Computer Vision and Image Processing: Homework 2

Chaitanya Chembolu – 50248987 – satyasiv@buffalo.edu

Implementation details:

The process of blob detection can be roughly divided into the following steps.

1) *Pre-processing*: Selecting the image and converting it into grayscale and double.
2) *Generate and apply filters to build a scale space*.
    - Applying the Laplacian of Gaussian filter can be done by two different ways.
        - Down sampling the image by a factor 1/k.
        - Up scaling the filter by factor k.

    While these two methods return similar results, the prior method is more time efficient as seen later in this report.

    - Difference of Gaussian, as described in David Lowe's paper, was also tried and results were recorded. The following extract was the basis for my implementation.

      "An efficient approach to construction of D (x, y, σ): The initial image is incrementally convolved with Gaussians to produce images separated by a constant factor k in scale space, shown stacked in the left column. We choose to divide each octave of scale space (i.e., doubling of σ) into an integer number, s, of intervals, so k = 2^(1/s). We must produce s +3 images in the stack of blurred images for each octave, so that final extrema detection covers a complete octave. Adjacent image scales are subtracted to produce the difference-of-Gaussian images shown on the right. Once a complete octave has been processed, we resample the Gaussian image that has twice the initial value of σ (it will be 2 images from the top of the stack) by taking every second pixel in each row and column. The accuracy of sampling relative to σ is no different than for the start of the previous octave, while computation is greatly reduced."
      Reference: *David G Lowe. Distinctive image features from scale-invariant key points. International Journal of Computer Vision, 60(2):91-110, 2004.*

3) *Perform non-maximum suppression in a scale space.*
   Different functions like nlfilter, colfilt, ordfilt2 were tried out and compared.
4) *Perform non-maximum suppression between scales.*
5) Using the set threshold, we *find the coordinates of the pixels of max values* and set the blob radius using the relation between sigma and radius. (r = sigma * squareroot(2))
6) *Display the blobs* on the image using the given show_all_circles function.

blob_detector function takes 6 different inputs. They are (in the following order)

- The filename of the image (placed in '../data' directory).
- Initial scale sigma for filtering.
- n: number of scale levels.
- k: the multiplication factor with which the sigma grows for every scale.
- threshold: the threshold over which the detections are considered.
- Filtering type {'logdown', 'logup', 'dog'}: This input switches between different methods of implementation. We have three different methods supported.
    - *'logdown'*: Laplacian of Gaussian by down sampling the image.
    - *'logup'*: Laplacian of Gaussian by upscaling the filter.
    - *'dog'*: Difference of Gaussian, method by Lowe.

NOTE: *'test_blob_detector.m' has a sample implementation of running blob_detector function. This allows us to run all the functions with ease.*

Example: `blob_detector('cityscape.jpg', 7, 5, 1.3, 0.007, 'logdown');`

The inputs are in the order above. We follow this structure in this report.

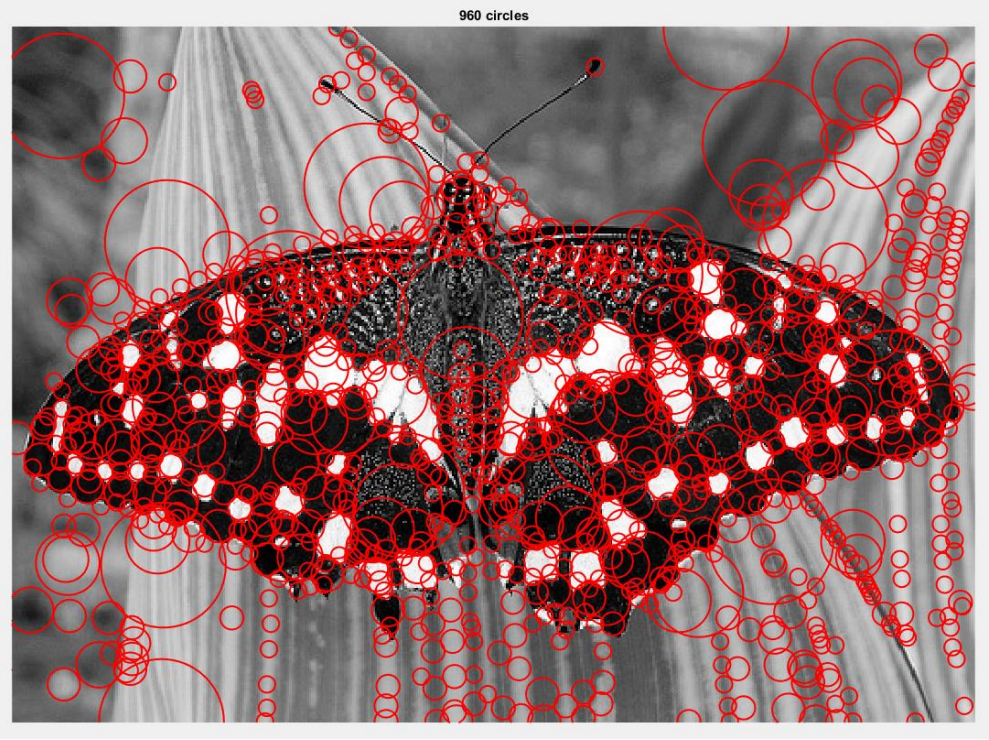
Results:

The implementation is run on the given four images and four other images, with different set of parameters and filtering types. The running times of different methods are compared.

The outputs and the respective parameters are as follows, also some observations specific to images are written below the respective images:

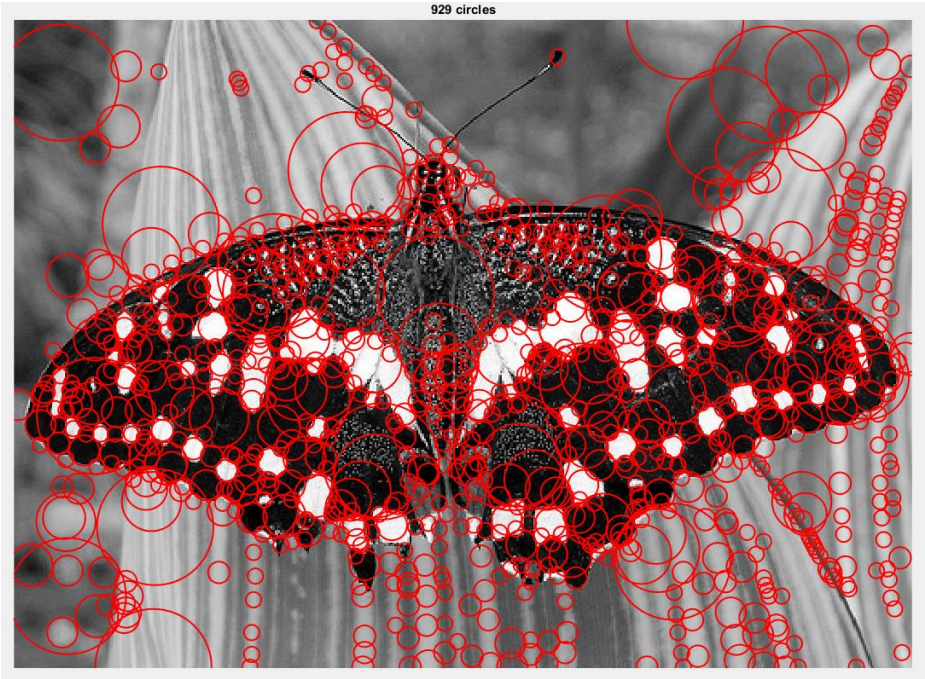
1) `blob_detector('butterfly.jpg', 3, 7, 1.4, 0.01, 'logdown');`

filename	butterfly.jpg
Initial sigma	3
n	7
k	1.4
Threshold	0.01
filter type	'logdown'



