Part 1

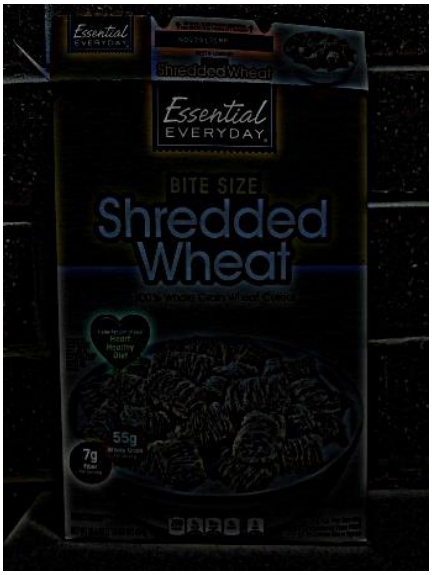
Comment on implementation:

For this part, I use scipy.ndimage.gaussian_filter to perform gaussian filtering. I read the source code of this function and found it convolves along each axis by a 1D gaussian filter, which is discussed in the lecture. It decreases the computational complexity drastically especially when sigma is large.

Example one: Cereal

The original images

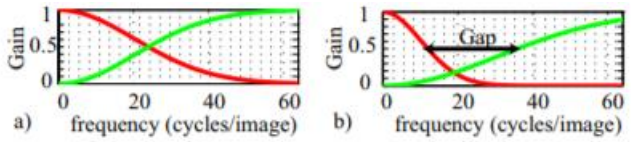
The filtered images



The hybrid images

Comment on result:

The sigma values I choose for this example are 4 and 5. The sigma of high frequency component should always be smaller than low frequency component. The reason is that frequency overlap of the outputs will produce an ambiguous interpretation. The higher the sigma of the gaussian filter, the lower the low pass filter cut off frequency of the output. If the sigma of high frequency component is higher than that of low frequency component, there will be a big overlap which makes the result ambiguous.

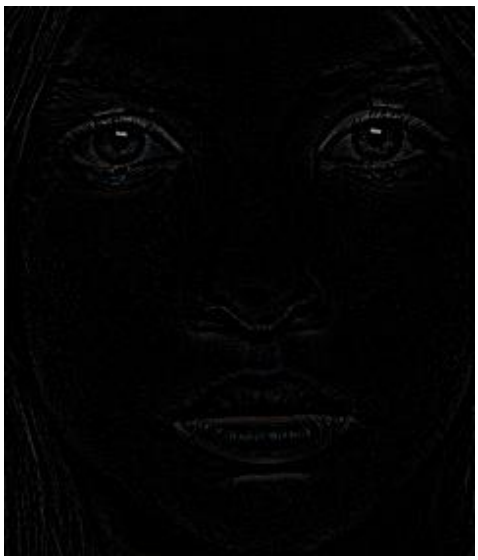


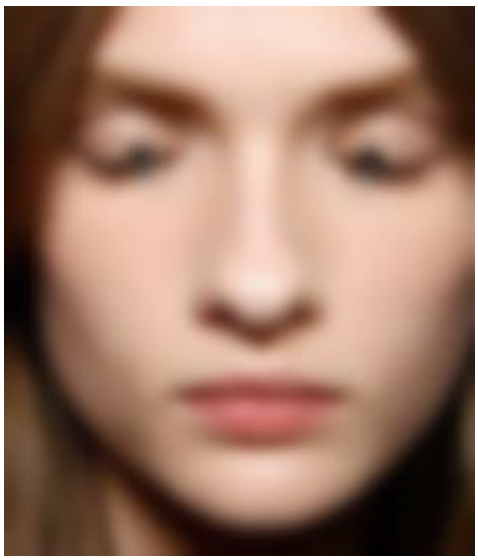
I choose 4 for the lower sigma because I want the letters “Shredded Wheat” to be less clear in low resolution. So it is good for it to be large enough. But I also want the plastic bag to be less clear in high resolution, so I choose 5 for the higher sigma. I think this is not a “successful example”. The plastic bag is too conspicuous and has a strong contrast with the letter.

Example two: Is her eyes closed or open?

