

Report

Part 1:

1. The original, filtered, and hybrid images for all 3 examples :

Original:



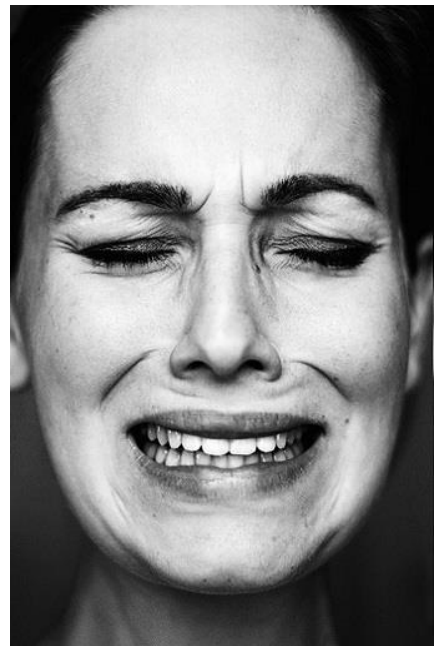
Filtered:



Hybrid:



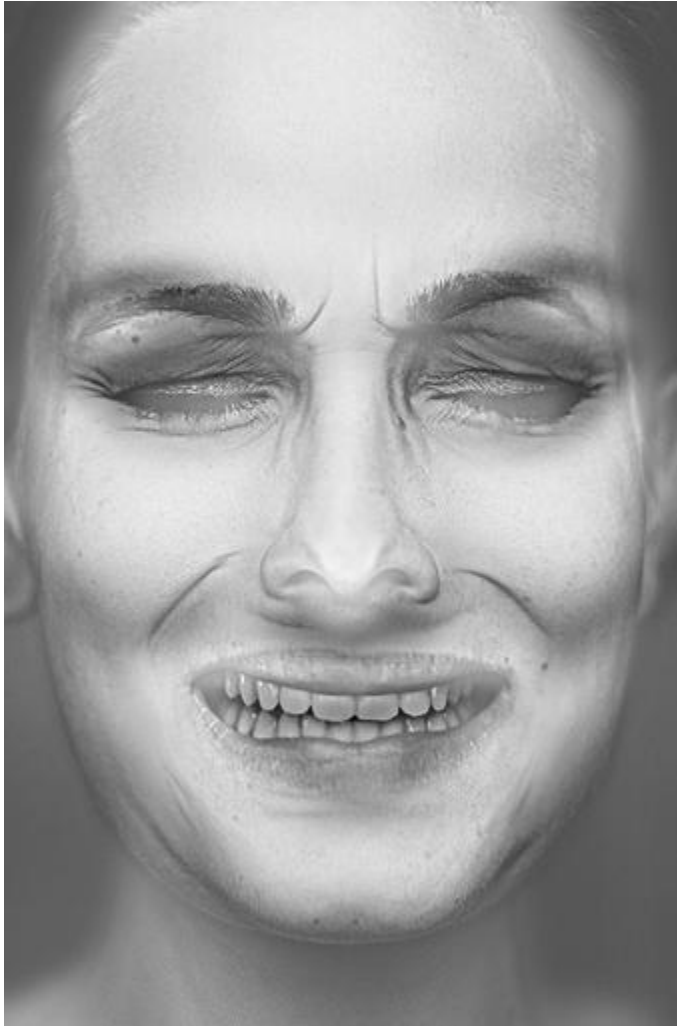
Original:



Filtered:



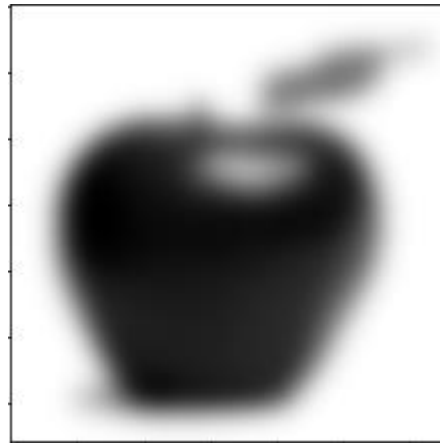
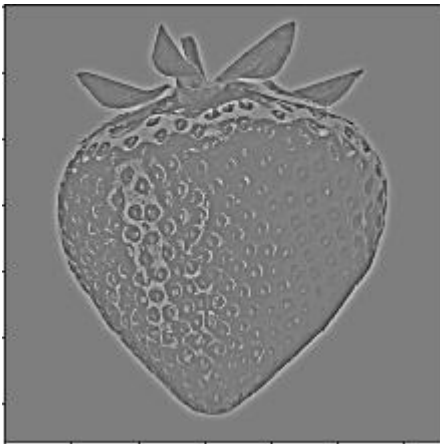
Hybrid:



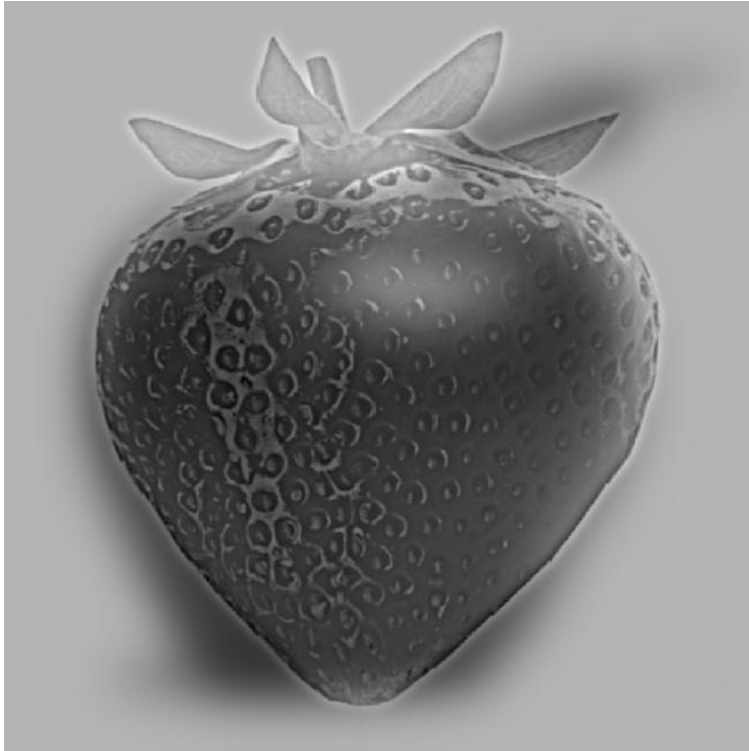
Original:



Filtered:



Hybrid:



2. Explanation of any implementation choices such as library functions or changes to speed up computation.

① When reading raw color image, the image was first read as grayscale, and converted the datatype into float64.

```
Img1 = np.float64(cv2.imread('apple.jpg', 0))
```

The underlying purpose is for the preservation of accuracy of the data during the image filter operation. Original unit8 datatype tends to lose data precision when dealing with decimal results, leading to irregular 'stains' on the processed image.

