Image Processing

Color Classification Project

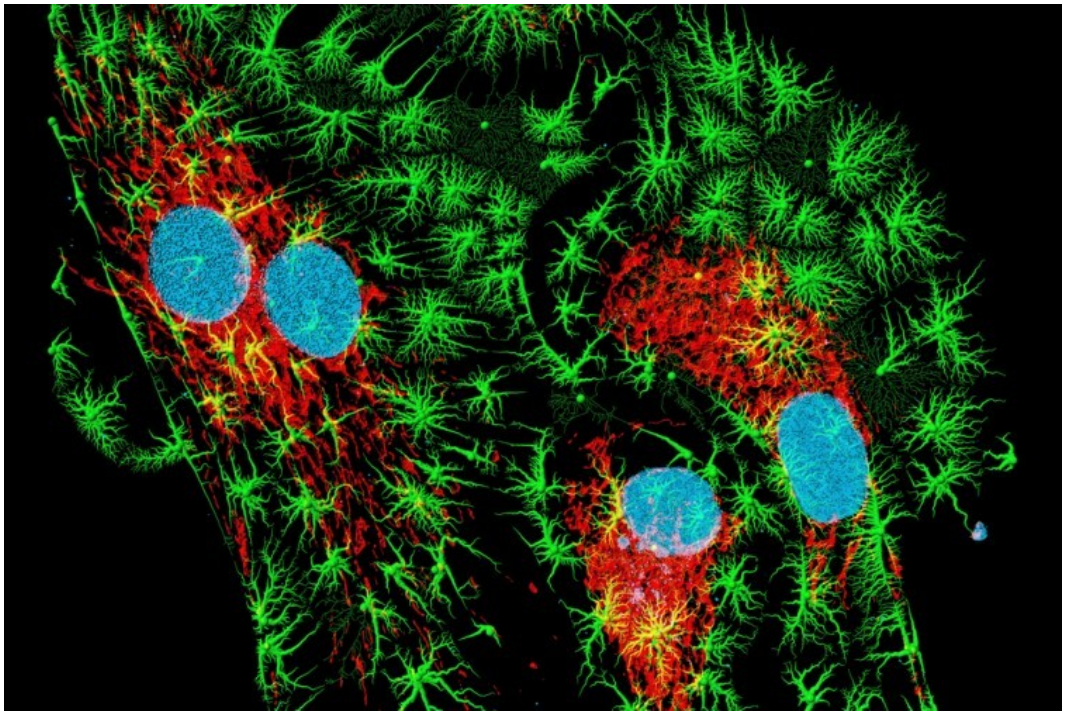
The goal of the project is to create an algorithm that will classify image objects (different cells or cells and tissues) based on their color characteristics (fluorescence color), and calculate the image surface coverage percentage of each object (Object Surface / Total Surface)

The algorithm works like this: First it splits the image to 3 images and then it creates 3 masks, one for each color channel. Each mask is tuned so that 1 color overpowers the other two in intensity. Take red for example:

> Red > Blue/2+ Green/2 + C (1)
> where C is a constant parameter specified by the user.

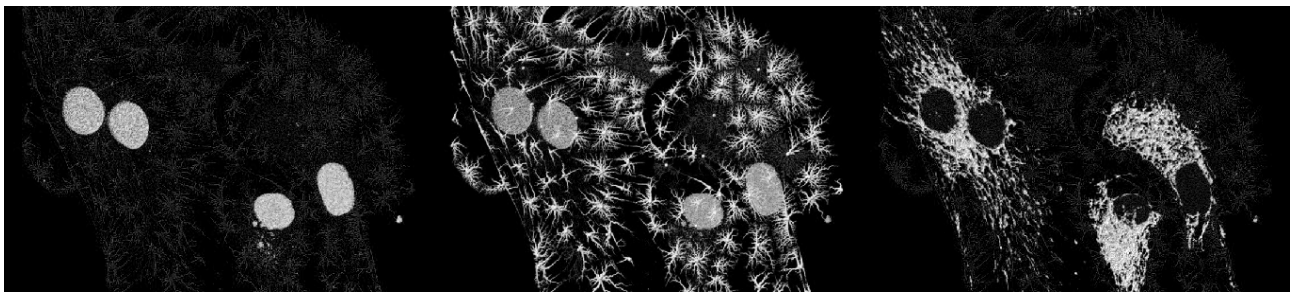
The program requests the C parameters from the user. Use the -C operator and input 3 numbers, one for each color. Use the the k1 and k2 operators to determine the size of kernels to be used in the denoising process (open and close respectively). If set to 0 the corresponding operation will be skipped. Again use 3 numbers to denote a kernel for each channel.

