CS543 Assignment 2

Your Name: Derek Yang
Your NetId: mcy3

Part 1 Fourier-based Alignment:

A: Channel Offsets

Low-resolution images (using channel Green as base channel):

Image	Blue (h,w) offset	Red (h,w) offset
00125v.jpg	(-6,-1)	(4,-1)
00149v.jpg	(-4,-2)	(5,0)
00153v.jpg	(-7,-2)	(7,1)
00351v.jpg	(-3,0)	(8,1)
00398v.jpg	(-5,-2)	(6,1)
01112v.jpg	(0,0)	(5,1)

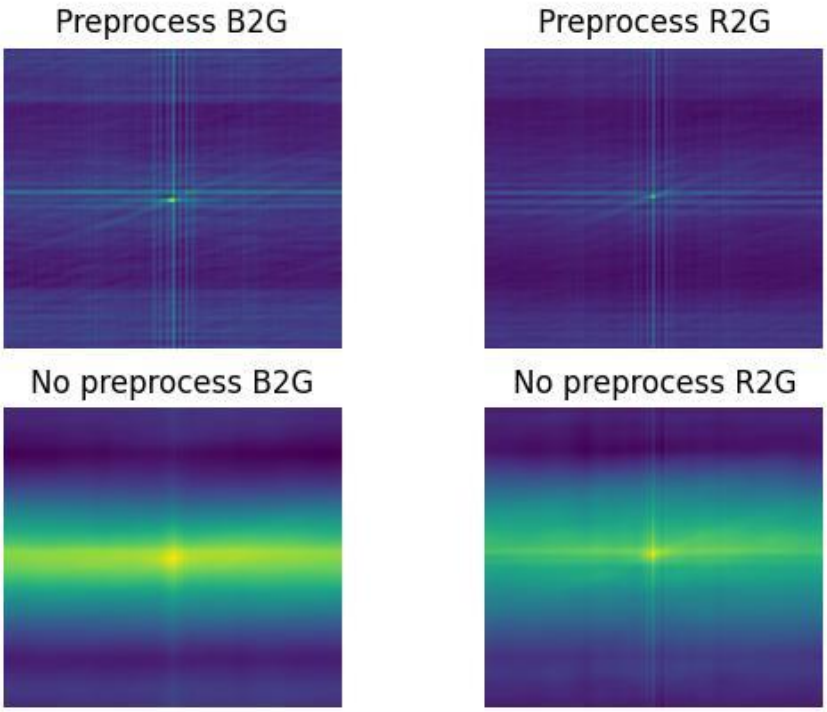
High-resolution images (using channel Blue as base channel):

Image	Green (h,w) offset	Red (h,w) offset
01047u.tif	(25,19)	(71,33)
01657u.tif	(56,9)	(118,13)
01861a.tif	(71,38)	(147,62)

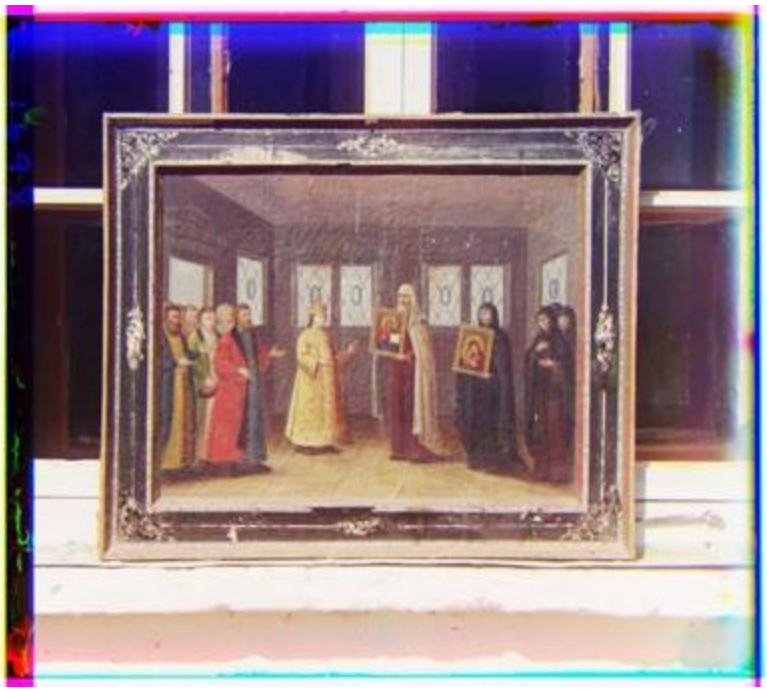
B: Output Visualizations

All the images shown is aligned with the original offset (original image height/3 as channel height) using the original image