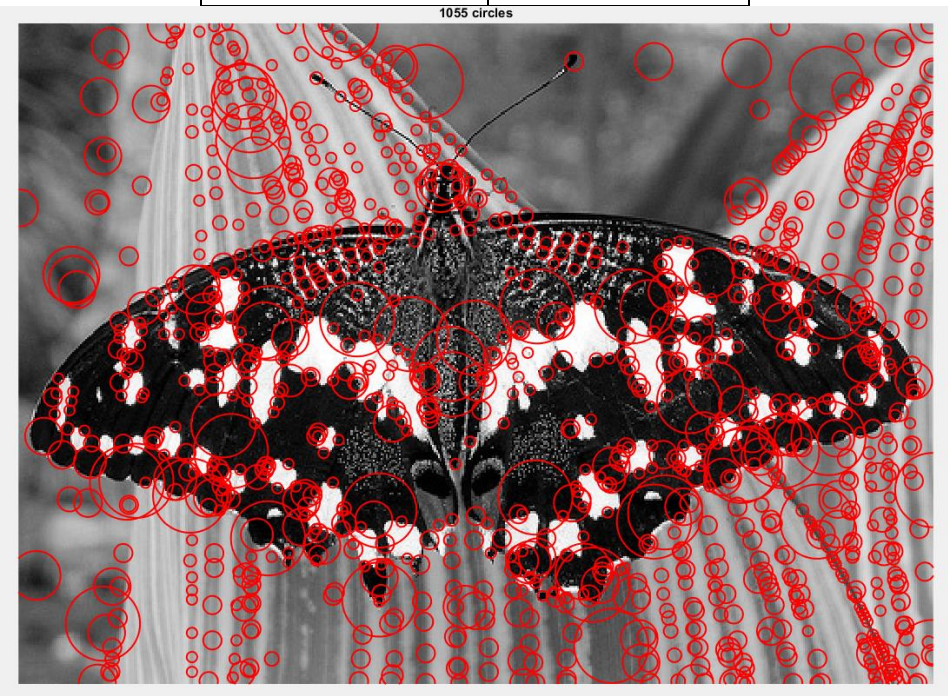
2) `blob_detector('butterfly.jpg', 3, 7, 1.4, 0.01, 'logup');`

filename	butterfly.jpg
Initial sigma	3
n	7
k	1.4
Threshold	0.01
filter type	'logup'



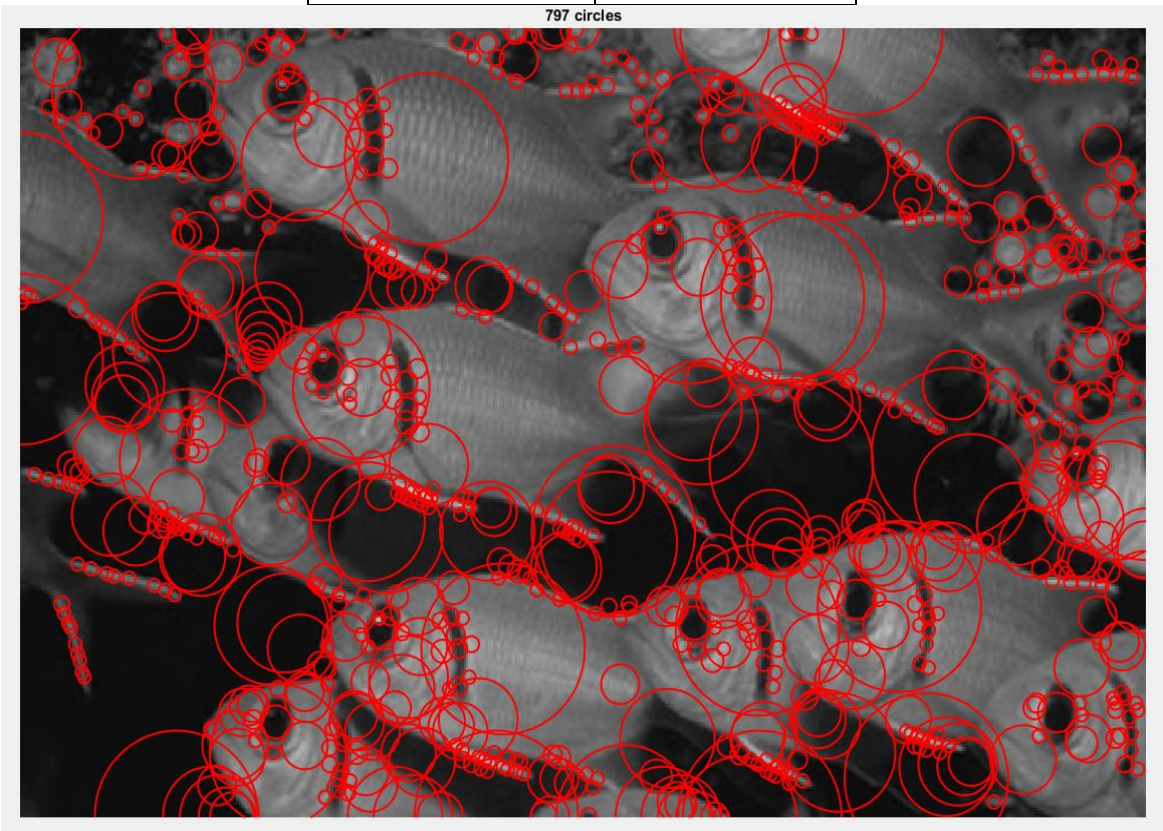
3) `blob_detector('butterfly.jpg', 2, 16, 1.15, 0.009, 'dog');`

filename	butterfly.jpg
Initial sigma	2
n	16
k	1.15
Threshold	0.009
filter type	'dog'



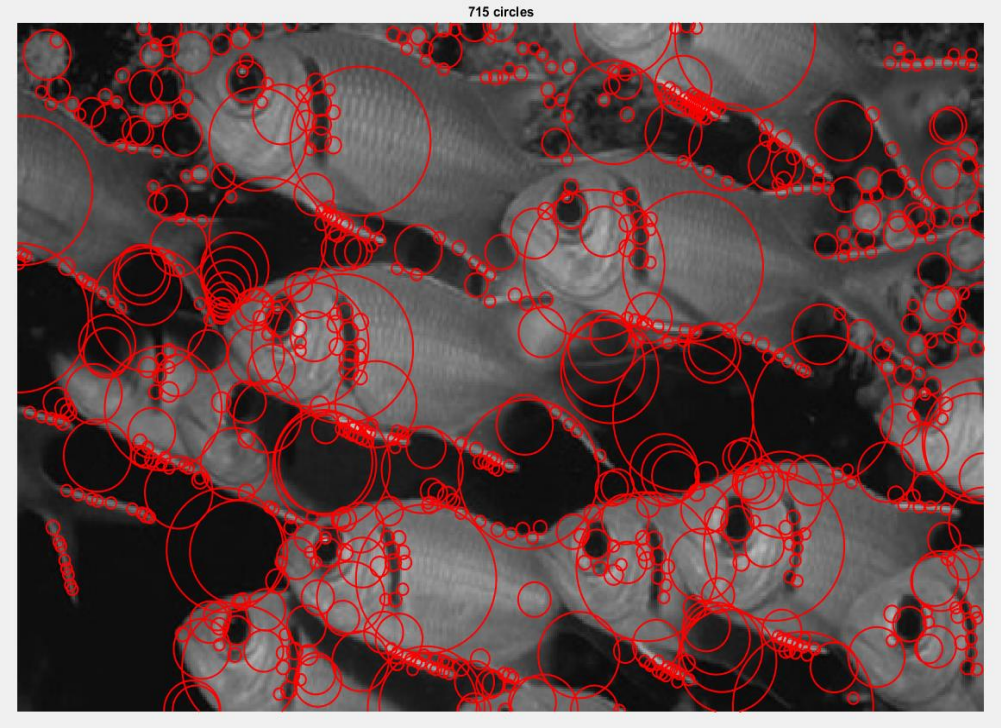
4) `blob_detector('fishes.jpg', 2, 15, 1.2, 0.007, 'logdown');`

filename	fishes.jpg
Initial sigma	2
n	15
k	1.2
Threshold	0.007
filter type	'logdown'