> fiber-optics-fluorescense-microscopy.jpg

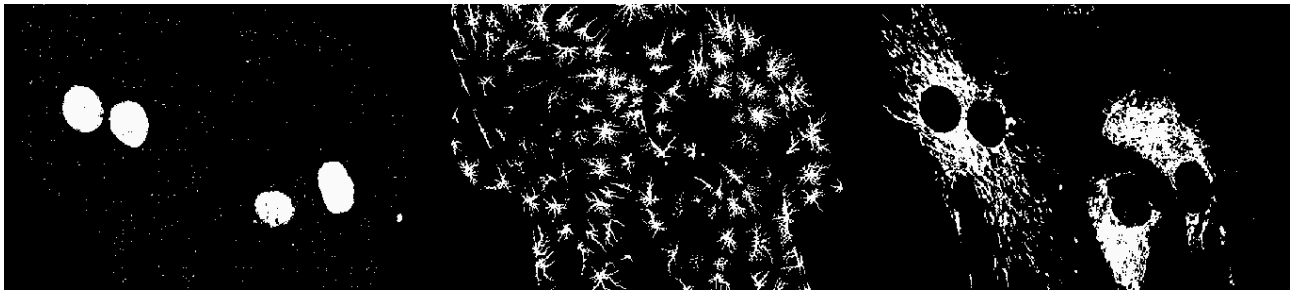


In order to process this picture use this command :
python ColorClassification.py fiber-optics-fluorescense-microscopy.jpg -C 20 100 50 -k1 5 0 0 -k2 5 0 0

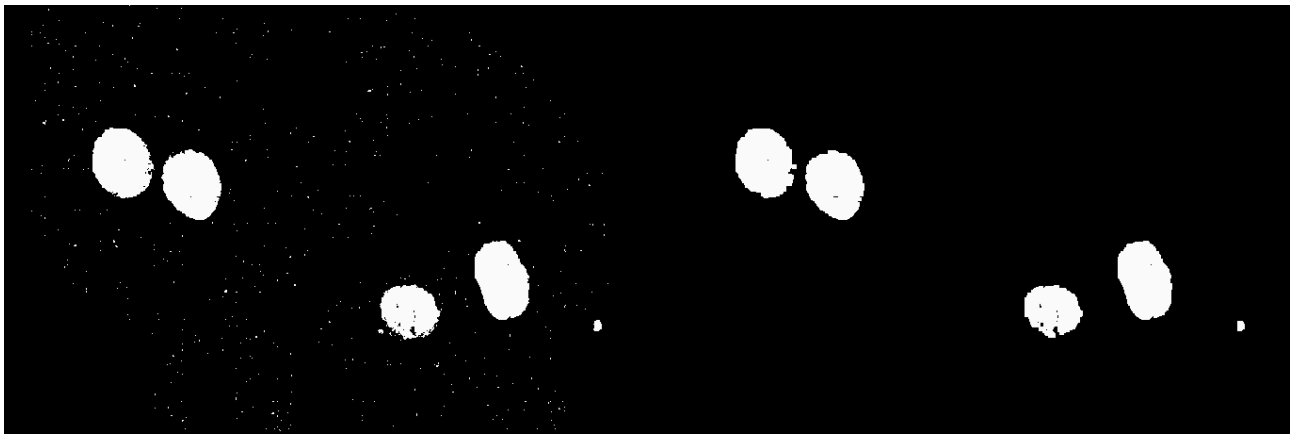
The image is first split into 3 pictures, one for each channel of RGB



