[MINI PROJECT 2: SCALE-SPACE BLOB DETECTION]

ABSTRACT

In computer vision, blob detection methods are aimed at detecting regions in a digital image that differ in properties, such as brightness or color, compared to surrounding regions. Informally, a blob is a region of an image in which some properties are constant or approximately constant; all the points in a blob can be considered in some sense to be similar to each other. The most common method for blob detection is convolution.

Contents:

Mini Project 1: Colorizing ProkudinGorskii images of the Russian Empire

1- Background:

blob detection methods are aimed at detecting regions in a digital image that differ in properties, such as brightness or color, compared to surrounding regions. Informally, a blob is a region of an image in which some properties are constant or approximately constant; all the points in a blob can be considered in some sense to be similar to each other. The most common method for blob detection is convolution.

Given some property of interest expressed as a function of position on the image, there are two main classes of blob detectors:

(i) differential methods, which are based on derivatives of the function with respect to position.
(ii) methods based on local extrema, which are based on finding the local maxima and minima of the function.

With the more recent terminology used in the field, these detectors can also be referred to as interest point operators, or alternatively interest region operators (see also interest point detection and corner detection).

2- Overview:

There are several motivations for studying and developing blob detectors. One main reason is to provide complementary information about regions, which is not obtained from edge detectors or corner detectors. In early work in the area, blob detection was used to obtain regions of interest for further processing. These regions could signal the presence of objects or parts of objects in the image domain with application to object recognition and/or object tracking. In other domains, such as histogram analysis,

blob descriptors can also be used for peak detection with application to segmentation. Another common use of blob descriptors is as main primitives for texture analysis and texture recognition. In more recent work, blob descriptors have found increasingly popular use as interest points for wide baseline stereo matching and to signal the presence of informative image features for appearance-based object recognition based on local image statistics. There is also the related notion of ridge detection to signal the presence of elongated objects.

3- Steps:

1- Don't forget to convert images to grayscale (rgb2gray command) and double (im2double).

2- For creating the Laplacian filter, use the fspecial function (check the options). Pay careful attention to setting the right filter mask size. Hint: Should the filter width be odd or even?

3- It is relatively inefficient to repeatedly filter the image with a kernel of increasing size. Instead of increasing the kernel size by a factor of k, you should downsample the image by a factor 1/k. In that case, you will have to upsample the result or do some interpolation in order to find maxima in scale space. For full credit, you should turn in both implementations: one that increases filter size, and one that downsamples the image. In your report, list the running times for both versions of the algorithm and discuss differences (if any) in the detector output. For timing, use tic and toc commands.

   Hint 1: think about whether you still need scale normalization when you downsample the image instead of increasing the scale of the filter.
   Hint 2: For the efficient implementation, pay attention to the interpolation method you're using to upsample the filtered images (see the options of the imresize function). What kind of interpolation works best?

4- You have to choose the initial scale, the factor k by which the scale is multiplied each time, and the number of levels in the scale space. I typically set the initial scale to 2, and use 10 to 15 levels in the scale pyramid. The multiplication factor should depend on the largest scale at which you want regions to be detected.

5- You may want to use a three-dimensional array to represent your scale space. It would be declared as follows:
   - *scale_space = zeros(h,w,n); % [h,w] - dimensions of image, n - number of levels in scale space*
   - Then scale_space(:,:,i) would give you the ith level of the scale space. Alternatively, if you are storing different levels of the scale pyramid at different resolutions, you may want to use a cell array, where each "slot" can accommodate a different data type or a matrix of different dimensions. Here is how you would use it:
   - *scale_space = cell(n,1); %creates a cell array with n "slots" scale_space{i} = my_matrix; % store a matrix at level i*

6- To perform nonmaximum suppression in scale space, you should first do nonmaximum suppression in each 2D slice separately. For this, you may find functions nlfilter, colfilt or ordfilt2 useful. Play around with these functions, and try to find the one that works the fastest. To extract the final nonzero values (corresponding to detected regions), you may want to use the find function.

7- You also have to set a threshold on the squared Laplacian response above which to report region detections. You should play around with different values and choose one you like best.

8- To display the detected regions as circles, you can use this function (or feel free to search for a suitable MATLAB function or write your own). Hint: Don't forget that there is a multiplication factor that relates the scale at which a region is detected to the radius of the circle that most closely "approximates" the region

4- Bonus Points:

1- Implement the difference-of-Gaussian pyramid as mentioned in class and described in David Lowe's paper. Compare the results and the running time to the direct Laplacian implementation.

- This approach has been named the Scale Invariant Feature Transform (SIFT), as it transforms image data into scale-invariant coordinates relative to local features. An important aspect of this approach is that it generates large numbers of features that densely cover the image over the full range of scales and locations. A typical image of size 500x500 pixels will give rise to about 2000 stable features (although this number depends on both image content and choices for various parameters). The quantity of features is particularly important for object recognition, where the ability to detect small objects in cluttered backgrounds requires that at least 3 features be correctly matched from each object for reliable identification. For image matching and recognition, SIFT features are first extracted from a set of reference images and stored in a database. A new image is matched by individually comparing each feature from the new image to this previous database and finding candidate matching features based on Euclidean distance of their feature vectors.

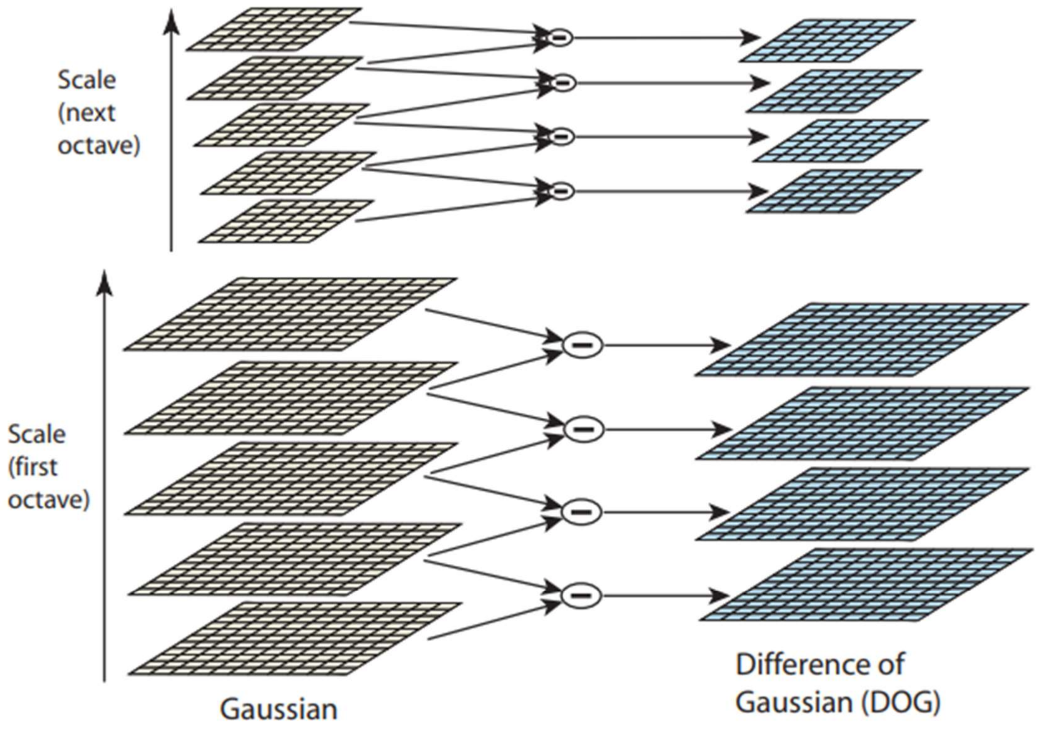


Figure 1: For each octave of scale space, the initial image is repeatedly convolved with Gaussians to produce the set of scale space images shown on the left. Adjacent Gaussian images are subtracted to produce the difference-of-Gaussian images on the right. After each octave, the Gaussian image is down-sampled by a factor of 2, and the process repeated.

- The key point descriptors are highly distinctive, which allows a single feature to find its correct match with good probability in a large database of features. However, in a cluttered 2 image, many features from the background will not have any correct match in the database, giving rise to many false matches in addition to the correct ones. The correct matches can be filtered from the full set of matches by identifying subsets of key points that agree on the object and its location, scale, and orientation in the new image. The probability that several features will agree on these parameters by chance is much lower than the probability that any individual feature match will be in error. The determination of these consistent clusters can be performed rapidly by using an efficient hash table implementation of the generalized Hough transform. Each cluster of 3 or more features that agree on an object and its pose is then subject to further detailed verification. First, a least-squared estimate is made for an affine approximation to the object pose. Any other image features consistent with this pose are identified, and outliers are discarded. Finally, a detailed computation is made of the probability that a particular set of features indicates the presence of an object, given the accuracy of fit and number of probable false matches. Object matches that pass all these tests can be identified as correct with high confidence.

