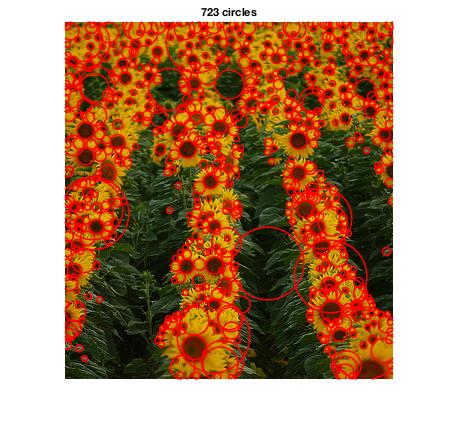
CSE 573

Hw2 Report

Enze Qian

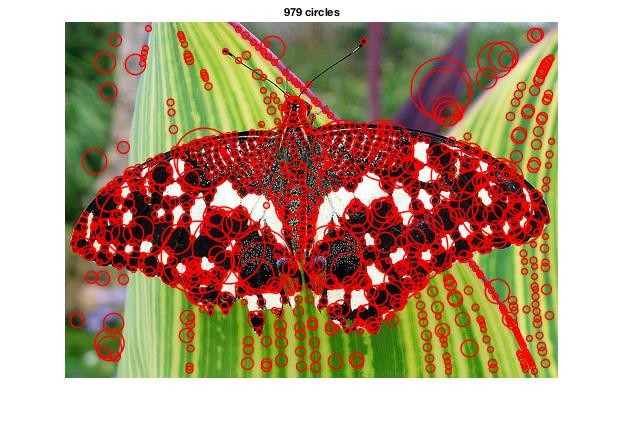
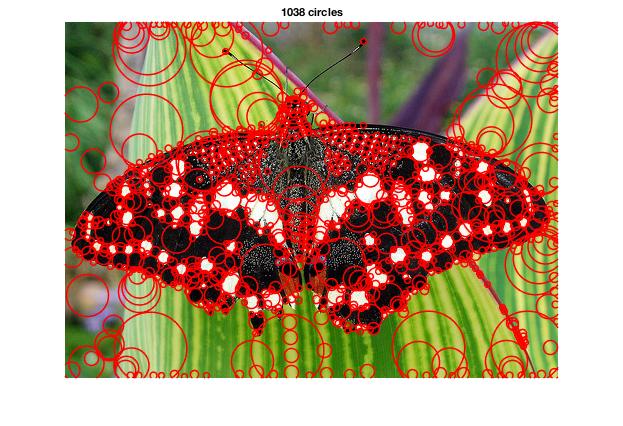
50091378

1. Output of circle detector on all images



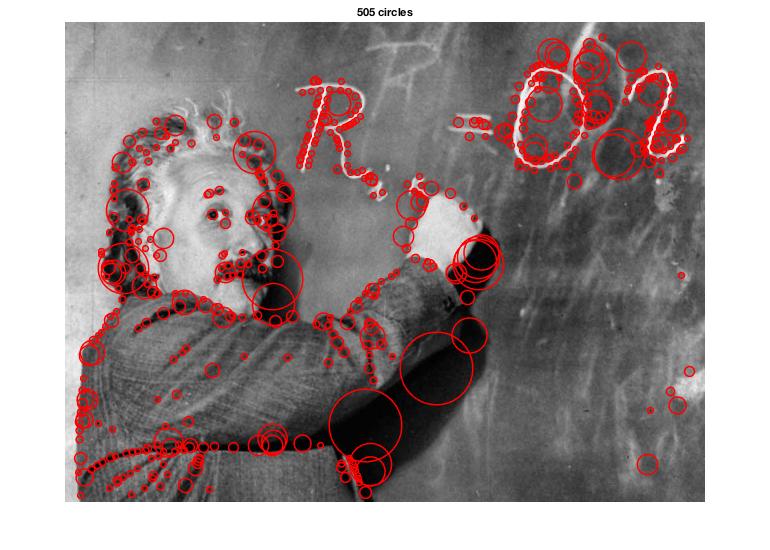
Left: With “inefficient” method, elapsed time is 1.055248 seconds.

Right: With “efficient” method, elapsed time is 0.083849 seconds.



Left: With “inefficient” method, elapsed time is 1.581091 seconds.

Right: With “efficient” method, elapsed time is 0.143090 seconds.