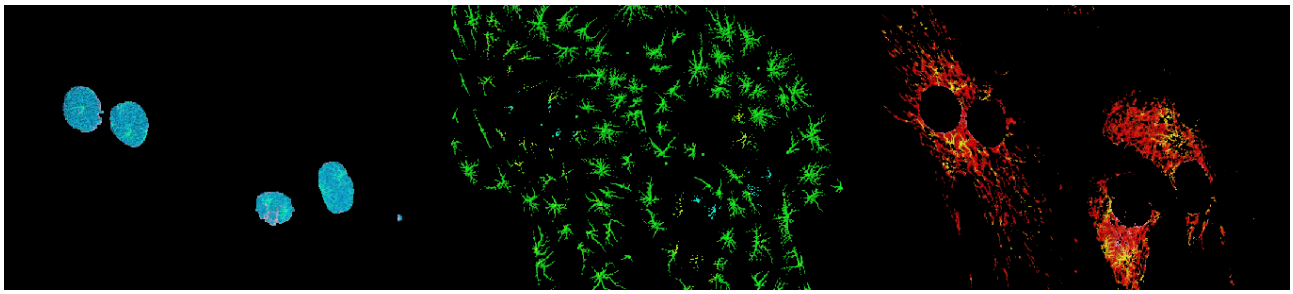
In the blue and the red channels the objects seem to be clearly distinct from the rest of the image. The green channel is gonna need a more strict threshold to get rid of the blue cells. In the next step masks are created, starting from a full 0 matrix of the same shape as the images. Then every number of the matrix becomes 250 if the corresponding pixel in the previous image follows the inequality (1). Pixels that simply have high intensity (>200) are also kept. After experimenting I have arrived to the conclusion that (20, 100, 50) provide the best results. As was expected the green channel needed a bigger threshold for the tissue to become disentangled from the blue cells.



Next step would be to remove noise where needed. Denoising tissue would only corrupt the picture and doesn't seem to be required. The blue channel, on the other hand, does need denoising. The parameters related to this are the kernels as mentioned above. To understand how the denoising is done we must understand what erosion and dilation mean. Take erosion for example, the kernel slides through the image and if any pixel under the kernel is a 0 then all pixels under the kernel become 0 (are "eroded"). This causes pixels near boundaries to be discarded and white holes to disappear. Depending on the size of the kernel the boundaries may erode greatly or we may be able to close up bigger holes. Dilation is simply the opposite, where a pixel becomes 250 if at least 1 pixel under the kernel is 250. With that we can close black holes. The operations used here are opening and closing. Opening is erosion followed by dilation while closing is dilation followed by erosion.
The kernel size that performed best after testing was k1 = 7 (open) and k2 = 5 (close).
The before and after pictures are demonstrated below:

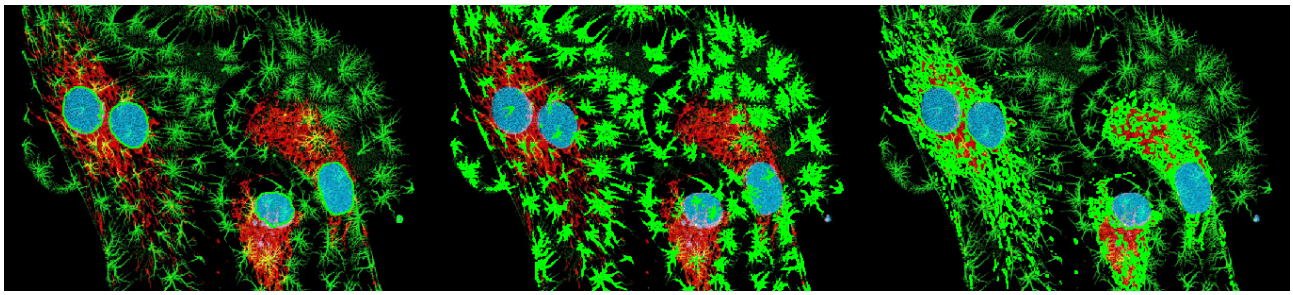


Finally by applying the masks on the original image we end up with our 3 distinct objects:



The objects have been successfully recognized, including even the green fibers hiding behind the blue cells!

The last step is to apply contours around the objects in order to calculate surface area.



