HW - 2 Scale-space blob detection

Name: Rajiv Ranjan
Ub no:5024 9099

Below is my finding for the implementation of both the efficient and inefficient methods for finding out the regions of interest in the different images. Total of 8 images have been considered for this. 4 of these were provided where as the other 4 images were supposed to be selected by us. I have considered the 4 images that were given to us for this assignment first. here the original images have been placed which is followed by the output of the inefficient implementation done by increasing the filter Size, which is followed by the efficient implementation method of down sampling the image. Both of these output images clearly shows the number of blob circles in them. Also their run time has been compared for the execution of each image. This is shown in a table which shows the run time of both implementations for each image.

The name of the two code files are in matlab folder are:
1)inc_filt_size.m : for the inefficient method
2) downsample.m : for the efficient method

The output images are:
inputfilename_out_1.jpg for the output after the first implementation by inefficient method and inputfilename_out_2.jpg for the output after the efficient method. So in total there are 16 images in the output folder for 16 input images.

1) An explanation of any “interesting" implementation choices that you made??
Ans: The choices for the inefficient implementation have been made in the following ways:
For every image firstly, it was converted into a grayscale image after that it was changed into a double image, which was mentioned in the implementation steps for this assignment.
A sigma value of 2 was taken for the inefficient implementation. Scale level was set to 12. This meant that the filter size would be increased 12 times. The scale multiplier k was taken to be 1.27, this value was achieved after multiple running of the implementations for varying different values of k and finally I selected it be 1.27 which gave an appropriate count of the blobs for this implementation. Also it be noted that that it was made sure that the filter size was made an odd number because of the property of the Gaussian so that there is equal distribution around some value and for that the hsize parameter was needed to made an odd value. This was done using “`2*ceil((3*sigma)) + 1`”. Also here sigma is the value which decides the window size. Also the squaring of the response of Laplacian filter was done as instructed.
Also the output after every filter response was normalized. The scaling factor k was used to change the value of sigma for scalecpace number of iterations (here 12), changing the value of sigma also meant that the value of window size in Laplacian filter would change. Each time the sigma would increase by a factor of k(1.27).
After obtaining the scale space it was also necessary to do the 2d non-maximum suppression was done and for that ordfilt2 function was used. For this function a value of 3 was selected for creating the window size of 3cross3 which would go across the entire image and find the maximum values and would suppress the non maximums around it. This would be done for the whole 12 different scale spaces. Now different combinations of values were tried and passed as parameter in the ordfilt2 function but 3 gave the most satisfactory results. As a smaller value gives a better result here among the neighbors. After this 3D non maximum suppression was done which would give us the maximum value for each pixel across the whole of 12 scalespace.

These values are what we needed to draw the circles. Once we have obtained these values we find these as the parameters to show_all_circles function along with the radius size. The value of radius would change for each iteration and depends on the value of sigma. It is defined as sigma times square root of 2. also we change the value of sigma for each scalespace iteration by the same scaling factor k. For each iteration we get circles around the maximum value co-ordinate in the scalespace (12) and it increases with each iteration as each iteration is multiplied by a factor of k(1.27). Also multiple values of the threshold were applied and were found that the number of the circles formed changes, so after many trials the value of 0.007 was selected.

All the values of the parameter sigma, scalespace, scalefactor (k), for this inefficient were chosen after multiple trials with many different combination of values, with this particular combination giving the most satisfactory results.

When we come to the efficient implementation one that downsamples the image, the parameters have been kept the same value, so the value of sigma, scalespace and scalefactor has been kept the same for the purpose of making a comparison of the two implementations. The difference comes in the implementation part. Here instead of changing the value of sigma by a constant multiple of 1.27(k), which then changes the window size in the filter, we rather divide the image by the same scalespace constant factor 1.27(k). the image is resized and filters is applied on this downsized image and then the other processes remain same other than the fact that the image has to up scaled to it's original size before we make a scalespace of size 12, same as what we did for the inefficient implementations. Other part of the implementation remains same for the sake of comparison of the run time of the two implementations.

The most important implementation detail to note here is that downsampling and upsampling works so much better if you take a "bicubic" instead of "bilinear" to maintain your spatial resolution

There are 2 major findings here:

1) The number of circles created for efficient implementation was always larger than what we had for the inefficient algorithms. Obviously it means it is able to find more areas of interest than the efficient algorithm
2) Secondly, comparing the time taken by each of these implementations suggests that the method of downscaling image runs much faster than the method that increases filter size. The efficient implementation is indeed more efficient than the inefficient implementations.

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

Soln: these finding were observed:

Increasing the value of threshold(thresh) decreases the number of circles.

Increasing the value of parameters in ordfilt2 decreases the number of circles.

Decreasing the value of sigma increases the number of the circles formed and decreases the run time of the algorithms.

Increasing the value of the scalespace increases the run time of the implementation but also increases the number of circles formed slightly.