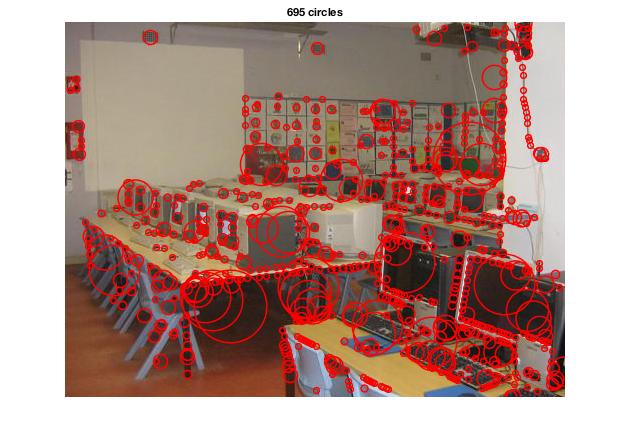
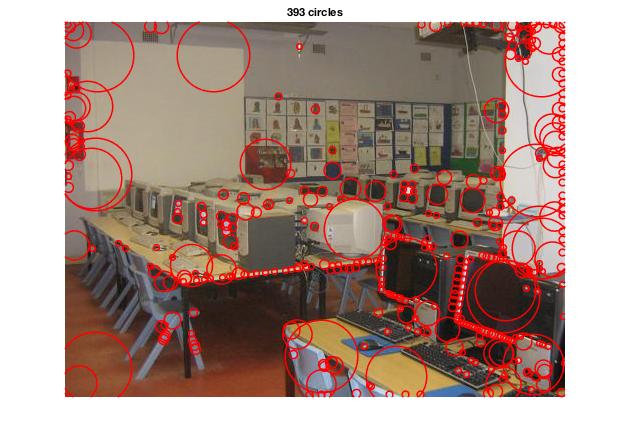
Left: With “inefficient” method, elapsed time is 2.545078 seconds.

Right: With “efficient” method, elapsed time is 0.187073 seconds.



Left: With “inefficient” method, elapsed time is 1.554247 seconds.

Right: With “efficient” method, elapsed time is 0.127936 seconds.



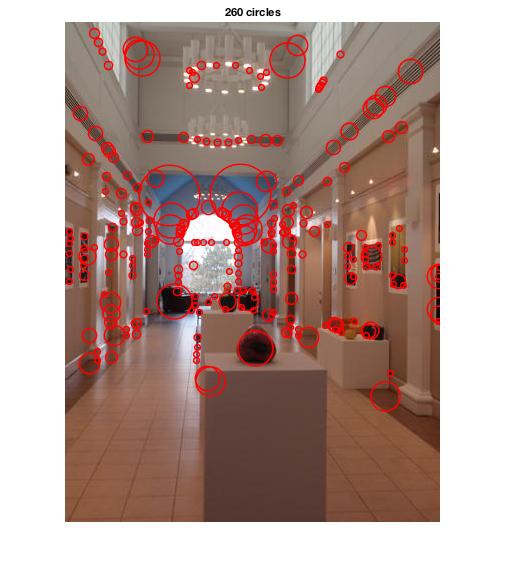
Left: With “inefficient” method, elapsed time is 1.827030 seconds.

Right: With “efficient” method, elapsed time is 0.134068 seconds.



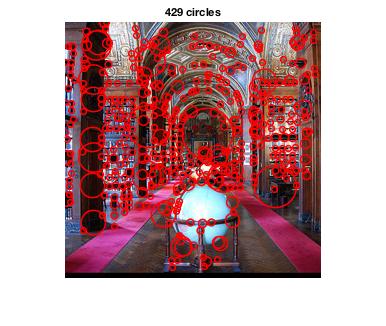
Left: With “inefficient” method, elapsed time is 1.469022 seconds.

Right: With “efficient” method, elapsed time is 0.116854 seconds.



Left: With “inefficient” method, elapsed time is 1.628208 seconds.

Right: With “efficient” method, elapsed time is 0.139809 seconds.



Left: With “inefficient” method, elapsed time is 0.696131 seconds.

Right: With “efficient” method, elapsed time is 0.067359 seconds.

The only difference between these two implementations are the way to get scale space, so I add ‘tic toc’ into SlowGetScaleSpace.m and FastGetScaleSpace.m to get running time for each method.During my implementation, I found that outputs of inefficient and efficient method are slightly different if I keep all other parameters same(except threshold, in fact; the value of threshold doesn’t affect these two functions above); However, in order to compare the running time, I have to keep all parameters(k, level and initial\_scale) staying same and give two outputs for each image.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Sunflower | Butterfly | Einstein | Fish | Computer | Ice | Gallery | Library |
| Inefficient method | 1.055248s | 1.581091s | 2.545078s | 1.554247s | 1.827030s | 1.469022s | 1.628208s | 0.696131s |
| Efficient method | 0.083849s | 0.143090s | 0.187073s | 0.127936s | 0.134068s | 0.116854s | 0.139809s | 0.067359s |

As We can see, the method that downsamples the images is a lot more efficient than the method that increases filter size.

2. An explanation of any “interesting” implementation choices that you made.

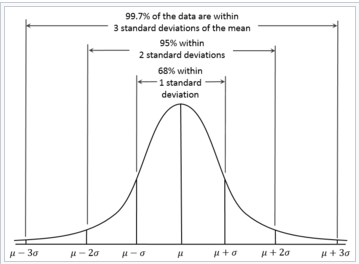
a. When I use the function: fspecial

[h](https://www.mathworks.com/help/images/ref/fspecial.html#outputarg_h) = fspecial('log',[hsize](https://www.mathworks.com/help/images/ref/fspecial.html" \l "inputarg_hsize),[sigma](https://www.mathworks.com/help/images/ref/fspecial.html#inputarg_sigma)) returns a rotationally symmetric Laplacian of Gaussian filter of size hsize with standard deviation sigma (positive).