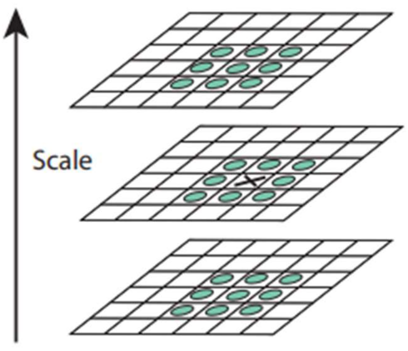


Figure 2: Maxima and minima of the difference-of-Gaussian images are detected by comparing a pixel (marked with X) to its 26 neighbors in 3x3 regions at the current and adjacent scales (marked with circles).

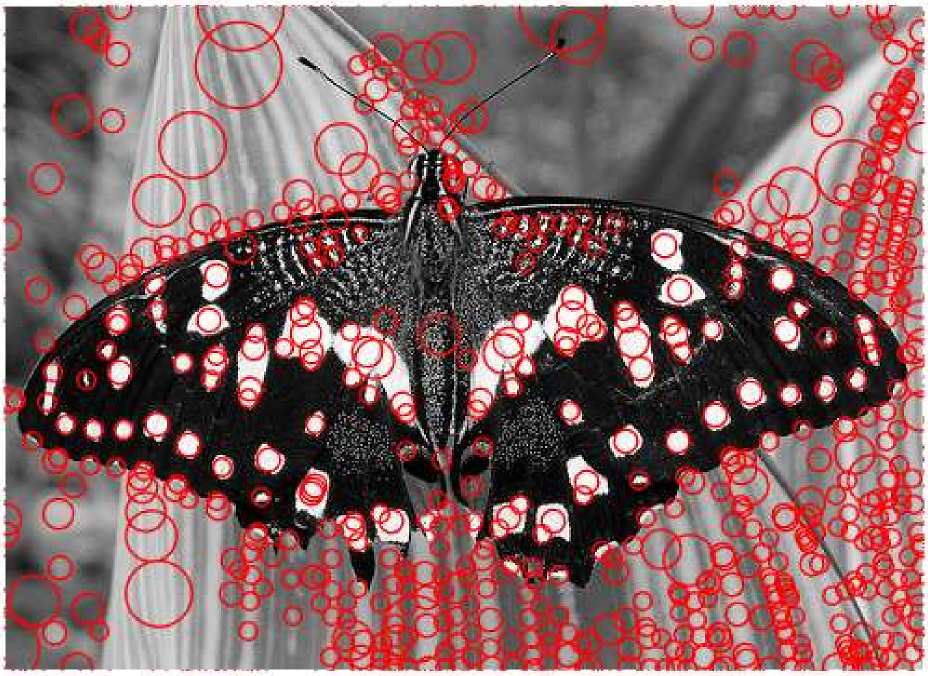
5- Implementation and results:

1. Laplacian blob detector: {Up Sampling}

- ***Algorithm steps:***

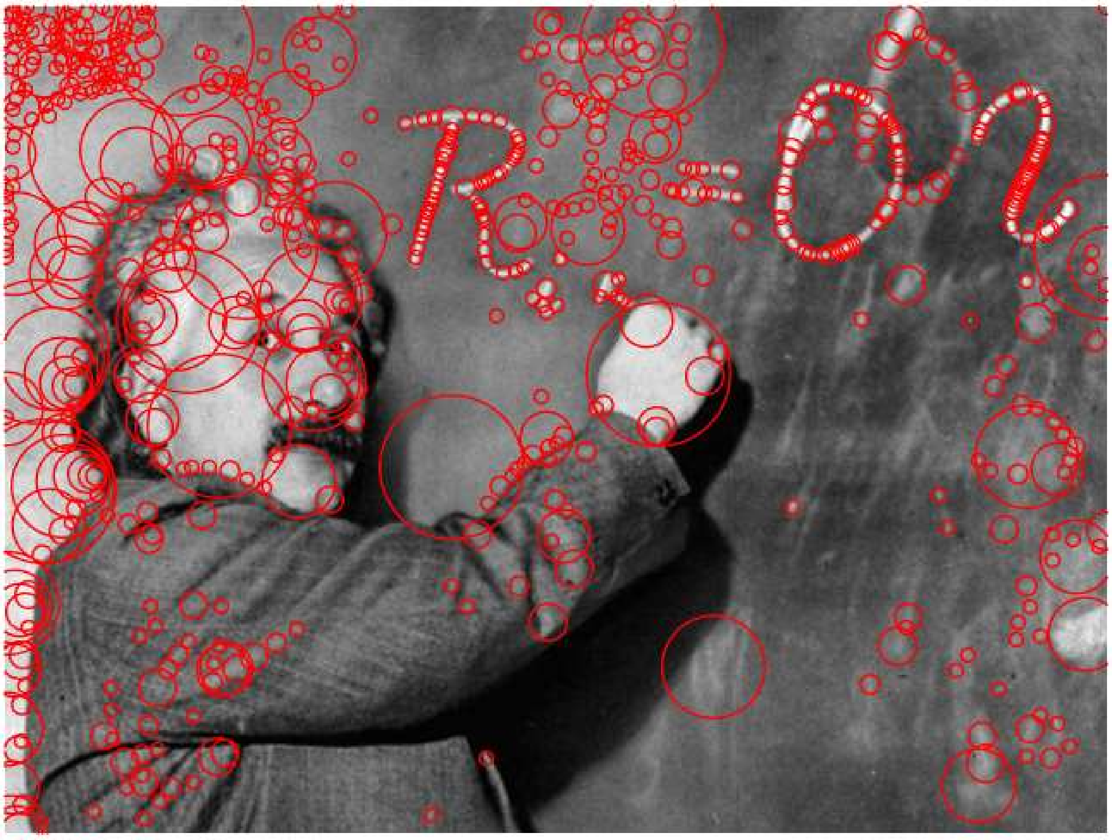
```
o Import image.
o convert it to gray
o convert it to double
• Perform LoG filter to image for several levels
o tic
o for i = 1:levels
o      Generate a Laplacian of Gaussian filter
o      Filter the img with LoG
o      Increase scale by a factor k
o end
o toc
• nonminimum suppression
• for i = 1:levels
•     % replaces each element by the orderth element
  in the sorted set of
•     neighbors by ordfit2.
• end
•
• for i = 1:levels
•     survive_space = scale_space.* img;
• end
• Find all the coordinates and corresponding sigma
• for num = 1:levels
•     % Find index and corresponding radius
• end
• show all circles
```

circles: 585, threshold: 0.300, scale: 2.000, factor k: 1.300

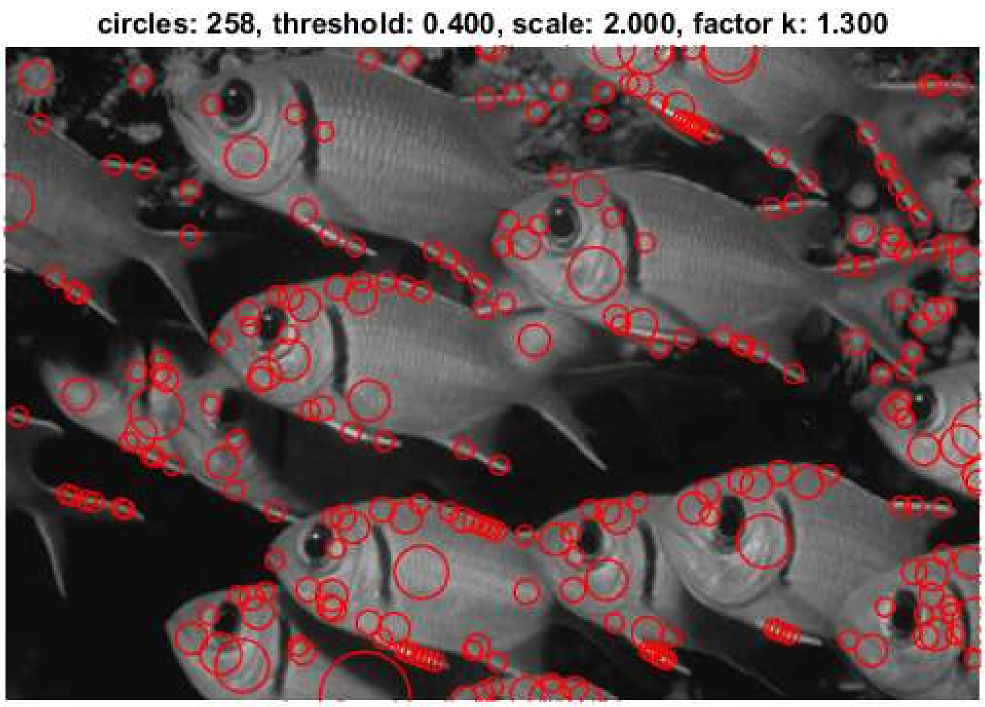


```
Elapsed time is 0.686530 seconds.
```

