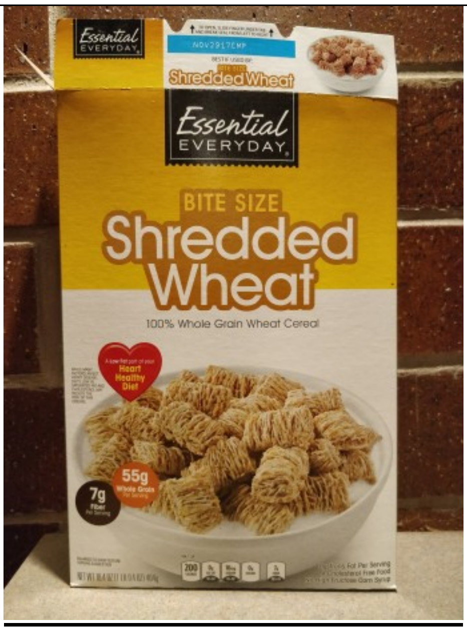
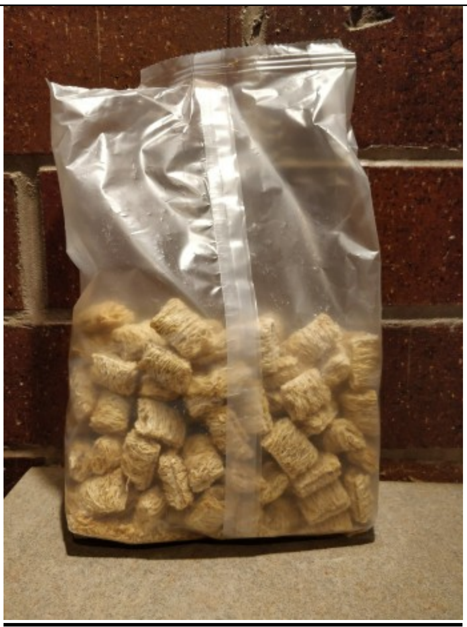
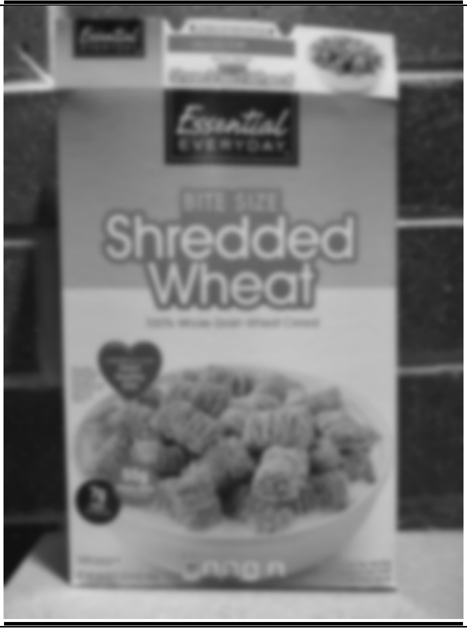
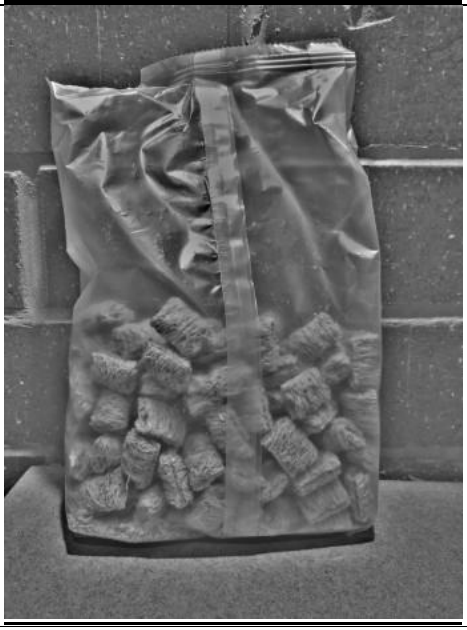
Name : Xiao Ma