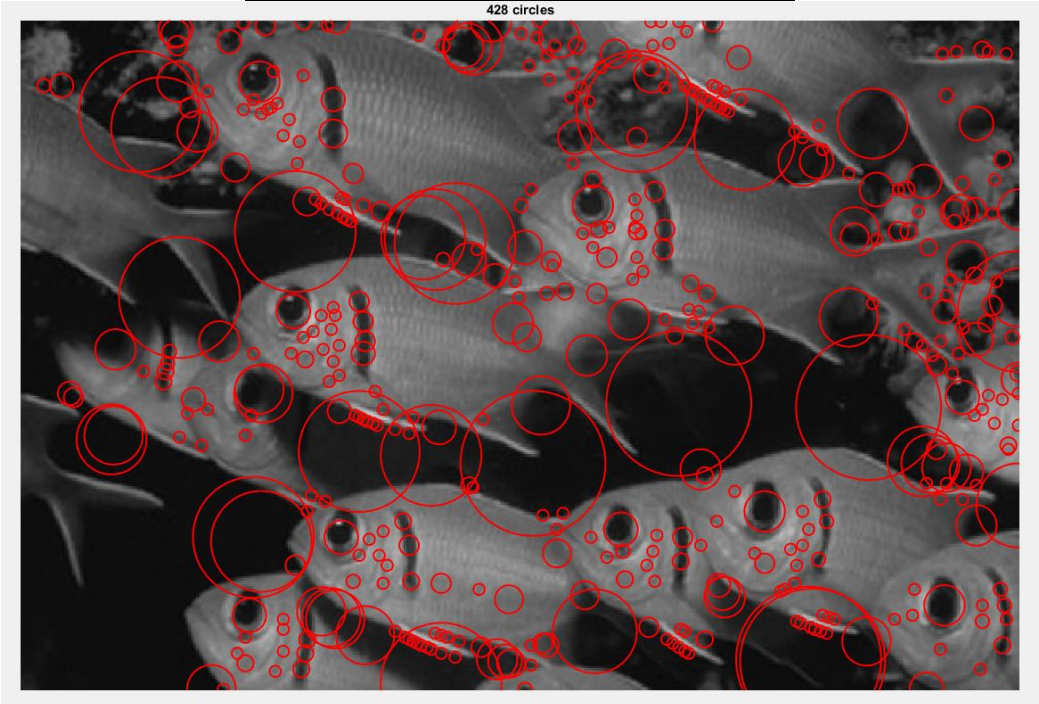
5) `blob_detector('fishes.jpg', 2, 15, 1.2, 0.007, 'logup');`

filename	fishes.jpg
Initial sigma	2
n	15
k	1.2
Threshold	0.007
filter type	'logup'



6) `blob_detector('fishes.jpg', 2, 15, 1.2, 0.005, 'dog');`

filename	fishes.jpg
Initial sigma	2
n	15
k	1.2
Threshold	0.005
filter type	'dog'



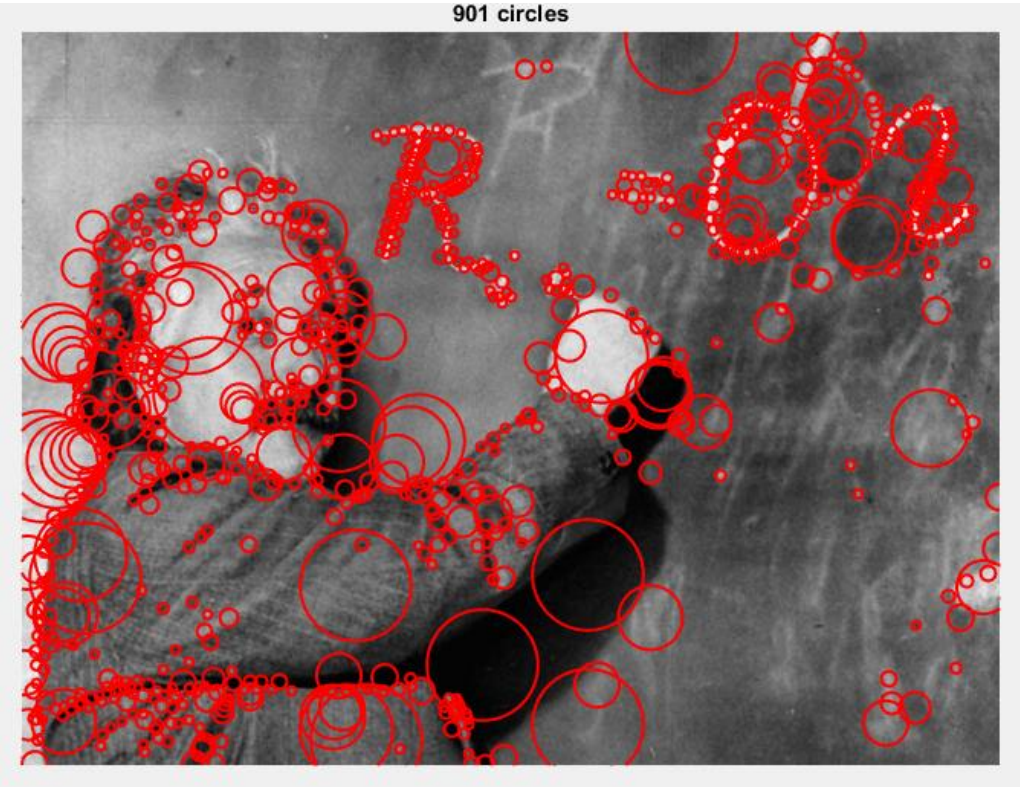
7) `blob_detector('einstein.jpg', 2, 15, 1.2, 0.007, 'logdown');`

filename	einstein.jpg
Initial sigma	2

n	15
k	1.2
Threshold	0.007
filter type	'logdown'

8) `blob_detector('einstein.jpg', 2, 15, 1.2, 0.007, 'logup');`

filename	einstein.jpg
Initial sigma	2
n	15
k	1.2
Threshold	0.007
filter type	'logup'

