ...  81  78  74]
 [ 17  17  17 ...  83  79  76]
 ...
```

image_B
```
[[15 15 15 ... 68 68 68]
 [15 15 15 ... 68 68 68]
 [15 15 15 ... 68 68 68]
 ...
```

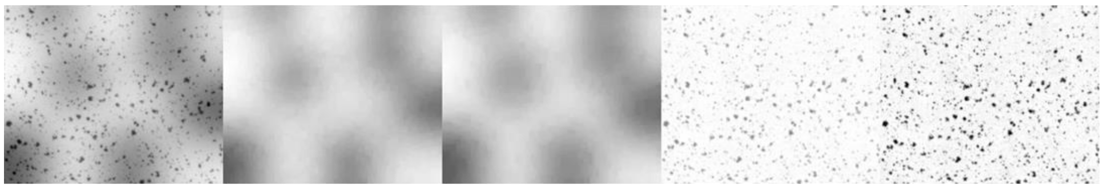
image_A
```
[[15 15 15 ... 50 61 68]
 [15 15 15 ... 47 57 68]
 [15 15 15 ... 44 51 61]
 ...
```

image_O
```
[[  3.   2.   2. ...  14.  11.   6.]
 [  1.   1.   1. ...  13.  10.   6.]
 [  2.   2.   2. ...  15.  11.   8.]
 ...
```

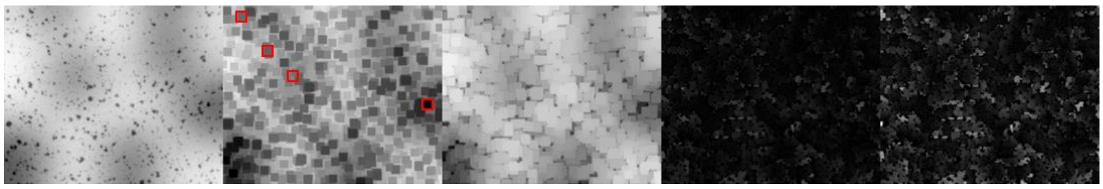
In simpler term, the function defined earlier in Task 1 is called twice in generalized_algorithm, where the sequence of Max-filtering and Min-filtering depends on user input, M, 0 refers to max-filtering then min-filtering and 1 refers to min-filtering then max-filtering.

In order to explain why M =0 or M = 1 for different images, consider **Particles.png** with different M value compared side by side:

***M* = 0**



***M* = 1**



Keep in mind, the purpose of this filter function is to *remove background from an image*. From this point:

- If background of the image is white, shown in **Particles.png**, and the subject of observation or 'objects', in this case the particles, are dark/black. This means the background has *higher* pixel intensity, under grey-scale, comparing to the target objects. Therefore, in order to 'split' the background from objects, *max-filtering* is first completed to remove objects from image, selecting 'high-pixel background intensity'. Note, as shown from ***Task 2 – Background Subtraction***, since background image has higher pixel values, a resultant negative matrix ('filtered image') requires an addition of 255.
- On the other hand, if background of the image is dark, shown in **Cells.png**, and the subject of observation or 'objects', in this case the cells, are bright/white. This means the background has *lower* pixel intensity, under grey-scale, comparing to the target objects. Therefore, in order to 'split' the background from objects, *min-filtering* is first completed to remove objects from image, selecting 'low-pixel background intensity'. Note, as shown from above matrix of M = 1 example, *image_B* actually has higher pixel value then *image_I*, hence there is no need of adding 255 in normalizing.

Now going back to output of different M values using **Particles.png**, as observed, the shading is properly removed via M = 0, where the output image clearly shows particles with a white background.

On the other hand, attempting in using M = 1, where min-filtering is completed first before max-filtering, results in 'drawing of black-squares', due to NxN window. In other words, 'enlarging' the particles, and when perform max filtering 're-shapes' the image by 'drawing white-squares', eventually the output doesn't mean anything, as it distorted the image.

