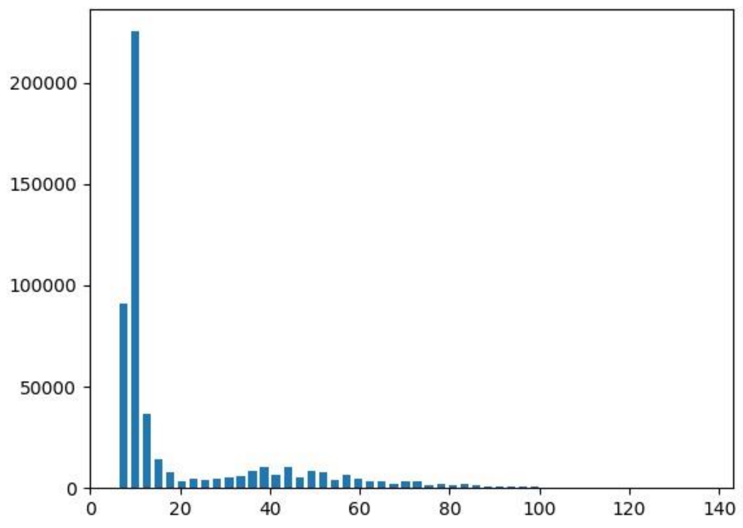
HW1 Cell Segmentation

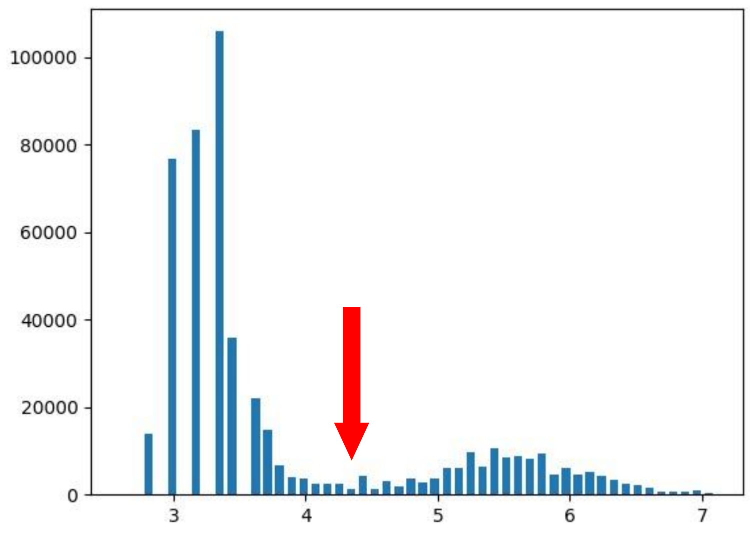
Source code can be seen in cell_segmentation.py:
https://github.com/Tonight1121/Biology-Image-Analysis.git

1. The advantages and disadvantages of using log2

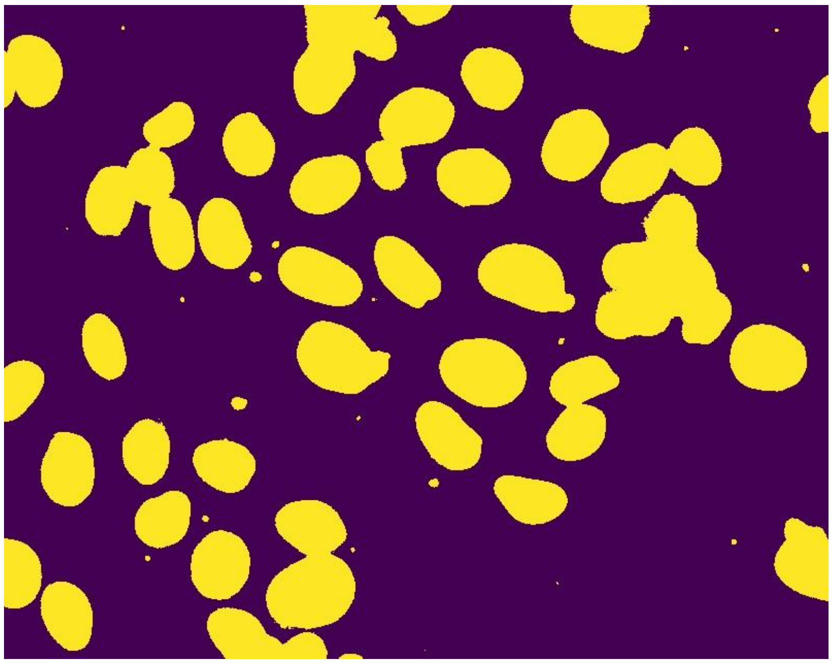
Before applying log2, the histogram appears like:



After applying log2, the histogram appears like:

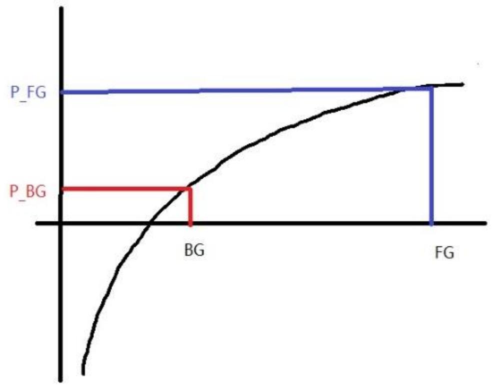


I chose a value 4.4 around here as a threshold. The binary segmentation mask is:



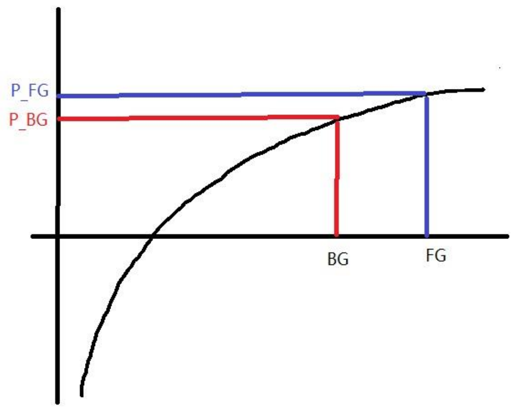
In a typical log function, with the x value grow to infinity, the slope of the function tends to decline consistently, which indicates that different large values can be normalized to equivalent (close to each other) values after applying log2 function. However, values close to 1 will still preserve relative difference from each other after log2.

Advantage: If the mean intensity of background has a significant difference from the mean values of foreground, applying log2 can well separate them apart. Or either background or foreground's mean intensity is close to 1, that is even better, as the slope around 1 is huge and can well separate two values through log2 projection.



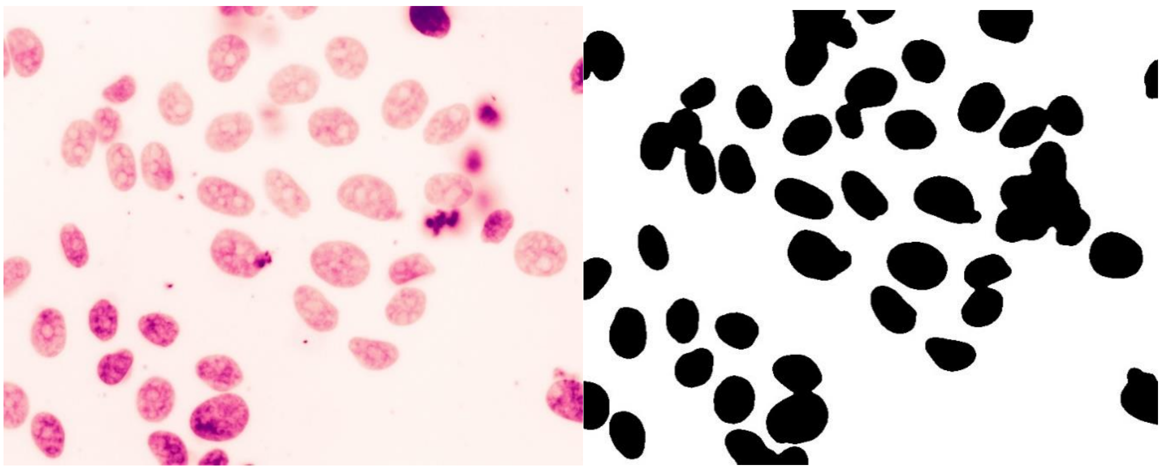
One example of good separation.

Disadvantage: If the mean intensity of background and mean intensity of foreground are all very large values, applying log2 will make these values even closer, which is the opposite purpose of segmentation.



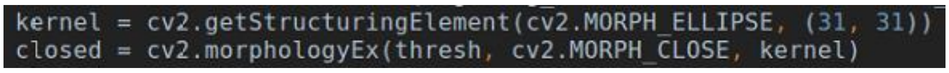
One example of bad separation

2. Images and thresholding results visualization

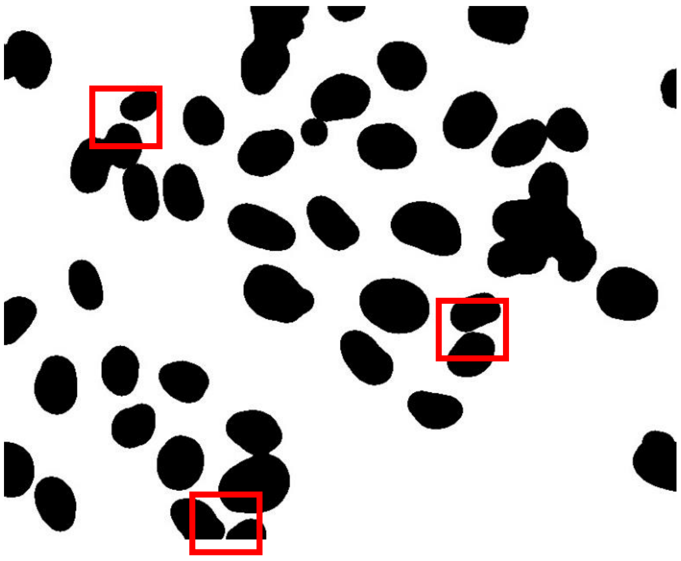


Using RdPu color map for raw image visualization, and segmentation mask visualization. 255 for background and 0 for foreground.