Increasing the value of scalefactor k increases the run time of the implementation significantly around 20 times but the number of circle formed were also more. For (k=1.27 to 1.5)

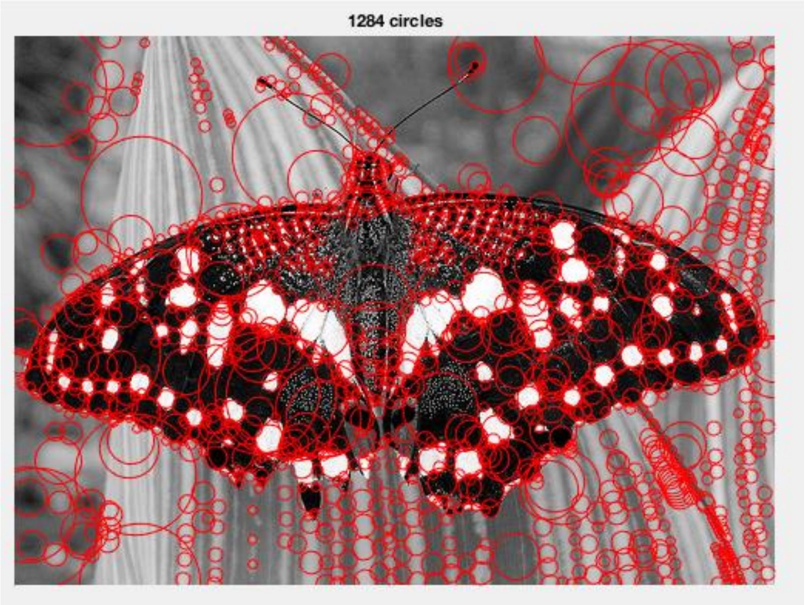
Similarly decreasing the value of k decreases the run time of the implementation and also decreases the value of the circles formed.

Based on these the parameters used in the implementation were decided and also the various trails have been commented out in the code.

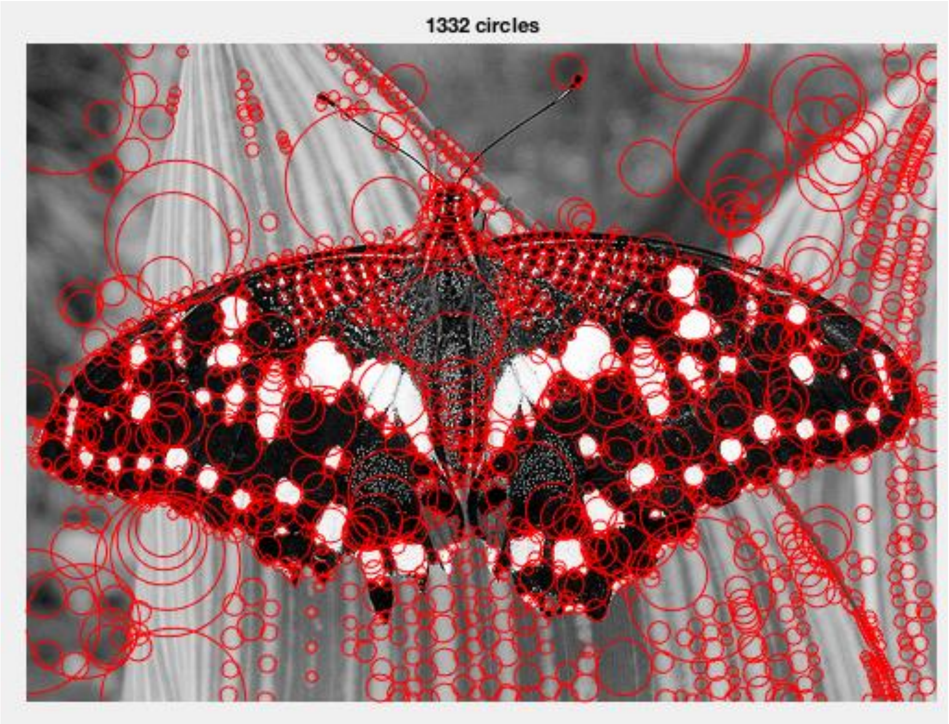
1) Butterfly.jpg

Original image:

a) Increasing the filter size Implementation output:



b) Down sampling the Image Implementation Output:

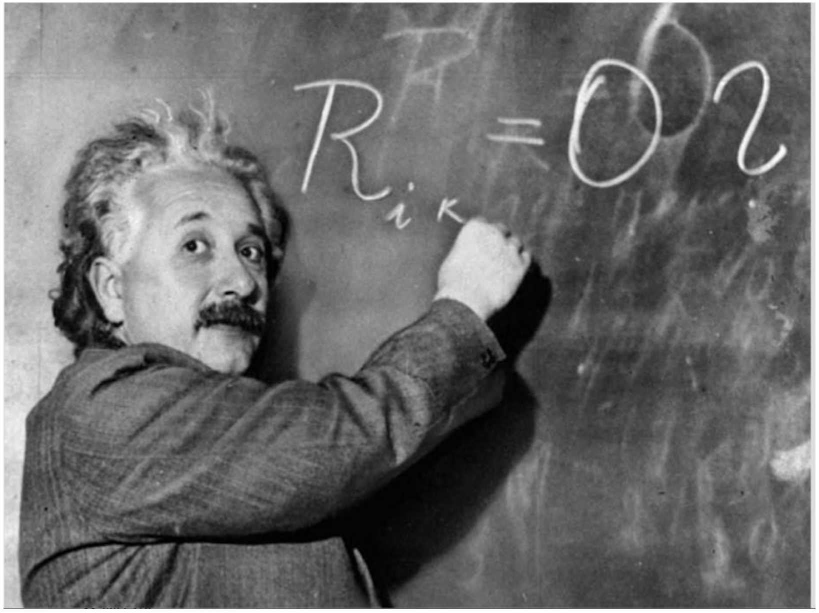


Time Comparison

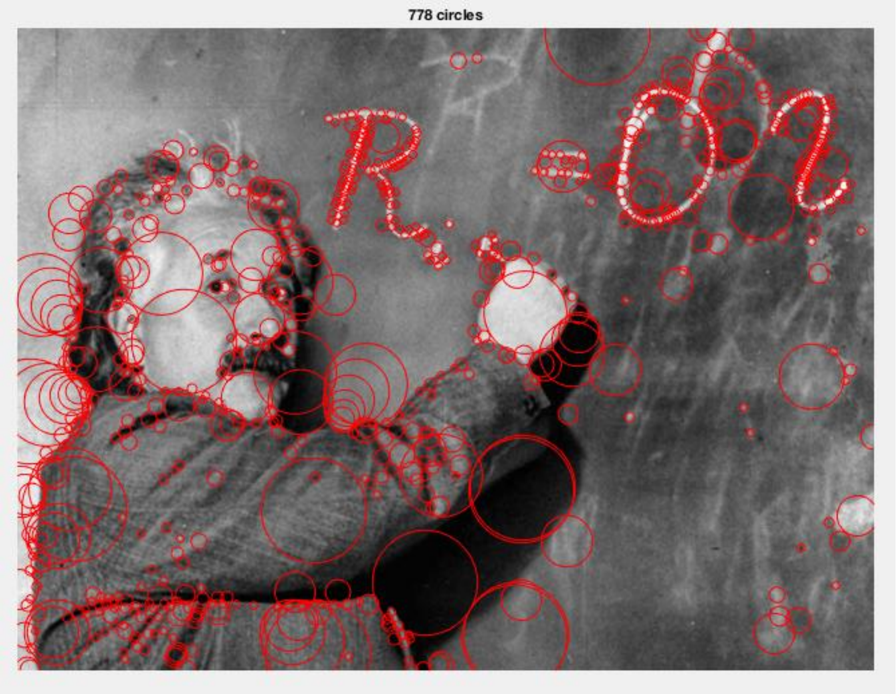
Time Comparison	Time taken	No of circles
Increasing the filter size Implementation	2.891573 seconds	1284
Down sampling the Image implementation	1.917930 seconds	1332

2) Einstein.jpg

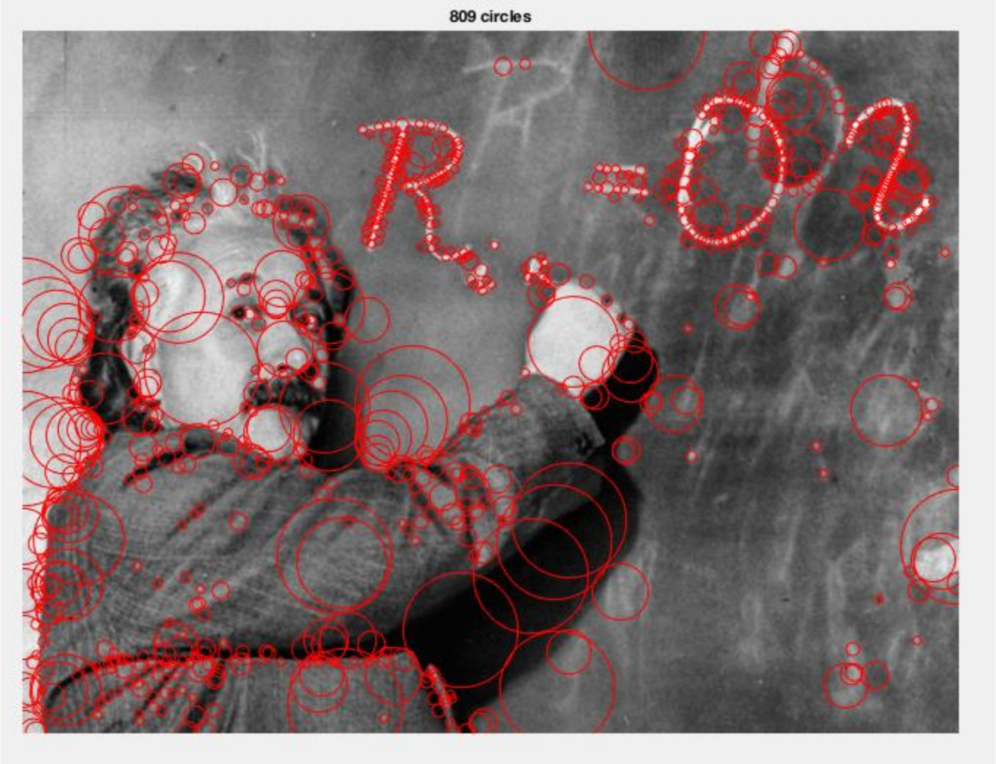
Original Image



a) Increasing the filter size Implementation output:



b) Down sampling the Image Implementation Output:



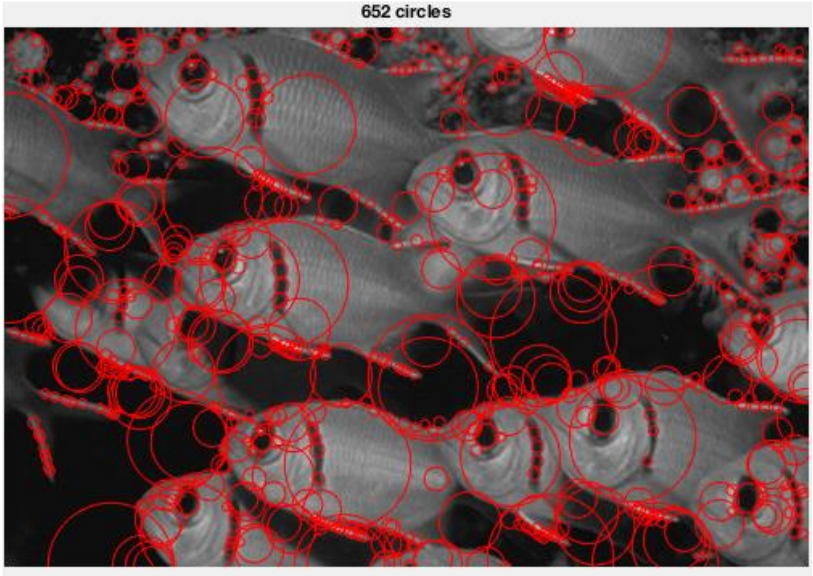
Time Comparison:

Time Comparison	Time taken	No of circles

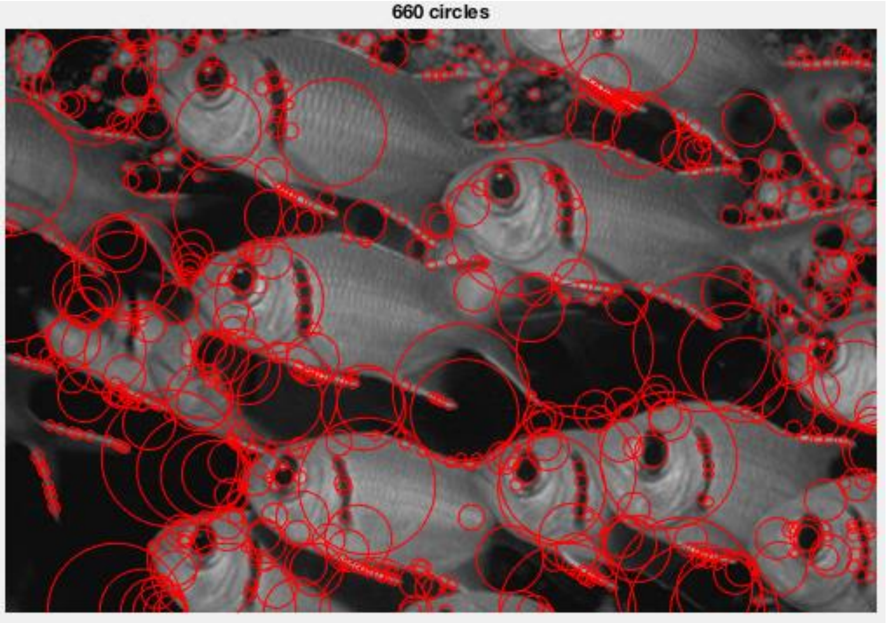
Increasing the filter size Implementation	3.842674 seconds	778
Down sampling the Image implementation	1.966670 seconds	809

3) fishes.jpg

a) Increasing the filter size Implementation output:



b) Down sampling the Image Implementation Output:

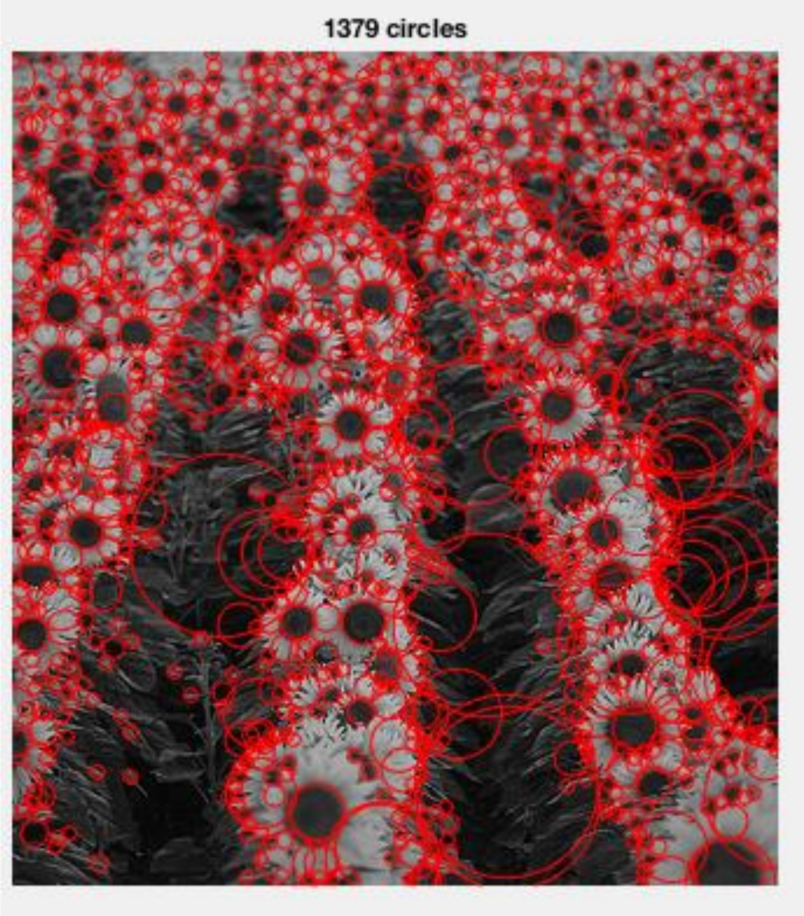


Time Comparison

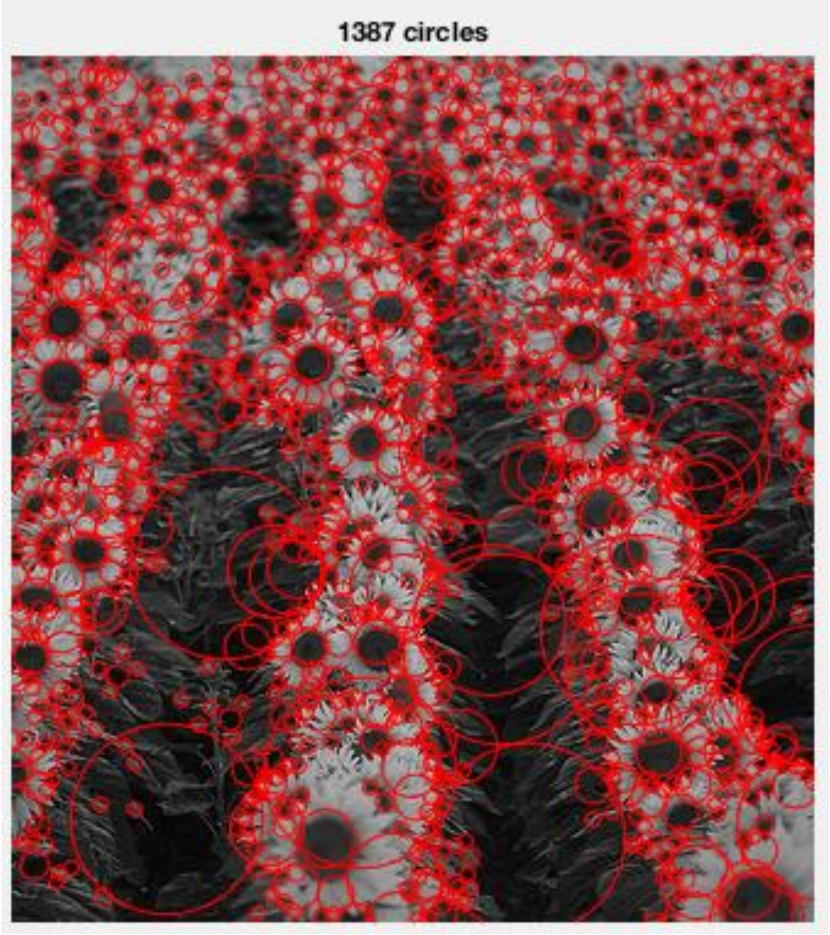
Time Comparison	Time taken	No of circles
Increasing the filter size Implementation	2.581322 seconds	652
Down sampling the Image implementation	1.612844 seconds	660

4) sunflowers.jpg (Original Image)

a) Increasing the filter size Implementation output:



b) Down sampling the Image Implementation Output:



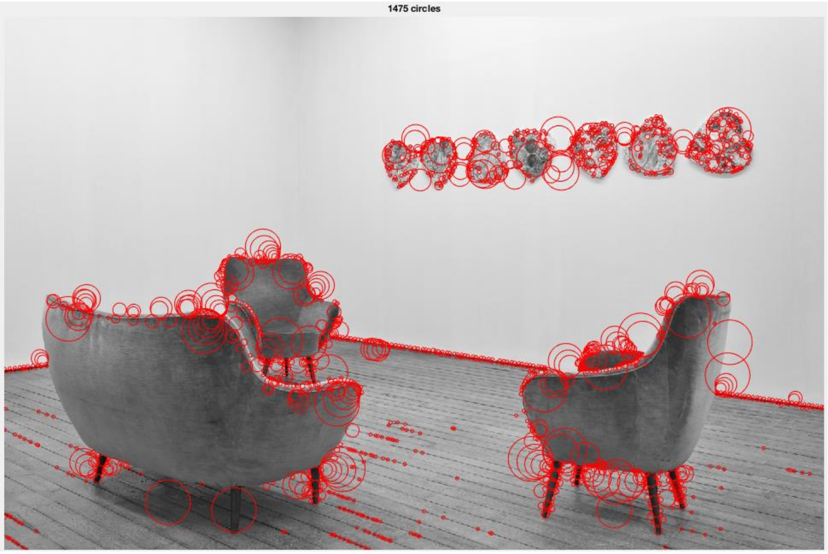
Time Comparison	Time taken	No of circles
Increasing the filter size Implementation	2.487762 seconds	1379
Down sampling the Image implementation	1.883150 seconds	1387

4 images that I took for comparison of the two implementations are:

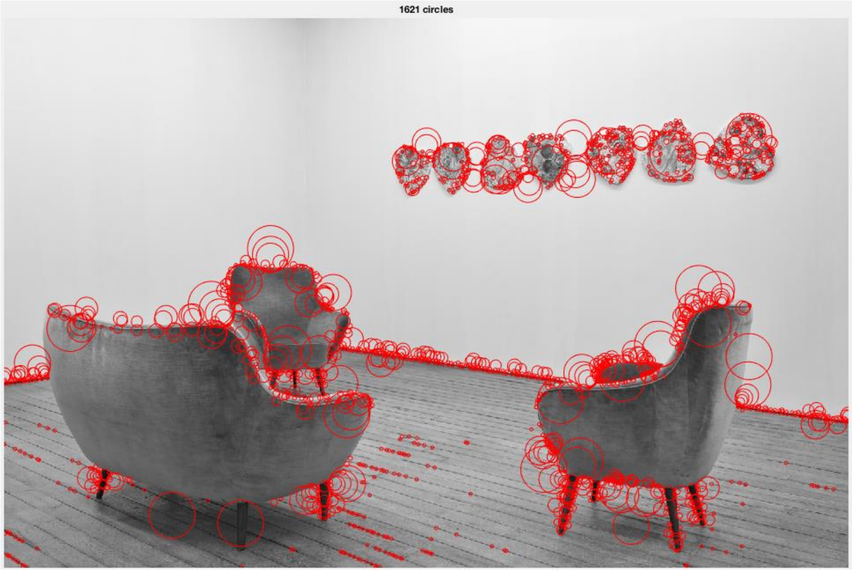
1) Im1.jpg

Original image:

a) Increasing the filter size Implementation output:



b) Down sampling the Image Implementation Output:

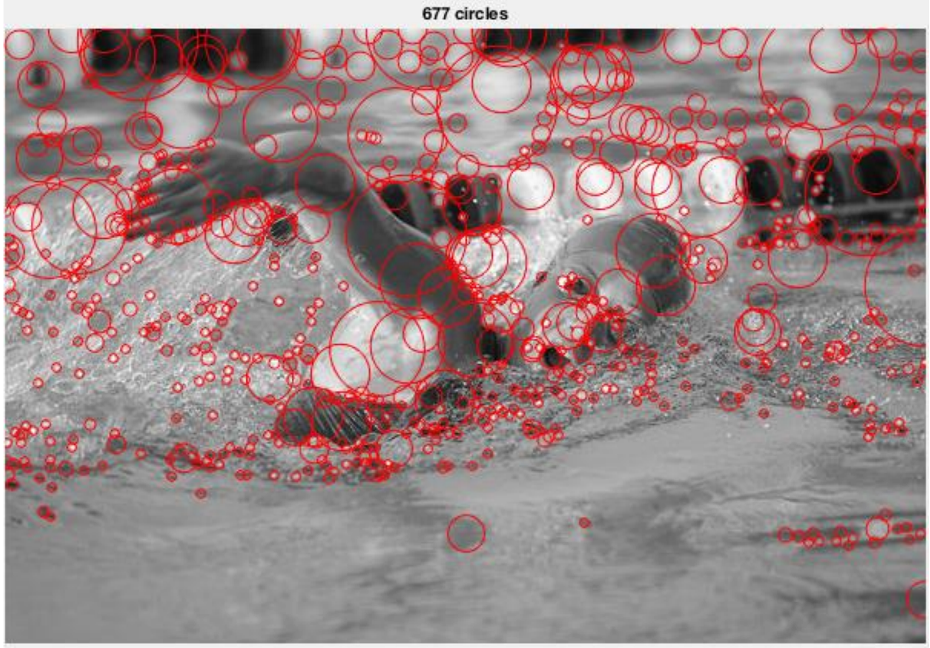


Time Comparison	Time taken	No of circles
Increasing the filter size Implementation	14.096155 seconds	1475
Down sampling the Image implementation	4.883662 seconds	1621

