haoyuan9_mp2_part2_code

March 4, 2019

```
In [53]: import cv2
         import numpy as np
         from scipy.ndimage.filters import gaussian_laplace, rank_filter
         from scipy.ndimage import gaussian_filter
         import skimage
         import matplotlib.pyplot as plt
         import time
         %matplotlib inline

In [54]: def show_all_circles(image, fname, cx, cy, rad, color='r'):
             """
             image: numpy array, representing the grayscsale image
             cx, cy: numpy arrays or lists, centers of the detected blobs
             rad: numpy array or list, radius of the detected blobs
             """
             import matplotlib.pyplot as plt
             from matplotlib.patches import Circle

             fig, ax = plt.subplots()
             ax.set_aspect('equal')
             ax.imshow(image, cmap='gray')
             for x, y, r in zip(cx, cy, rad):
                 circ = Circle((x, y), r, color=color, fill=False)
                 ax.add_patch(circ)

             plt.title('%i circles' % len(cx))
             plt.savefig(fname + "_blob.png")
             #plt.imsave(fname + "_blob.gif", image)
             plt.show()

In [55]: def get_scale_space(input, num_sigma, min_sigma, sigma_ratio, octave_size, method="do

             x_dim = input.shape[0]
             y_dim = input.shape[1]
             scale_space = np.empty((x_dim, y_dim, num_sigma))

             if method == "normal":
```

```
        for i in range(num_sigma):
            sig = min_sigma * np.power(sigma_ratio, i)
            scale_space[:, :, i] = (sig * gaussian_laplace(input, sigma=sig, mode="nea
#             plt.figure("Sigma=%f" % sig)
#             plt.imshow(scale_space[:, :, i], cmap="gray")
#             plt.show()
    elif method =="downsample":
        for i in range(num_sigma):
            sig = min_sigma * np.power(sigma_ratio, i % 3)
            if i % 3 == 0 and i != 0:
                input = skimage.transform.resize(input, (input.shape[0] // 2, input.sh
                intermidiate = (gaussian_laplace(input, sigma=sig, mode="nearest")) **
            else:
                intermidiate = (sig * gaussian_laplace(input, sigma=sig, mode="nearest
            if i >= 3:
                scale_space[:, :, i] = skimage.transform.resize(intermidiate, (x_dim,
            else:
                scale_space[:, :, i] = intermidiate
    elif method == "dog":

#         Correct but slow
#
#         sigma_list = np.array([min_sigma * (sigma_ratio ** i)
#                                for i in range(num_sigma + 1)])
#         gaussian_images = [gaussian_filter(input, s) for s in sigma_list]
#         for i in range(num_sigma):
#             scale_space[:, :, i] = (gaussian_images[i] - gaussian_images[i+1]) * si

#         Faster but the result is not good

        cur_sigma = min_sigma
        gaussian_image = input
        cur_gaussian = gaussian_filter(gaussian_image, cur_sigma)
        pre_gaussian = None

        for i in range(num_sigma):
            pre_gaussian = cur_gaussian
            cur_sigma = (sigma_ratio - 1) * cur_sigma
            cur_gaussian = gaussian_filter(pre_gaussian, cur_sigma)
            scale_space[:, :, i] = (pre_gaussian - cur_gaussian)
            cur_sigma = cur_sigma * sigma_ratio / (sigma_ratio - 1)

#         The approach exactly the same as the paper, but didn't work
#
#         sigma_ratio = np.power(2, (1.0 / octave_size))
#         gaussian_image = input
#         cur_sigma = min_sigma
#         cur_gaussian = gaussian_filter(gaussian_image, cur_sigma)
#         pre_gaussian = None
#         for i in range(num_sigma):
#             if i != 0 and i % octave_size == 0:
#                 gaussian_image = skimage.transform.resize(gaussian_image, (gaussian
#                                         // 2, gaussian_image.shape[1] // 2), anti_a
#                 cur_sigma = min_sigma
#                 cur_gaussian = gaussian_filter(gaussian_image, cur_sigma)

#             pre_gaussian = cur_gaussian
#             cur_sigma = sigma_ratio * cur_sigma
#             cur_gaussian = gaussian_filter(gaussian_image, cur_sigma)
#             scale_space[:, :, i] = skimage.transform.resize(cur_gaussian - pre_gaus

    else:
        print("Error in method: No %s" % method)

    return scale_space
```