② Gaussian filter was adopted as low-pass filter:

```
from scipy.ndimage import gaussian_filter  
a1 = gaussian_filter(img1, sigma=5)
```

The high-pass filter was implemented by subtracting Gaussian filtered image from the original one.

```
img2 = np.float64(cv2.imread('apple.jpg', 0))  
a2 = gaussian_filter(img2, sigma=20)  
hf = img2 - a2
```

③ final image is derived by the average of the low-pass and high-pass filtered images, and rescaled to 0~255:

```
h = (img1 - a1 + a2) / 2  
h = (h - np.min(h)) / (np.max(h) - np.min(h)) * 255
```

3. For each example, give the two σ values you used. Explain how you arrived at these values. Discuss how successful your examples are or any interesting observations you have made.

To select the best σ values, I have generated hybrid images by iteratively matching σ pairs, from 1 to 20, as below.

```
for i in range(1,20):
    for j in range(1,20):
        img1 = np.float64(cv2.imread('s.jpg', 0))
        a1 = gaussian_filter(img1, sigma=i)

        img2 = np.float64(cv2.imread('apple.jpg', 0))
        a2 = gaussian_filter(img2, sigma=j)

        h = (img1 - a1 + a2) / 2

        h = (h - np.min(h)) / (np.max(h) - np.min(h)) * 255
        plt.imshow(h, cmap='gray')
        plt.show()
```

The optimal hybrid image is selected by naked eye observation...

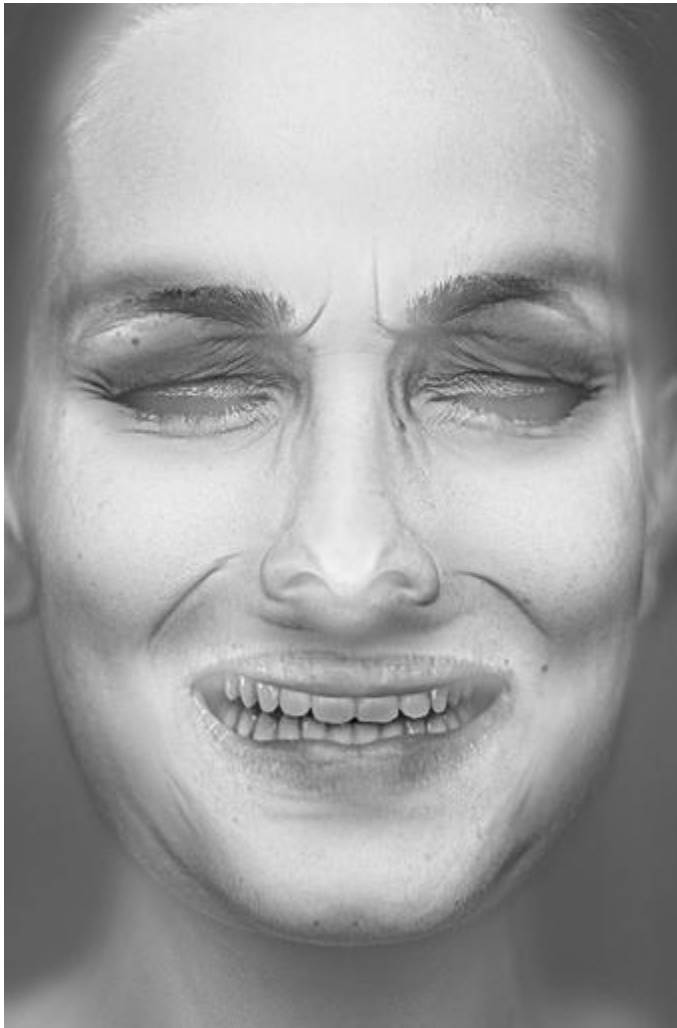
Accordingly, the two σ values I used are listed below for each example:

Example1:



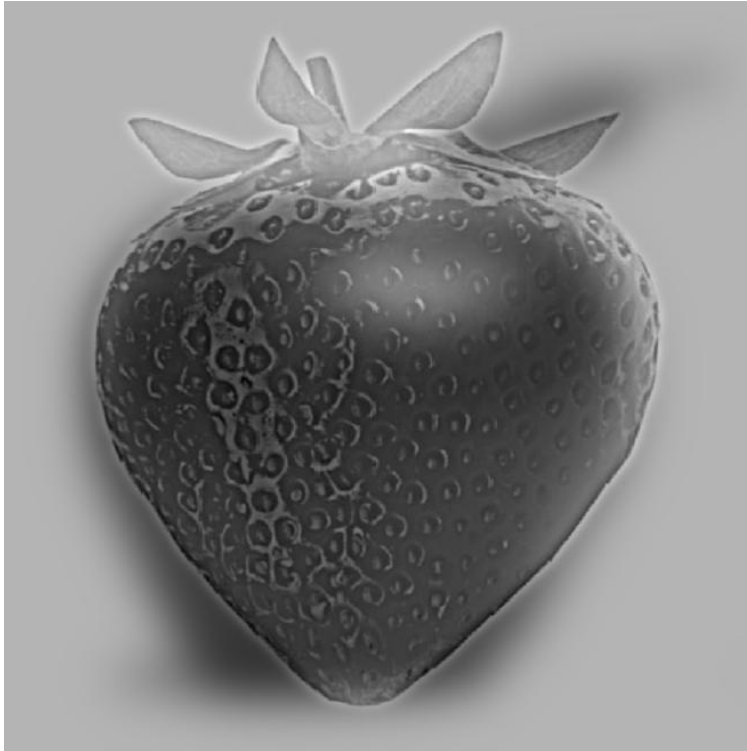
The σ value for high-pass filter is selected as 2, while 3 for low-pass filter.

Example2:



The σ value for high-pass filter is selected as 4, while 6 for low-pass filter.

Example3:



The σ value for high-pass filter is selected as 5, while 20 for low-pass filter.

As far as I can notice, all 3 examples work pretty well with the selected sigma values pairs. When zoomed out, the image is perceived as basically pure smoothed image, while zooming in would unveil the outlines of the sharpened image.

The potential defective part might lie in the 2nd example:



When zoomed out, the teeth seem vague but still perceivable. This might be due to the fact there's a low-frequency component within the seemingly high-frequency teeth part. So, although the teeth sharpening effect is alleviated by zooming out, its low-frequency component still remains.