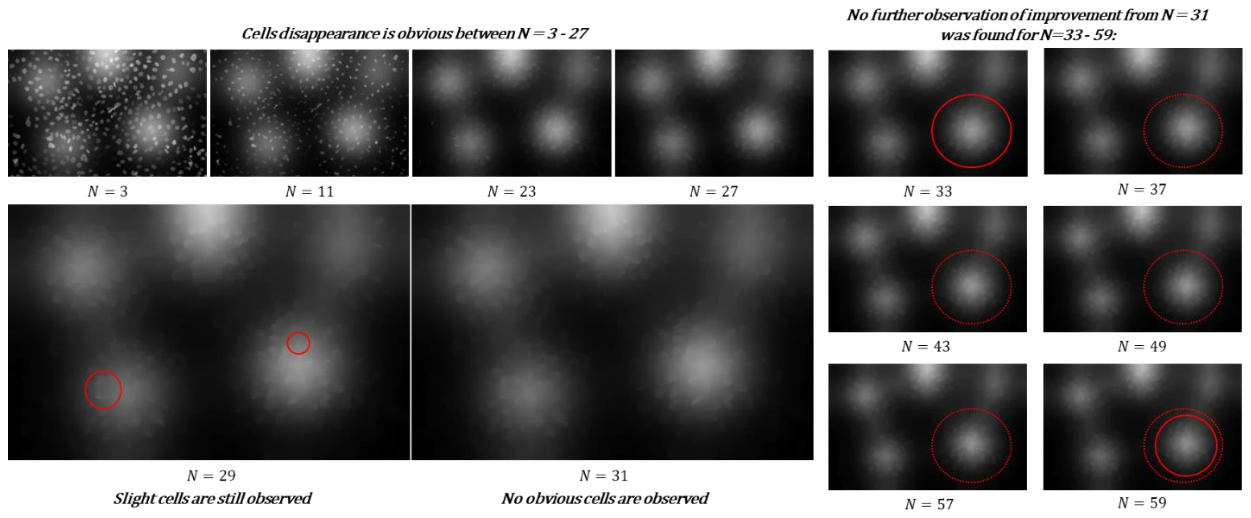
To conclude,

- M = 0 : Max then Min, when image background is brighter/white and object is darker/black.
- M = 1 : Min then Max, when image background is darker/black and object is brighter/white.

Cells.png Output:

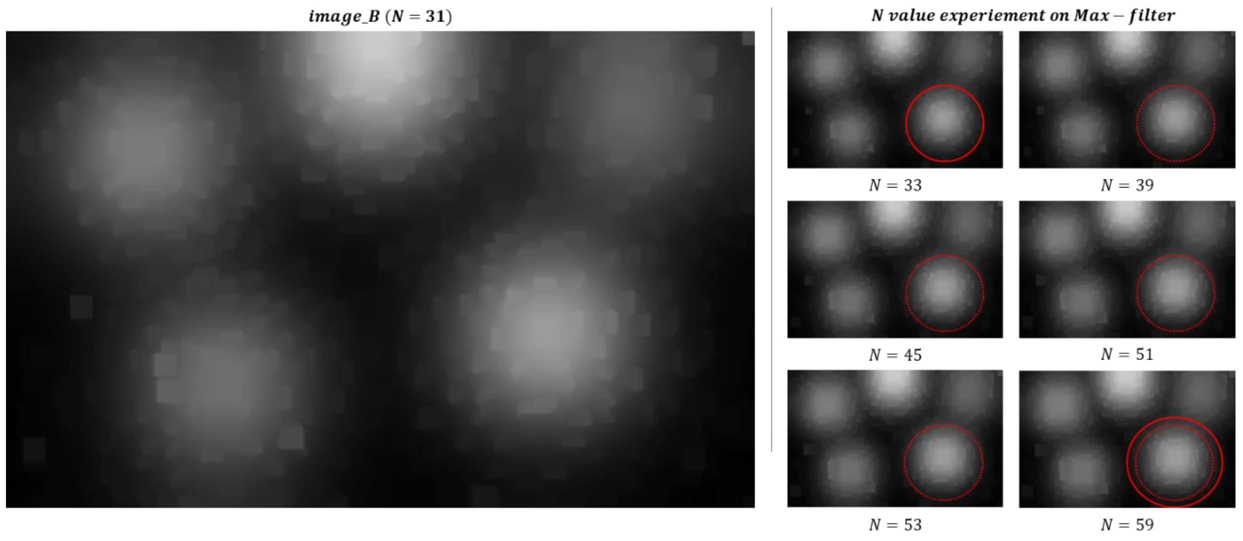
From previous explanation, as observed from **Cells.png**, the background of the image is darker/black and the subject of observation, in this case, cells are brighter/white, hence M=1 is used in performing Min-filtering then Max-filtering.