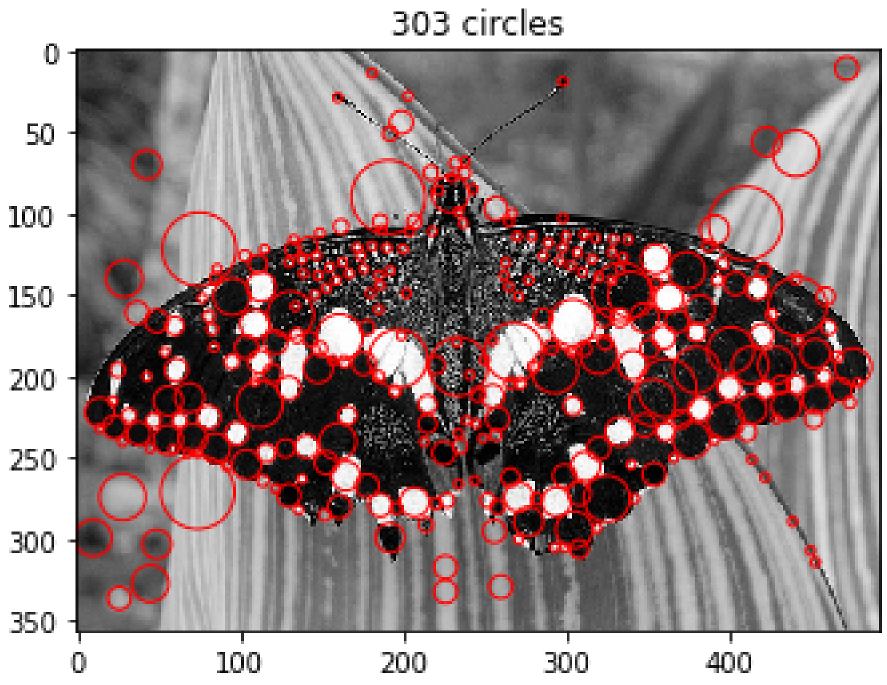
```
In [56]: def non_maximum_suppression(scale_space, num_sigma, threshold, nms_size, exclude_borde

             # Get the local maxima in the across all scales with size(nms_size, nms_size)
             local_max = rank_filter(scale_space, rank=-1, size=(nms_size, nms_size, num_sigma
             local_max[local_max != scale_space] = 0

             # Eliminate maxima near to the border
             if exclude_border:
                 local_max[:exclude_border, :, :] = local_max[-exclude_border:, :, :] = 0
                 local_max[:, :exclude_border, :] = local_max[:, -exclude_border:, :] = 0

             nonzero_element = np.where(local_max > threshold)

             return nonzero_element
```

```
In [57]: def blob_detection(input, fname, num_sigma=10, min_sigma=2, sigma_ratio=1.2599, octave
                            threshold=0.01, nms_size=10, exclude_border=10):

             print("<---------------------Blob Detection-------------------->")
             start = time.time_ns()
             scale_space = get_scale_space(input, num_sigma, min_sigma, sigma_ratio, octave_si
             end = time.time_ns()
             print("Implementation: %s Time:: %d" % (method, (end - start)))
             print("<------------------------------------------------------>")

             nonzero_element = non_maximum_suppression(scale_space, num_sigma, threshold, nms_

             # Transform sigma into radius
```

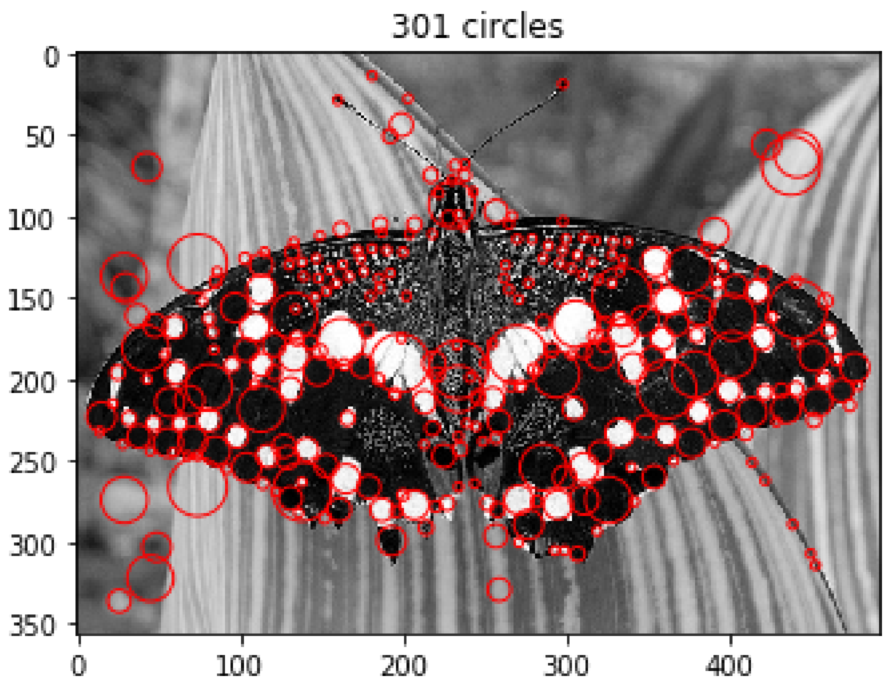
```
            radius = np.empty(nonzero_element[2].shape)
            for i in range(num_sigma):
                radius[nonzero_element[2]==i] = np.sqrt(2) * min_sigma * np.power(sigma_ratio
            center = nonzero_element[:2]
            show_all_circles(input, fname, center[1], center[0], radius)
```

```
In [67]: k = 1.2599
         num_sigma = 10
         min_sigma = 2
         threshold = 0.02
         nms_size = 10

         butterfly = cv2.imread("butterfly.jpg", 0).astype("float") / 255
         blob_detection(butterfly, "butterfly", num_sigma, method="normal", threshold=threshol
         blob_detection(butterfly, "butterfly", num_sigma, method="downsample", threshold=thres
```