Part 2:

1. The output of your circle detector on all the images

Image 1:

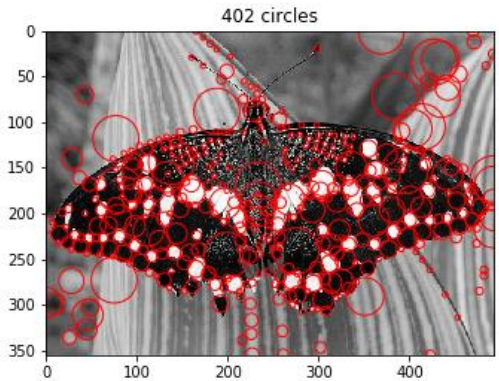
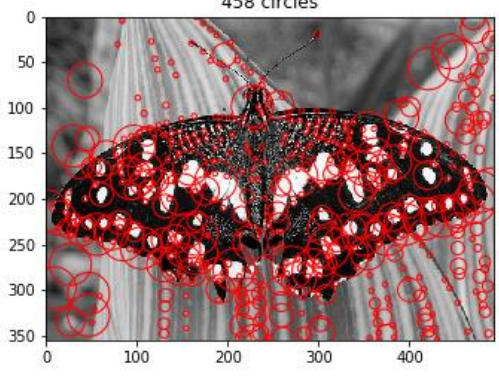
Implementa tion	Blob detected	Number of levels	k	Running time (s)
Image downsampling		12	1.22	0.6891548633 57544
filter size Increase		12	1.1	0.7928786277 770996

Image 2:

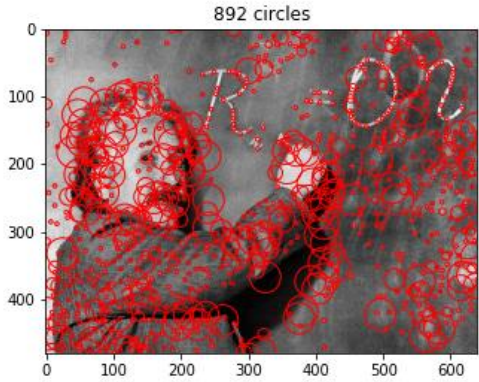
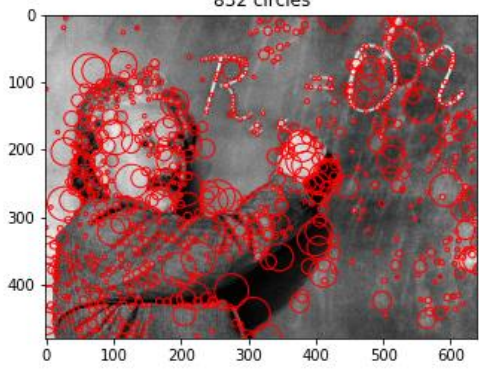
Implementa tion	Blob detected	Number of levels	k	Running time (s)
Image downsampling		12	1.22	1.2297096252 441406
filter size Increase		12	1.1	1.4052391052 246094

Image 3:

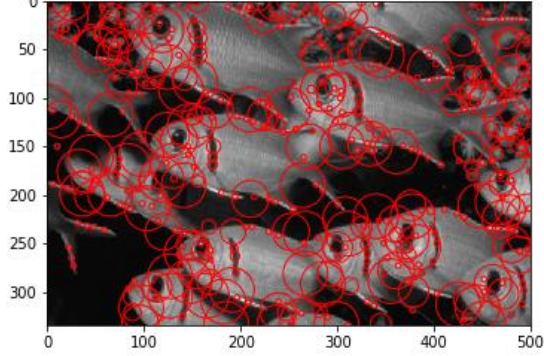
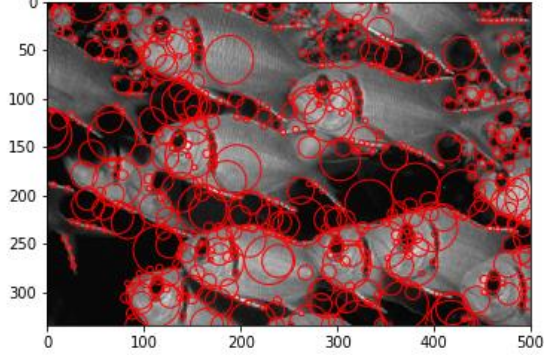
Implementa tion	Blob detected	Number of levels	k	Running time (s)
Image downsampling	<p>446 circles</p> 	12	1.22	0.7629582881 92749
filter size Increase	<p>542 circles</p> 	12	1.1	0.6622281074 523926

Image 4:

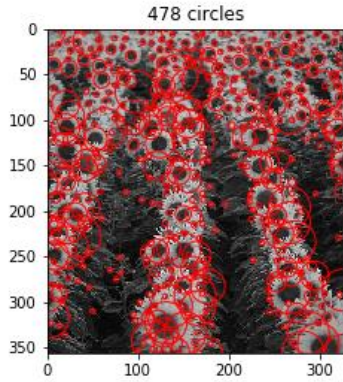
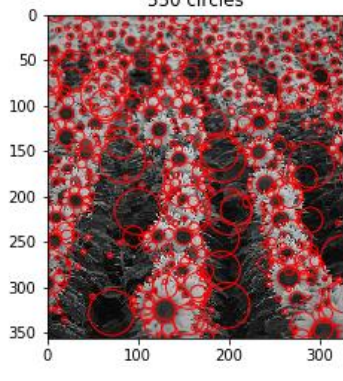
Implementa tion	Blob detected	Number of levels	k	Running time (s)
Image downsampling		12	1.22	0.5415523052 215576
filter size Increase		12	1.1	0.4248626232 147217

Image 5:

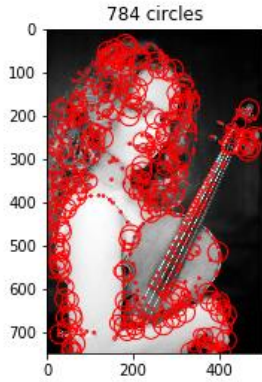
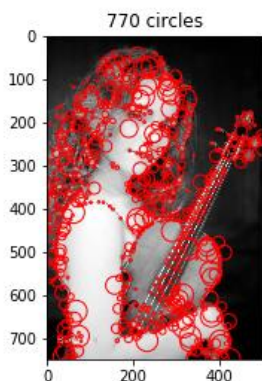
Implementa tion	Blob detected	Number of levels	k	Running time (s)
Image downsampling		12	1.22	1.5603861808 776855
filter size Increase		12	1.1	1.3302795886 993408

Image 6:

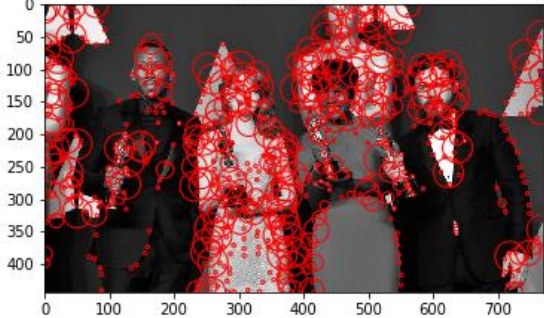
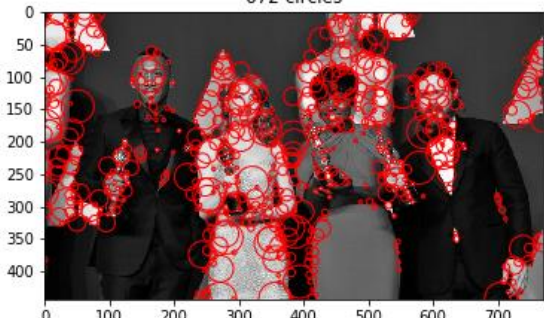
Implementa tion	Blob detected	Number of levels	k	Running time (s)
Image downsampling	<p>744 circles</p> 	12	1.22	1.4395985603 33252
filter size Increase	<p>672 circles</p> 	12	1.1	1.2372653484 344482

Image 7:

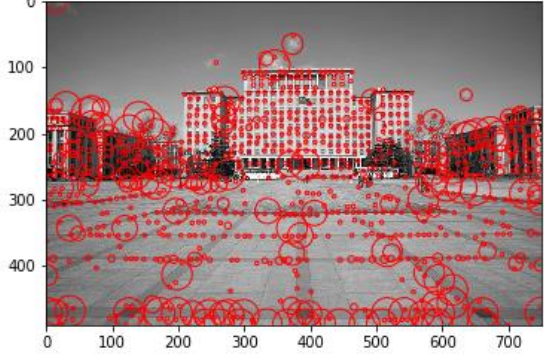
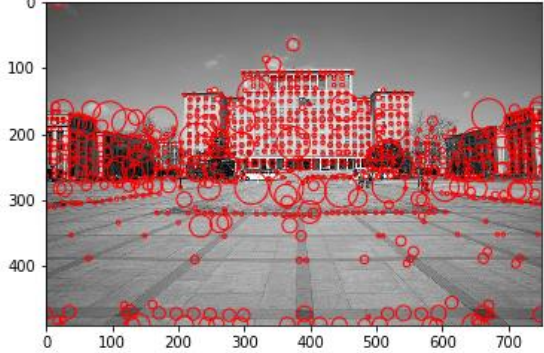
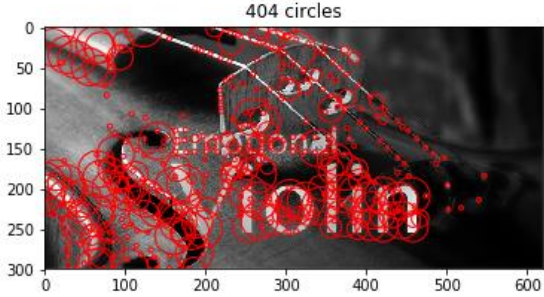
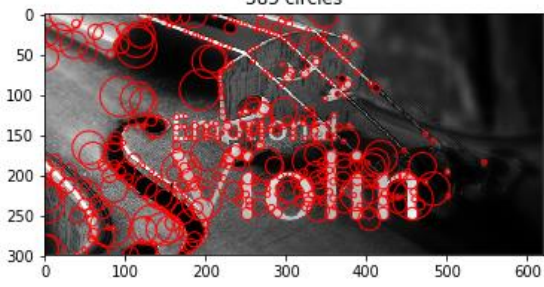
Implementation	Blob detected	Number of levels	k	Running time (s)
Image downsampling	<p>938 circles</p> 	12	1.22	1.5378913879 394531
filter size Increase	<p>780 circles</p> 	12	1.1	1.3252248764 038086

Image 8:

Implementation	Blob detected	Number of levels	k	Running time (s)
Image downsampling		12	1.22	0.8066668510 437012
filter size Increase		12	1.1	0.7072038650 512695

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

During downscaling implementation, scale normalization is not used because the sigma is not changed, so the problem of Laplacian response decaying with scale is not of our concern now. Besides, to better preserve the interpolation precision, quadratic interpolation is adopted instead of linear interpolation.

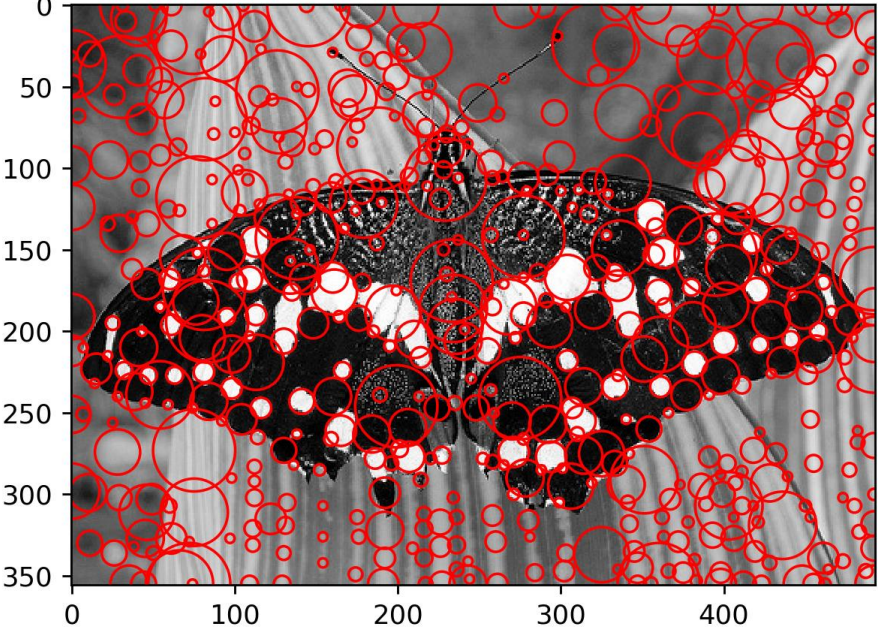
3. An explanation of parameter values you have tried and which ones you found to be optimal.

For filter size Increase method, the parameters I tested are k, filter size and threshold on the squared Laplacian response.