circles: 636, threshold: 0.500, scale: 2.000, factor k: 1.400

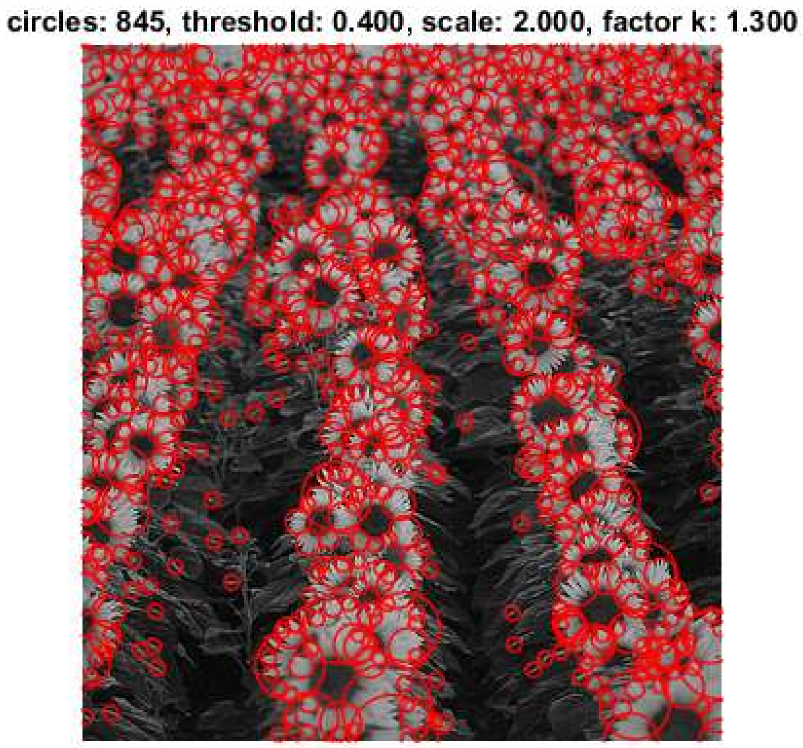


```
Elapsed time is 2.545424 seconds.
```



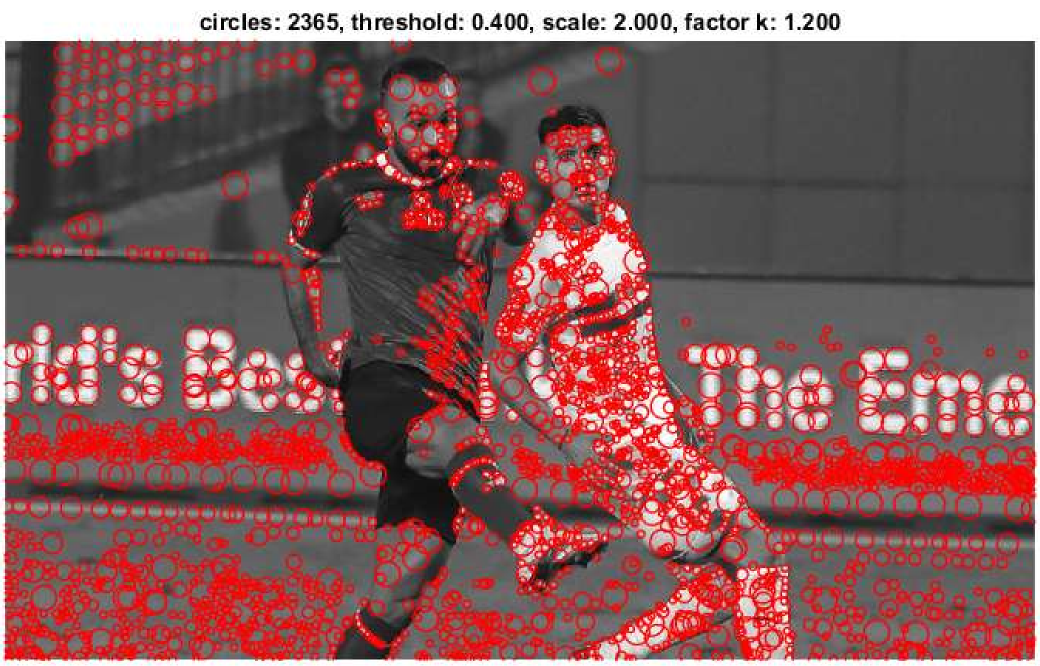
circles: 258, threshold: 0.400, scale: 2.000, factor k: 1.300

```
Elapsed time is 0.919862 seconds.
```

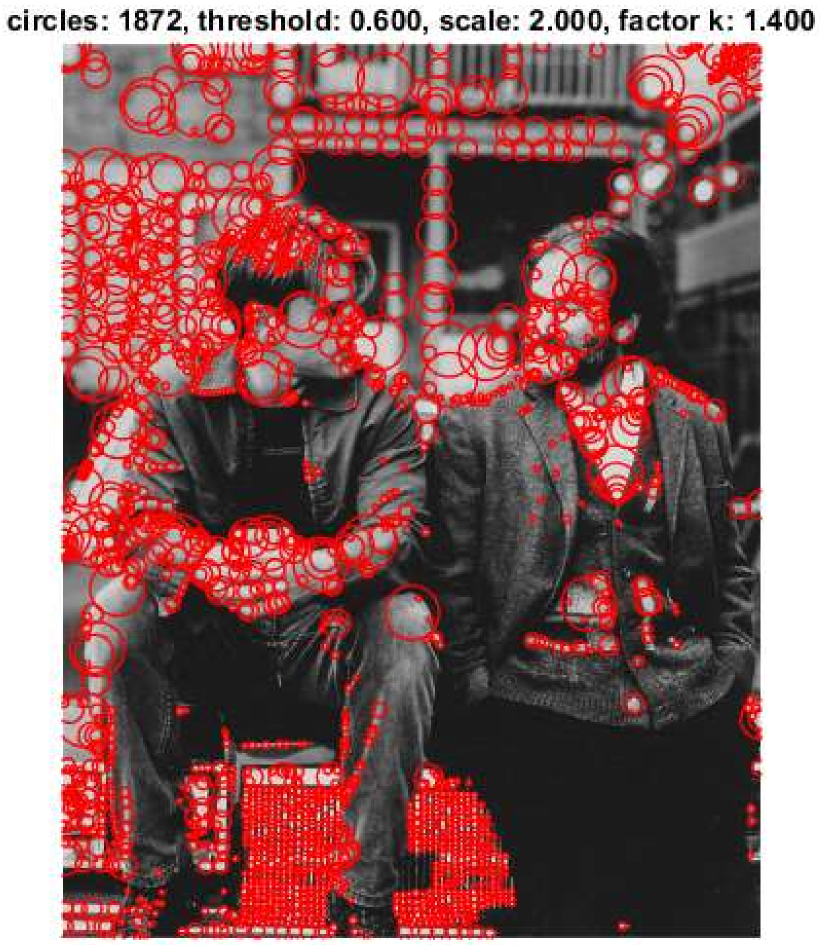


circles: 845, threshold: 0.400, scale: 2.000, factor k: 1.300

```
Elapsed time is 0.656524 seconds.
```

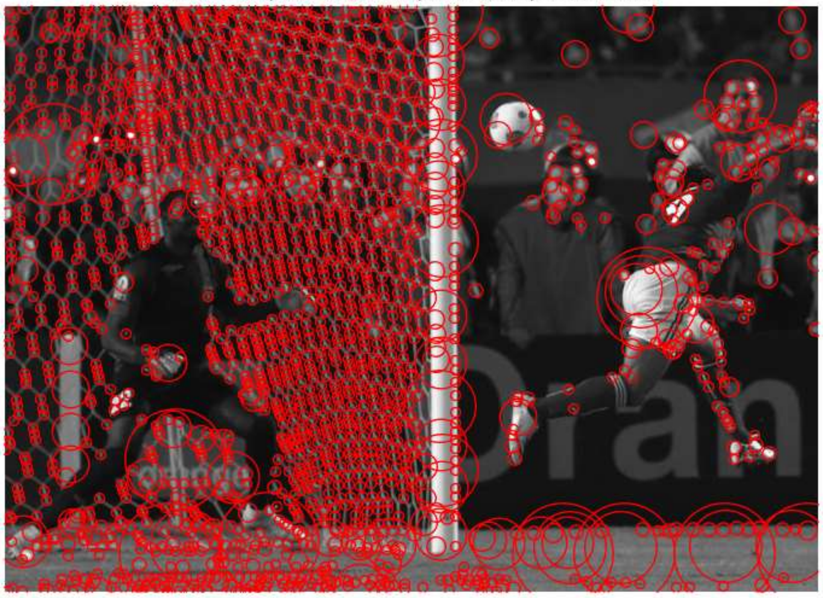

circles: 2365, threshold: 0.400, scale: 2.000, factor k: 1.200

Elapsed time is 0.972935 seconds.