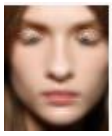
The original images

The filtered images





The hybrid images

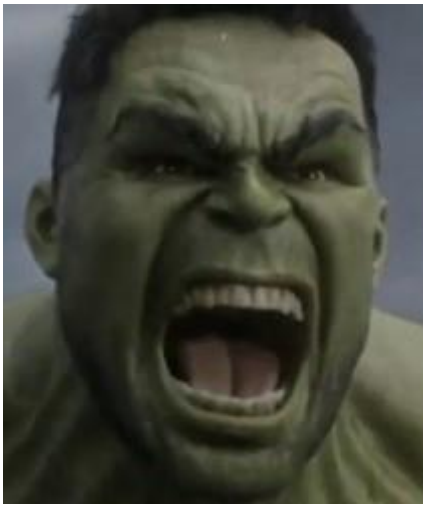


Comment on result:

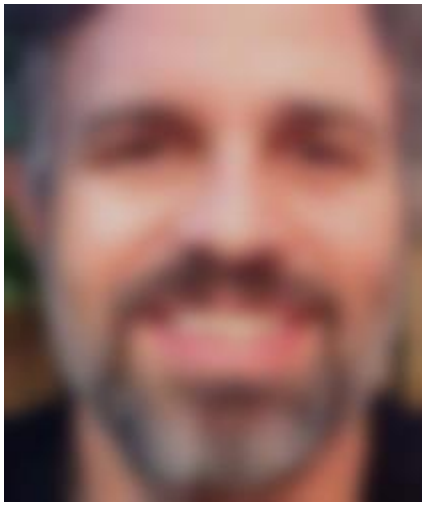
The sigma for the two images are 2 and 4. In order to make the high frequency part invisible in the image of low resolution I display above. And for the higher sigma, 4 is just good to make the eye clear in the high resolution image. The implementation is basically the same as above. The only thing to notice is that I actually take a while to find the proper images. It is not easy to find a pair of images with good alignment. I crop the input to align the eyes. The result is pretty good. I can see the eyes open clearly in high resolution and the eyes closed in low resolution.

Example three: Bruce Banner and Hulk

The original images



The filtered images



The hybrid images

Comment on result:

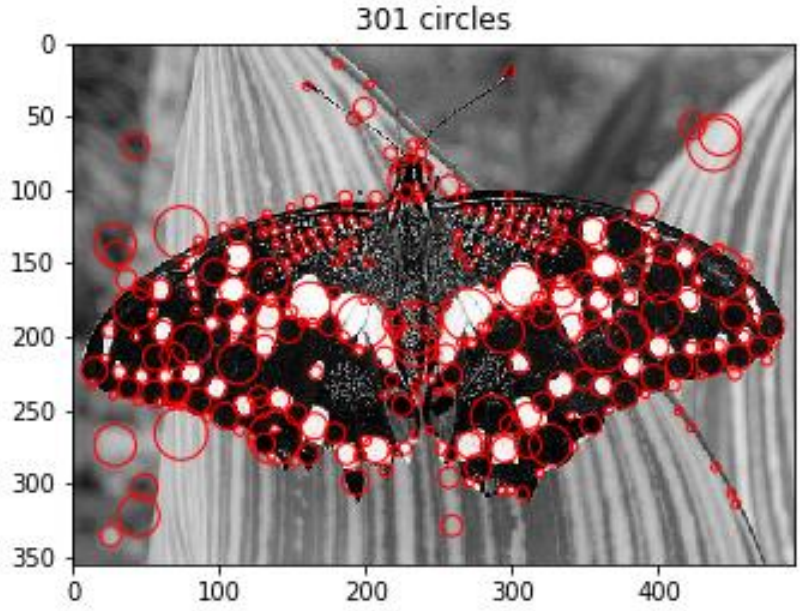
The sigma for these two images are 4 and 5 correspondingly. They are just good to make Bruce Banner looks like the angry Hulk in high resolution image, and has a smiling face in the low resolution image. An interesting thing to note is that I align the eyebrows and the mouths for these two images. This makes the Bruce Banner in high resolution image look like he is yelling and frowning.

Extra points:

I implemented this part using colored images. And as I state above, a good failure example, in my opinion, is the first image. The two original images have strong color contrast, which makes both of them in most resolutions.