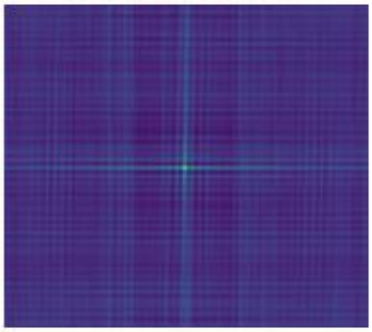
00125v.jpg



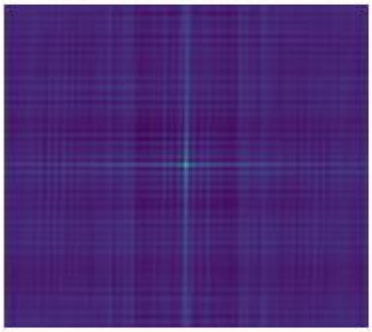
00149v.jpg



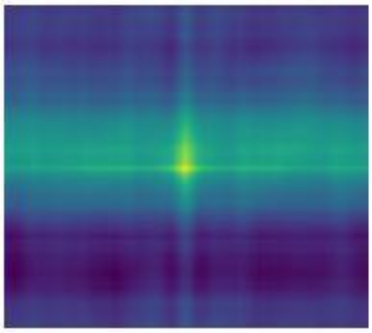
Preprocess B2G



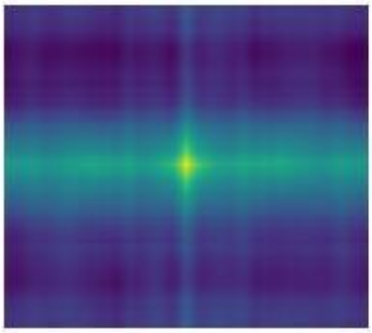
Preprocess R2G



No preprocess B2G

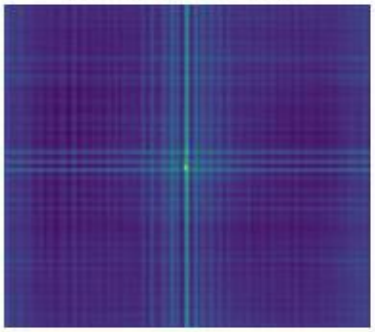


No preprocess R2G

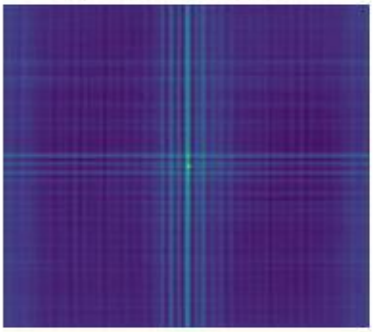


00153v.jpg

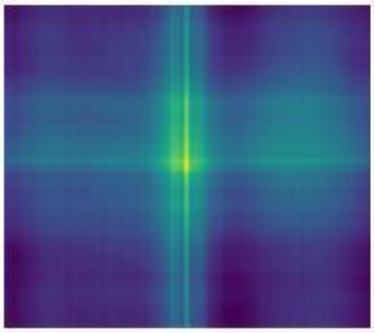
Preprocess B2G



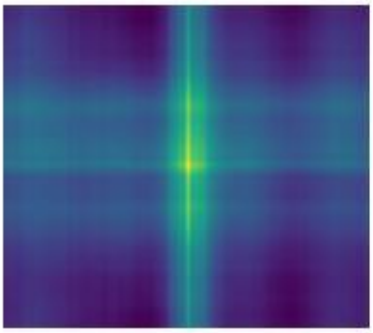
Preprocess R2G



No preprocess B2G

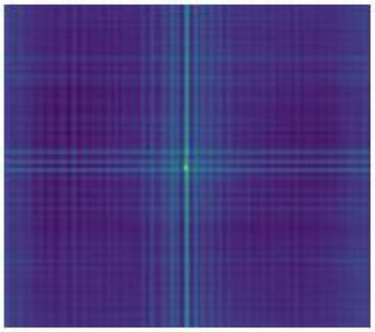


No preprocess R2G

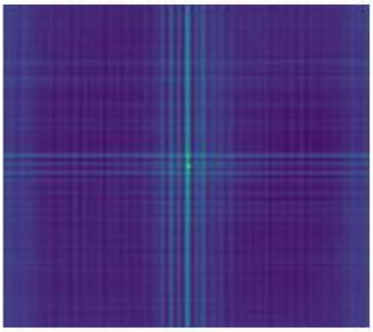


00351v.jpg

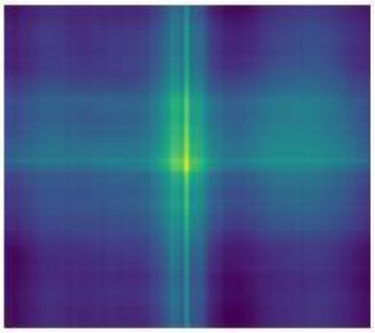
Preprocess B2G



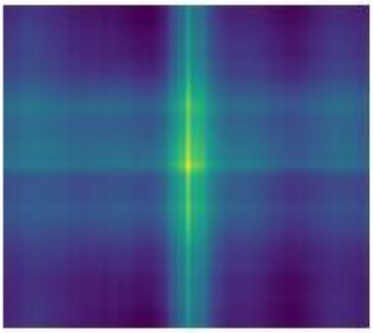
Preprocess R2G



No preprocess B2G

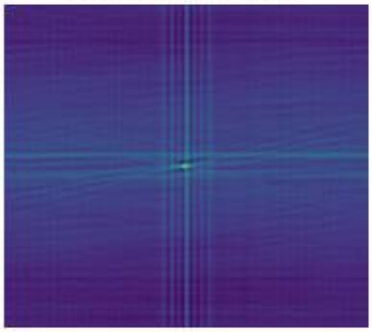


No preprocess R2G

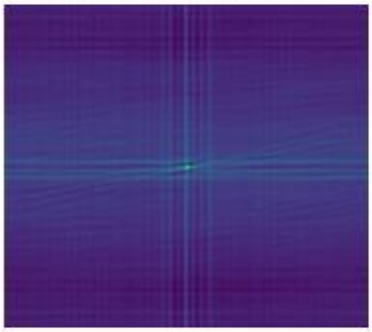


00398v.jpg

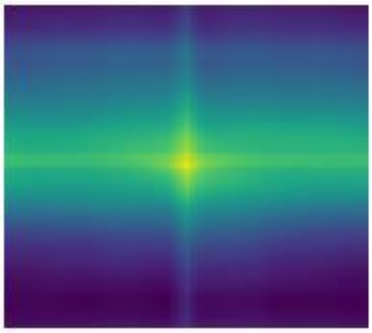
Preprocess B2G



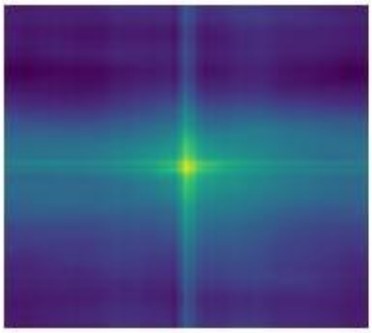
Preprocess R2G



No preprocess B2G

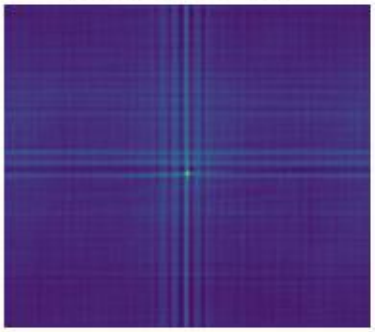


No preprocess R2G

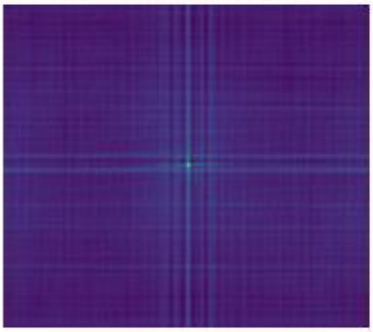


01112v.jpg

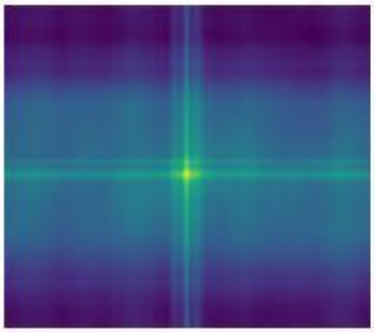
Preprocess B2G



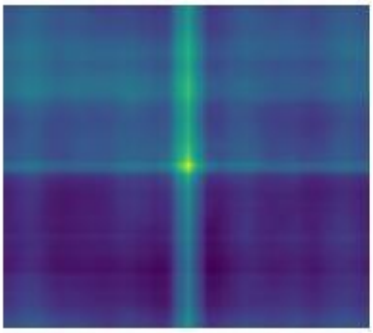
Preprocess R2G