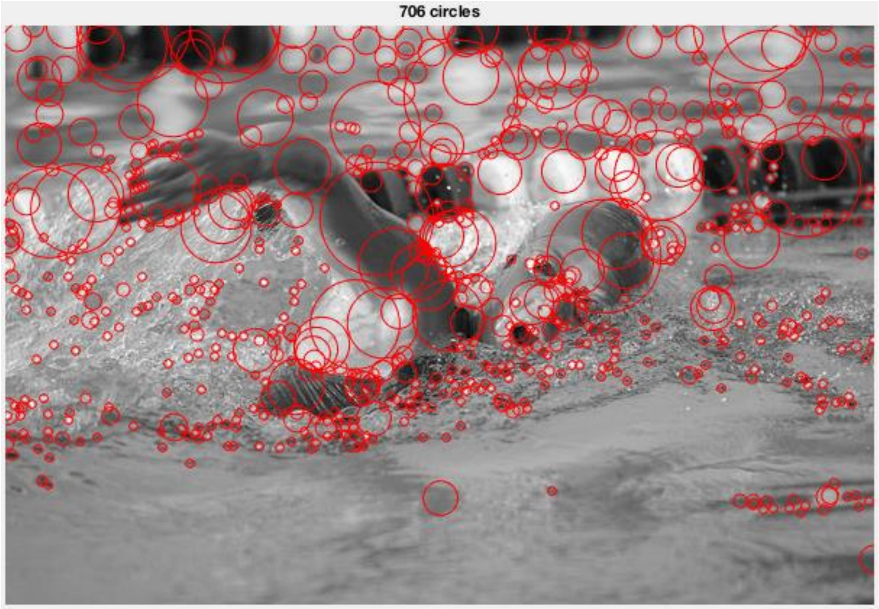
2) Im2.jpg

Original image:

a) Increasing the filter size Implementation output:



b) Down sampling the Image Implementation Output:

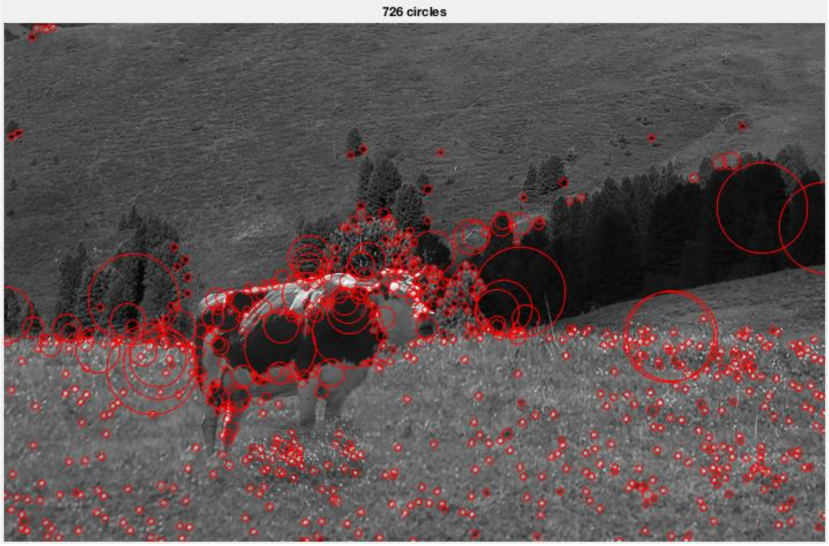


Time Comparison	Time taken	No of circles
Increasing the filter size Implementation	3.219553 seconds	677
Down sampling the Image implementation	1.823791 seconds	706

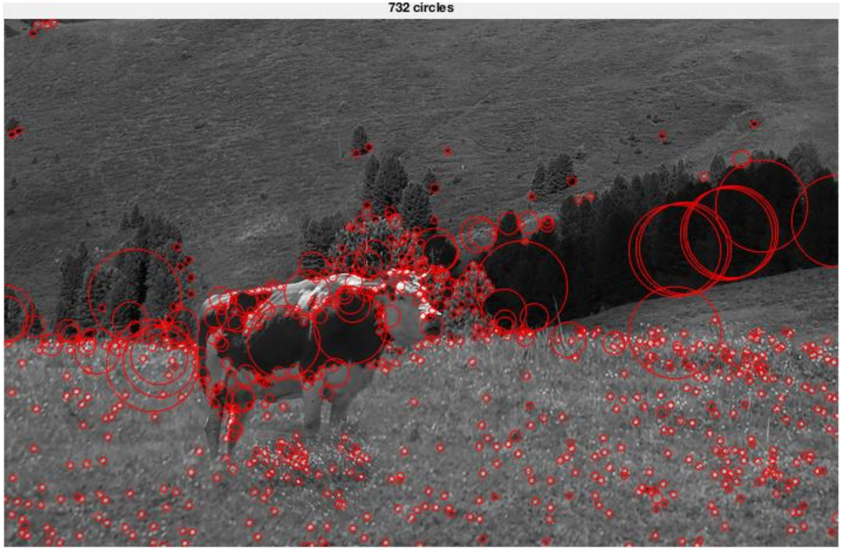
3) Im3.jpg

Original image:

a) Increasing the filter size Implementation output:



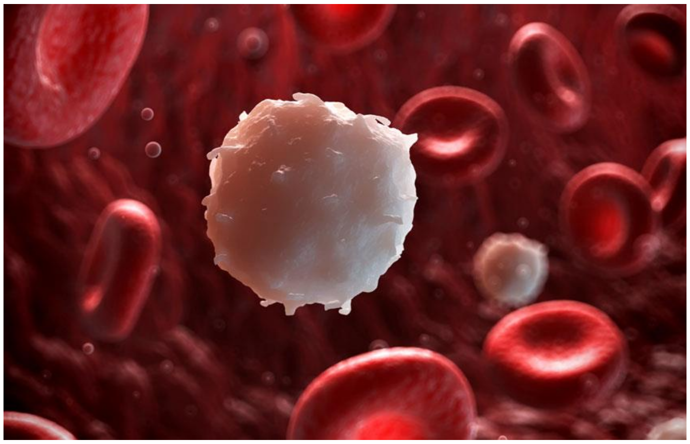
b) Down sampling the Image Implementation Output:



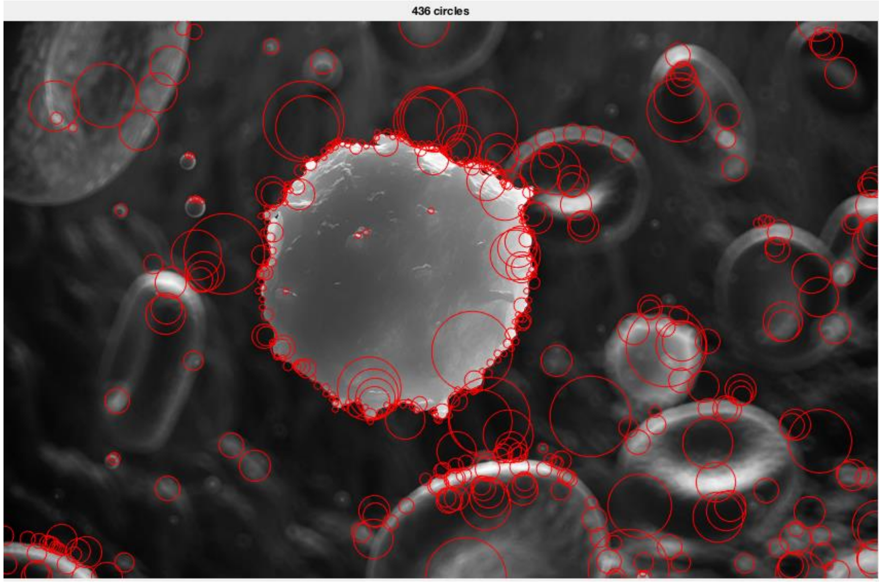
Time Comparison	Time taken	No of circles
Increasing the filter size Implementation	4.070590 seconds	726
Down sampling the Image implementation	1.966248 seconds	732

4) Im4.jpg

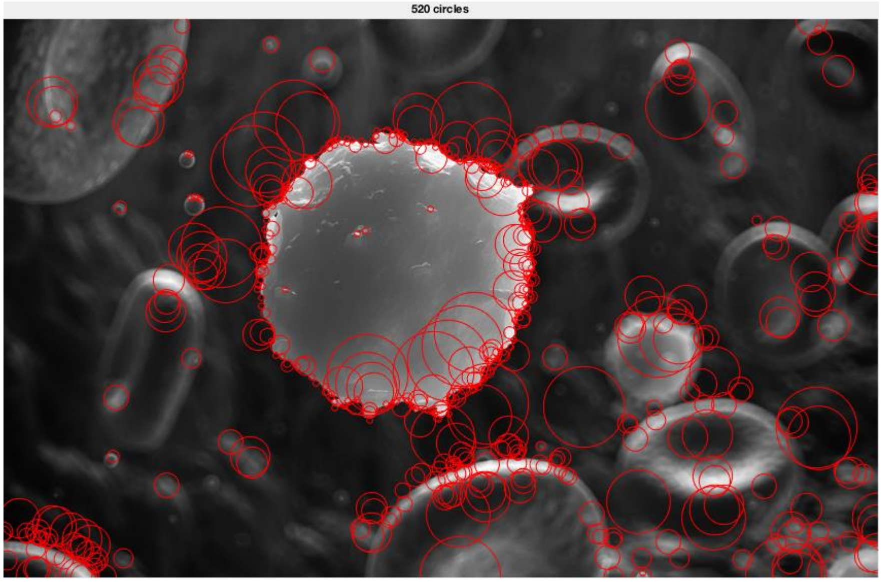
Original image:



a) Increasing the filter size Implementation output:



b) Down sampling the Image Implementation Output:



Time Comparison	Time taken	No of circles
Increasing the filter size Implementation	5.058795 seconds	436
Down sampling the Image implementation	2.389106 seconds	520