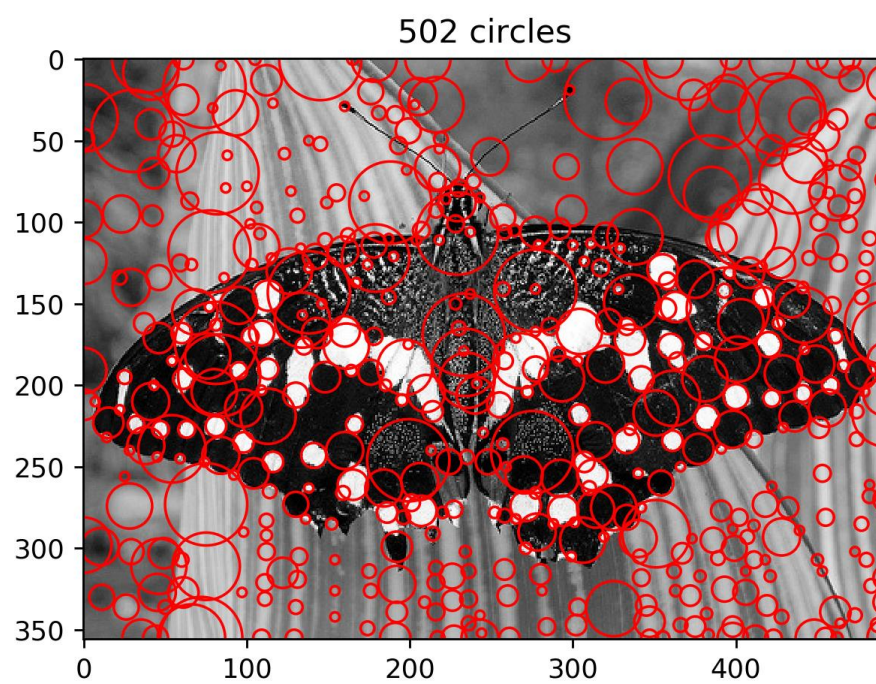
The threshold was selected as the percentile of 2D filtered slice:

```
threshold_min = np.percentile(p, 80)
result[:, :, i] = np.clip(p, a_min=threshold_min, a_max=None)
```

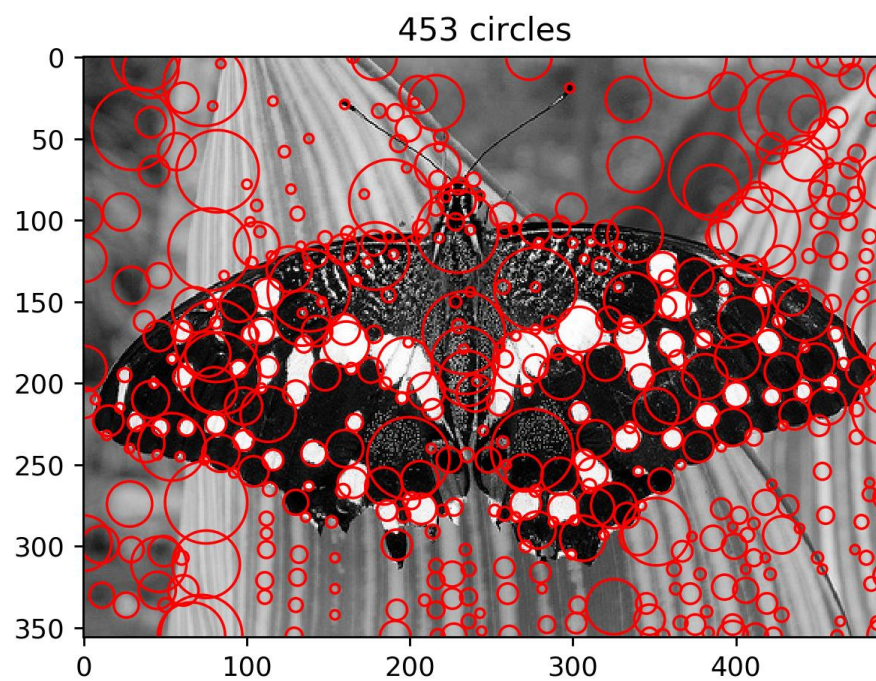
The higher the threshold, the less circles detected:

threshold_min	
10 percentile	<div><div>545 circles</div></div>

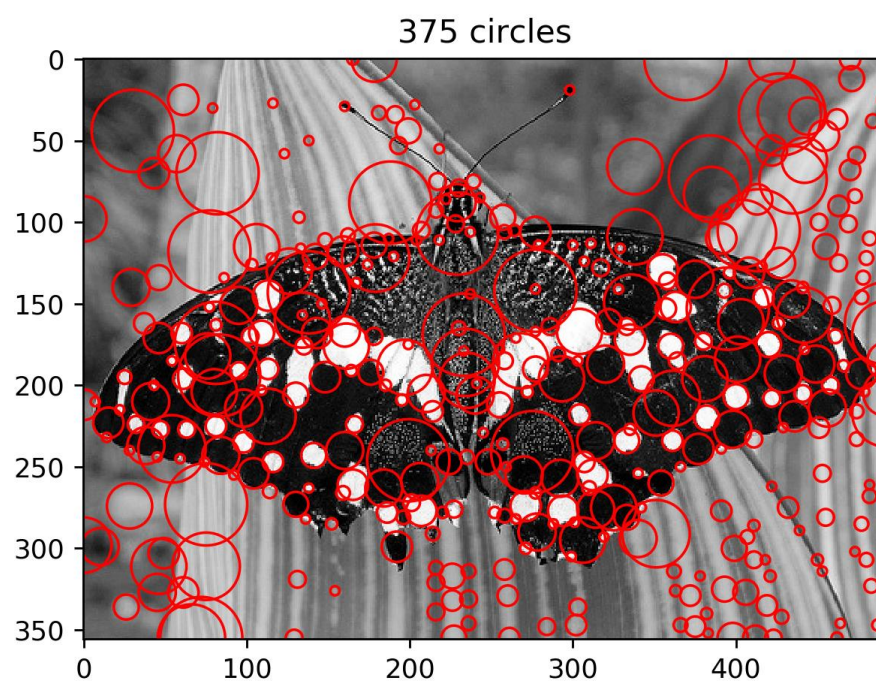
20 percentile



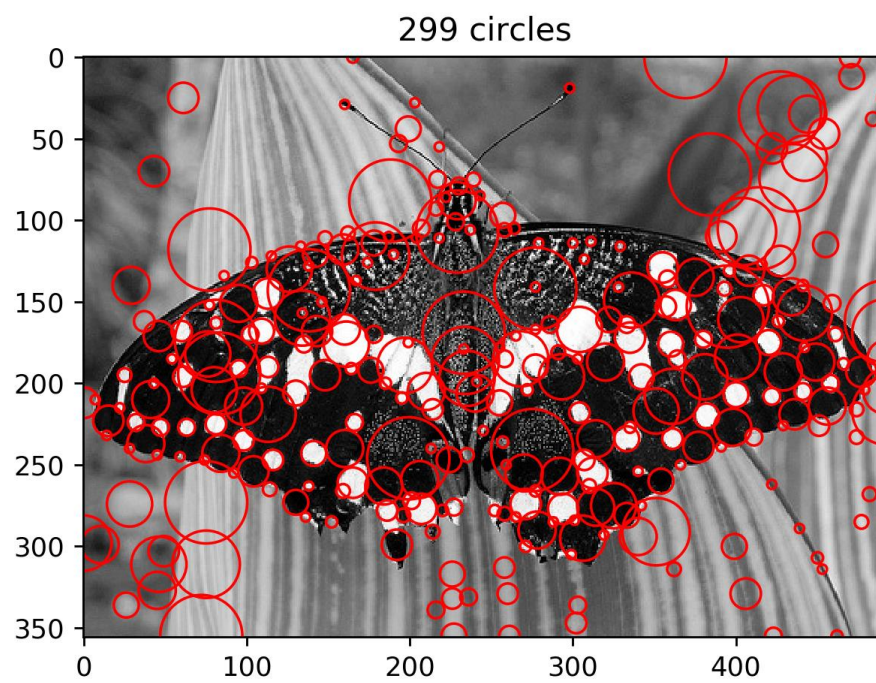
30 percentile



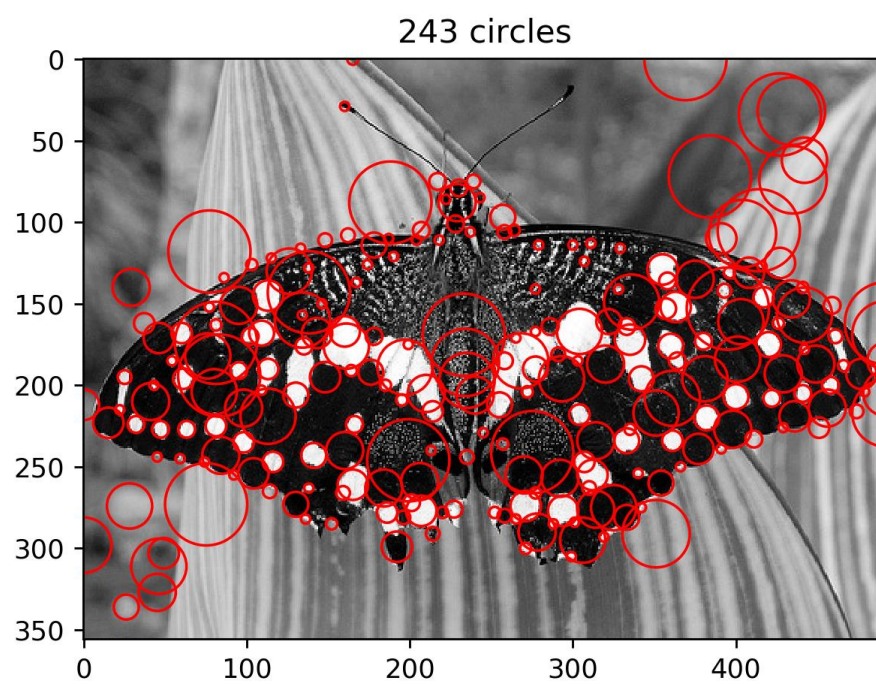
40 percentile



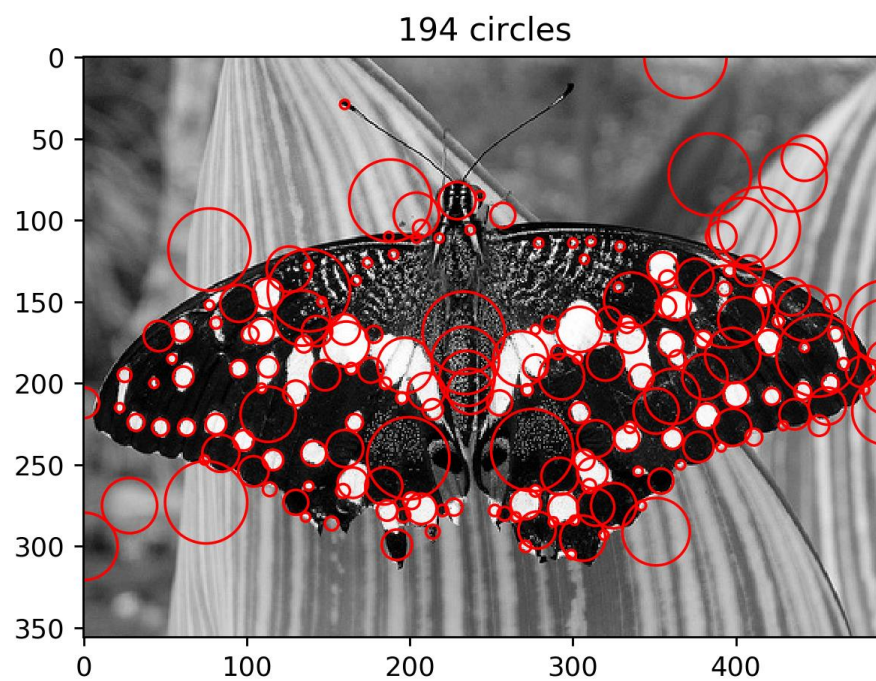
50 percentile



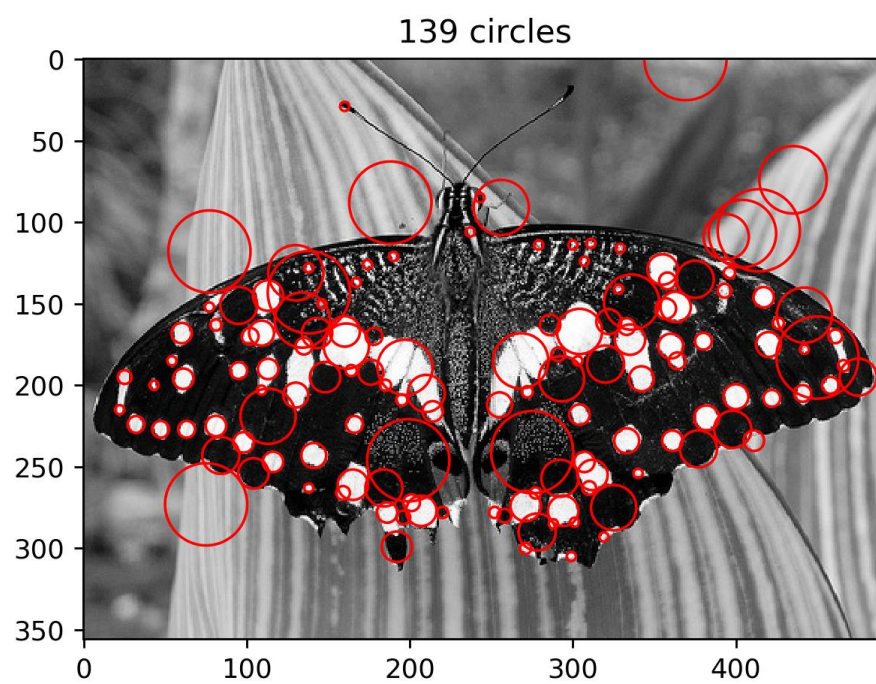