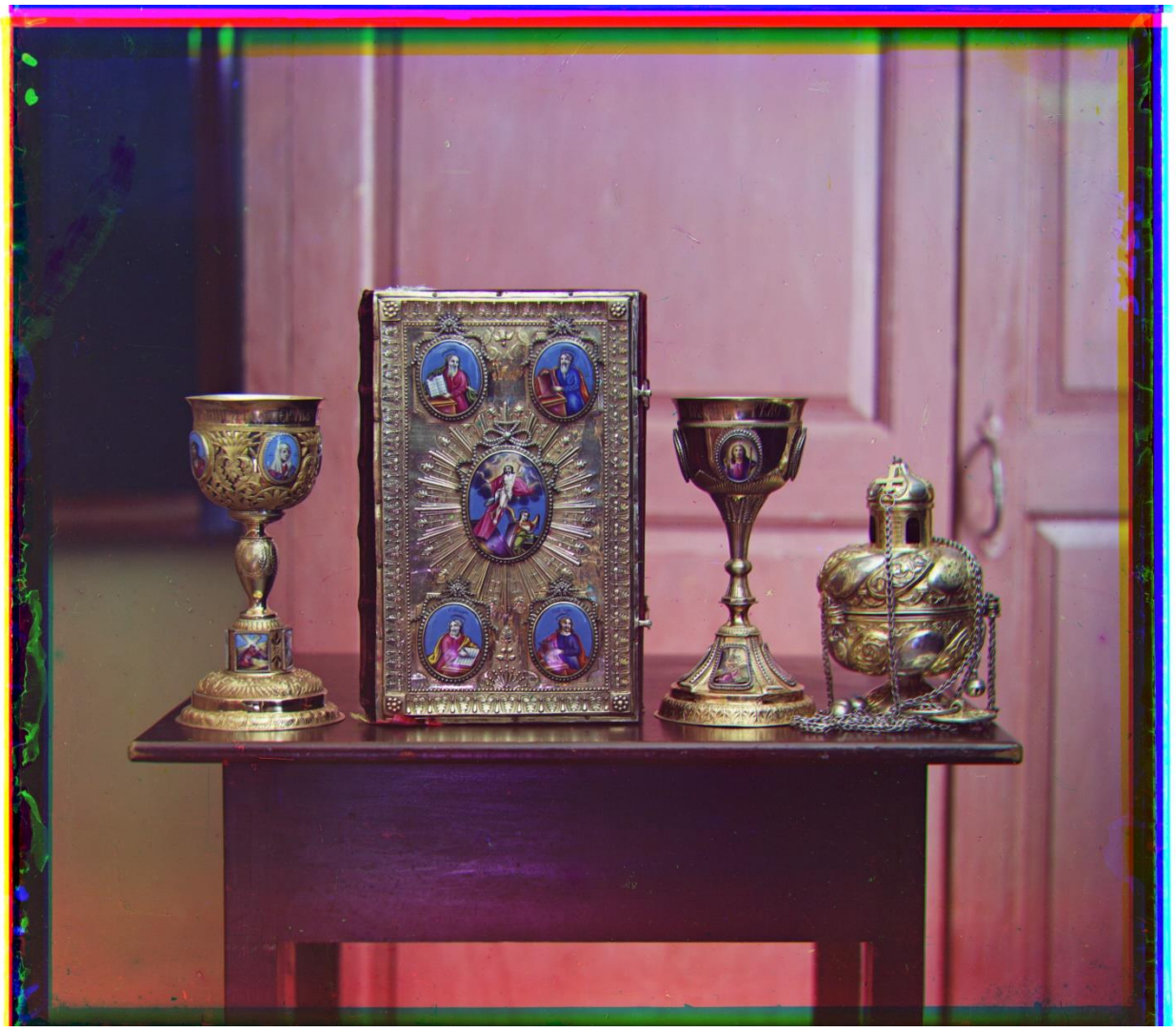
No preprocess B2G



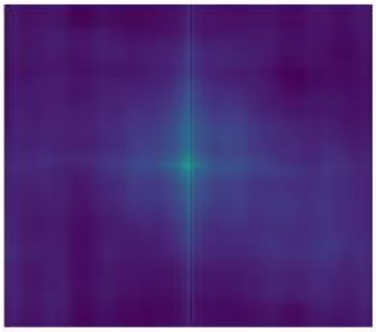
No preprocess R2G



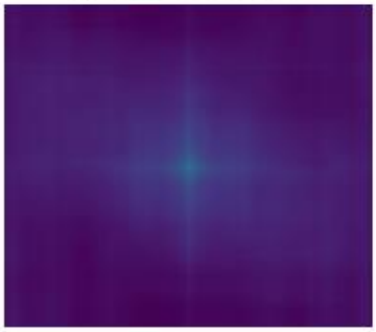
01047u.tif



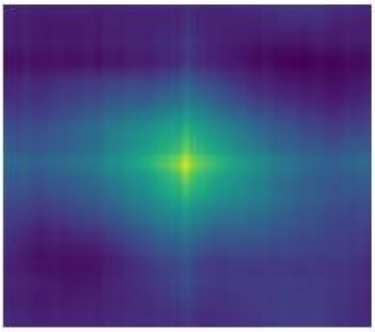
Preprocess B2G



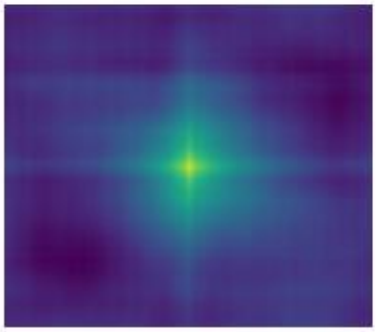
Preprocess R2G



No preprocess B2G

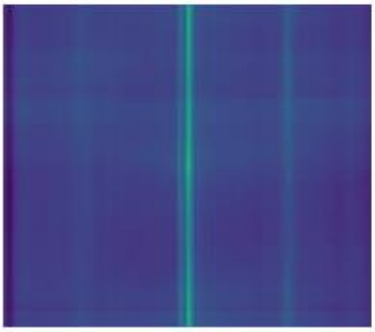


No preprocess R2G

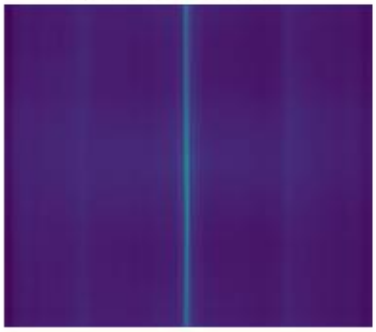


01657u.tif

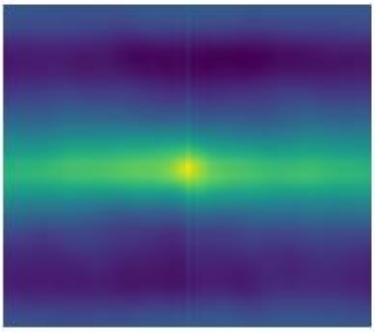
Preprocess B2G



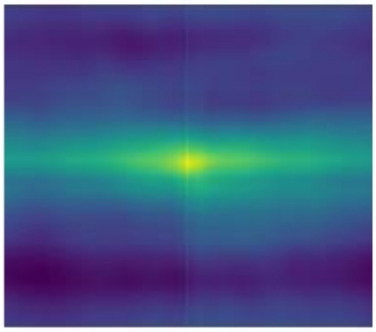
Preprocess R2G



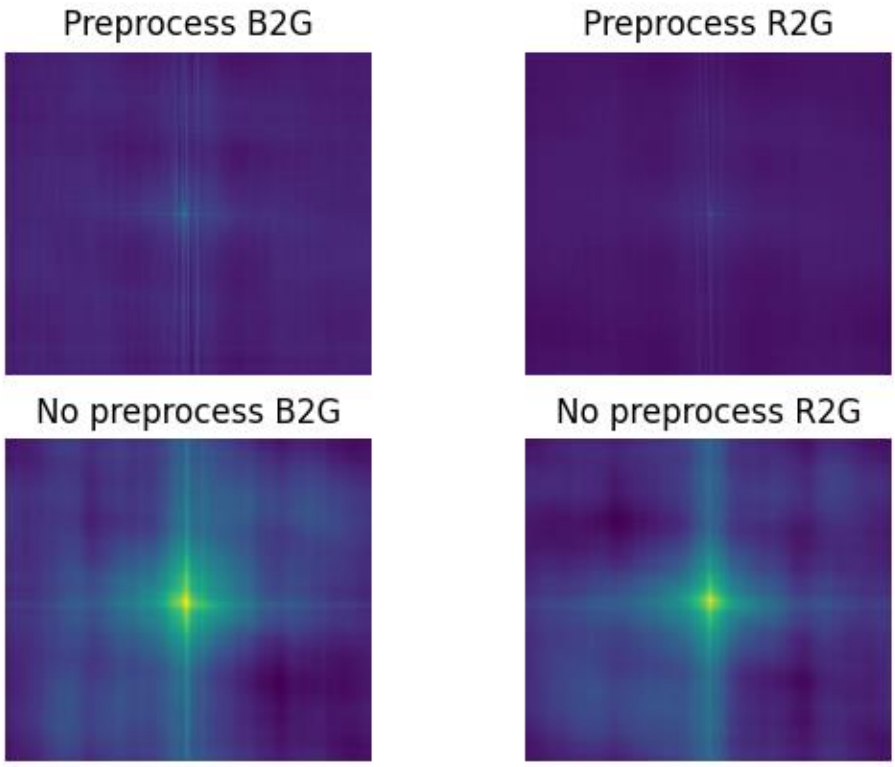
No preprocess B2G



No preprocess R2G



01861a.tif



C: Discussion and Runtime Comparison

Preprocessing:

Similar to the last assignment, I crop the white border by deleting any row/col with a mean value larger than 210.

For the channel preprocessing I implemented Laplacian of Gaussian with an extra step of edge sharpen to further enhance the outline because of one of the pictures (01657u.tif) having difficulties lining up.

```python
def LoG(img):
    img = gaussian_filter(img, sigma=1.5)
    lap_kernal =np.array([  [  0,  0, -1,  0,  0 ],
                            [  0, -1, -2, -1,  0 ],
                            [ -1, -2, 16, -2, -1 ],