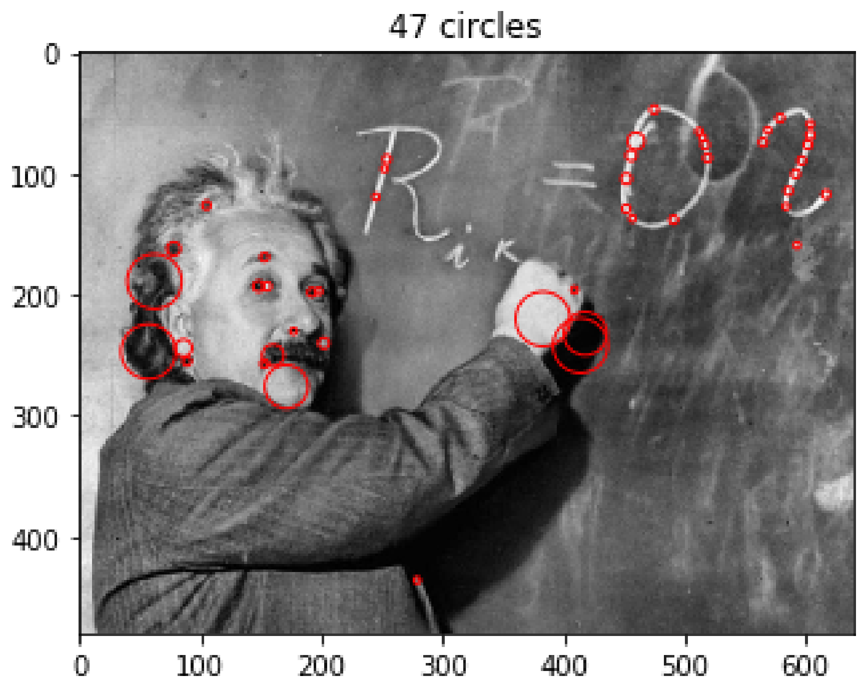
```
<--------------------Blob Detection-------------------->
Implementation: normal Time:: 253553100
<------------------------------------------------------>
```



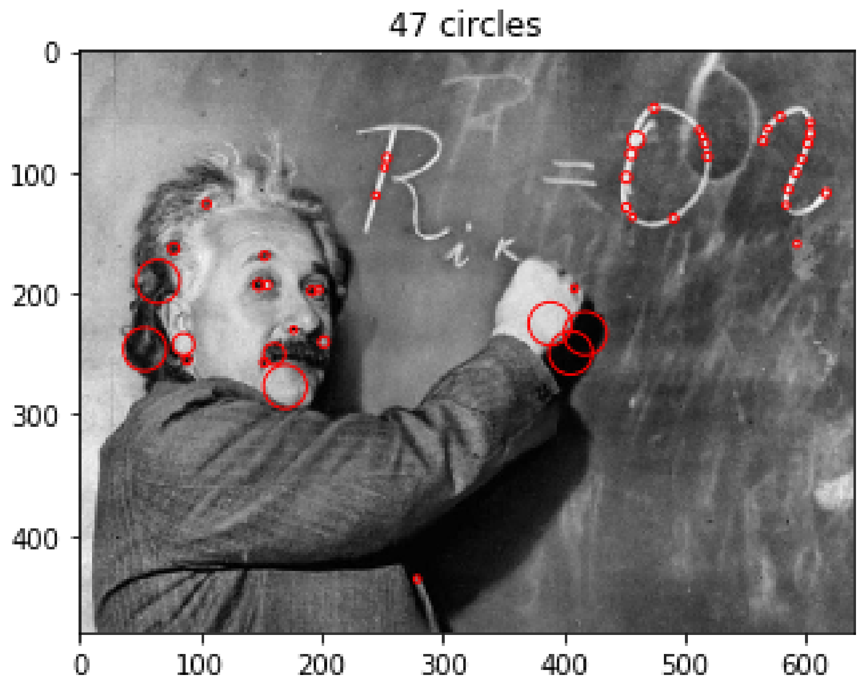
```
<--------------------Blob Detection-------------------->
Implementation: downsample Time:: 134346600
<------------------------------------------------------>
```



```
In [68]: einstein = cv2.imread("einstein.jpg", 0).astype("float") / 255
         blob_detection(einstein, "einstein", num_sigma, method="normal", threshold=0.05, nms_s
         blob_detection(einstein, "einstein", num_sigma, method="downsample", threshold=0.05, n

<--------------------Blob Detection-------------------->
Implementation: normal Time:: 452309000
<------------------------------------------------------>
```