circles: 1872, threshold: 0.600, scale: 2.000, factor k: 1.400

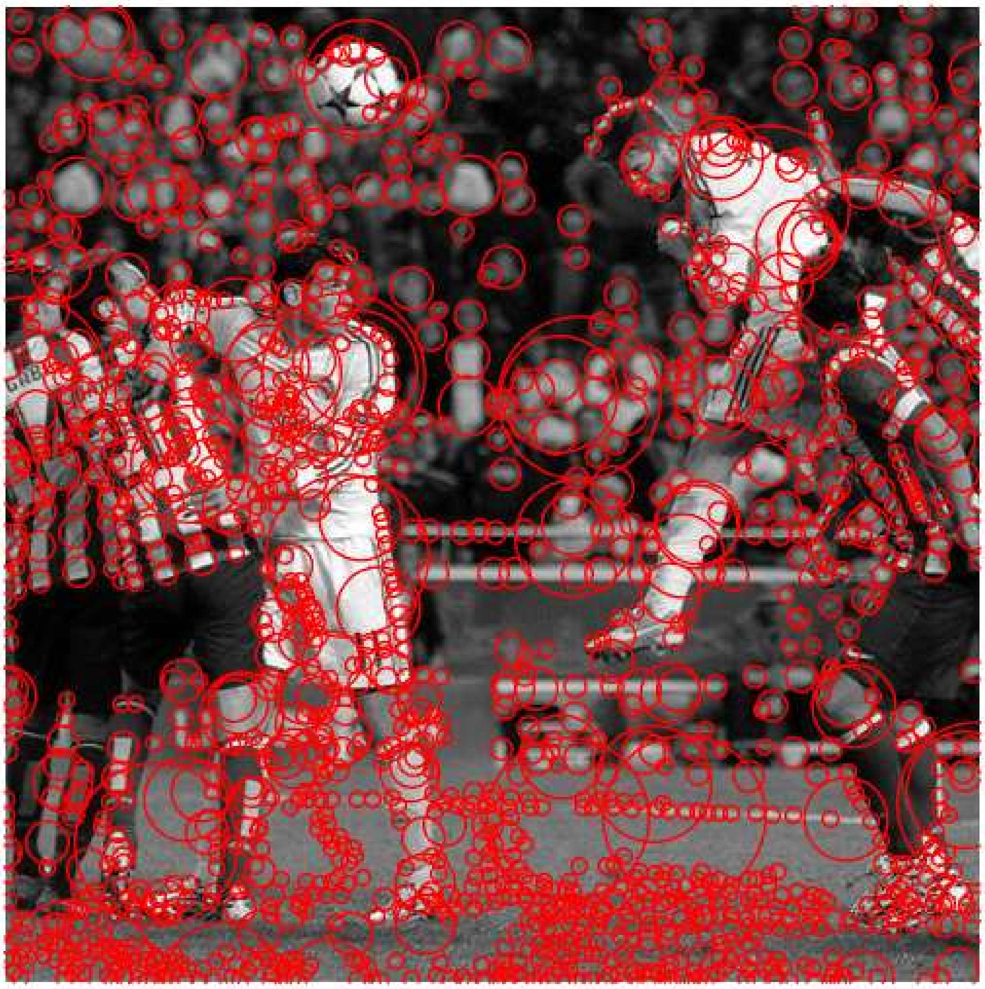
Elapsed time is 11.714062 seconds.

circles: 1914, threshold: 0.300, scale: 2.000, factor k: 1.400



Elapsed time is 2.822294

circles: 1436, threshold: 0.300, scale: 2.000, factor k: 1.400



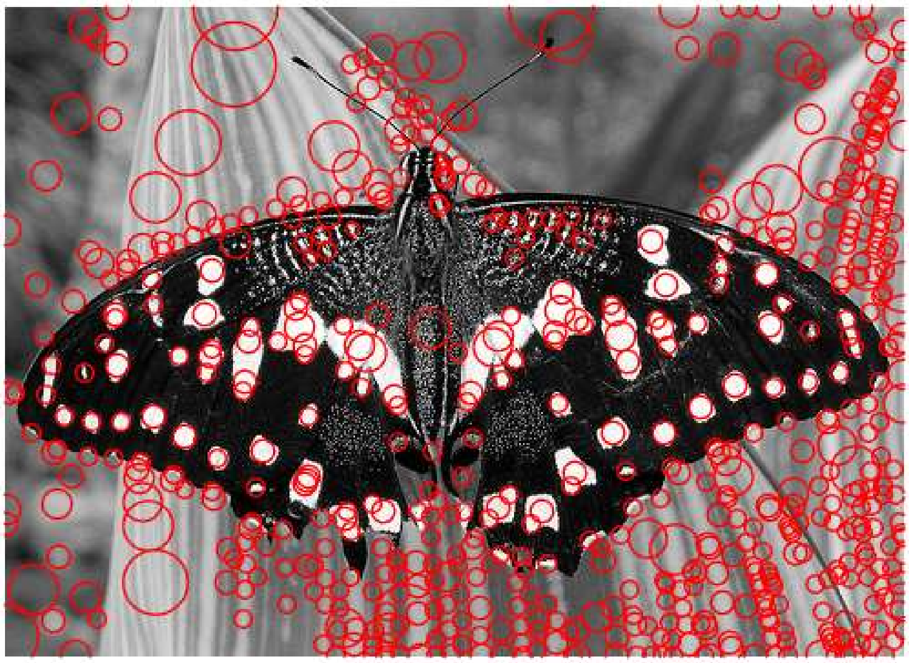
Elapsed time is 1.917761 seconds.

2. Laplacian blob detector: {Down Sampling}

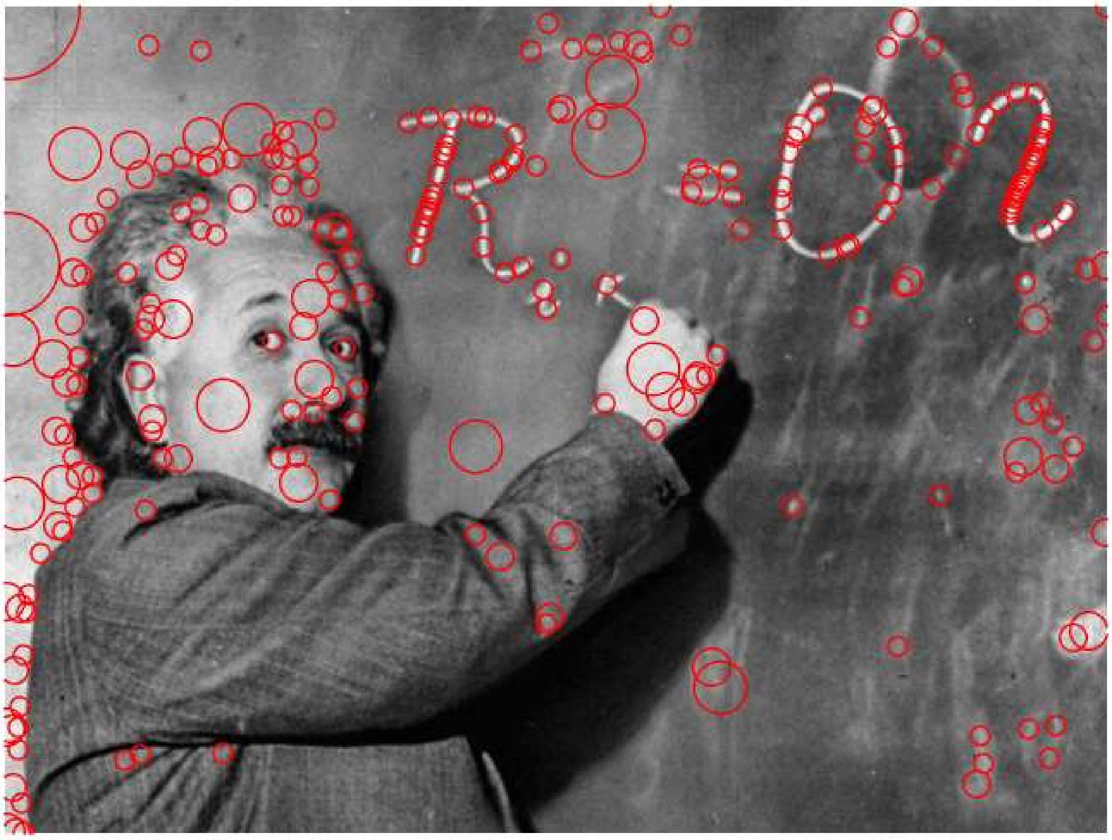
- ***Algorithm steps:***
  - Import image.
  - convert it to gray
  - convert it to double
- *Perform LoG filter to image for several levels*

```
tic
for i = 1:levels
    % Generate a Laplacian of Gaussian filter
% Filter the img with LoG
    % return the image to its original scale
    % Increase scale by a factor k
    % Downsample the img
end
toc
nonminimum suppression
for i = 1:levels
    % replaces each element by the orderth element in the sorted set of
    neighbors by ordfit2.
end

for i = 1:levels
    survive_space = scale_space.* img;
end
Find all the coordinates and corresponding sigma
for num = 1:levels
    % Find index and corresponding radius
end
show all circles
```