                            [  0, -1, -2, -1,  0 ],
                            [  0,  0, -1,  0,  0 ]])
    lap_kernal2 = np.array([[0,-1,0],[-1,8,-1],[0,-1,0]])
    return cv2.filter2D(cv2.filter2D(img,-1,lap_kernal),-1,lap_kernal2)
```

The image would have some noticeable displacements happening comparing to the result image in the previous section without the preprocessing.

I resume the offset of cropping by adding back what was cropped according to base channel

```
g2b_L,g2b_L_ifft = offset(blue_L,green_L)
g2b_L[0] += off//3 # offset of cropping
plt.subplot(221)
plt.imshow(g2b_L_ifft)
plt.title("Preprocess G2B")
plt.axis('off')
r2b_L,r2b_L_ifft = offset(blue_L,red_L)
r2b_L[0] += (off//3)*2 # offset of cropping
plt.subplot(222)
```

Runtime:

*All the time record is timed with jupyter notebook cell timer

The previous assignment I estimated that the alignment of 3 images with no pyramid would take at least 3 hours 22 minutes even for searching a [-60,60] area.

Even with pyramid under a [-15,15] range the runtime would still be 8 minutes and 50.5 seconds

```
✓ 8m 50.5s
[-4, 20] [16, 34]
[-16, 6] [-32, 12]
[-6, 40] [-6, 62]
```

By aligning channels with Fourier transform, we effectively reduced the time complexity from O(N^2 M^2) into O(NM log NM)