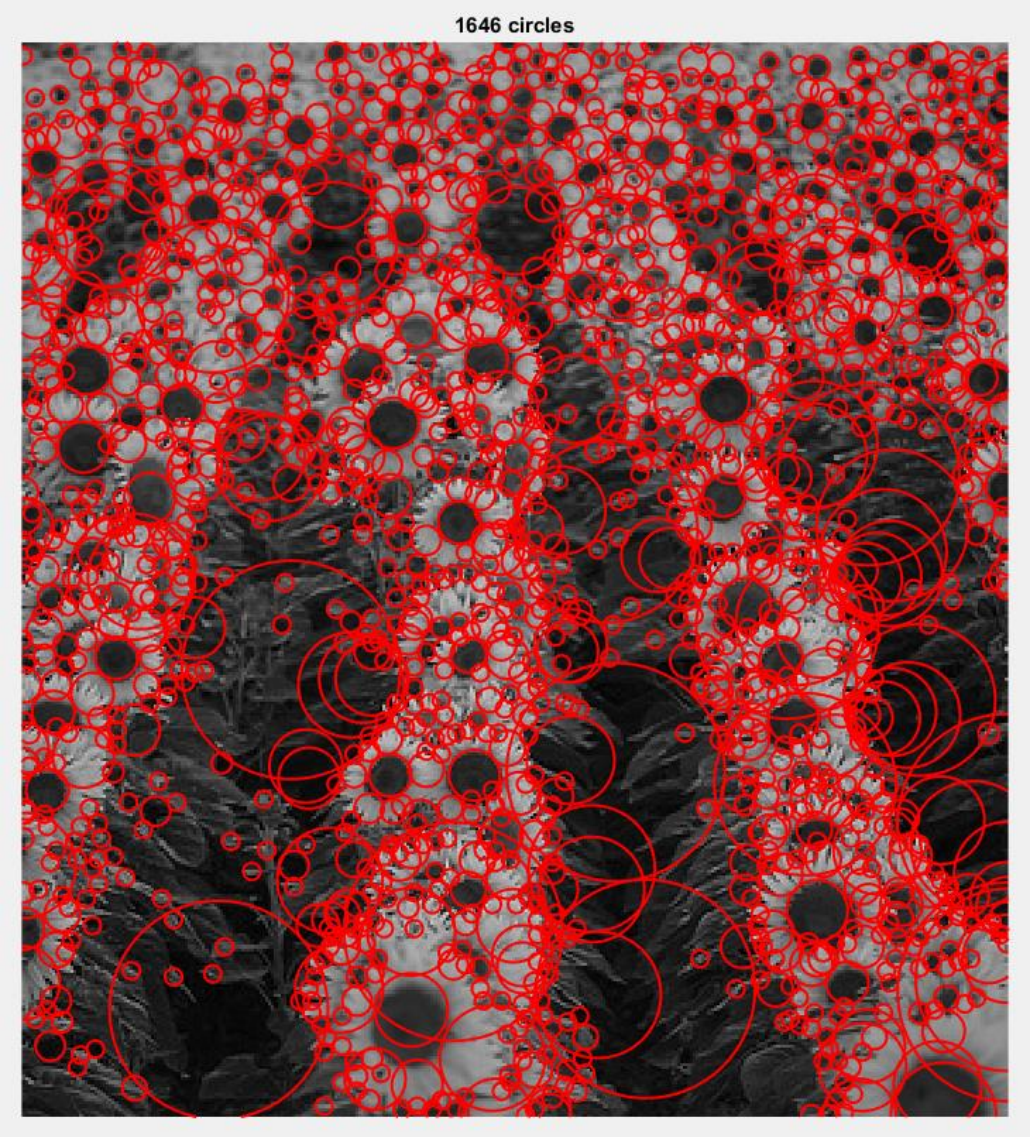


9) `blob_detector('einstein.jpg', 2, 15, 1.2, 0.007, 'dog');`

filename	einstein.jpg
Initial sigma	2
n	15
k	1.2
Threshold	0.007
filter type	'dog'

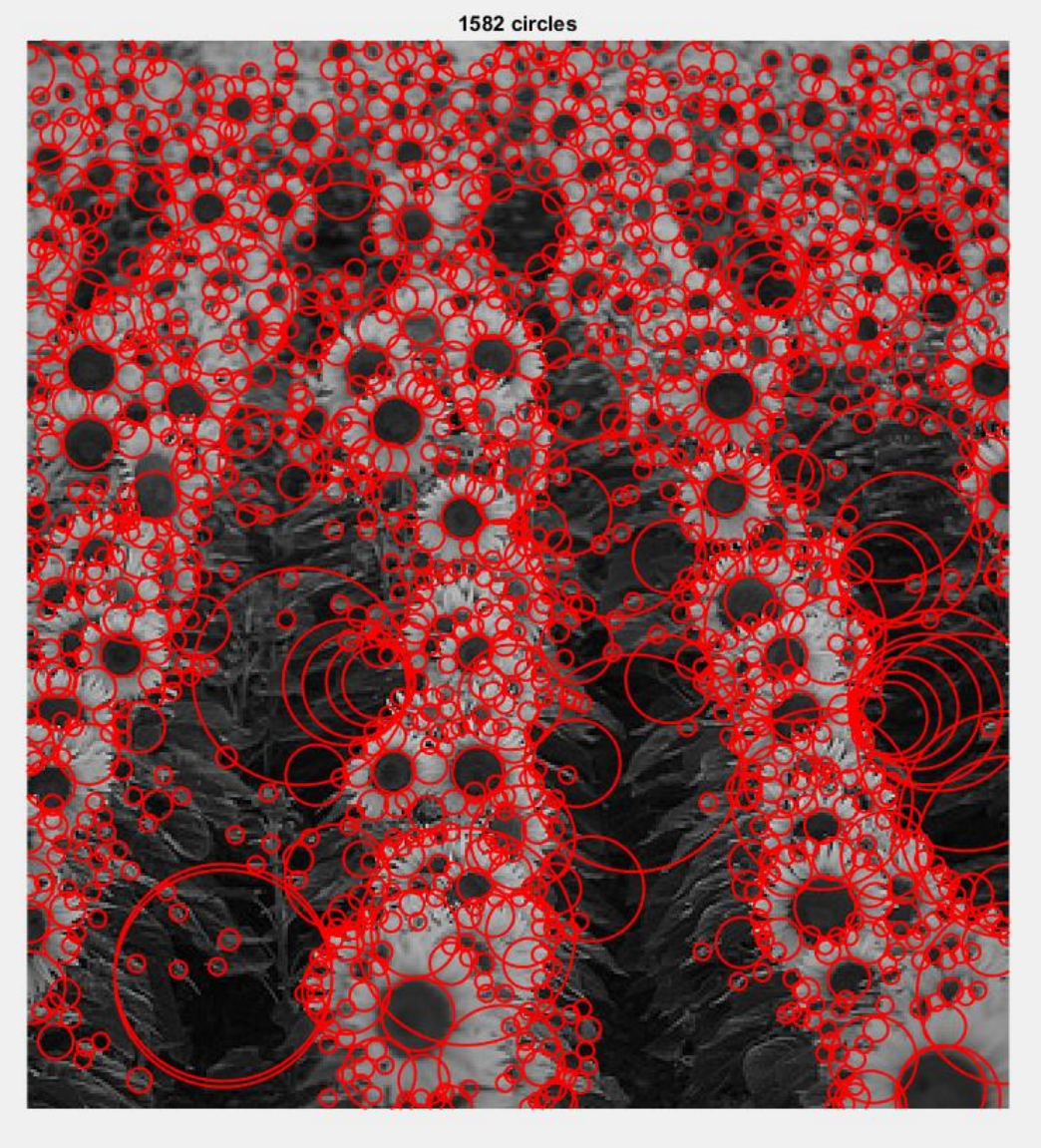
10) `blob_detector('sunflowers.jpg', 2, 15, 1.2, 0.007, 'logdown');`

filename	sunflowers.jpg
Initial sigma	2
n	15
k	1.2
Threshold	0.007
filter type	'logdown'



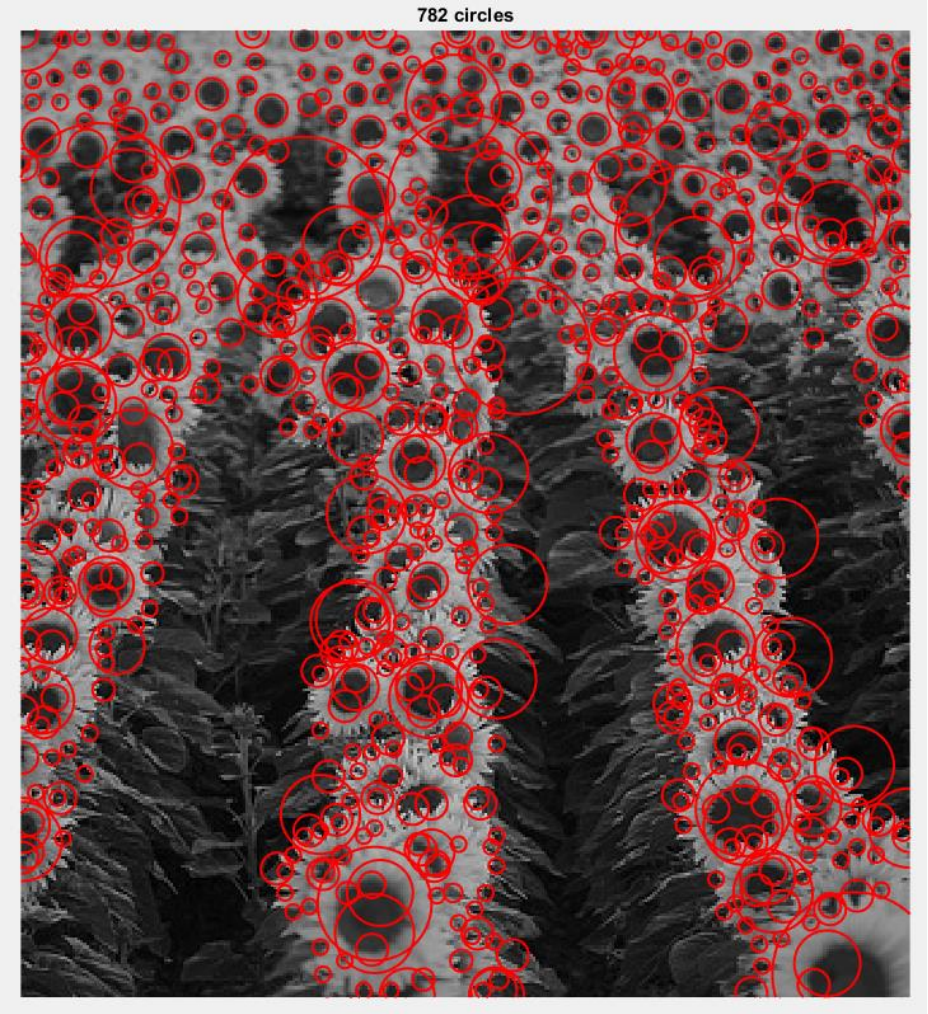
11) `blob_detector('sunflowers.jpg', 2, 15, 1.2, 0.007, 'logup');`