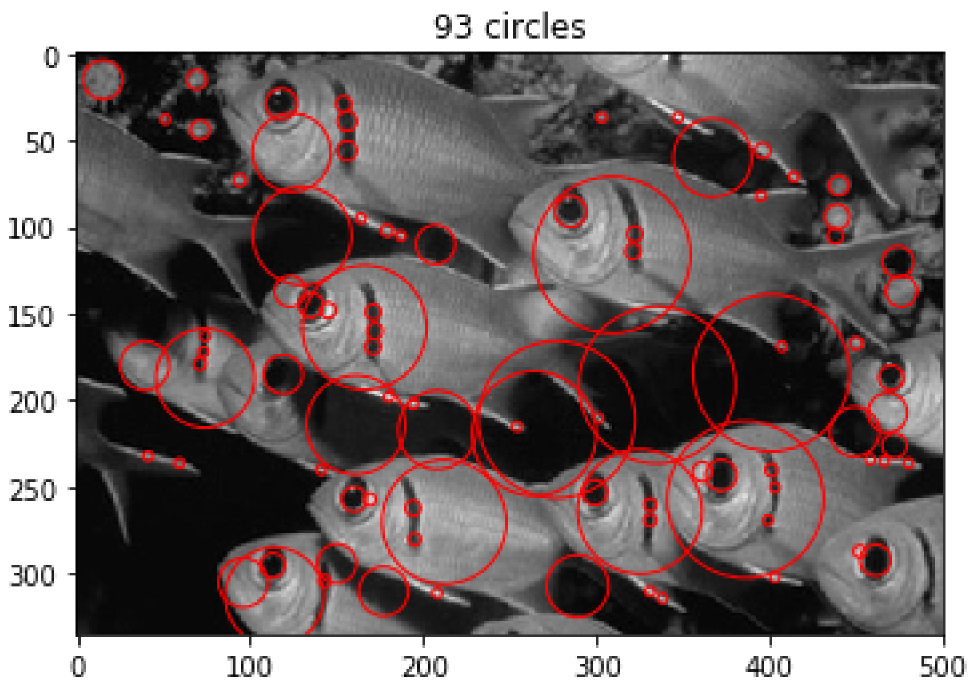


```
<--------------------Blob Detection-------------------->
Implementation: downsample Time:: 220807500
<------------------------------------------------------>
```

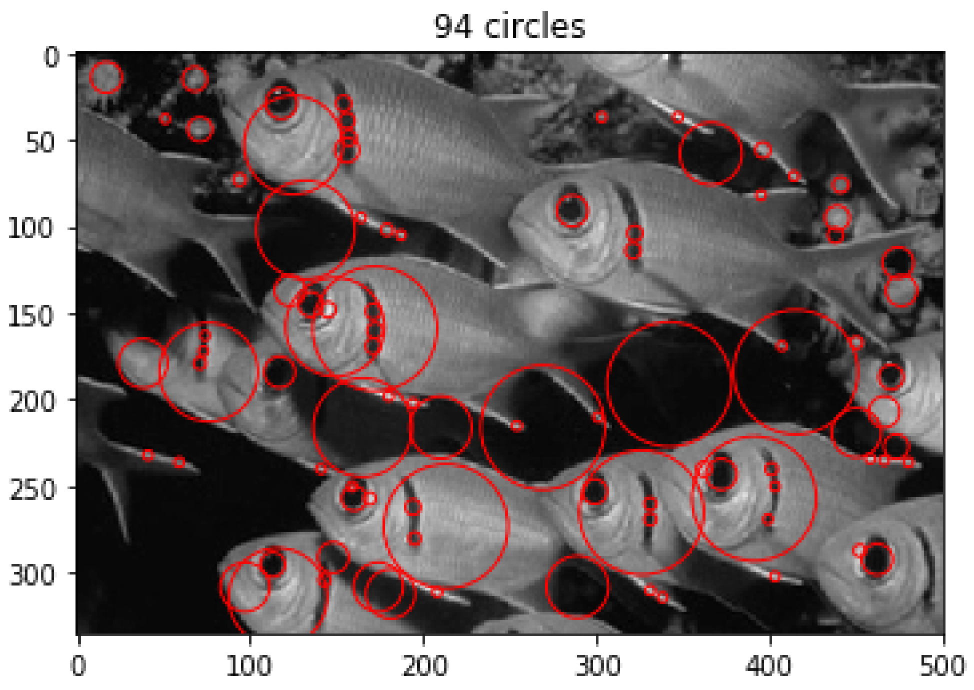


```
In [71]: fishes = cv2.imread("fishes.jpg", 0).astype("float") / 255
         blob_detection(fishes, "fishes", num_sigma=13, method="normal", threshold=threshold, r
         blob_detection(fishes, "fishes", num_sigma=13, method="downsample", threshold=threshol
```