60 percentile



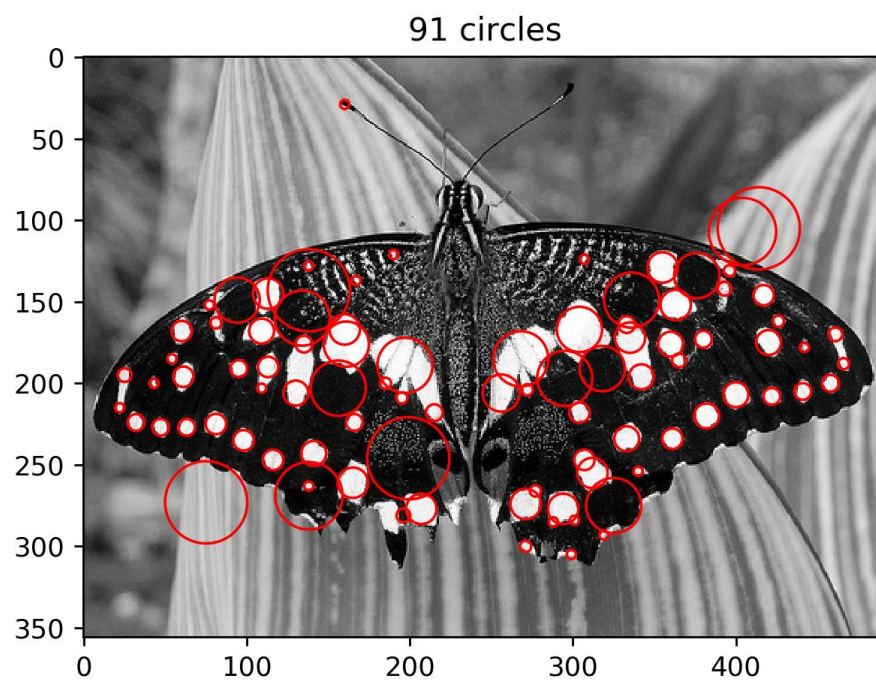
70 percentile



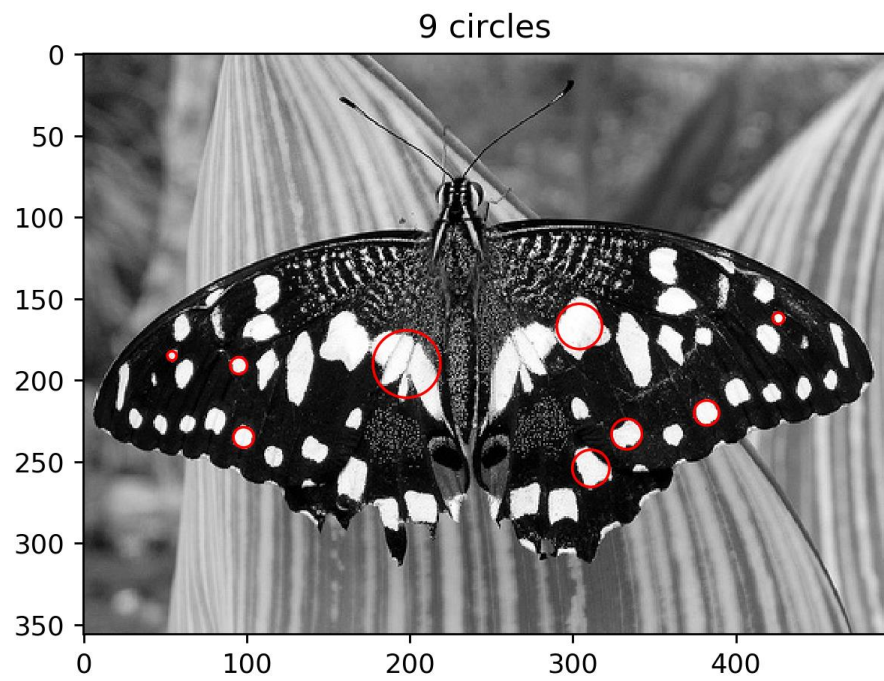
80 percentile



90 percentile



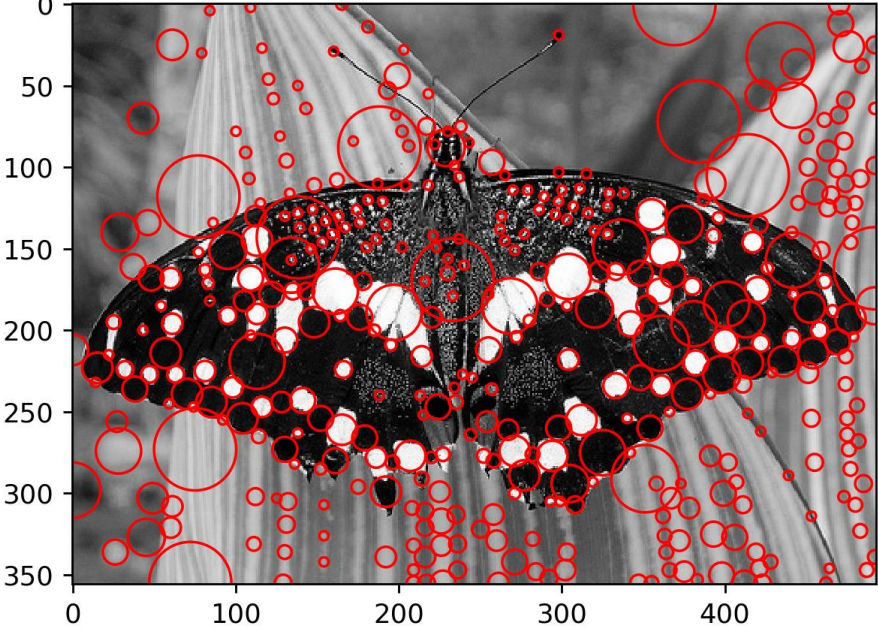
100 percentile



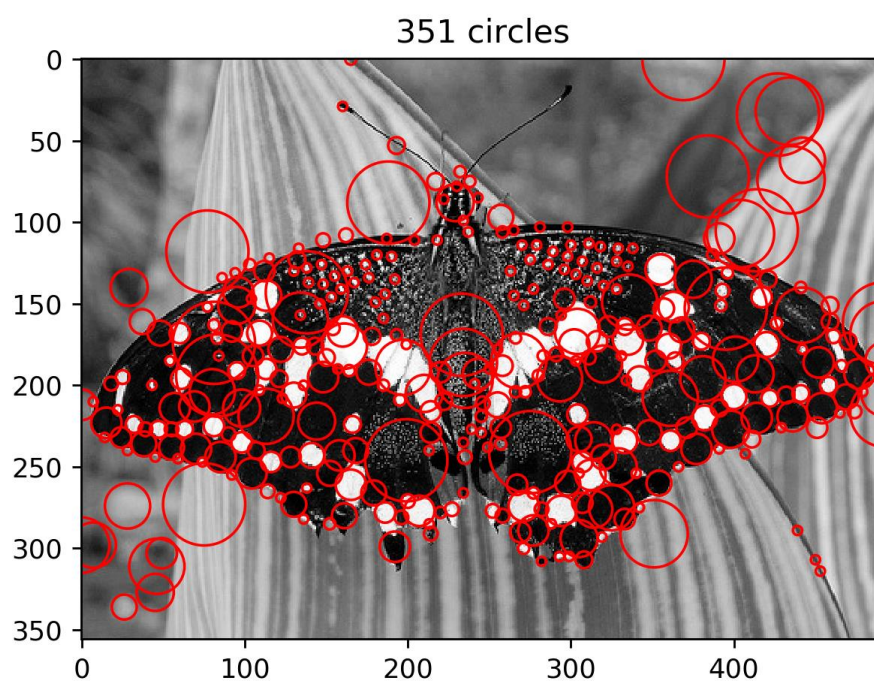
For k values, it determines the range of circle diameters.