filename	sunflowers.jpg
Initial sigma	2
n	15
k	1.2
Threshold	0.007
filter type	'logup'

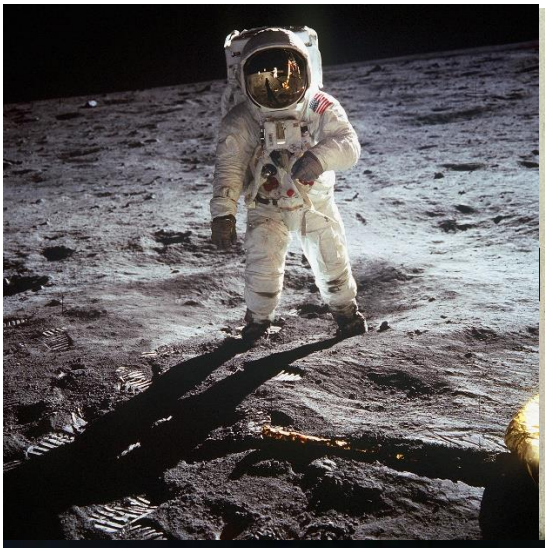


12) `blob_detector('sunflowers.jpg', 2, 15, 1.2, 0.007, 'dog');`

filename	`sunflowers.jpg`
Initial sigma	`2`
n	`15`
k	`1.2`
Threshold	`0.007`
filter type	`'dog'`

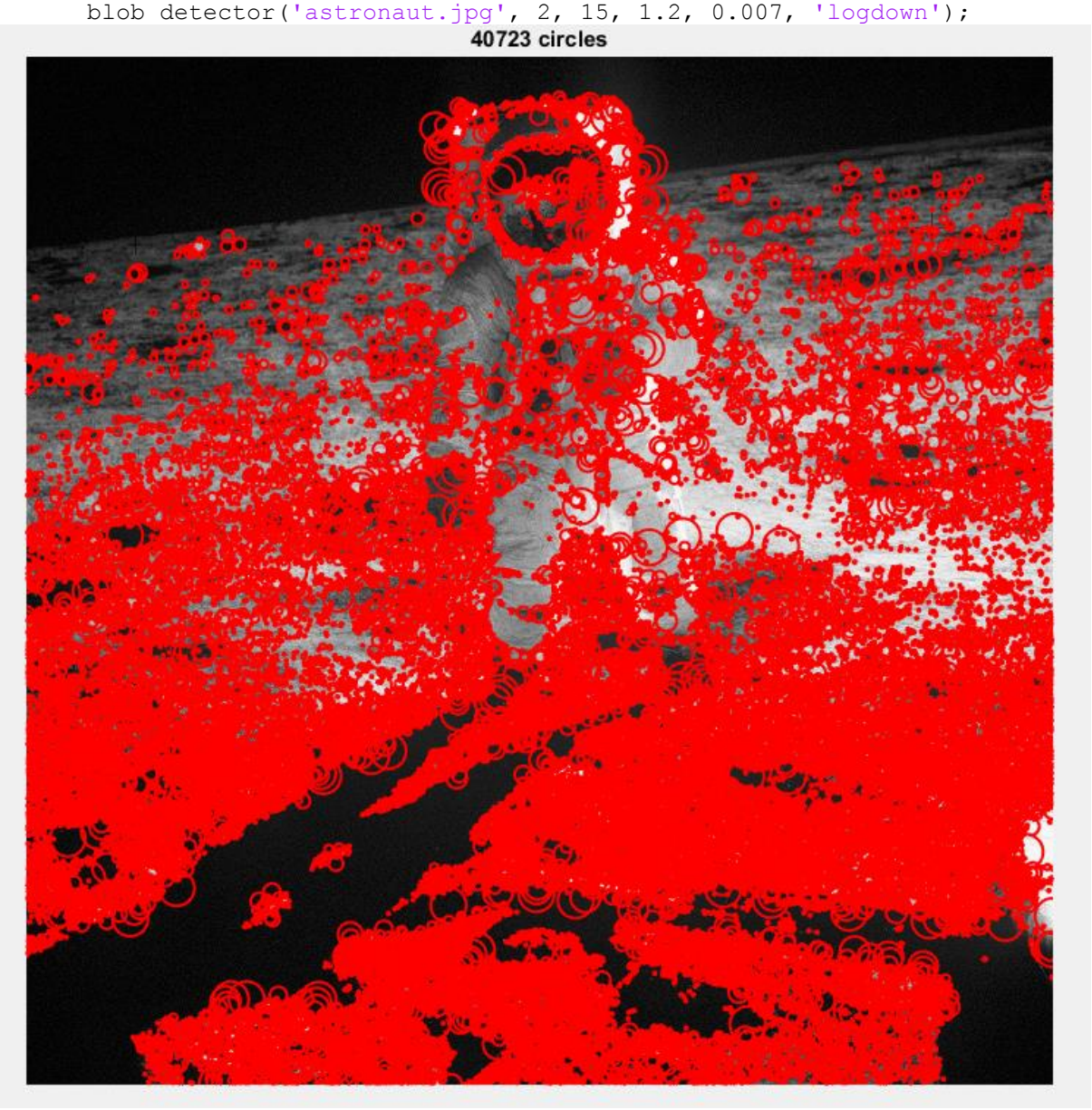


- Similarly, blob detection was implemented on the following additional images (astronaut.jpg, shining.jpg, lm.jpg, cityscape.jpg respectively). Selection of these images was based on the variation in resolutions, the scope of background/front objects, and also the number of objects in focus (like buildings in cityscape.jpg, people in lm.jpg and dust particles in astronaut.jpg).