In order to find the best value for N, similar experiment is done and shown below different min-filtered *Image_A*, on the other hand, it is observed that increasing N value shrinks background white shading circle:

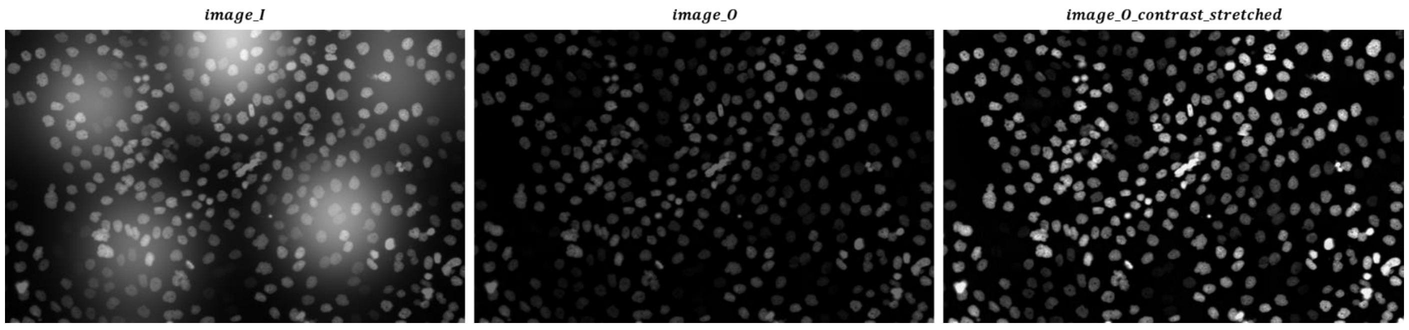


Cells disappearance is obvious between N = 3 - 27
No further observation of improvement from N = 31 was found for N=33 - 59:
N = 3
N = 11
N = 23
N = 27
N = 33
N = 37
N = 43
N = 49
N = 29
Slight cells are still observed
N = 31
No obvious cells are observed
N = 57
N = 59

Hence N = 31 is selected, therefore by directly implementing max-filter on *image_A* with same size of N, *image_B* is obtained, experiment is conducted in examining effect of N value:



The output of **Cells.png,** with N =31 and M = 1, is as the following:



The overall process is shown below (matrix of *Image_I, A, B and O* are shown in previous section):

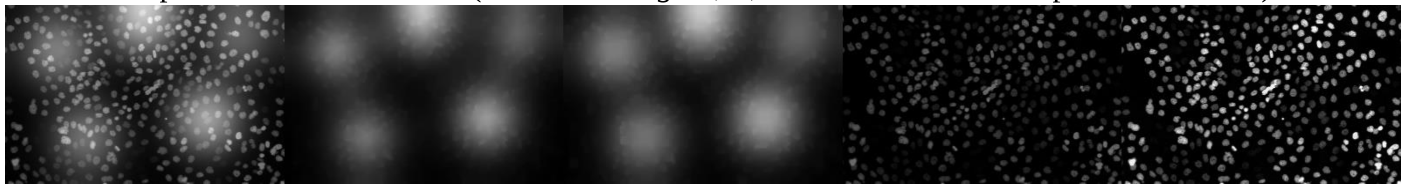


Figure 2: (1) - Input Image, (2) - Image_A, (3) - Image_B, (4) - Image_O, (5) - Contrast_streteched_O