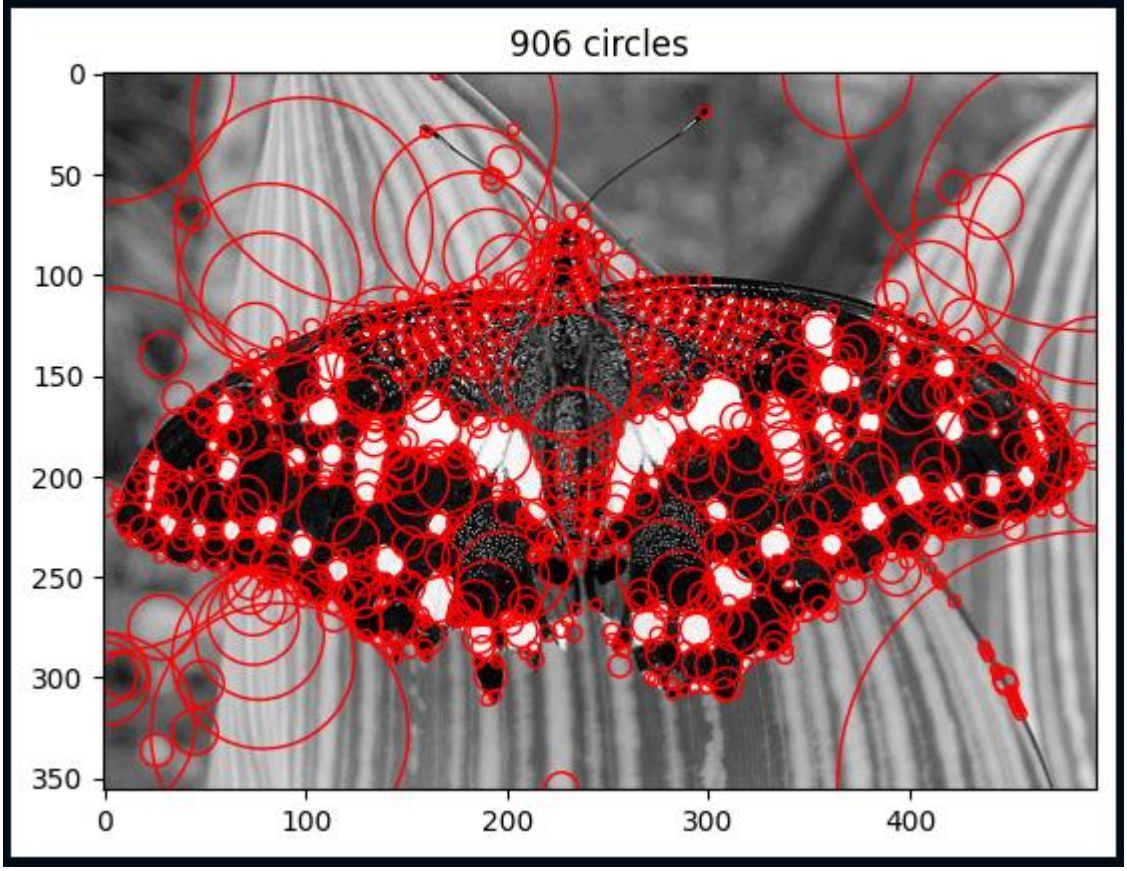
```
✓ 51.6s
01047u.tif (-3, 19) (15, 33)
01657u.tif (-13, 9) (-20, 13)
01861a.tif (-6, 38) (-7, 62)
```

You could speed up the non-Fourier method even faster with lower search ranges, but it wouldn’t be possible without any further knowledge of the input data/image.

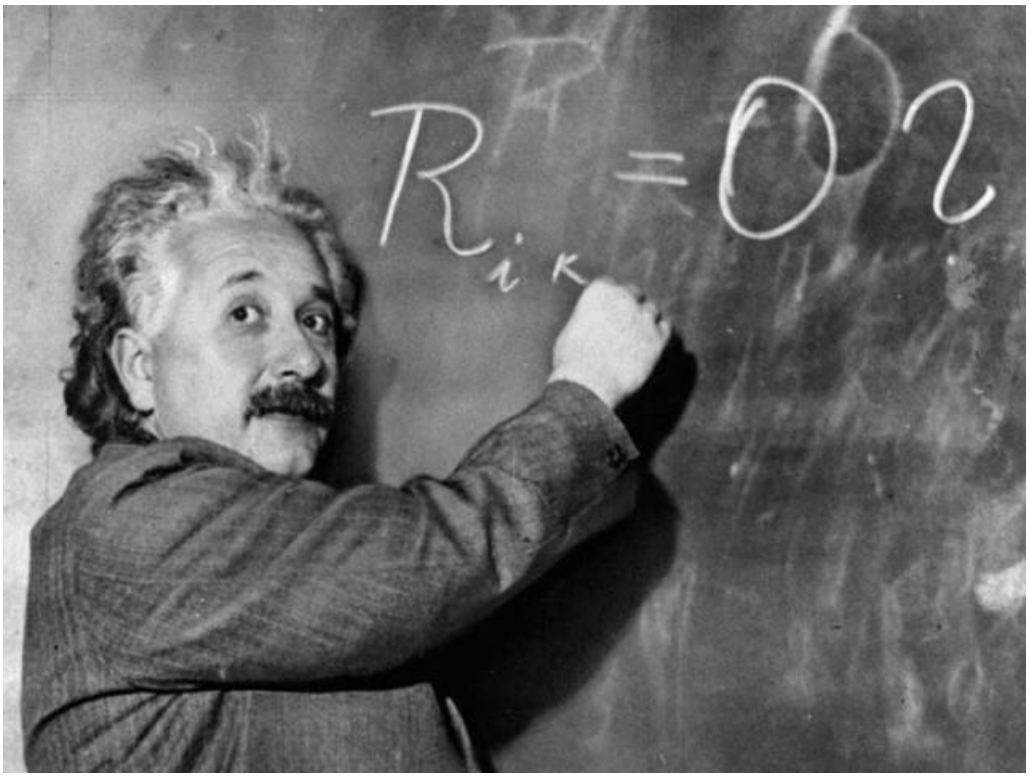
Part 2 Scale-Space Blob Detection:

Example 1:

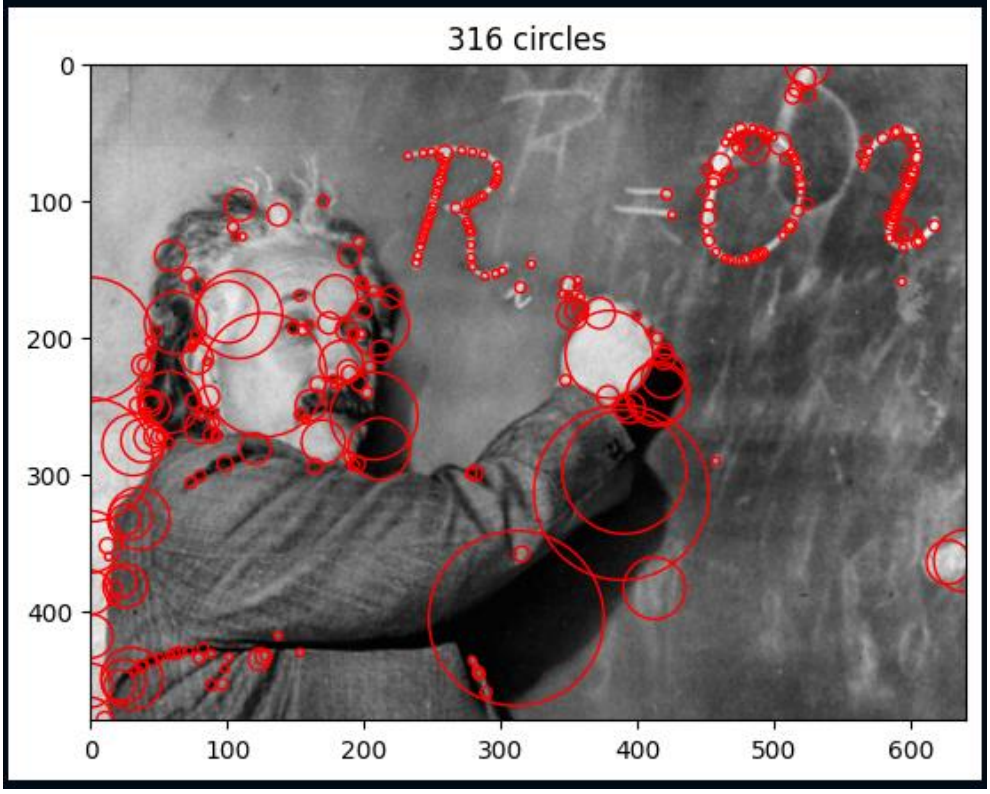
```
increase filter 0.664513111114502
downsample 0.09400582313537598
```