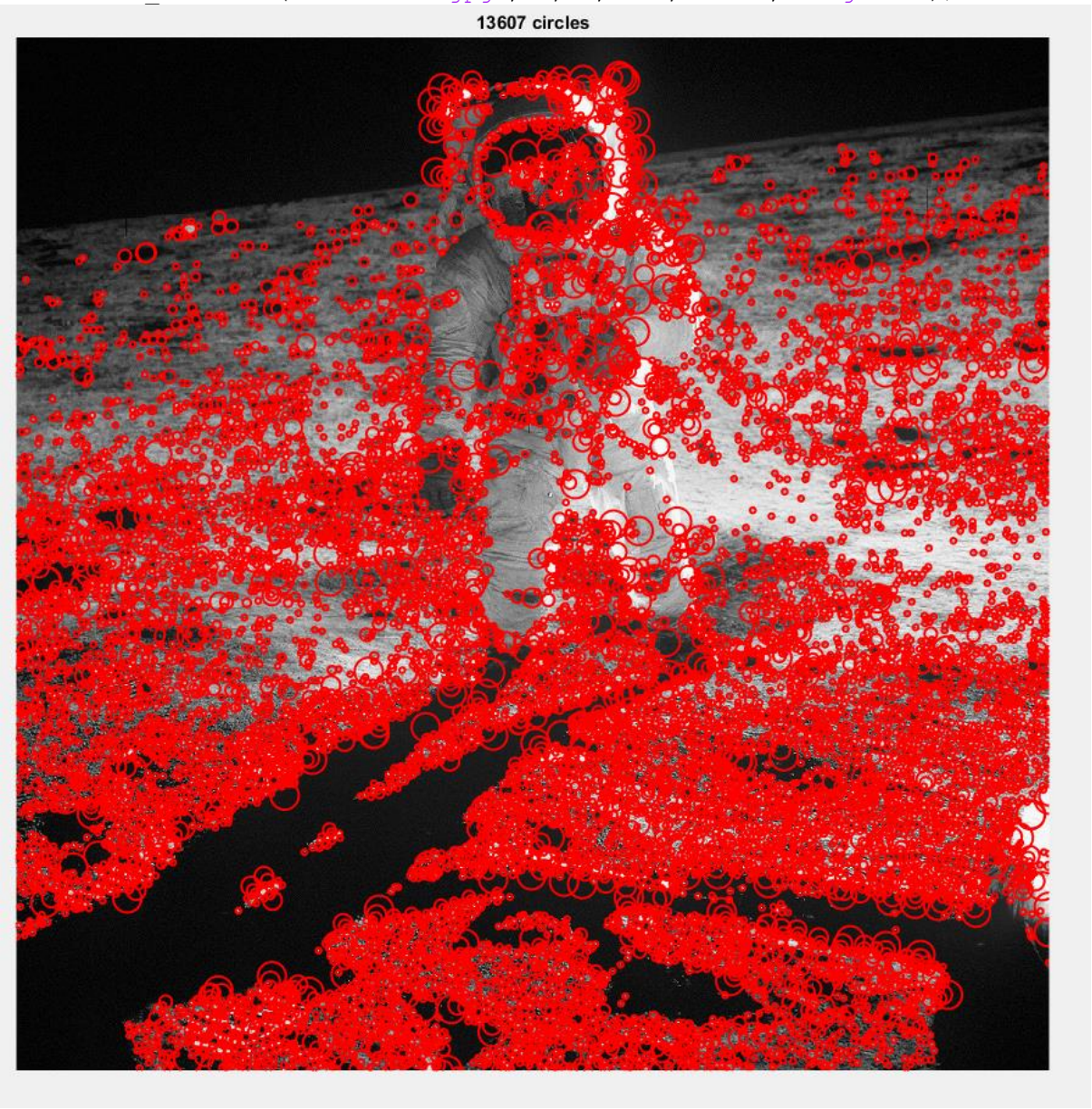
Results and observations are as follows:

```
13)       blob_detector('astronaut.jpg', 2, 15, 1.2, 0.007, 'logdown');
```



- As shown above, the density of blobs over the minute particles is expectedly very high; In order to reduce the density of blobs, the value of n is reduced and k is increased to differentiate blobs of different scales. Sigma is also increased to improve the size of smallest blob.

```
14)       blob_detector('astronaut.jpg', 4, 5, 1.5, 0.007, 'logdown');
```



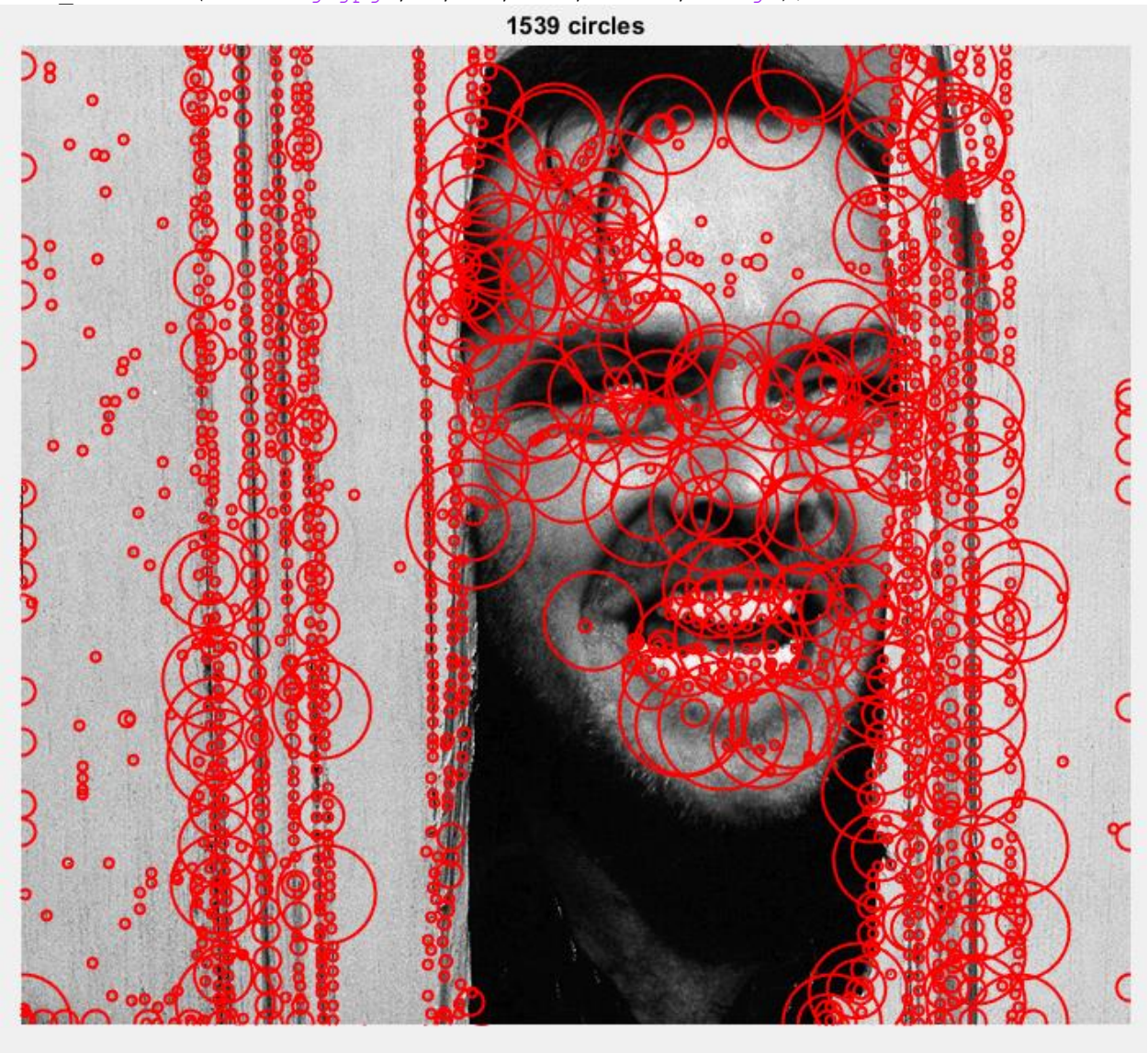
15) `blob_detector('shining.jpg', 2, 12, 1.3, 0.007, 'logdown');`

- The blobs here are finely placed considering the face and the lines on the wall.