I tried k from 1.01 to 1.5, and find $k=1.22$ renders best circle sets to accommodate characteristic patterns in the image.

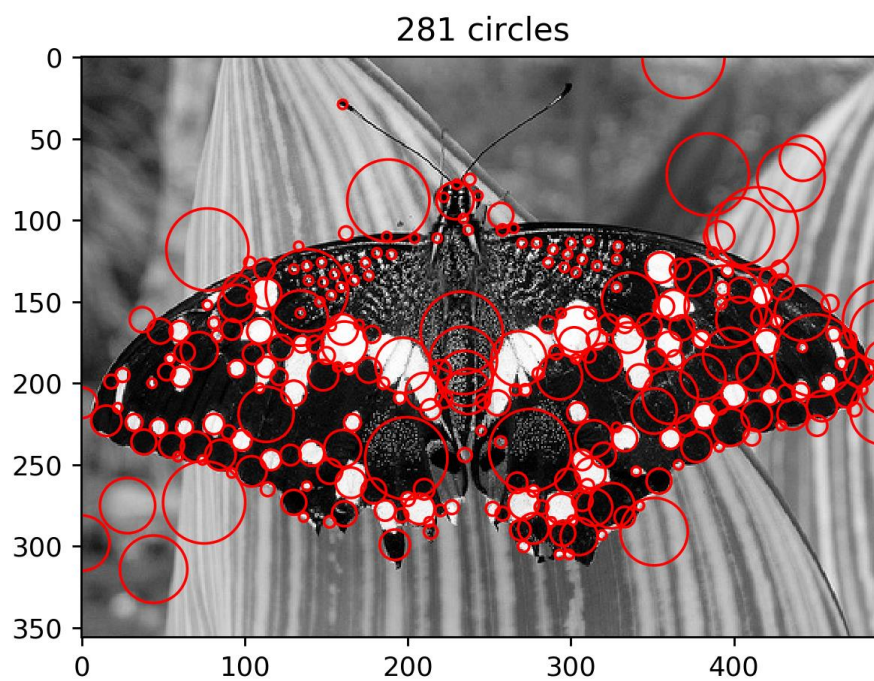
For filter size, larger size would filter out more blobs, thus less circles would be detected.

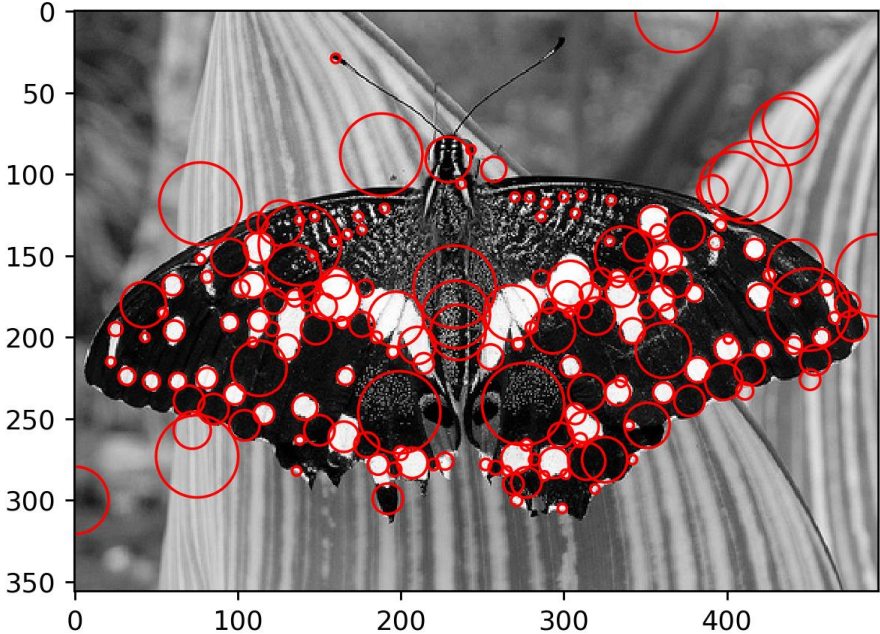
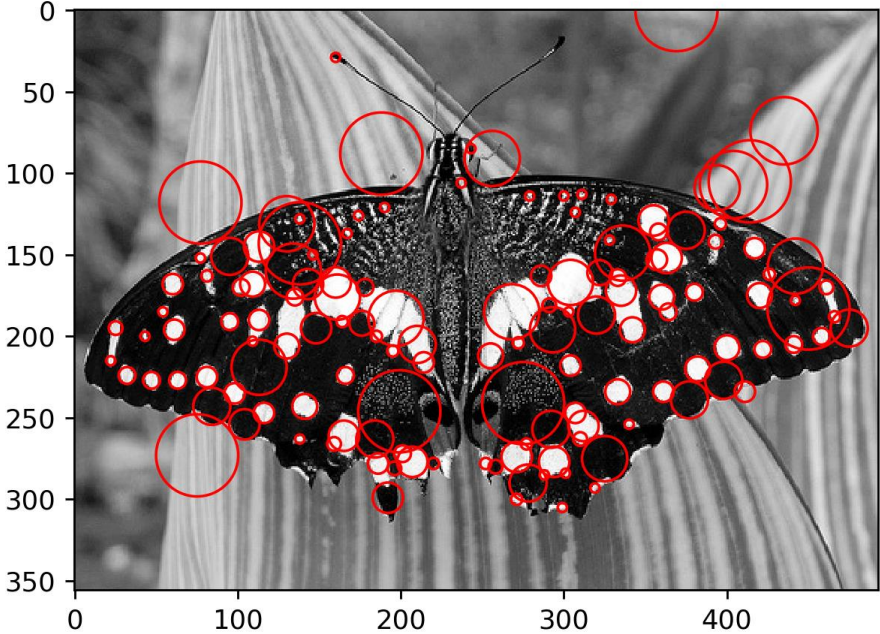
Filter Size for 2D filtering	Blob detection
1	<div>397 circles</div> 

5



10



15	<p>184 circles</p> 
20	<p>139 circles</p> 

Accordingly, for 2D slice filtering, $\sigma=3$ was adopted and for 3D scale space filtering, $\sigma = 12$ was used, considering the blob density and accuracy, among all σ pairs compared.