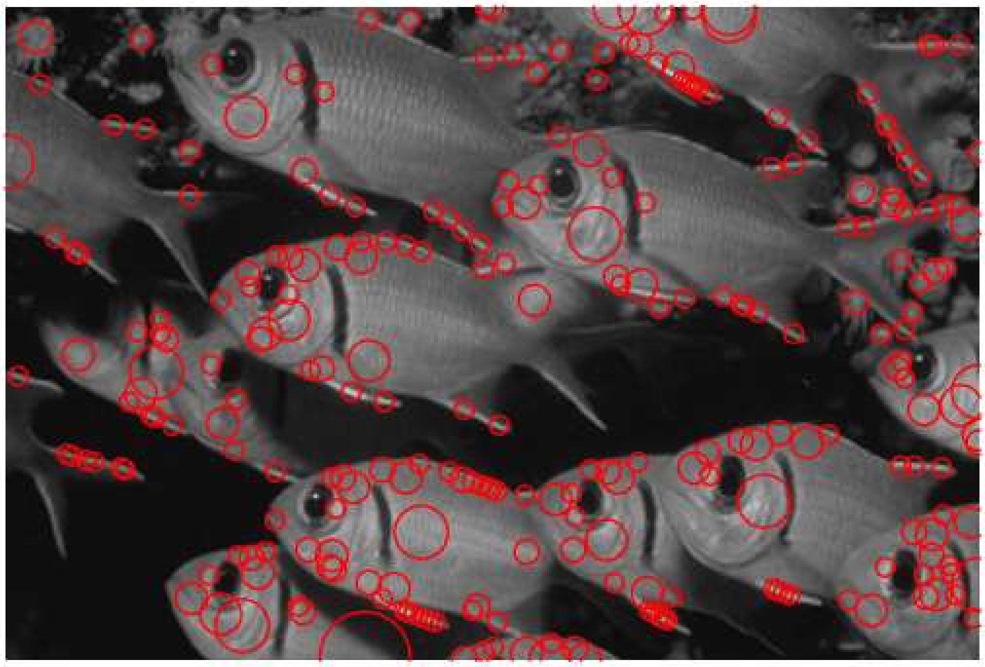
circles: 585, threshold: 0.300, scale: 2.000, factor k: 1.300



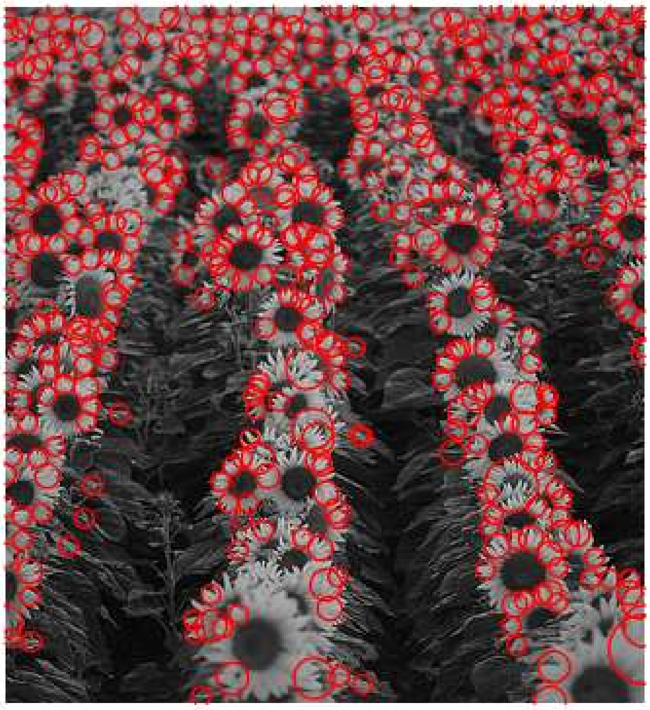
circles: 256, threshold: 0.500, scale: 2.000, factor k: 1.400



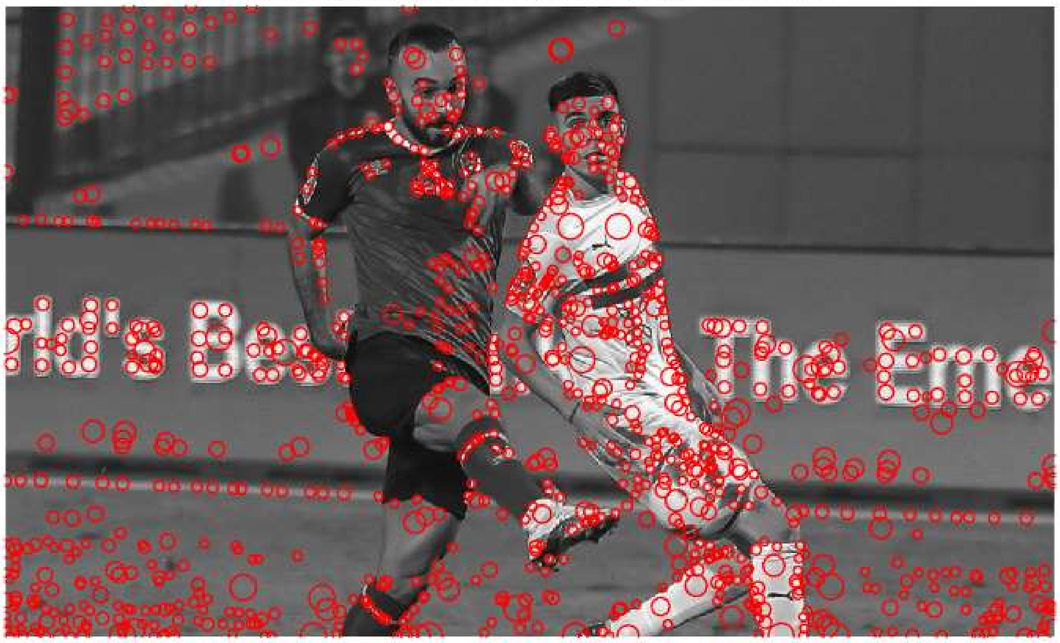
circles: 258, threshold: 0.400, scale: 2.000, factor k: 1.300



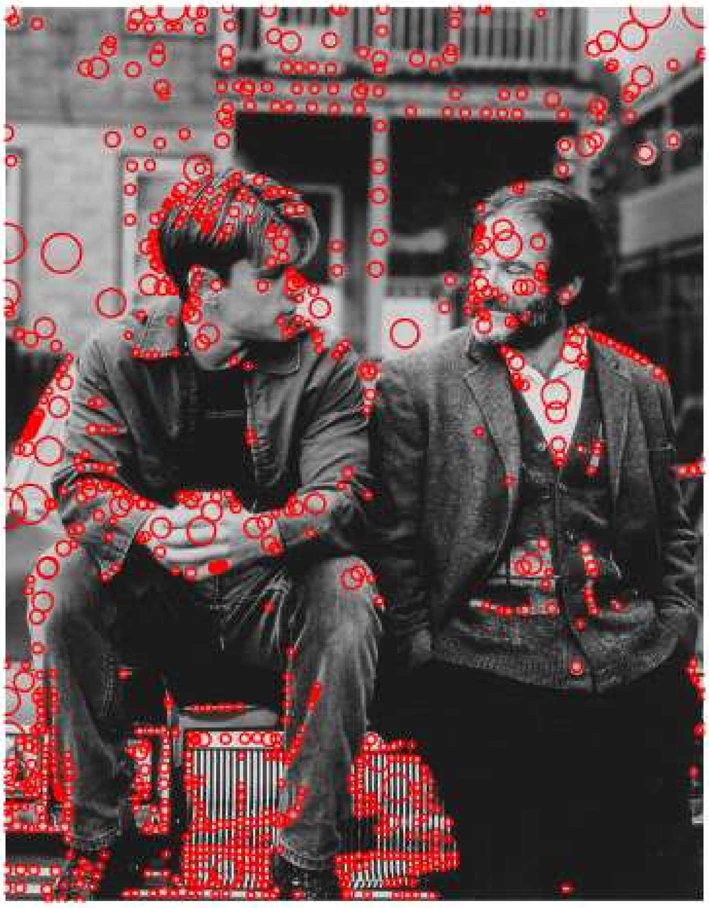
circles: 470, threshold: 0.400, scale: 2.000, factor k: 1.300