Part 2

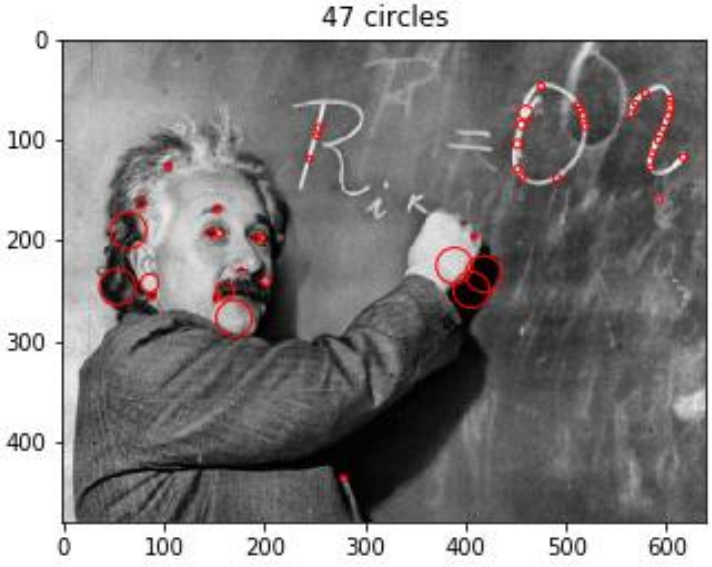
Butterfly:



Inefficient implementation: 253553100 ns

Efficient implementation: 134346600 ns

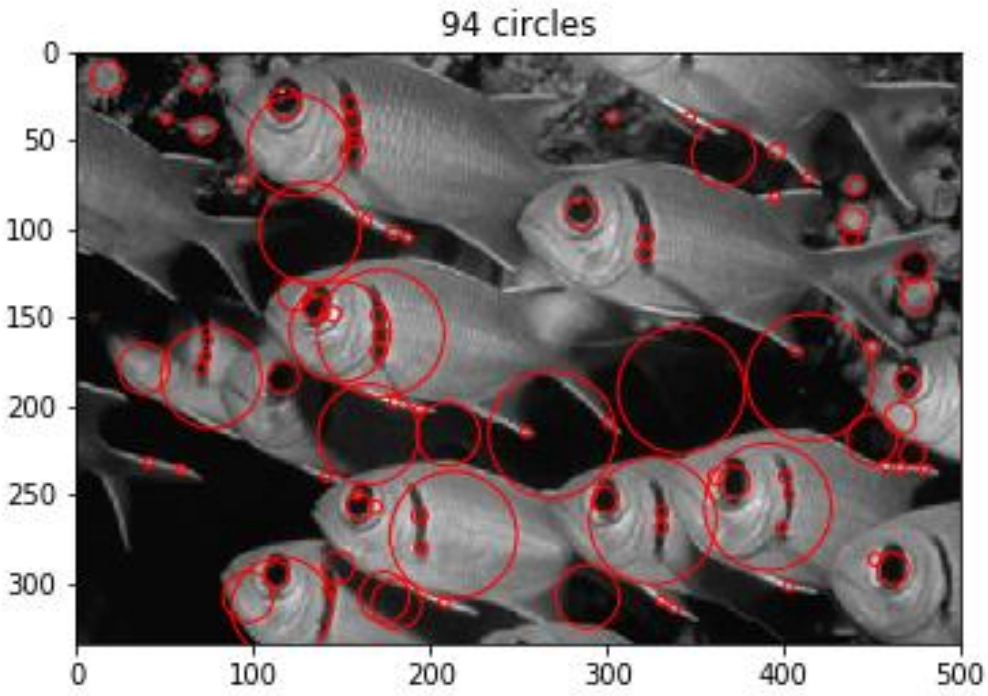
Einstein:



Inefficient implementation: 452309000 ns

Efficient implementation: 220807500 ns

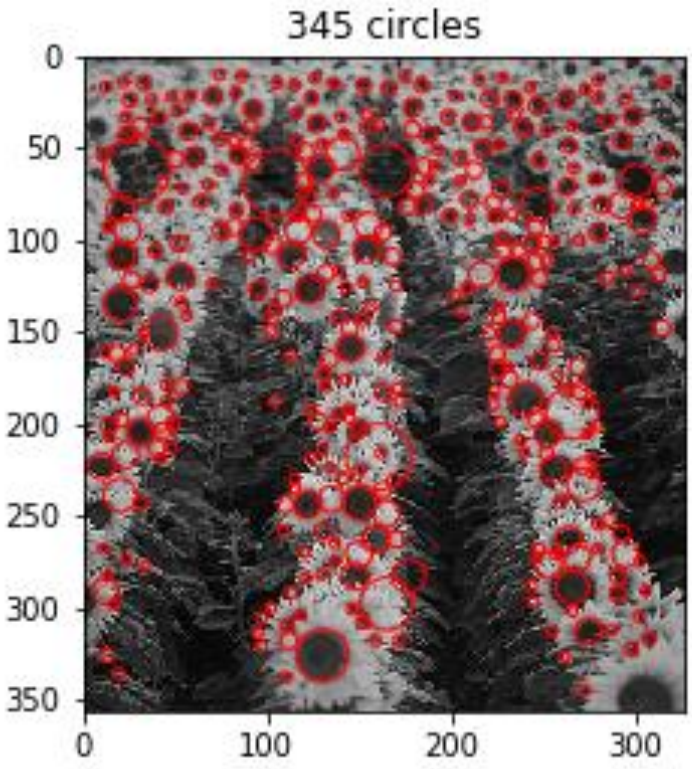
Fishes:



Inefficient implementation: 489198600 ns

Efficient implementation: 185093800 ns

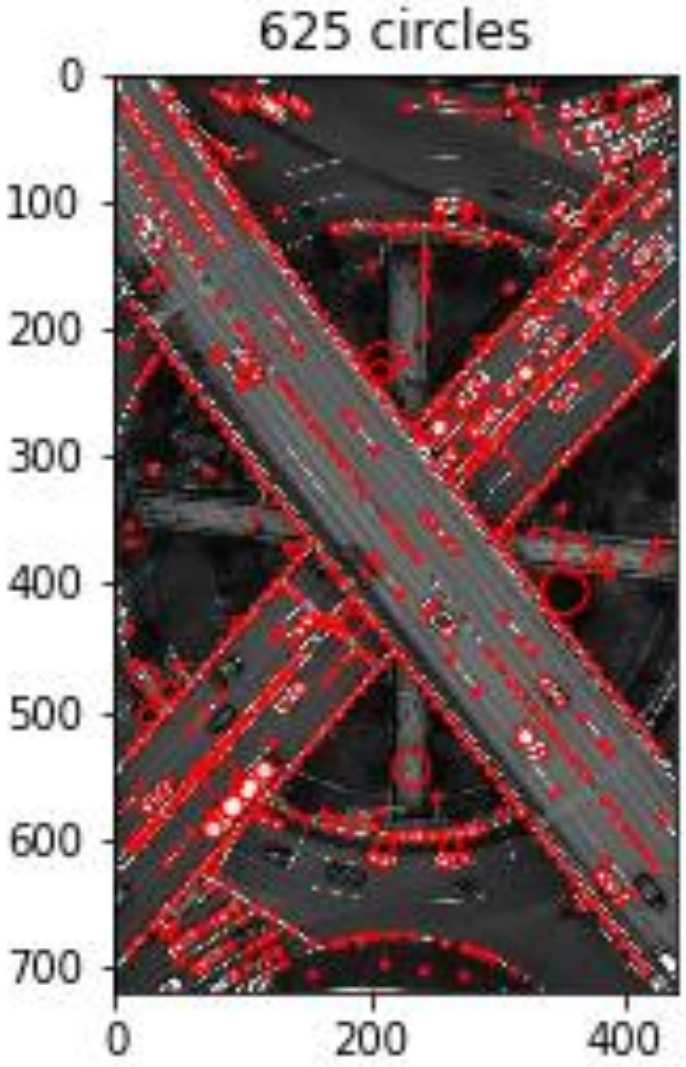
Sunflowers:



Inefficient implementation: 211793400 ns

Efficient implementation: 134354100 ns

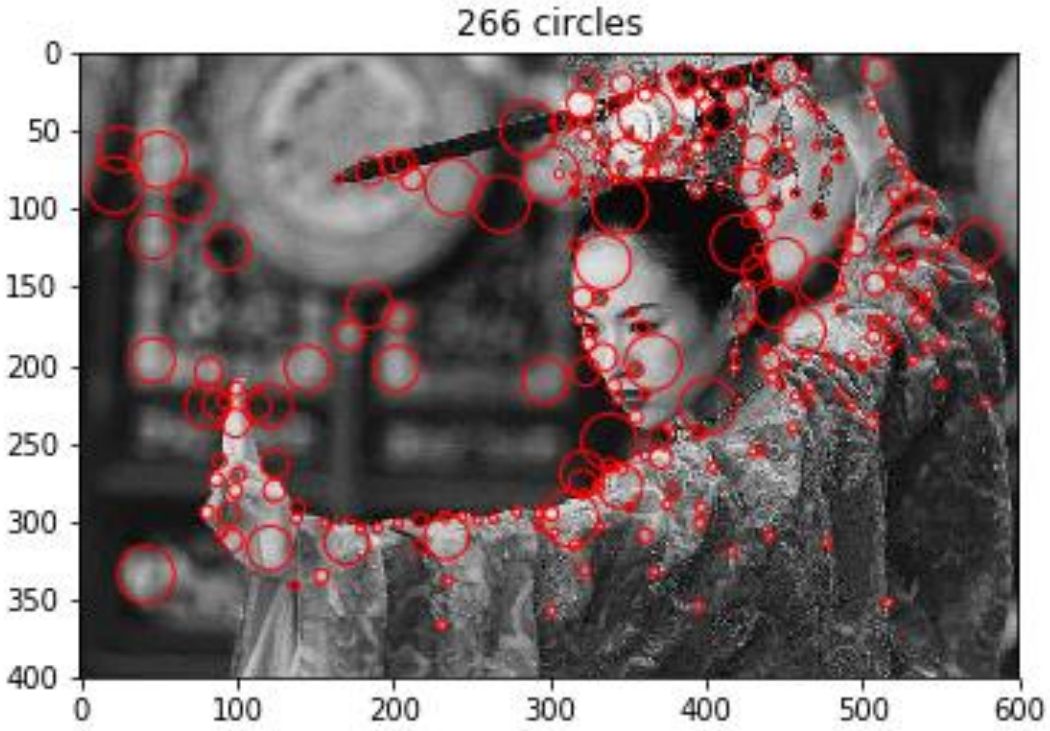
Bridge:



Inefficient implementation: 558379900 ns

Efficient implementation: 308197800 ns

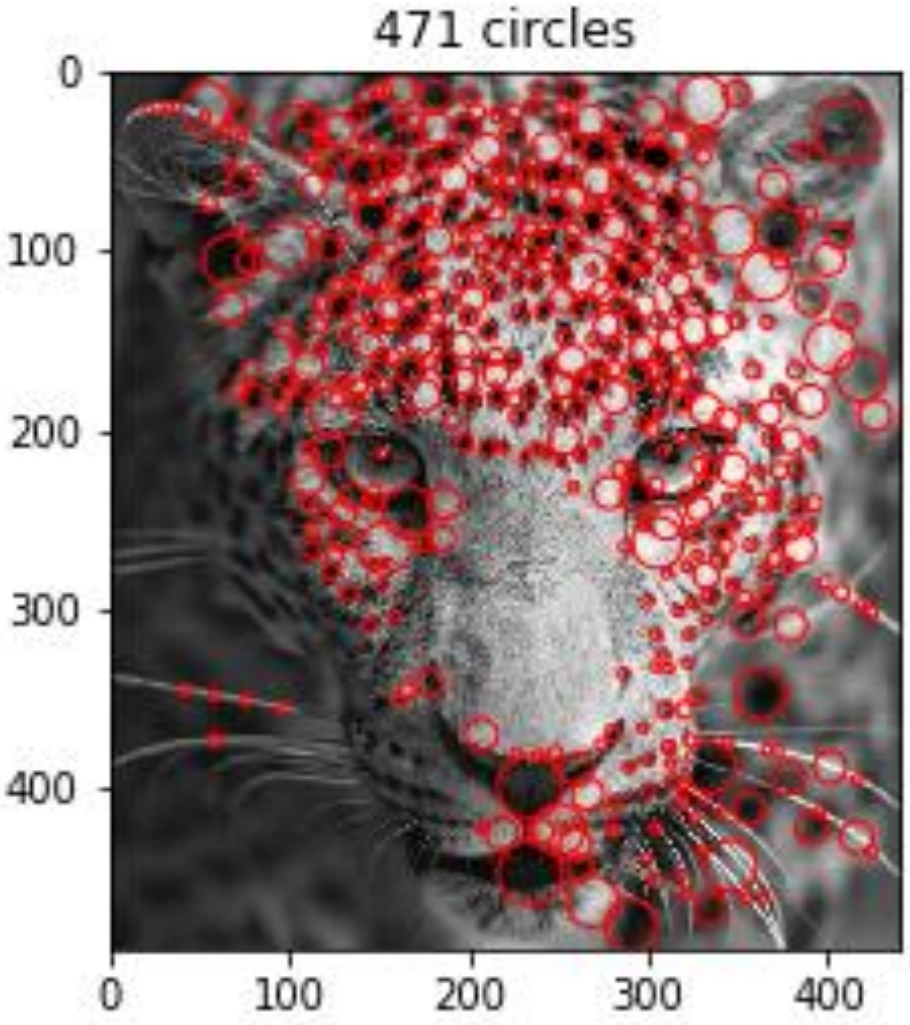
Beauty:



Inefficient implementation: 419424400 ns

Efficient implementation: 236427800 ns

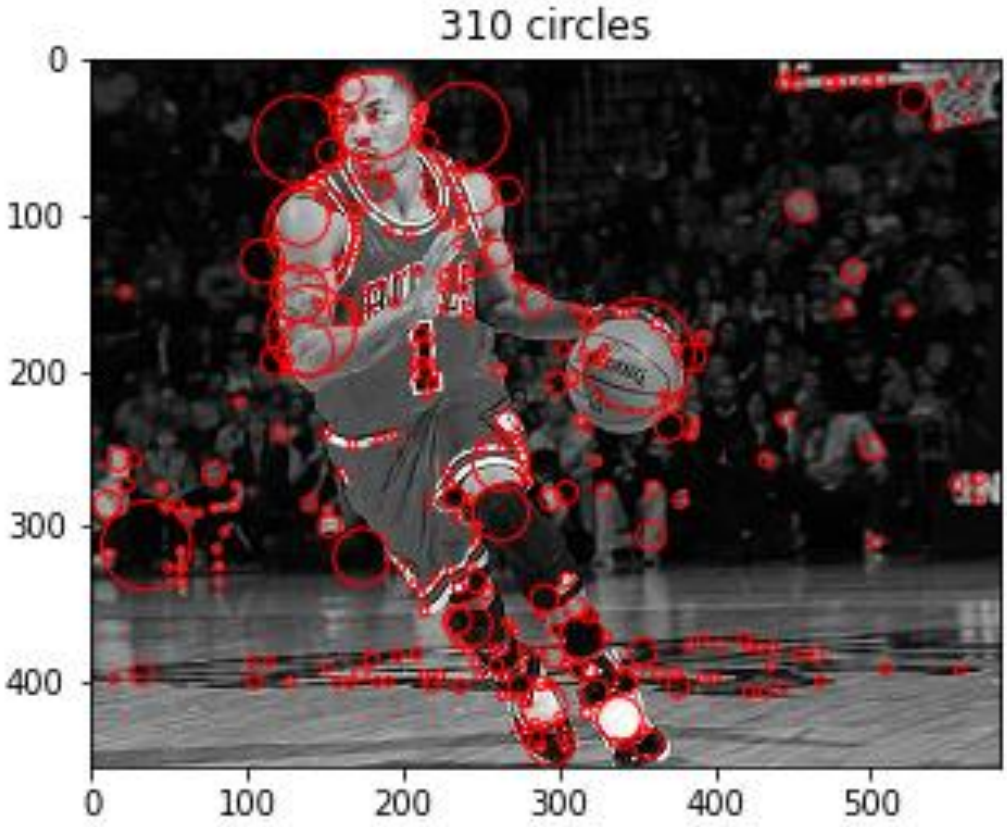
Leopard:



Inefficient implementation: 317149800 ns

Efficient implementation: 153590100 ns

Rose:



Inefficient implementation: 605415100 ns

Efficient implementation: 220403200 ns

Explanation of implementation:

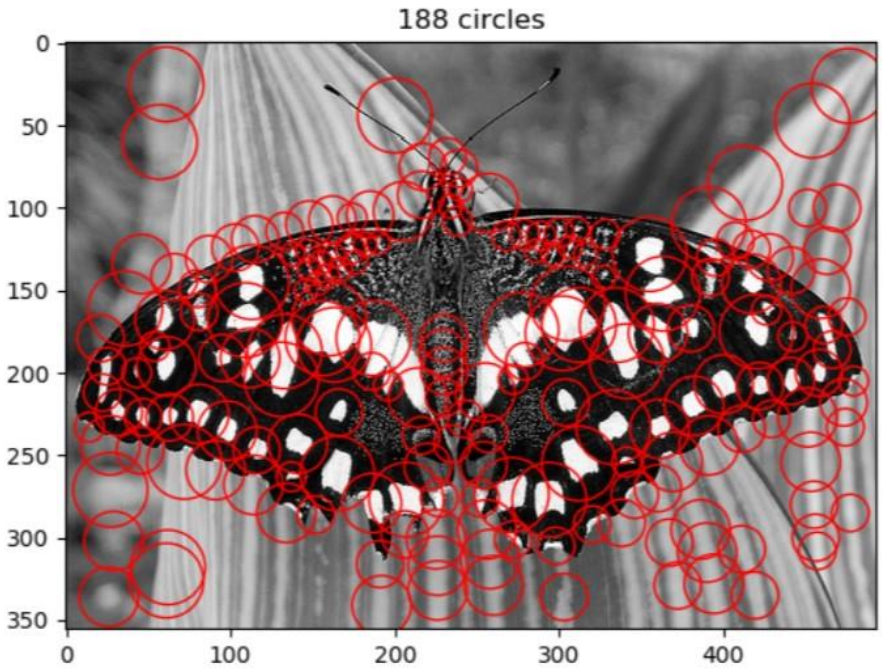
When I finished the basic part, I found that there were plenty of circles around the border. So I implement a border exclusion on all the output images. What I did was to wipe out all the local maxima whose distance to the border is less than exclude_border. I also noticed that even after this, some circles still went out of the border. So I also try to use the maximum radius of the circle (square 2 times sigma) as the exclude_border. But the result showed it didn't work because it will also eliminate some useful blobs located close to the border.

Explanation of parameter values:

I set the min_sigma to 2 since it is small enough to detect most small blobs. I set the sigma_ratio to 1.2599, whose cube is 2. It makes it easier to implement. In downsample implementation, I down sample the input by 2 every time the sigma is doubled. I set the exclude_border to 10 to eliminate most of the out of bound blobs. Initially, I set the NMS size to 3, but there were many overlapped circles. So I increased it to 10, and the result became better.

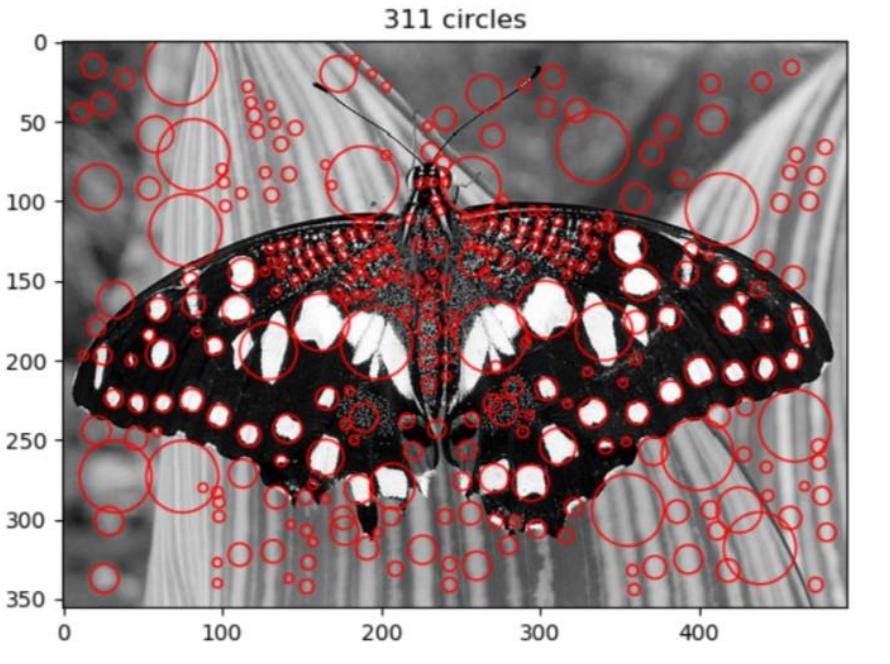
Extra point:

I tried to implement DoG. My first approach is to iteratively filter input with Gaussian filter. By doing so, I got a sequence of output with $\sigma=k\sigma, k2\sigma$.



It seems that the maxima have the correct center position but wrong radius.

I also tried to implement it in another naive way, that is to compute all the sigma values in advance.



The result is pretty good and totally makes sense to me.