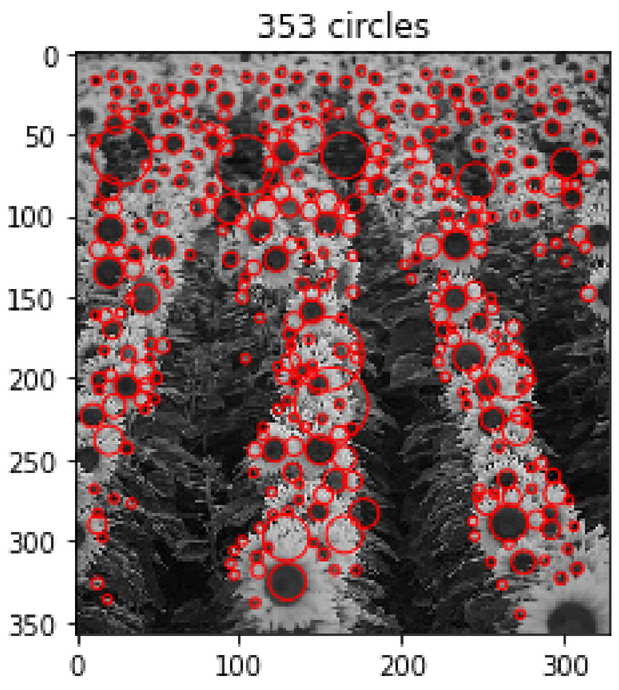
```
<--------------------Blob Detection-------------------->
Implementation: normal Time:: 489198600
<------------------------------------------------------>
```



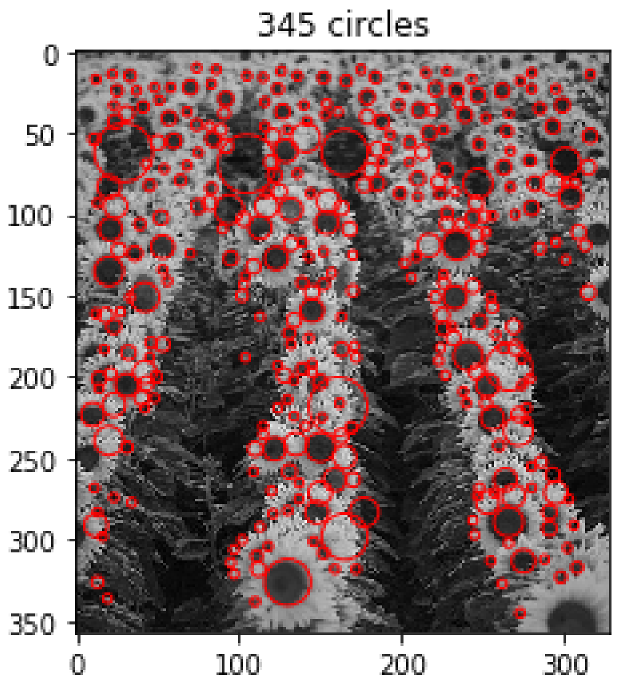
```
<--------------------Blob Detection-------------------->
Implementation: downsample Time:: 185093800
<------------------------------------------------------>
```



```
In [61]: sunflowers = cv2.imread("sunflowers.jpg", 0).astype("float") / 255
         blob_detection(sunflowers, "sunflowers", num_sigma, method="normal", threshold=thresho
         blob_detection(sunflowers, "sunflowers", num_sigma, method="downsample", threshold=thr

<--------------------Blob Detection-------------------->
Implementation: normal Time:: 211793400
<------------------------------------------------------>
```