Example 2:

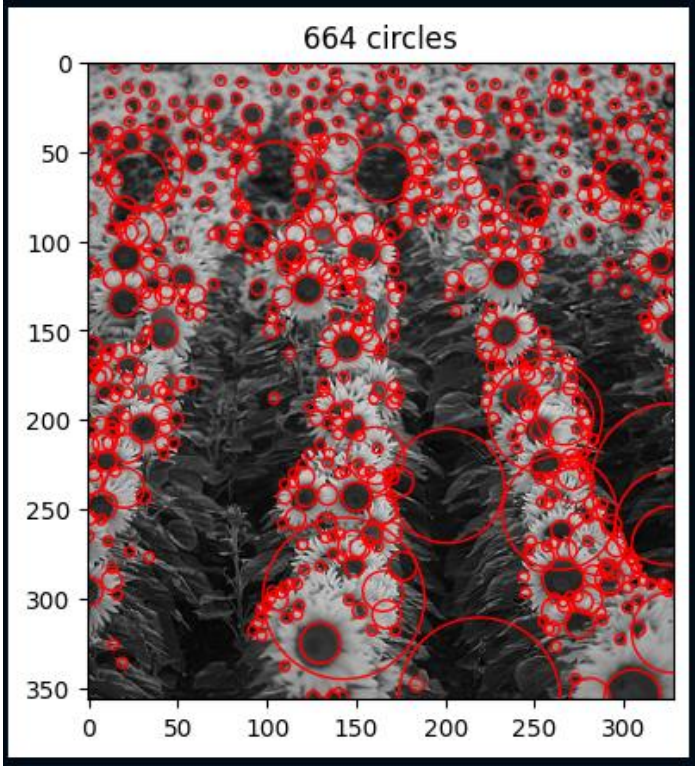


```
increase filter 1.1930091381072998
downsample 0.1780281066894531 2
```



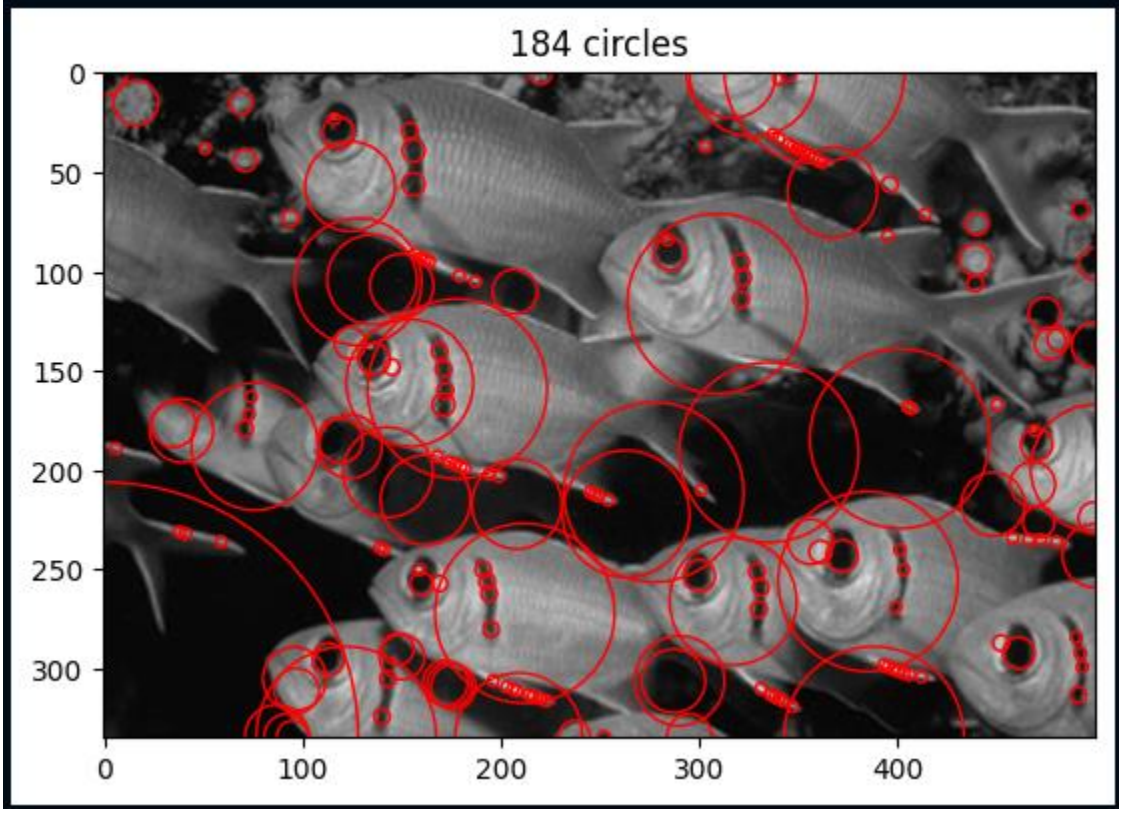
Example 3:

```
increase filter 0.45697927474975586
downsample 0.07101917266845703
```