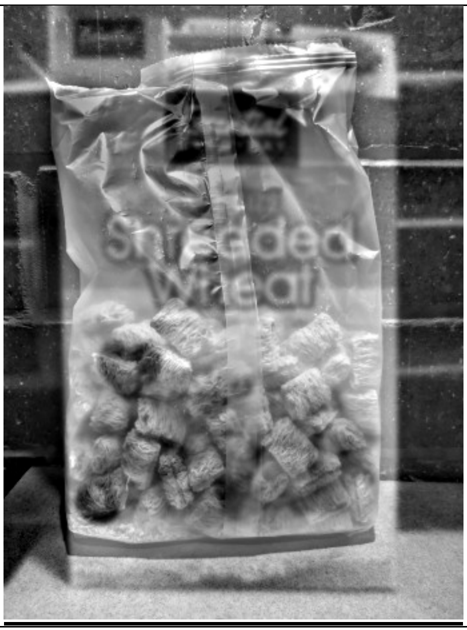
<u>Part-1 : Hybrid Images</u>

- **The original, filtered, and hybrid images for all 3 examples (i.e., 1 provided and 2 pairs of your own).**

1. **Cereal**

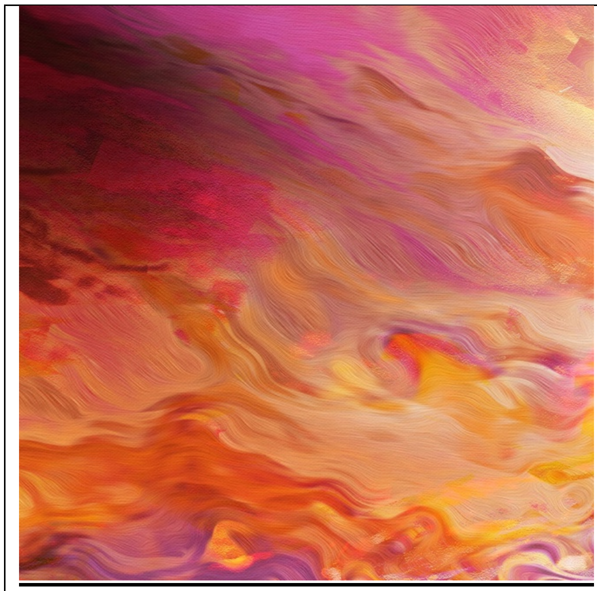
High Pass Filter $\sigma = 10$
Low Pass Filter σ =2

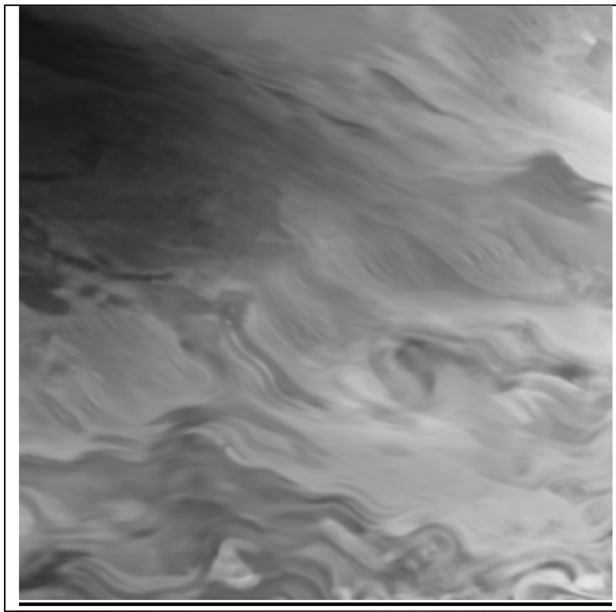
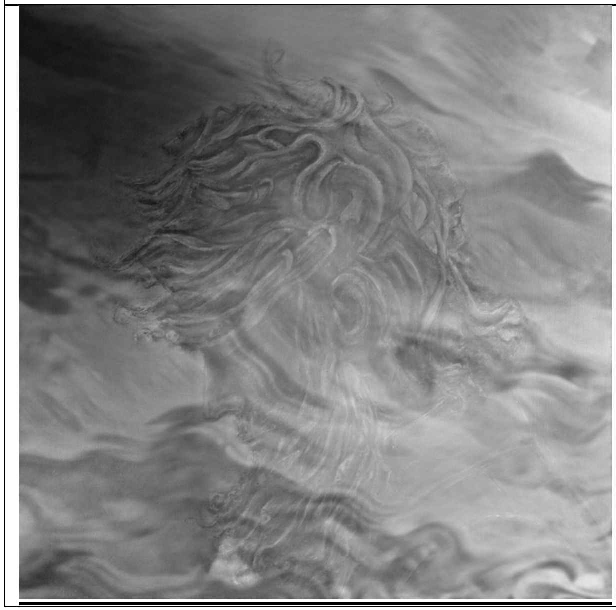
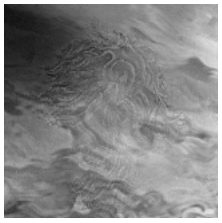
	