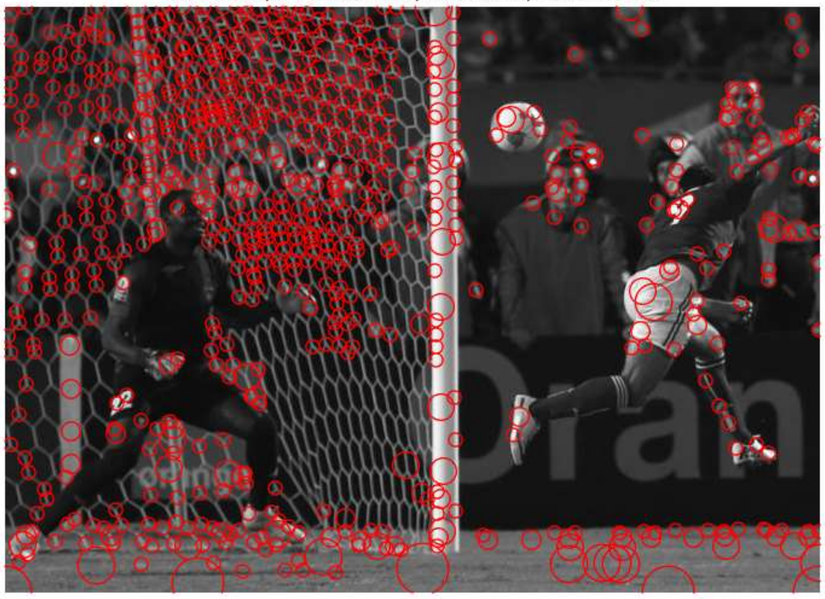
circles: 998, threshold: 0.400, scale: 2.000, factor k: 1.200



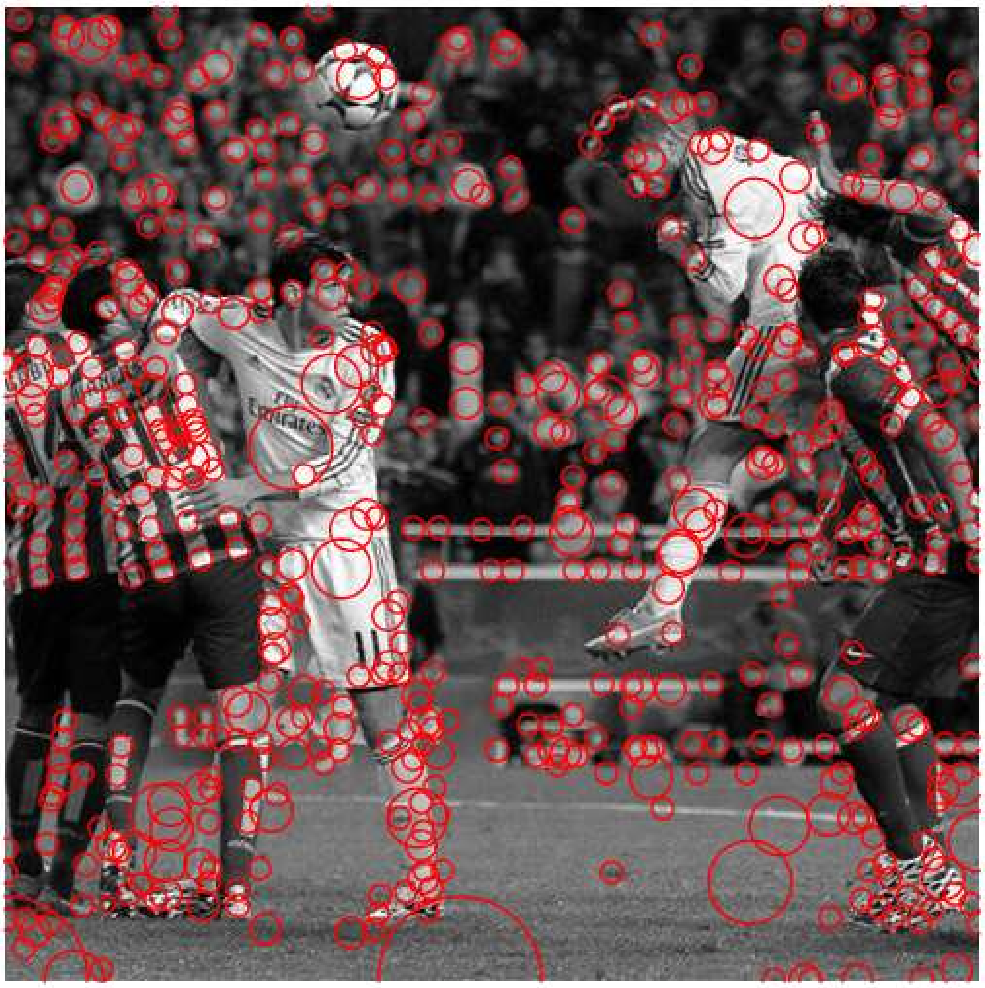
circles: 843, threshold: 0.600, scale: 2.000, factor k: 1.400