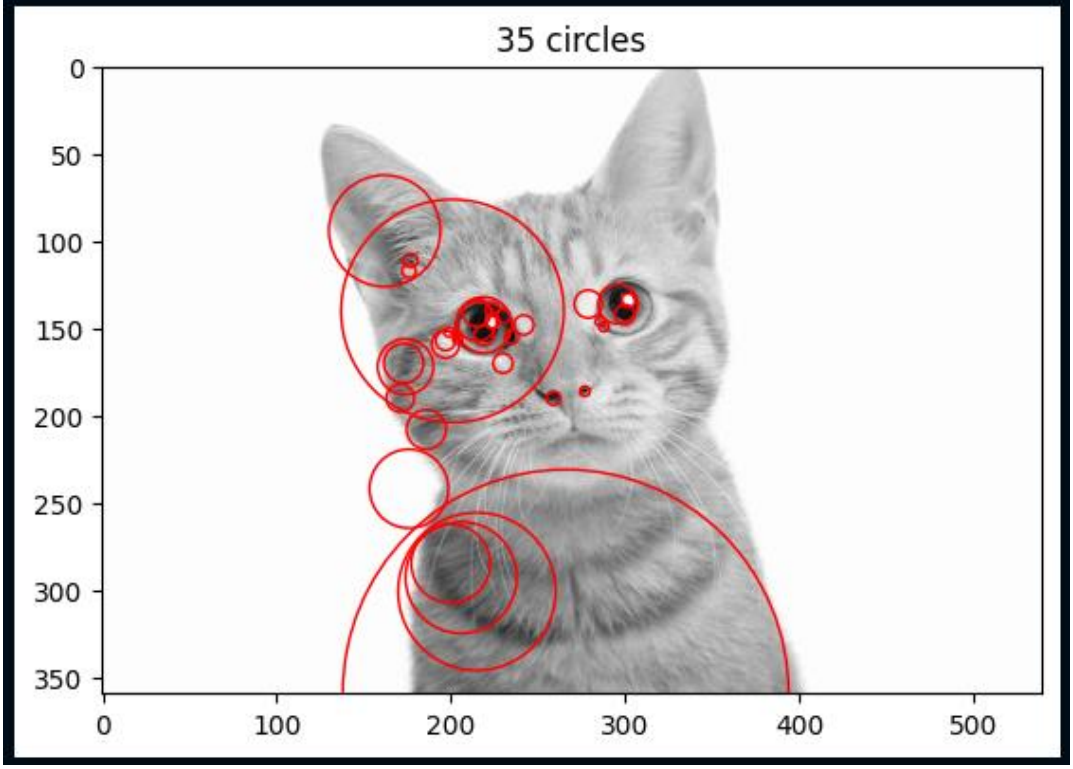
Example 4:

```
increase filter 0.6299874782562256
downsample 0.08798360824584961
```

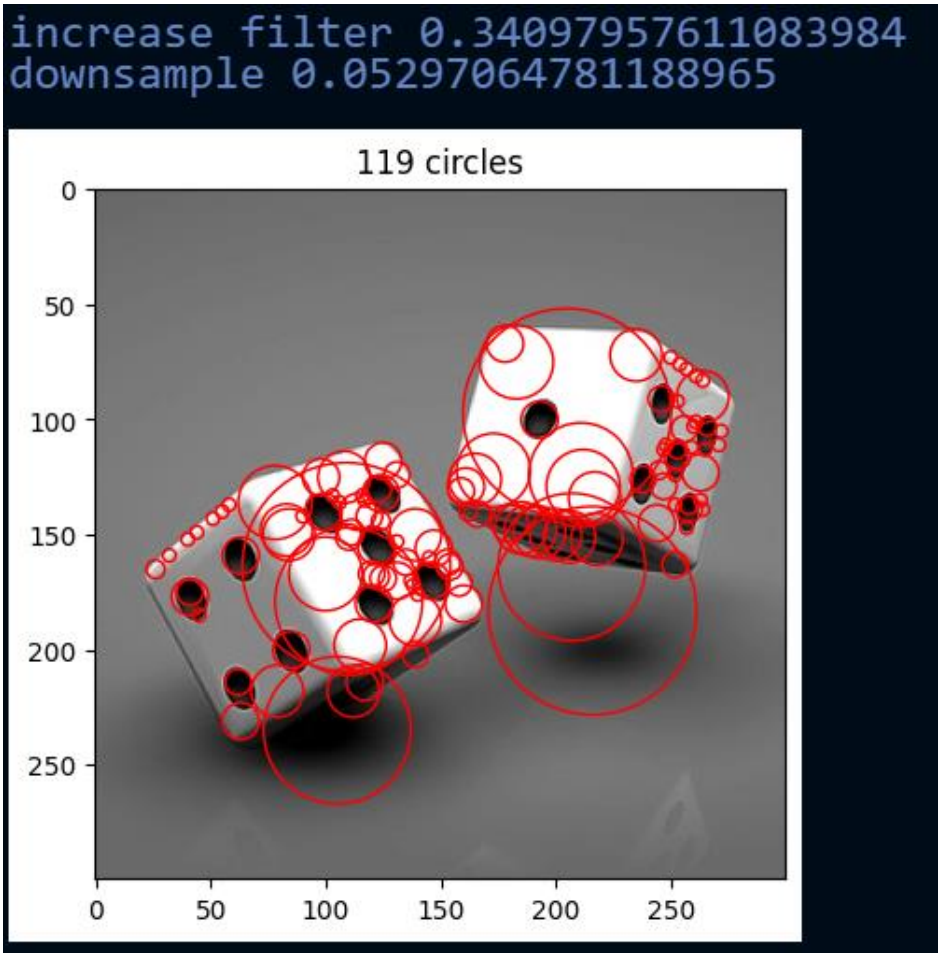


Example 5:

```
increase filter 0.7309999465942383
downsample 0.1060218811035156
```