	

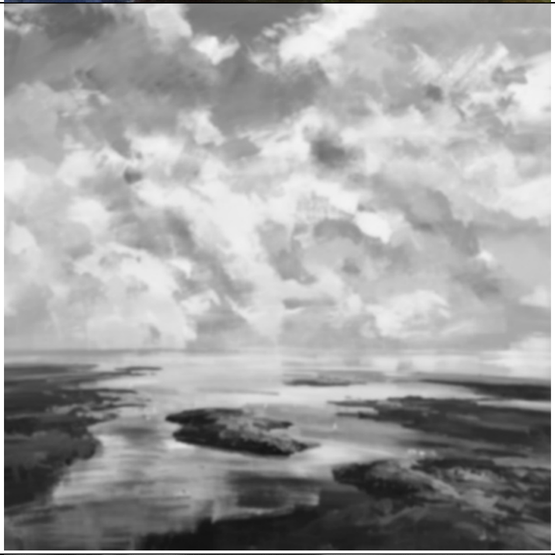
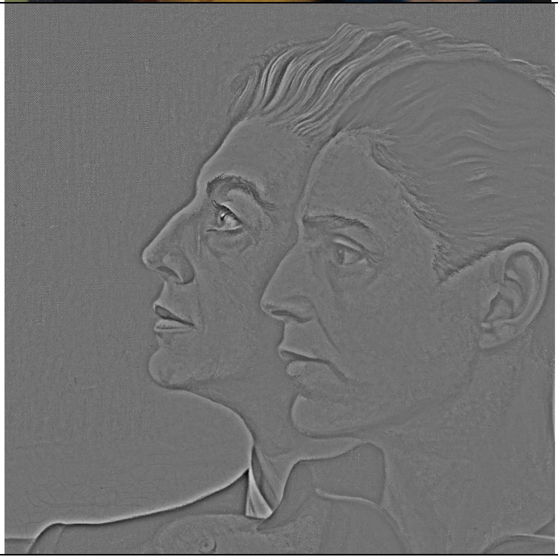
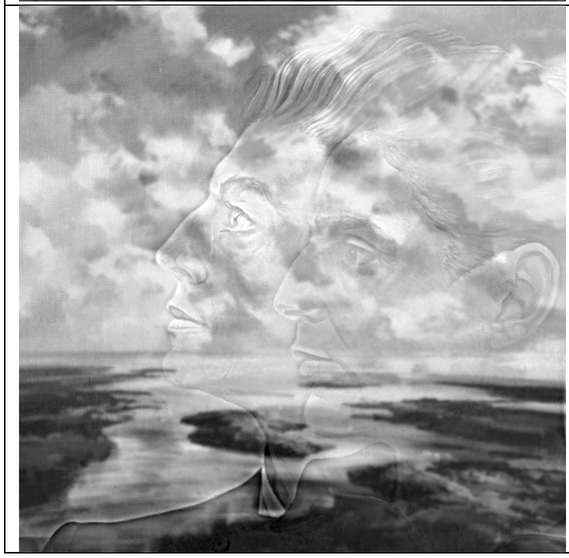
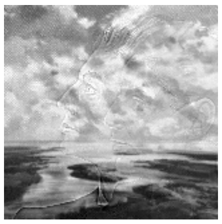
2. **Wind and Man**

High Pass Filter $\sigma = 10$
Low Pass Filter σ =2

3. **Sky and People**
   High Pass Filter $\sigma = 5$
   Low Pass Filter $\sigma = 2$

- **Explanation of any implementation choices such as library functions or changes to speed up computation.**

  I have used `scipy.ndimage.gaussian_filter`. from scipy library, and for the high pass filter since it is a combination of the identity filter – gaussian filter, therefore, I use the original image minus the gaussian filtered image, since the operation is linear. The guassian_filter is faster than building the gaussian function and then use the convolve function. To make sure that the quality of the hybrid images, makes sure after the filter operation, now of the pixel goes to negative.