16) `blob_detector('shining.jpg', 2, 12, 1.3, 0.007, 'logup');`

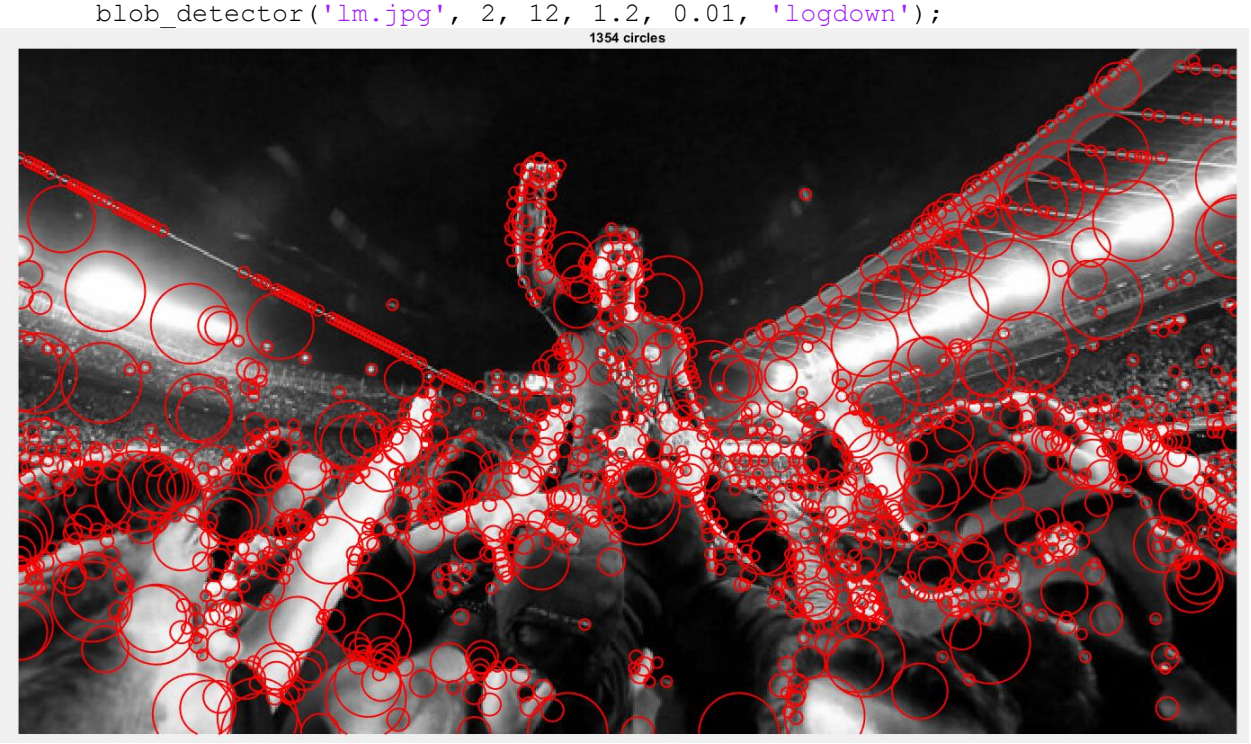
- We can see that the filter up scaled versions give results very similar to that of the image downscaled versions.

17) `blob_detector('shining.jpg', 2, 12, 1.3, 0.007, 'dog');`

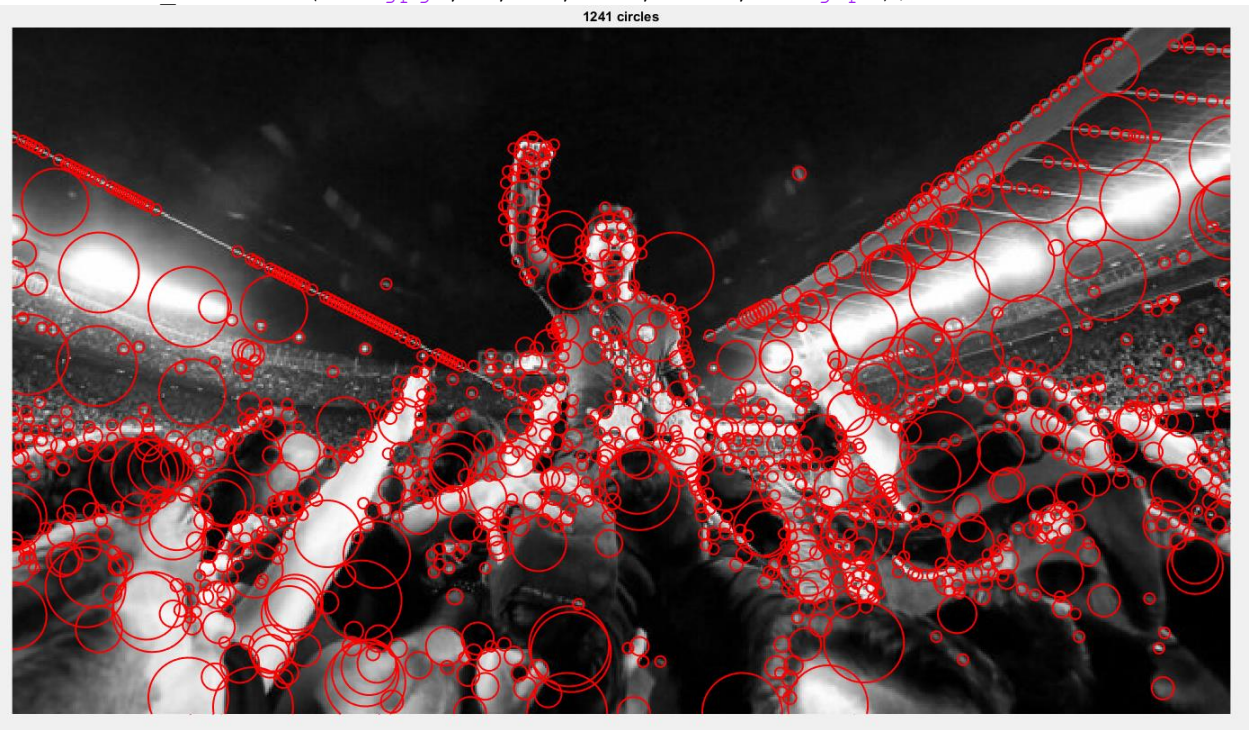


- While the output of the difference of Gaussian version is not so similar to the other two implementations, It still recognizes the features decently and with a change in threshold, gives similar results to that of LoG.

```
18)       blob_detector('lm.jpg', 2, 12, 1.2, 0.01, 'logdown');
```



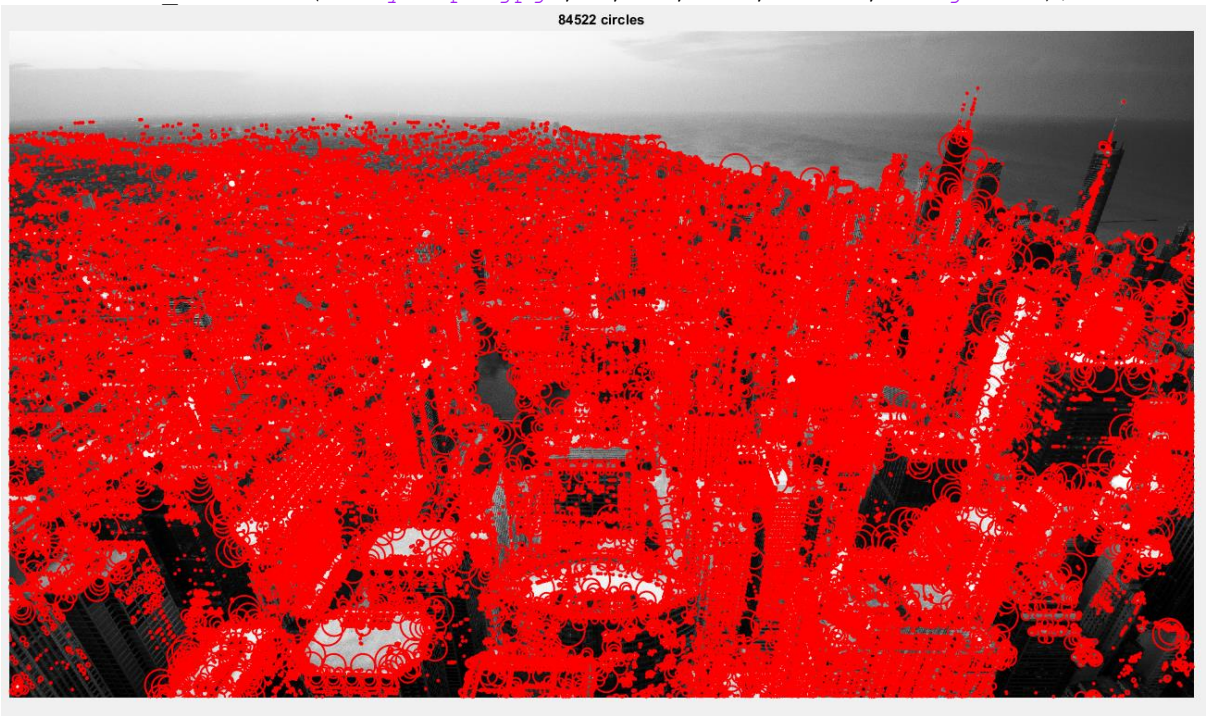

```
19)       blob_detector('lm.jpg', 2, 12, 1.2, 0.01, 'logup');
```



20) `blob_detector('lm.jpg', 2, 12, 1.2, 0.006, 'dog');`

- Now we consider a very high resolutioned image, cityscape.jpg (3840*2160). By intuition, we should get a result with very high density of blobs spread across the buildings etc.