circles: 696, threshold: 0.300, scale: 2.000, factor k: 1.400



circles: 568, threshold: 0.300, scale: 2.000, factor k: 1.400

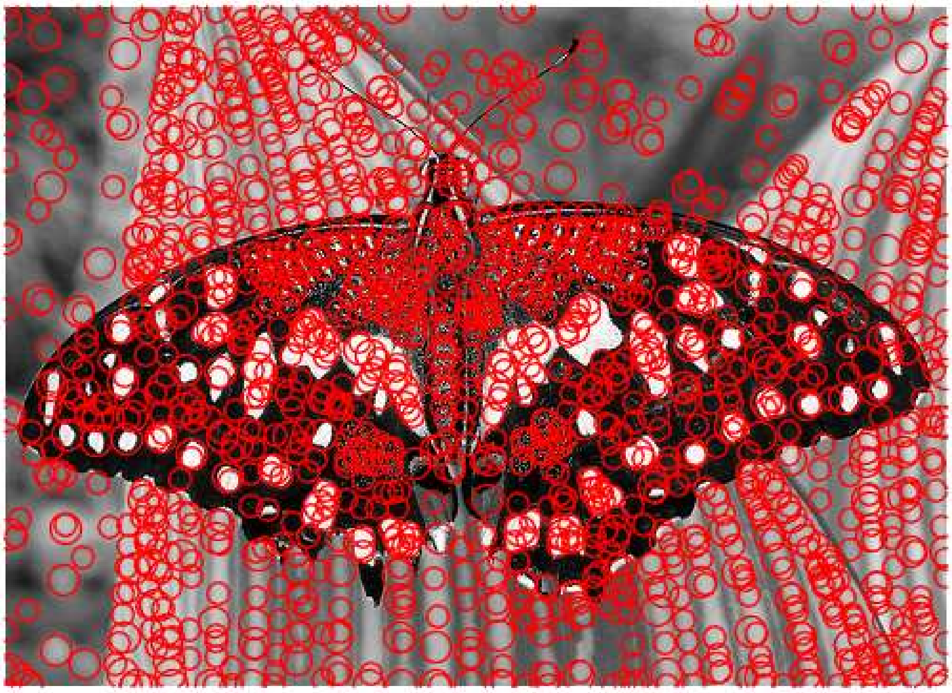


2. [BONUS POINT] Differences of Gaussian pyramid

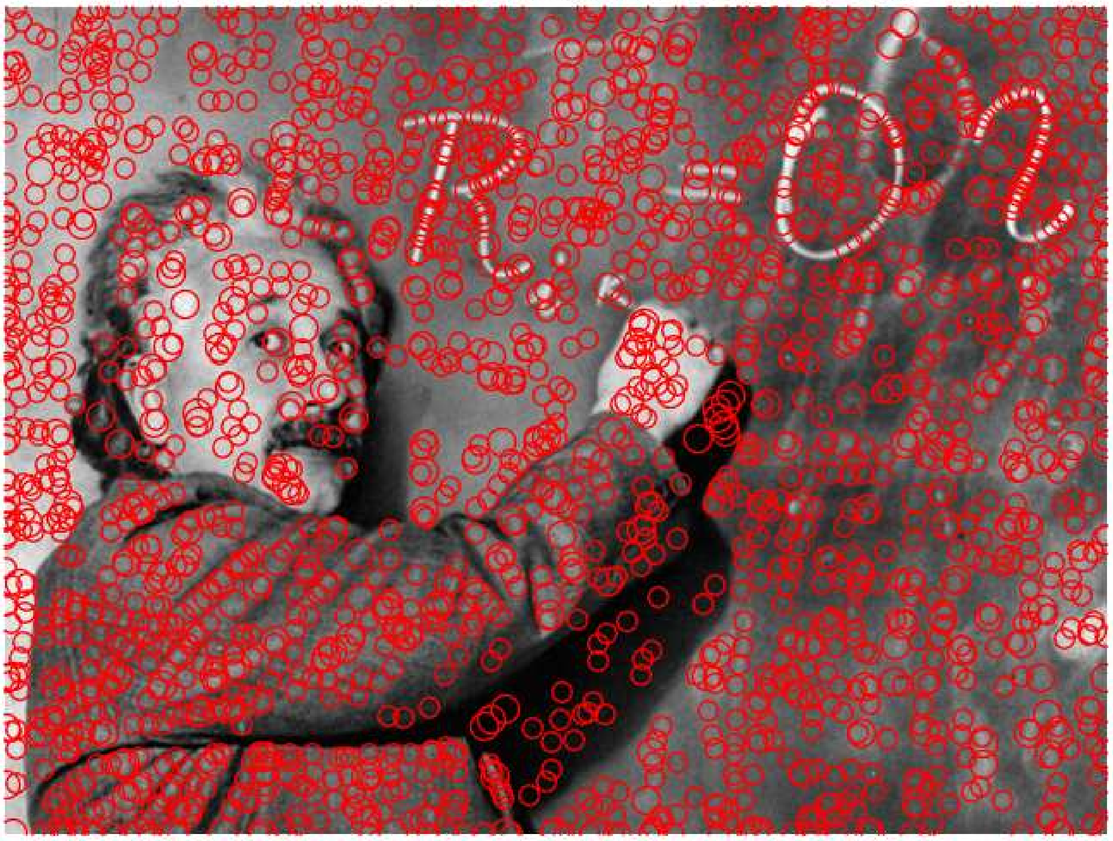
- ***Algorithm steps:***

```
o convert img to gray
o convert it to double
o Parameters initialization
First octave
o tic
o img_octave_1 = img;
o for i = 1:levels
o       Generate a Laplacian of Gaussian filter
o       Filter the img with LoG
o       Increase scale by a factor k
o end
o Subtract the previous octave
Second octave
o img_octave_2 = imresize(img, 0.5);
o for i = 1:levels
o End
o Subtract the previous octave
Third octave
o img_octave_3 = imresize(img_octave_2, 0.5);
o for i = 1:levels
o End

o Subtract the previous octave

o resize to original image

o Apply threshold in each 2D slice.

o Perform Nonmaximal Suppression in octaves

o show_all_circles
```

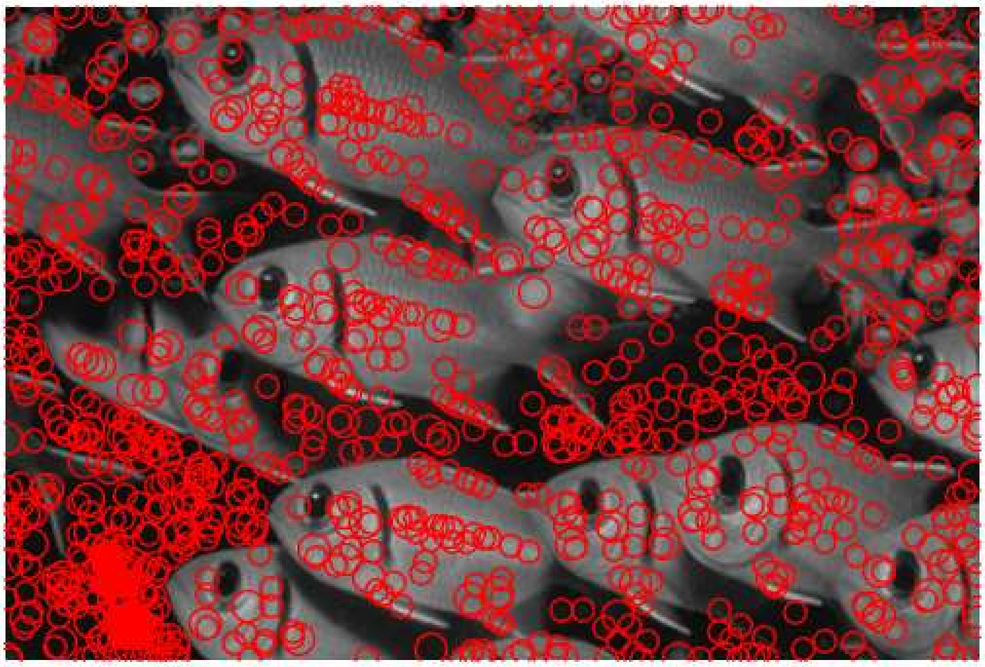
circles: 1416, threshold: 0.900, scale: 2.000, factor k: 1.414



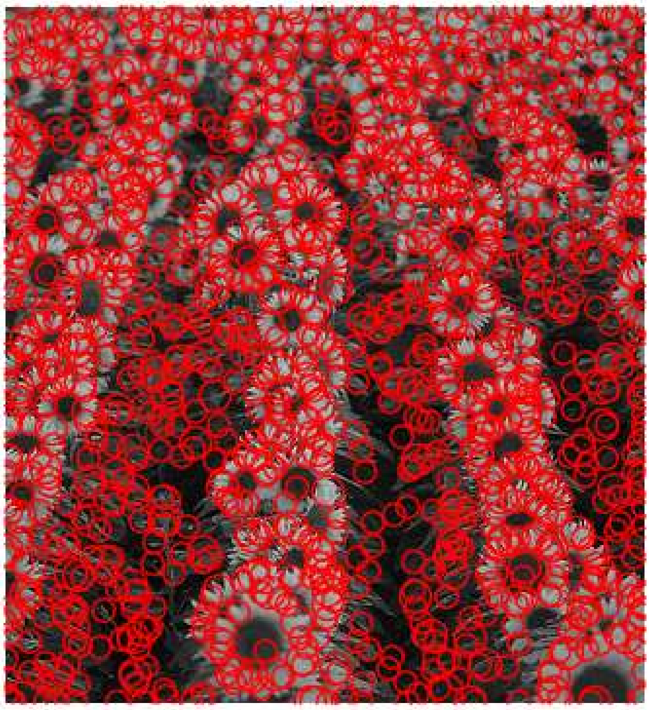
circles: 1819, threshold: 0.900, scale: 2.000, factor k: 1.414



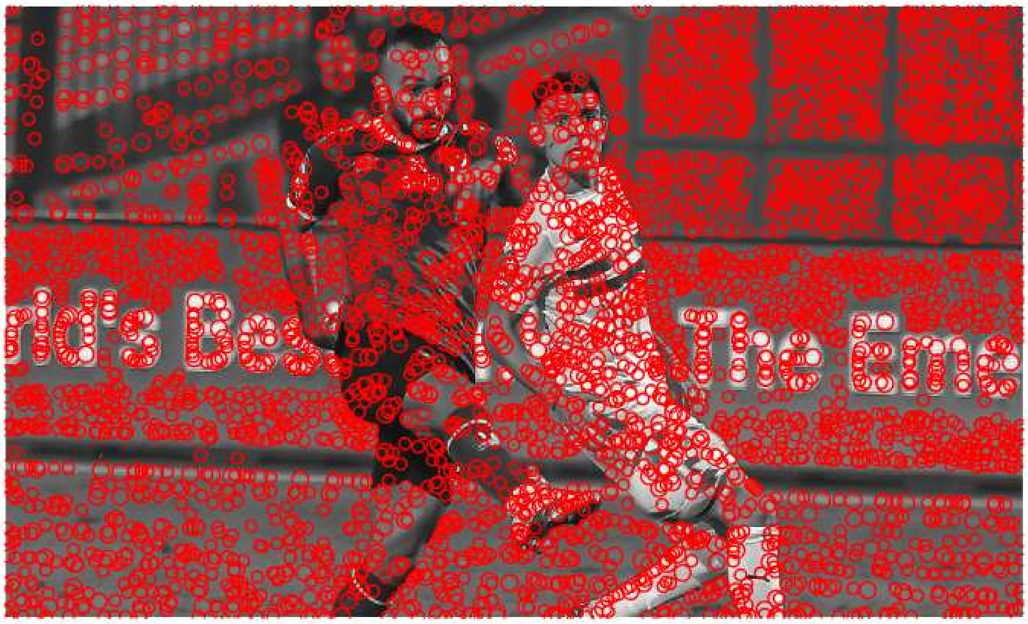
circles: 1097, threshold: 0.900, scale: 2.000, factor k: 1.414