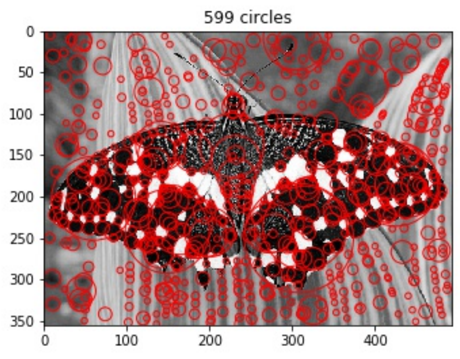
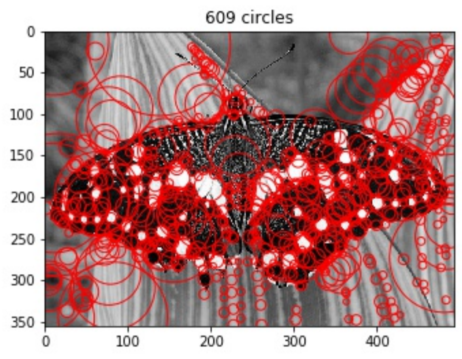
- An explanation of parameter values you have tried and which ones you found to be optimal.

1. Cereal:
   The resulting hybrid images are well constructed, for the high resolution figure we can see the cereal box, and with the low resolution case we can clear see a bag of cereal. After iterating through the sigma for the high pass filter and low pass filter, I come to the conclusion that the high frequency filter size has to be larger than the low frequency filter size.
2. Wind and Man:
   I choose the cloud picture as the smooth input, and the man painting as the textured image input. The generated hybrid image is really well constructed. As one can see, the man's painting is seamlessly embedded into the smooth background ,and if you zoom out, you can see the texture of the man's portrait clearly. The generation takes some iteration for the high and low pass filter size.
3. Sky and People
   For this pictures I choose two pictures that have large contrast, I would like to test if the hybrid images would be more obvious. And the generated hybrid image is really successfully. And it gives an artisan look. I have the decrease the high-frequency filter size here to 5, and this sets the smooth input to be less obsecure, and make the whole hybrid picture smooth.

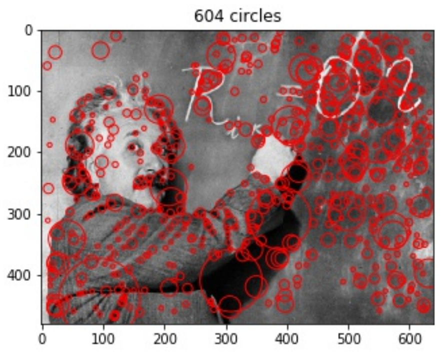
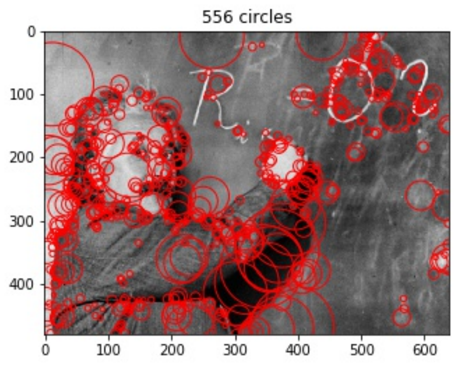
Part-2 : Scale-space blob detection

- The output of your circle detector on all the images (four provided and four of your own choice), together with running times for both the "efficient" and the "inefficient" implementation.

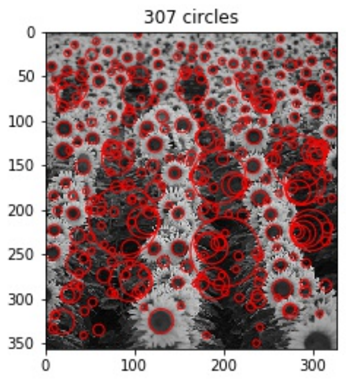
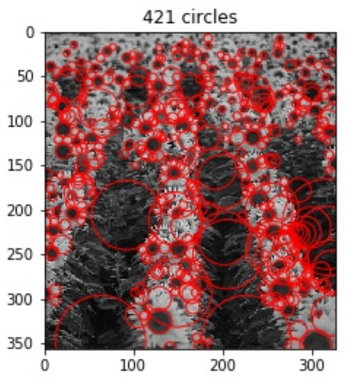
1. **Butterfly:**

Increasing Filter Size (Inefficient)	**Downsample Image (Efficient)**
	
initial_sigma = 2 **num_iter = 15** **factor = 1.3** **thred = 0.0015**	**initial_sigma = 2** **num_iter = 15** **factor = 1.3** **thred = 0.00075**
`Time elapse =  1.2571468353271484`	`Time elapse =  0.4231309890747070`