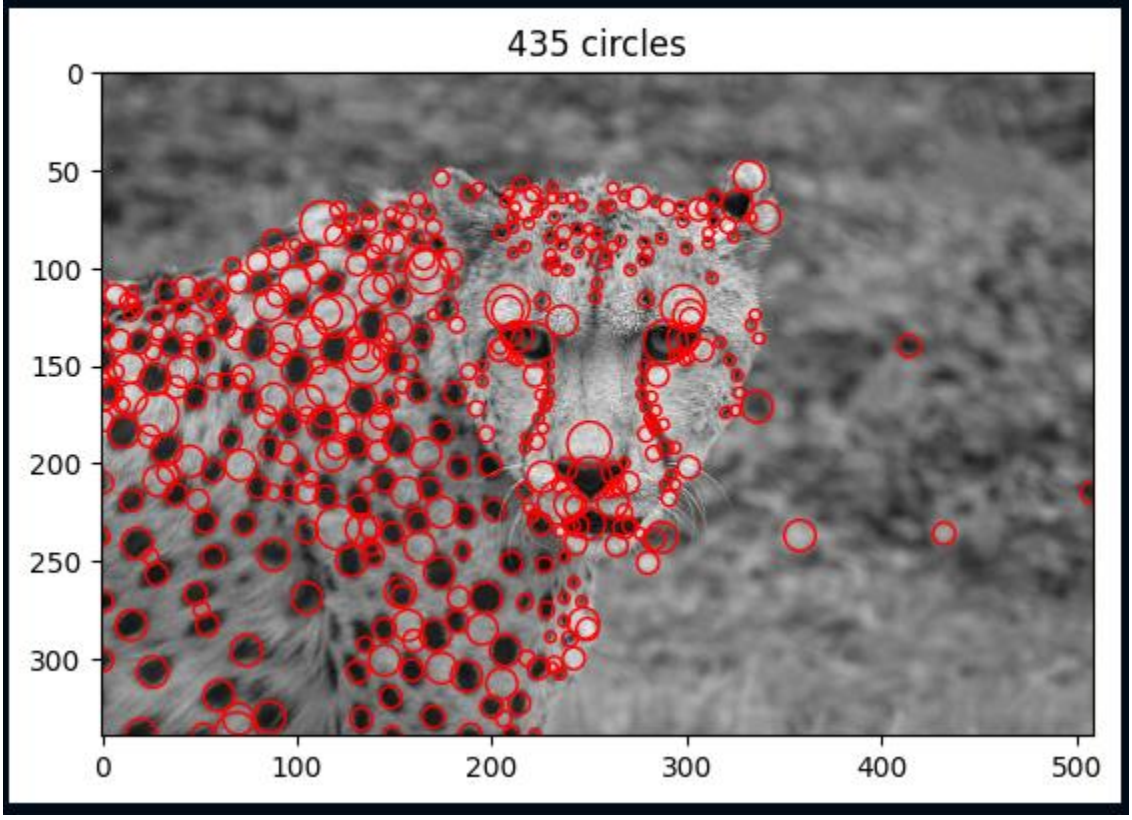


Example 6:


increase filter 0.3409795761108398
downsample 0.05297064781188965
119 circles
0
50
100
150
200
250
0
50
100
150
200
250

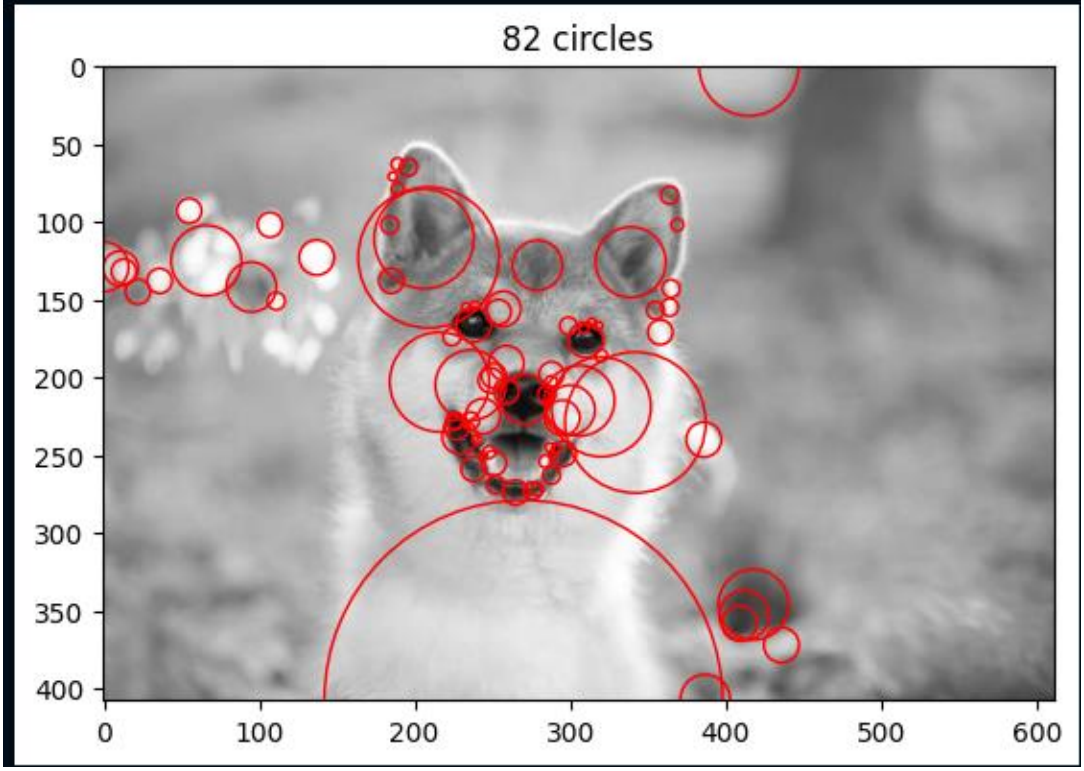
Example 7:

```
increase filter 0.6490204334259033
downsample 0.10004234313964844
```



Example 8:

```
increase filter 0.9399926662445068
downsample 0.1329896450042724 6
```



Discussion:

I used a rank filter instead of generic filter because rank filter was three times faster when I tested it and yielded the same results.

```
for i in range(12):
    local_max[:,:,i] = rank_filter(input=scale_list[:,:,i],rank=-1,size=(3,3))
# for i in range(12):
#         maxx = lambda a:np.amax(a)
#         local_max[:,:,i] = generic_filter(scale_list[:,:,i], maxx, (3, 3))
```

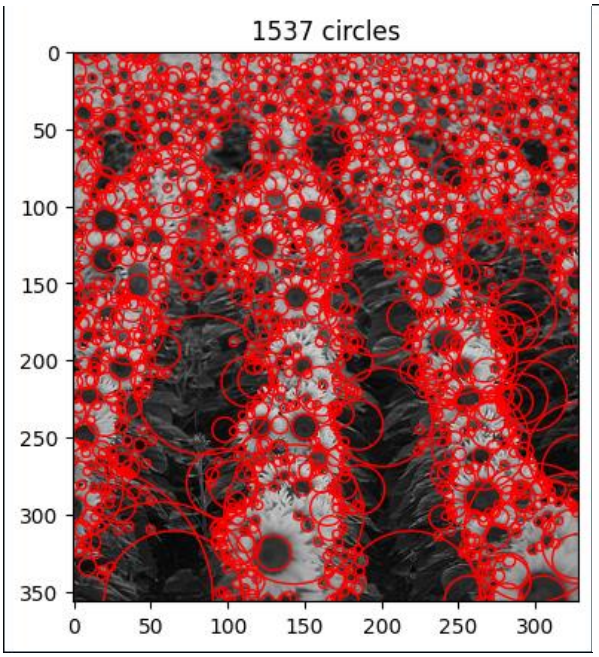
The initial scale I set was 2 and I scale up/down sample sqrt (2) every iteration with 12 in total

```
k = 2**0.5
sigma = 2.0
scale_list = np.empty((h, w, 12))
```

After testing a lot of thresholds, I set my threshold to be 0.02 because I want to really focus more on the actual blobs

```
    idx = np.argmax(local_max[i,j,:])
    if maxx == scale_list[i][j][idx] and maxx >= 0.02:
        maxima.append((i,j,idx))
```

Threshold = 0.005



Threshold = 0.02