If we apply the following simple preprocessing to the image, we can divide some connected cells into separate ones. But for one disadvantage, the size of each cell got a little shrink.



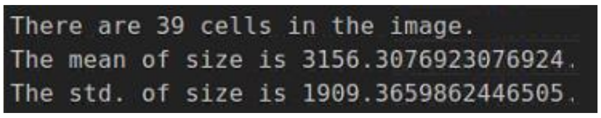
```
kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (31, 31))
closed = cv2.morphologyEx(thresh, cv2.MORPH_CLOSE, kernel)
```



3. Cell analysis

a) Mean and std of cells

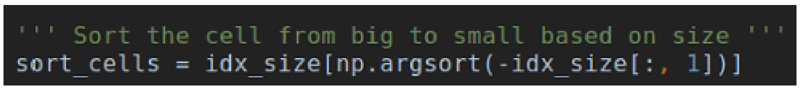
After applying the preprocessing, we are closer to the real number of cells. The size here represent the number of pixels in that connected area. Mean and std are calculated through np.mean and np.std.



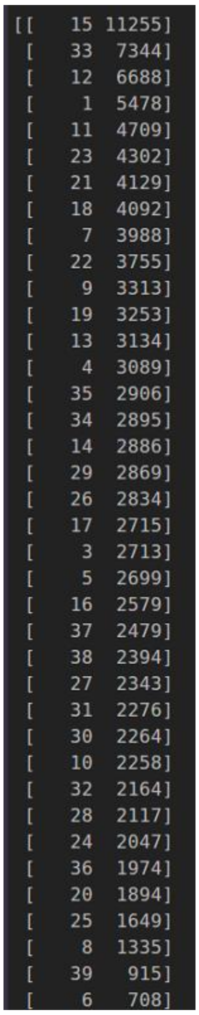
```
There are 39 cells in the image.
The mean of size is 3156.3076923076924.
The std. of size is 1909.3659862446505.
```

b) Sort of cells based on size

I stored each cell's index and size to a 39*2D numpy array, and sort by the second column which stands for the size.



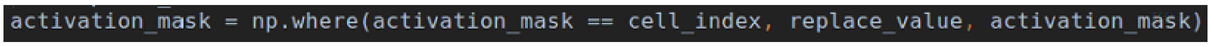
```
''' Sort the cell from big to small based on size '''
sort_cells = idx_size[np.argsort(-idx_size[:, 1])]
```



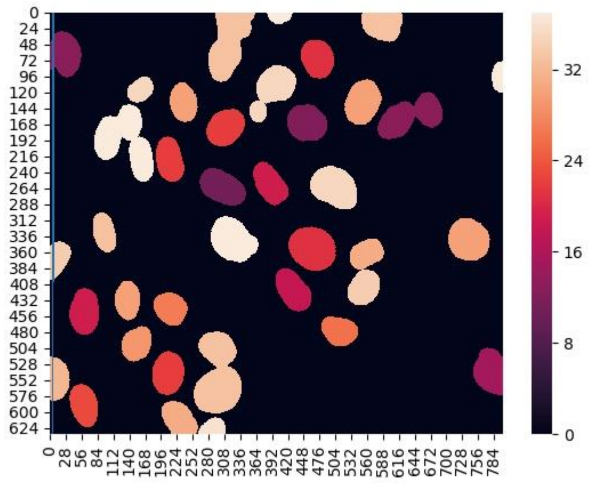
```
[[   15 11255]
 [   33  7344]
 [   12  6688]
 [    1  5478]
 [   11  4709]
 [   23  4302]
 [   21  4129]
 [   18  4092]
 [    7  3988]
 [   22  3755]
 [    9  3313]
 [   19  3253]
 [   13  3134]
 [    4  3089]
 [   35  2906]
 [   34  2895]
 [   14  2886]
 [   29  2869]
 [   26  2834]
 [   17  2715]
 [    3  2713]
 [    5  2699]
 [   16  2579]
 [   37  2479]
 [   38  2394]
 [   27  2343]
 [   31  2276]
 [   30  2264]
 [   10  2258]
 [   32  2164]
 [   28  2117]
 [   24  2047]
 [   36  1974]
 [   20  1894]
 [   25  1649]
 [    8  1335]
 [   39   915]
 [    6   708]
```

c) Color code the cells based on size rank

After sorting the array according to the size, I replace the value in each cell of ILabel with its rank in the sorted array. Because we are going to draw the heatmap in a robust way, there is no need to normalize anymore.



```
activation_mask = np.where(activation_mask == cell_index, replace_value, activation_mask)
```



0
24
48
72
96
120
144
168
192
216
240
264
288
312
336
360
384
408
432
456
480
504
528
552
576
600
624
0
28
56
84
112
140
168
196
224
252
280
308
336
364
392
420
448
476
504
532
560
588
616
644
672
700
728
756
784
32
24
16
8
0