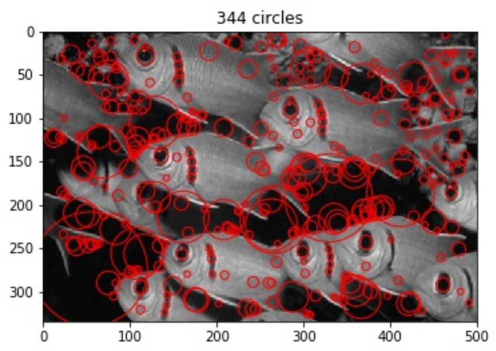
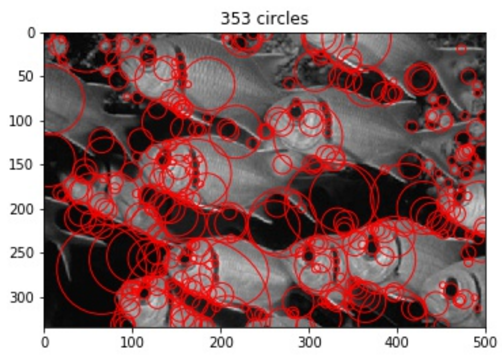
2. **Einstein**

Increasing Filter Size (Inefficient)	**Downsample Image (Efficient)**
	
initial_sigma = 2 **num_iter = 15** **factor = 1.3** **thred = 0.025**	**initial_sigma = 2** **num_iter = 15** **factor = 1.3** **thred = 0.0004**
`Time elapse =  2.1532888412475586`	`Time elapse =  0.8174808025360107`

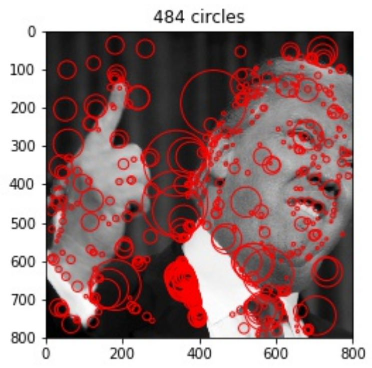
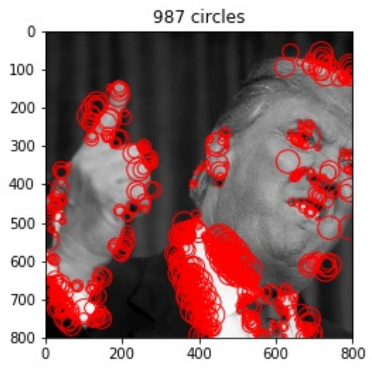
3. Sunflowers

Increasing Filter Size (Inefficient)	Downsample Image (Efficient)
	
initial_sigma = 2 **num_iter = 15** **factor = 1.3** **thred = 0.025**	**initial_sigma = 2** **num_iter = 15** **factor = 1.3** **thred = 0.0008**
`Time elapse =  0.8343658447265625`	`Time elapse =  0.2984631061553955`

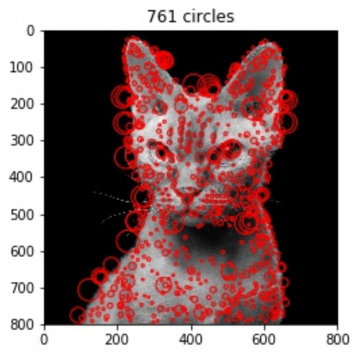
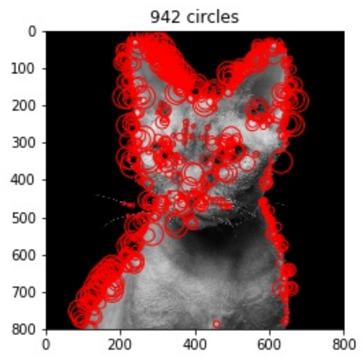
4. Fishes

Increasing Filter Size (Inefficient)	Downsample Image (Efficient)
	
initial_sigma = 2 **num_iter = 15** **factor = 1.3** **thred = 0.025**	**initial_sigma = 2** **num_iter = 15** **factor = 1.3** **thred = 0.0005**
`Time elapse =  1.2124390602111816`	`Time elapse =  0.41802000999450684`