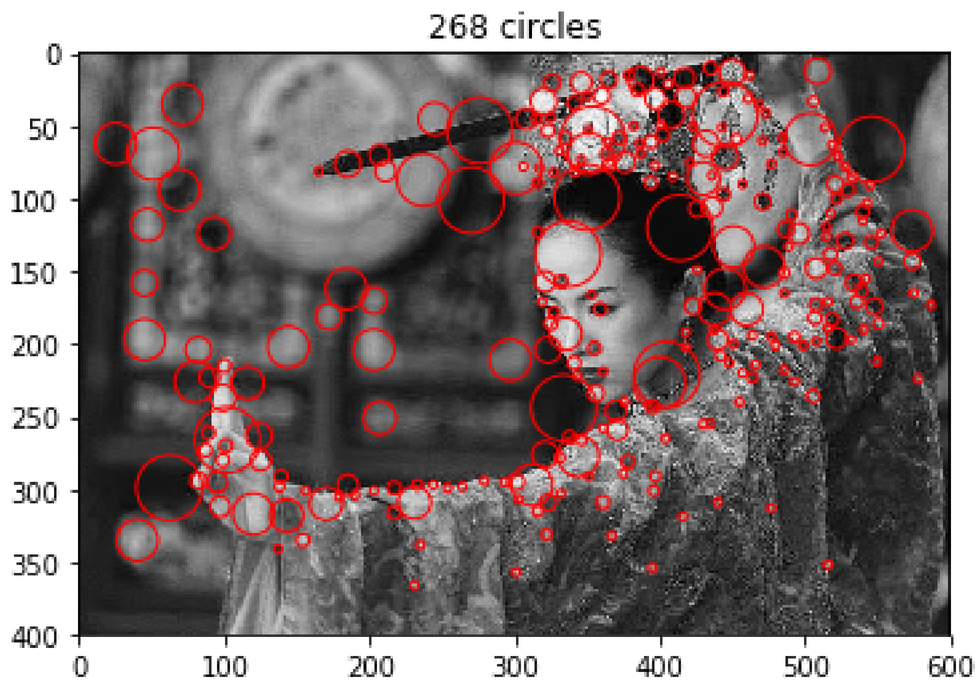


```
<--------------------Blob Detection-------------------->
Implementation: downsample Time:: 134354100
<------------------------------------------------------>
```

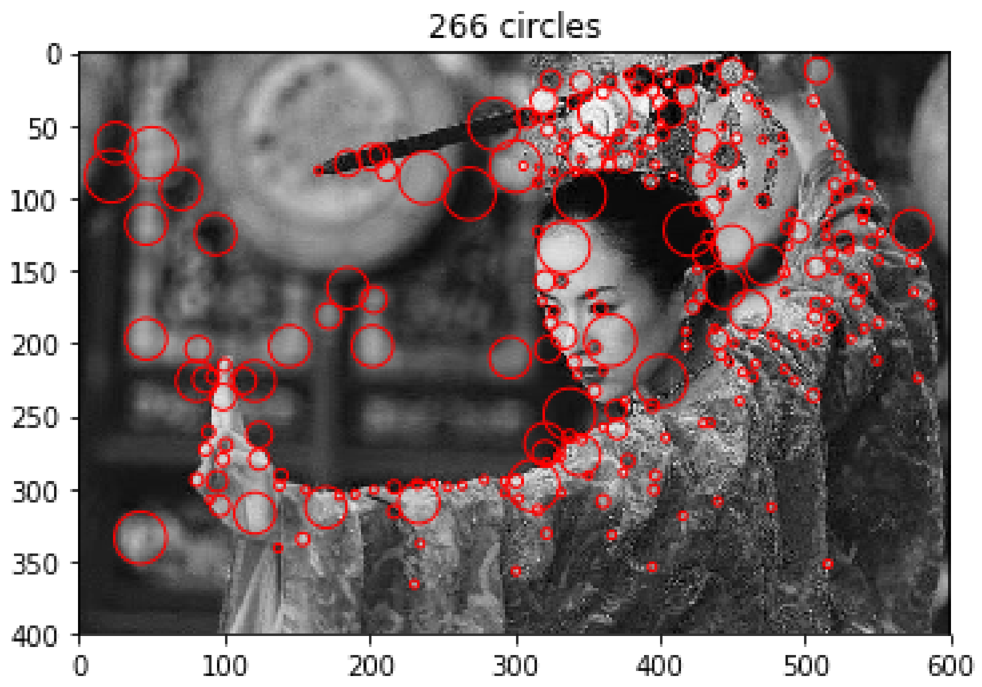


```
In [62]: beauty = cv2.imread("beauty.jpg", 0).astype("float") / 255
         blob_detection(beauty, "beauty", num_sigma, method="normal", threshold=threshold, nms_
         blob_detection(beauty, "beauty", num_sigma, method="downsample", threshold=threshold,
```

```
<--------------------Blob Detection-------------------->
Implementation: normal Time:: 419424400
<------------------------------------------------------>
```