21) `blob_detector('cityscape.jpg', 2, 12, 1.3, 0.007, 'logdown');`



- As expected, the blobs are too densely placed with the parameters we used for other images. Also, the sigma value seems too low as most of the blobs appear as points. Increasing sigma, decreasing n, we get the following

22) `blob_detector('cityscape.jpg', 4, 5, 1.3, 0.007, 'logdown');`

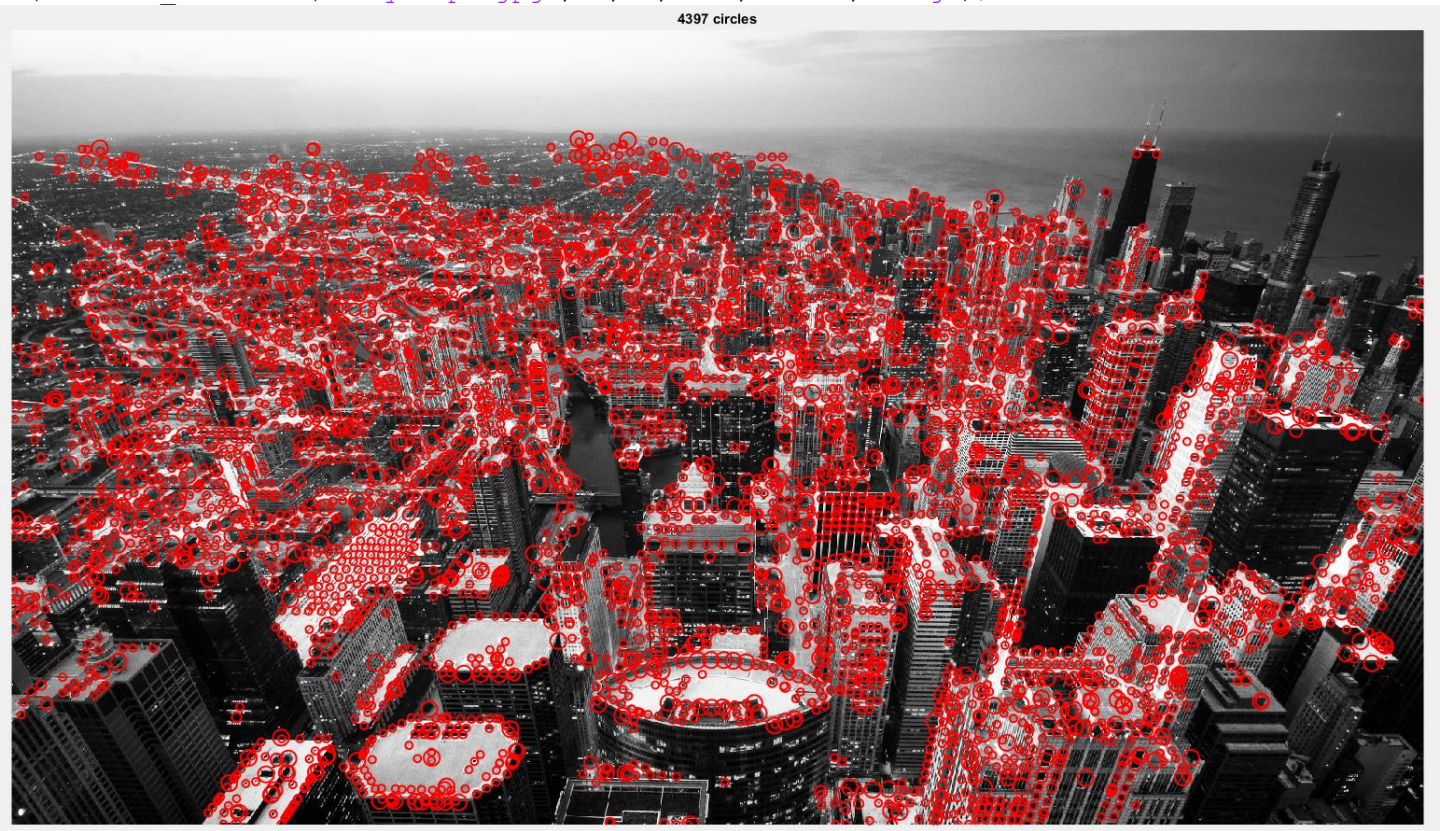
- The blobs still are too tiny for the image, so increasing the sigma further,

23) `blob_detector('cityscape.jpg', 7, 5, 1.3, 0.007, 'logdown');`

- The buildings and the roads are well detected.

- Finally, we consider the DoG for this image.

```
24)    blob_detector('cityscape.jpg', 7, 5, 1.3, 0.007, 'dog');
```



- for the same threshold and other parameters as for the LoG filter, DoG gives a slightly less dense blob output.

Observations and Conclusion:

Let us observe the times taken by different methods of implementation tried out. All times are in seconds.

Image	LoG Filter by down-sampling image	LoG Filter by Upscaling filter	DoG filter	In-scale non-maximum suppression		Between scales non-maximum suppression	blob centres and radii	Drawing blobs
				colfilt	ordfilt2			
butterfly.jpg	0.052239	0.423944	0.073041	0.199762	0.064491	0.017011	0.012830	0.398412
fishes.jpg	0.093341	0.808830	0.088403	0.312586	0.136782	0.033564	0.022435	0.343139
einstein.jpg	0.115916	1.406621	0.130781	0.557314	0.255136	0.061856	0.046788	0.458095
sunflowers.jpg	0.080428	0.616729	0.081169	0.245392	0.113282	0.032574	0.023139	0.660024
astronaut.jpg	0.501216	5.641705	0.596015	2.646698	1.059311	0.281595	0.197243	5.503286
shining.jpg	0.106770	2.510592	0.174116	0.570196	0.235124	0.064306	0.043772	0.901470
lm.jpg	0.062108	0.373270	0.051392	0.339324	0.057360	0.015313	0.014274	1.379213
cityscape.jpg	2.188261	13.537472	1.223416	5.081423	1.650287	0.525569	0.351120	1.862884

Observations:

- The times taken for all functions increases with the resolution of the image, which is obvious.
- Laplacian of Gaussian (LoG) by down sampling image clearly takes lesser time and is more efficient compared to LoG filtering by upscaling filter, which proves our assumption from before. The results from both these methods are similar.
- The up-sampling-filter method gets worse as the image size increases, as seen with cityscape.jpg.
- Derivative of Gaussian (DoG) takes similar times compared to the LoG by downsampling the image.

- The outputs of DoG are not quite similar to those of different LoG implementations for the same parameters, but the features and edges detected are quite similar for most images in all three implementations (can be seen in the output images).
- nlfilter takes a lot more time than the coltfilt and ordfilt2 so it wasn't considered for comparison. It was also mentioned in the nlfilter documentation page that coltfilt is faster. (Reference:https://www.mathworks.com/help/images/ref/nlfilter.html)
- Between ordfilt2 and colfilt, ordfilt2 is clearly the better performing function from the above comparisons.

Observations on parameter values selection:

- Parameters, for ideal blobs number, vary based on the image considered.
- Images with higher resolution give blobs in higher density than those with lower resolutions for the same set of parameters.
- Increasing the k value too much results in very high computations as it is exponentially increased for every scale.
- Too high threshold (e.g. = 0.1) results in too less detections while too small threshold could result in high density of blobs. So, an ideal threshold is necessary.