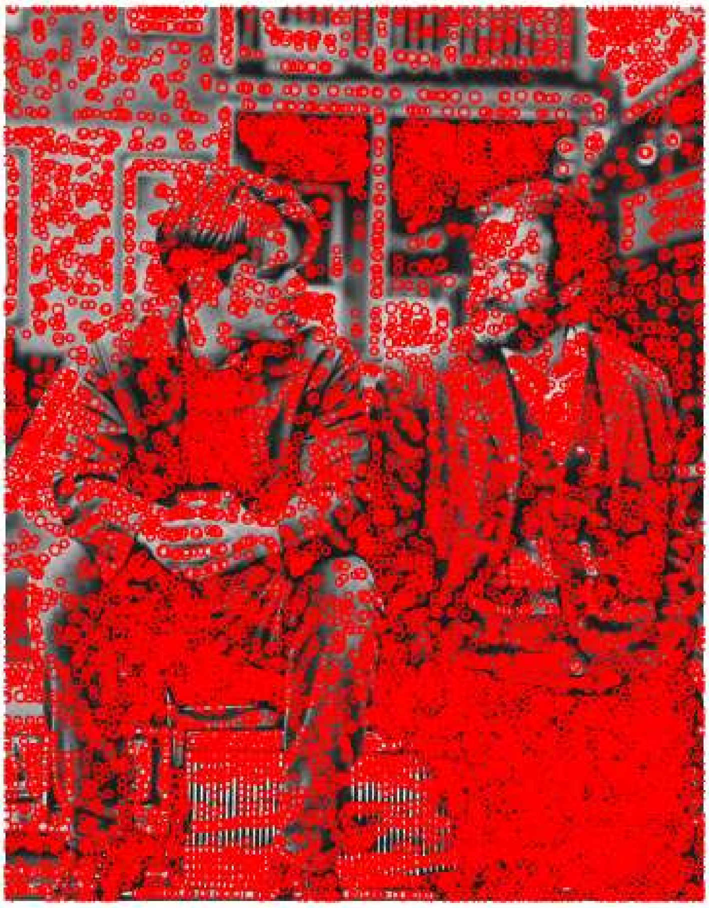
circles: 1149, threshold: 0.900, scale: 2.000, factor k: 1.414



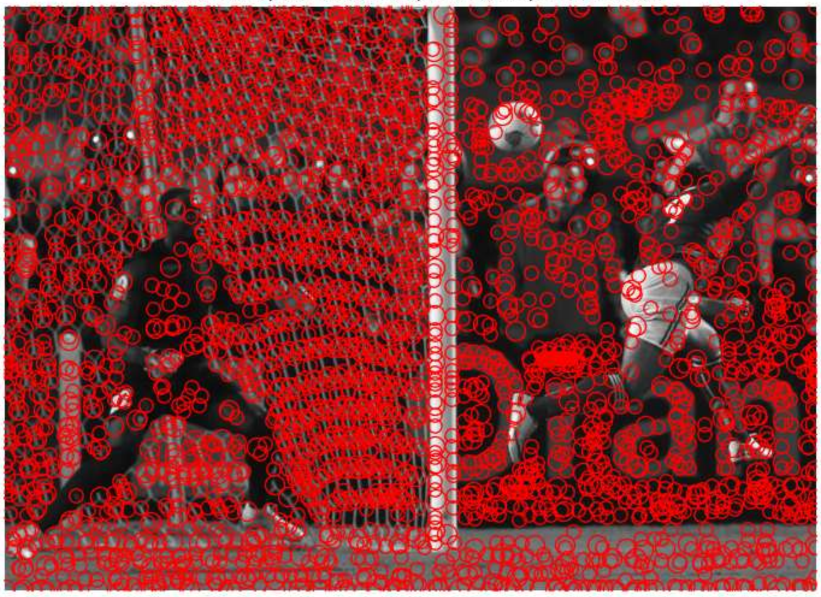
circles: 4084, threshold: 0.900, scale: 2.000, factor k: 1.414



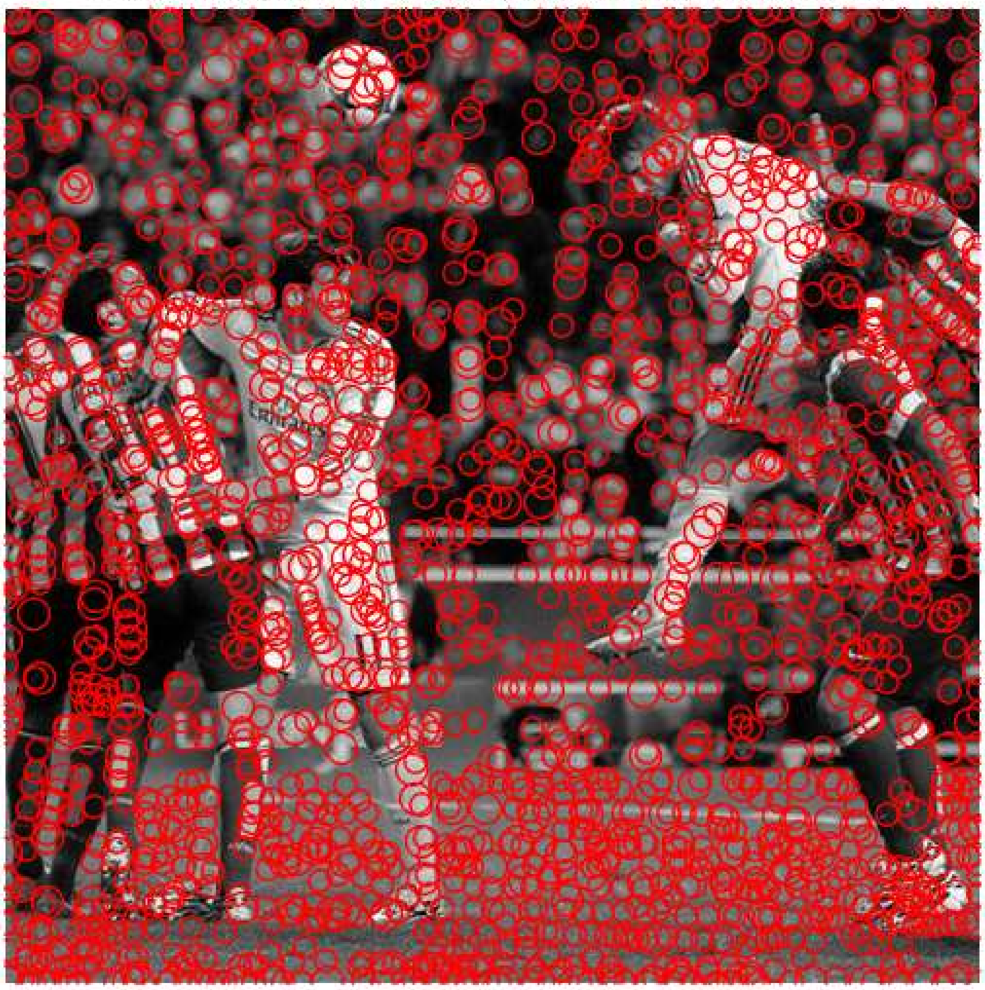
circles: 10048, threshold: 0.900, scale: 2.000, factor k: 1.414



circles: 2809, threshold: 0.900, scale: 2.000, factor k: 1.414



circles: 1723, threshold: 0.900, scale: 2.000, factor k: 1.414



6- Conclusion:

We have explored how different the blob detection methods — such as the Laplacian of Gaussian (LoG), Difference of Gaussian (DoG), and Determinant of Hessian (DoH) — can be used to detect, label, and measure objects in our image. We have also tackled the importance of preprocessing the image before conducting such blob detection methods. Now, it should be noted that specific applications will require specific blob detection methods, and it is upon the discretion of the data scientist to choose the proper techniques. With this, I hope the findings raised in this blog can help you select the most appropriate blob detection method in your projects.

we have presented a multi-scale representation of grey-level image. It can be used for extraction of important regions from a image in, without any a priori assumptions about the shape of the primitives. The representation, which is essentially to tuning parameters and, gives a qualitative description of the grey-level landscape with information about approximate location, we have demonstrated how such information can serve as a guide to an edge detection scheme working at a Laplacian blob detector.

7- References:

1- https://en.wikipedia.org/wiki/Blob_detection
2- https://cs.gmu.edu/~kosecka/cs482/cs482-blob.pdf
3- https://link.springer.com/article/10.1007%2Fs10851-012-0378-3