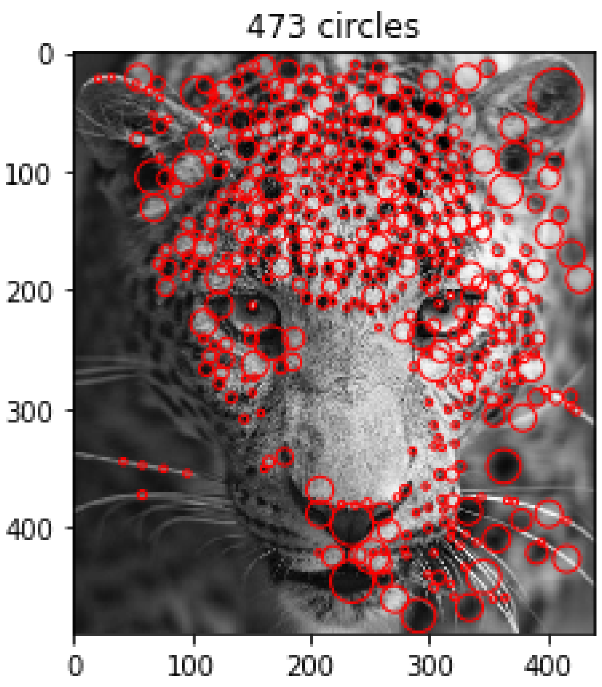


```
<--------------------Blob Detection-------------------->
Implementation: downsample Time:: 236427800
<------------------------------------------------------>
```

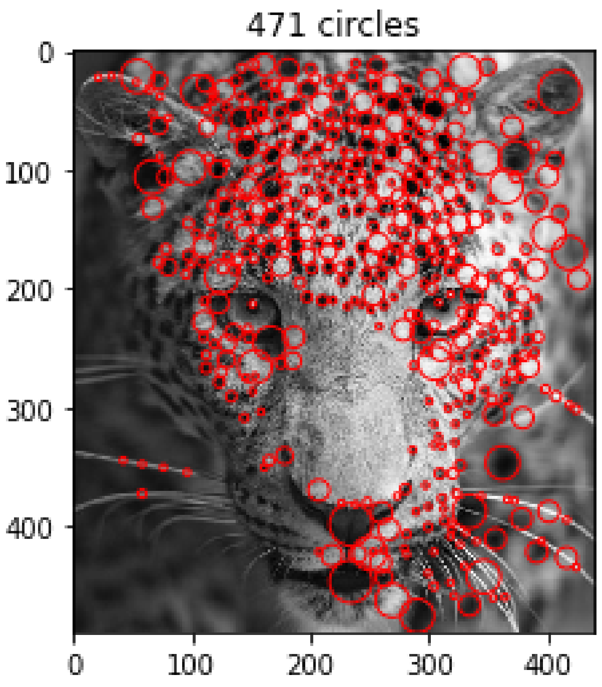


```
In [74]: leopard = cv2.imread("leopard.jpg", 0).astype("float") / 255
         blob_detection(leopard, "leopard", num_sigma=10, method="normal", threshold=threshold
         blob_detection(leopard, "leopard", num_sigma=10, method="downsample", threshold=thresh
```

```
<--------------------Blob Detection-------------------->
Implementation: normal Time:: 317149800
<------------------------------------------------------>
```

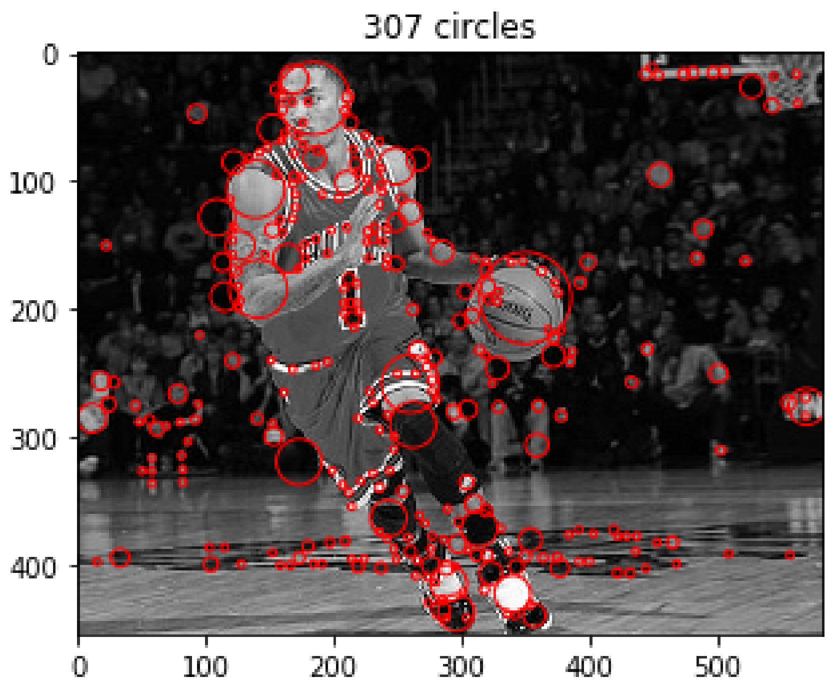


```
<--------------------Blob Detection------------------->
Implementation: downsample Time:: 153590100
<----------------------------------------------------->
```