I decided use 2\*3\*sigma+1 for the value of hsize.

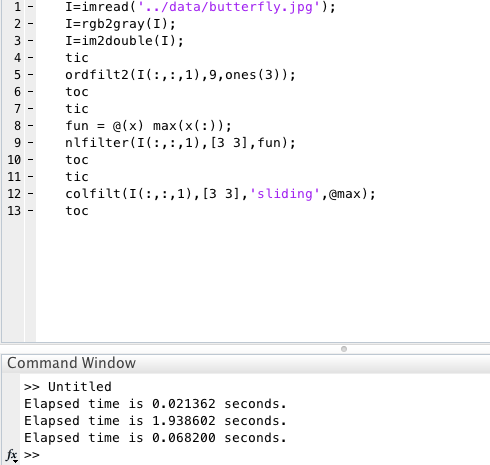
First, add one to ensure the filter width be odd, because we can find a center in an odd width.

Second, I use 2\*3\*sigma as the filter size because by the property of normal distribution, the value less than three standard deviations account for 99.73%



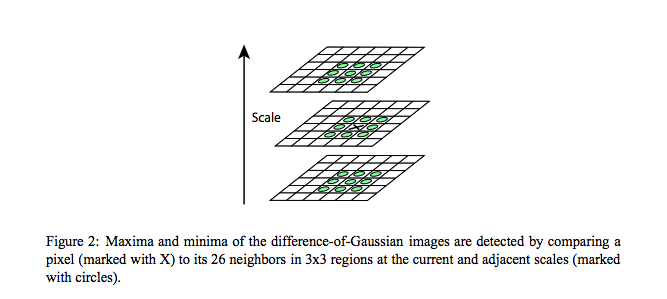
Reference: Normal distribution, Wikipedia.

b. When I do nonmaximum suppression for each layer, I may use three functions nlfilter, colfilt orordfilt2. I run some tests on these functions as following:



It turns out ordfilt2 is the fastest one. Then I choose ordfilt2 in my implementation.

When I do nonmaximum suppression for the entire scale space, for each layer, I compare it with its two neighborhoods because “Maxima and minima of the difference-of-Gaussian images are detected by comparing a pixel to its 26 neighbors in 3x3 regions at the current and adjacent scales.”

For the first and last layer, I only compared it with one other layer because it only has one neighborhood.

Reference: David G Lowe. Distinctive image features from scale-invariant keypoints.

c. There are three interpolation methods 'nearest', 'bilinear' and 'bicubic' when I use imresize to upsample the filtered images. I try all of these method and get outputs.



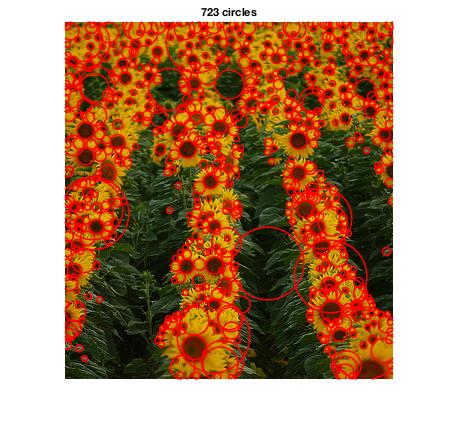
(Left to right: 'nearest', 'bilinear' and 'bicubic')

‘bicubic’ works best and I choose it in my imresize function.

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

Like I mentioned above, I found that outputs of inefficient and efficient method are slightly different. Hence, I found different parameters for two methods in order to get optimal output. I decide to use sunflower image to do my test because it contains most amount of circles. (Initial scale is 2.)

For inefficient method:

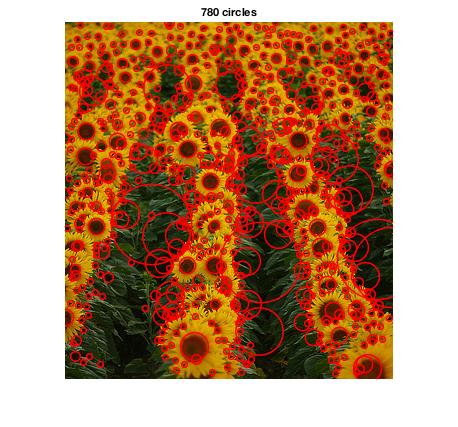


k=1.2;

level=15;

threshold=0.02;

For efficient method:



k=1.19;

level=15;

threshold=0.001;

4.Extensions.

The function fspecial only takes integer as size input. In order to get integer, I can either use floor or ceil function to get nearest integer. In my experience, it seems that there is no apparent difference between output images weather I use ceil or floor.