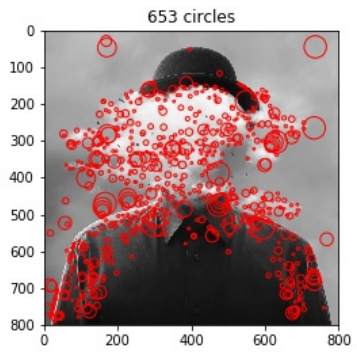
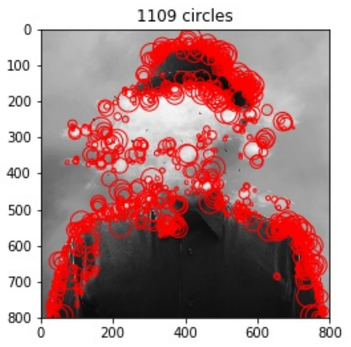
5. **Trump**

Increasing Filter Size (Inefficient)	**Downsample Image (Efficient)**
	
initial_sigma = 2 **num_iter = 15** **factor = 1.3** **thred = 0.025**	**initial_sigma = 2** **num_iter = 15** **factor = 1.2** **thred = 0.0005**
`Time elapse =  4.31630802154541`	`Time elapse =  1.2506718635559082`

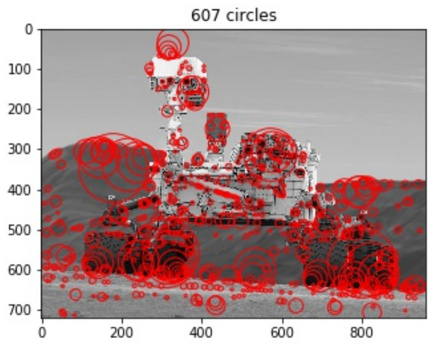
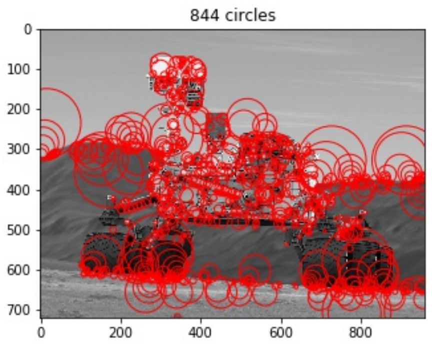
6. **Cat**

Increasing Filter Size (Inefficient)	**Downsample Image (Efficient)**
	
initial_sigma = 2 **num_iter = 15** **factor = 1.2** **thred = 0.01**	**initial_sigma = 2** **num_iter = 15** **factor = 1.2** **thred = 0.0005**
`Time elapse =  2.3197691440582275`	`Time elapse =  1.3327178955078125`

7. **Man**

Increasing Filter Size (Inefficient)	**Downsample Image (Efficient)**
	
initial_sigma = 2 **num_iter = 15** **factor = 1.2** **thred = 0.01**	**initial_sigma = 2** **num_iter = 15** **factor = 1.2** **thred = 0.0001**
`Time elapse =  2.118086814880371`	`Time elapse =  1.1998817920684814`

8. **Mars Rover:**

Increasing Filter Size (Inefficient)	**Downsample Image (Efficient)**
	
initial_sigma = 2 **num_iter = 15** **factor = 1.3** **thred = 0.025**	**initial_sigma = 2** **num_iter = 15** **factor = 1.3** **thred = 0.0005**
`Time elapse =  4.744199752807617`	`Time elapse =  1.5782489776611328`

- An explanation of any "interesting" implementation choices that you made.

  There is this function “peak_local_max()” from the scikit-image library and the functionality is designed for finding the local peak maximum value and it is sutaible for the implementation of non_maximum_suppresion function. Basically I cloud use the peak_local_max function to find each local in each stack frame, and then I could set up a threshold for non maximum suppression, the function is more efficient than using rank_filter() from scipy library. And np.where helps my implementation to find the points that exceed the threshold. And in the opencv libarary, there is already some functionality exisit in doing non_maximum suppression and it will be more efficient to use that functionality insteady of implementing it step by step.

- An explanation of parameter values you have tried and which ones you found to be optimal.

  The downsample method is more efficient from the computational time list in the table than the increasing filter size approach. But generally speaking, that you need to use lower threshold for the downsampling approach than the increasing filter size approach. The tables in each figures are showing the optimal parameters that I come to after doing parametric studies. And from the results the downsampling approach is giving slightly better result in detecting the edges and feature objects.