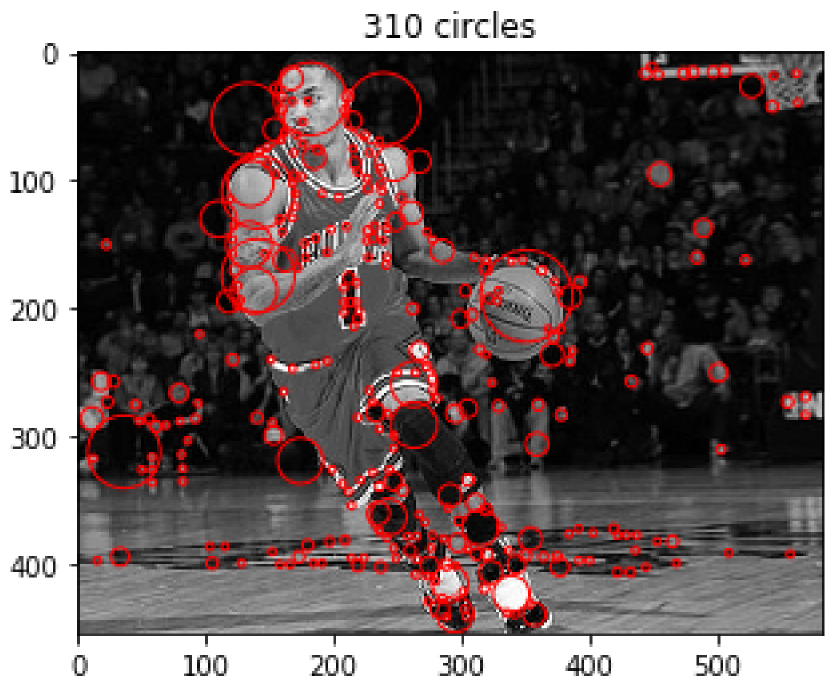


```
In [79]: rose = cv2.imread("rose.jpg", 0).astype("float") / 255
         blob_detection(rose, "rose", num_sigma=12, method="normal", threshold=threshold, nms_s
         blob_detection(rose, "rose", num_sigma=12, method="downsample", threshold=threshold, n
```

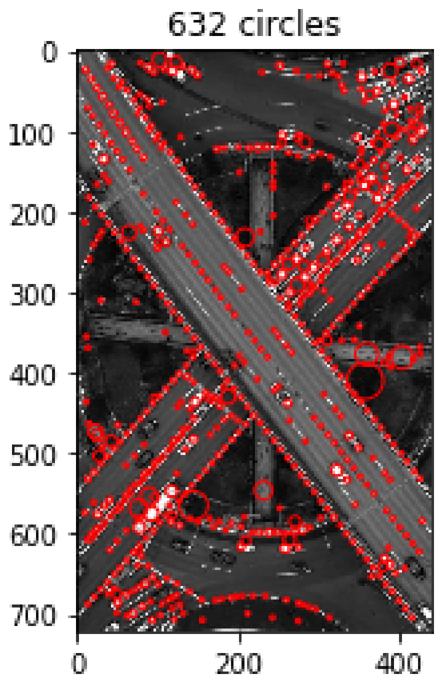
```
<--------------------Blob Detection-------------------->
Implementation: normal Time:: 605415100
<------------------------------------------------------>
```



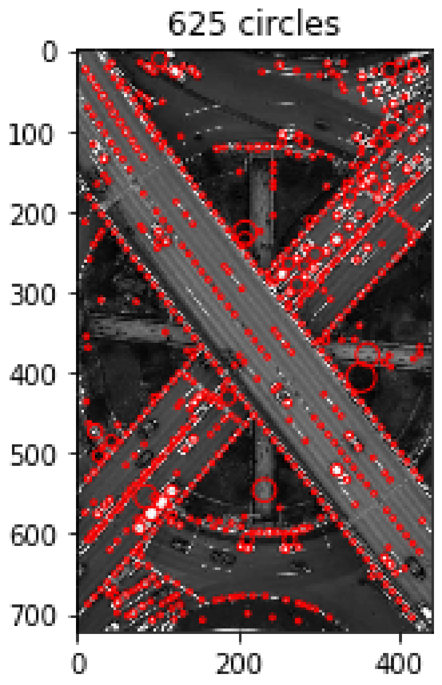
```
<--------------------Blob Detection-------------------->
Implementation: downsample Time:: 220403200
<------------------------------------------------------>
```



```
In [65]: bridge = cv2.imread("bridge.jpg", 0).astype("float") / 255
         blob_detection(bridge, "bridge", num_sigma, method="normal", threshold=threshold, nms_
         blob_detection(bridge, "bridge", num_sigma, method="downsample", threshold=threshold,

<--------------------Blob Detection-------------------->
Implementation: normal Time:: 558379900
<------------------------------------------------------->
```



```
<--------------------Blob Detection-------------------->
Implementation: downsample Time:: 308197800
<------------------------------------------------------>
```



```
In [ ]:
```