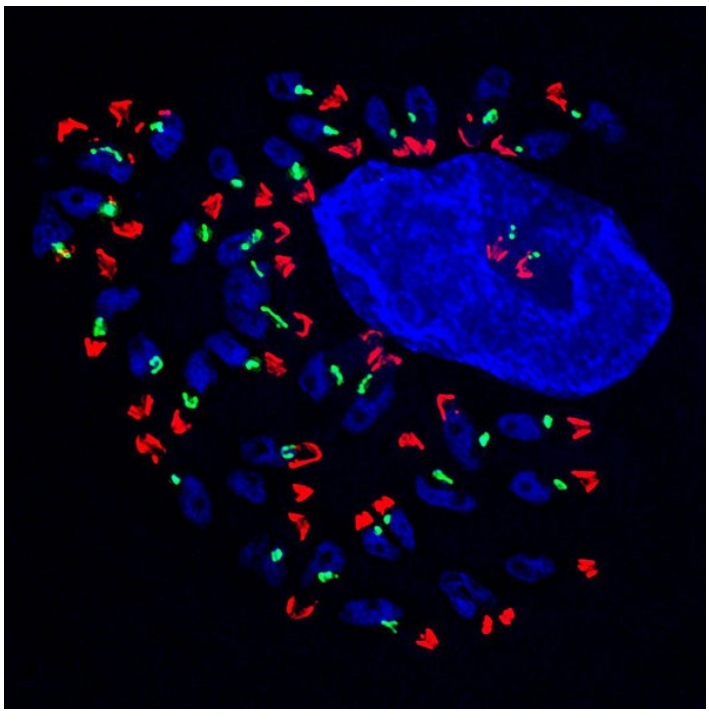
Based on these contours the calculated area coverage surface is respectively:

Blue Object: 4.19% Coverage

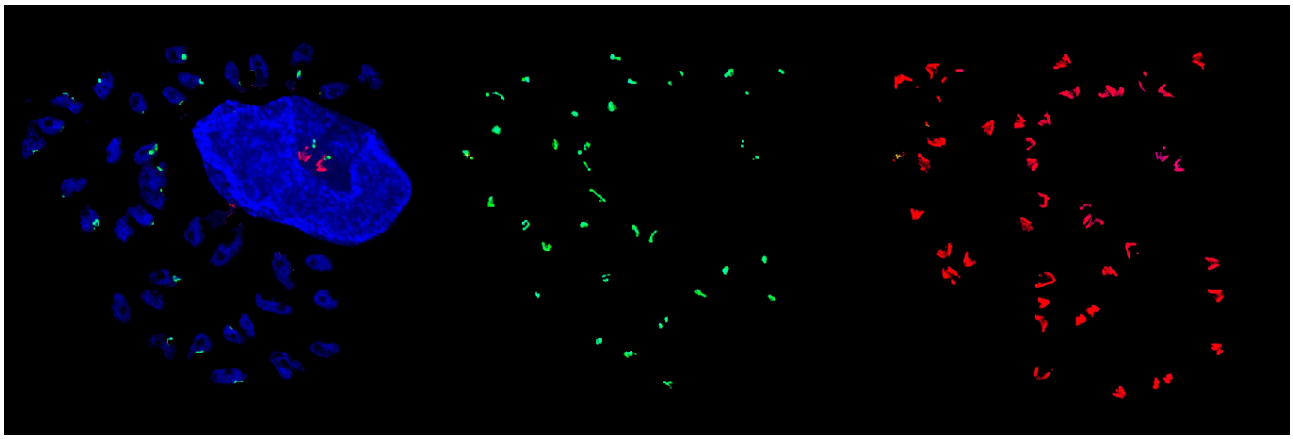
Green Object: 7.77% Coverage

Red Object: 14.9% Coverage

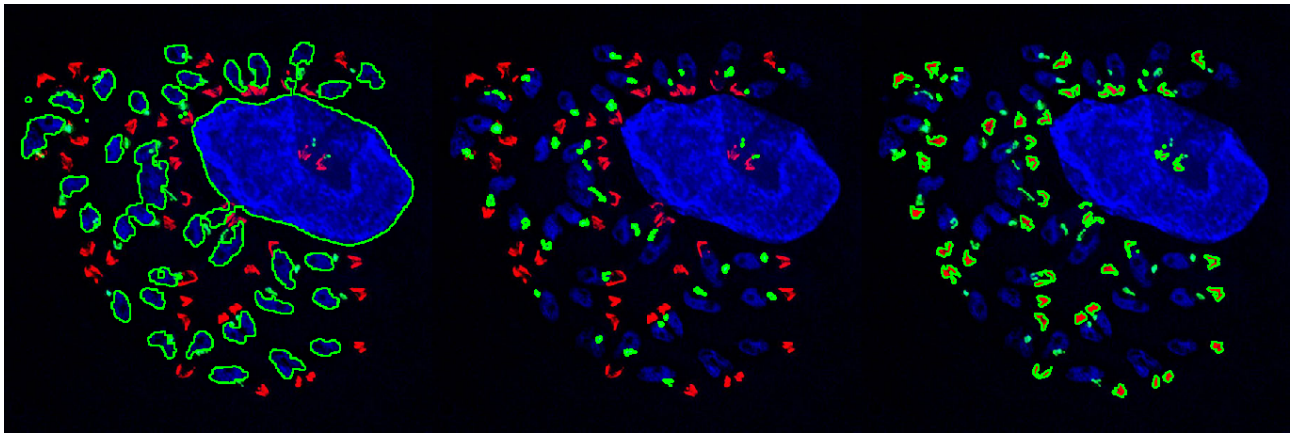
Next picture python ColorClassification.py lg-VR0qQ1440.jpg -C 30 70 50 -k1 4 0 0 -k2 7 0 0



Following the same process we disentangle the objects:



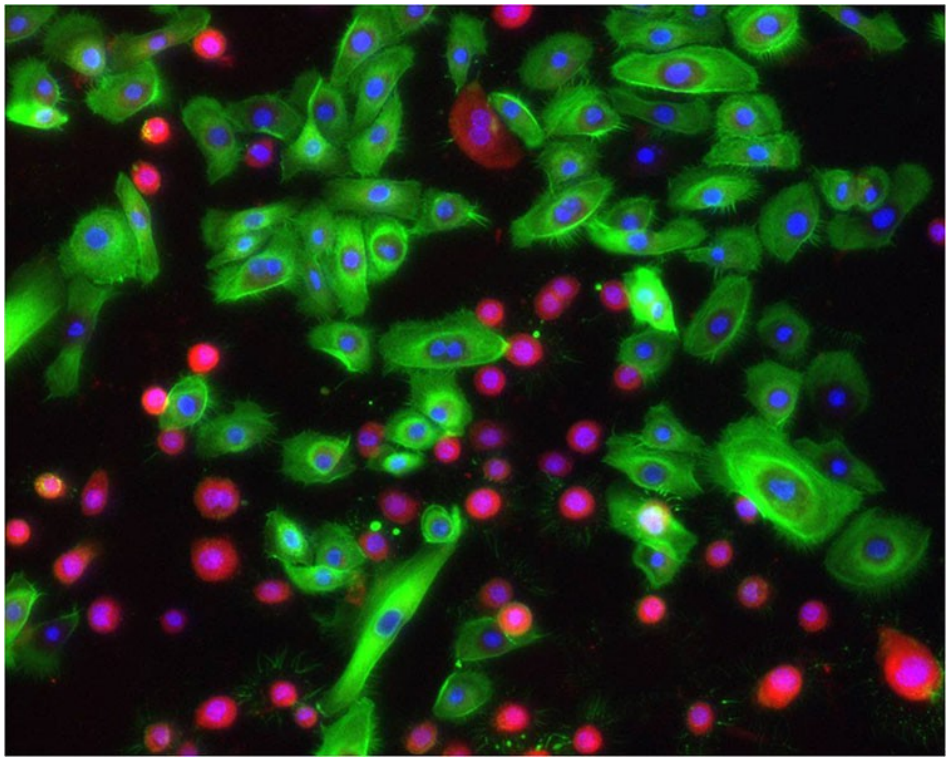
And then use contours:

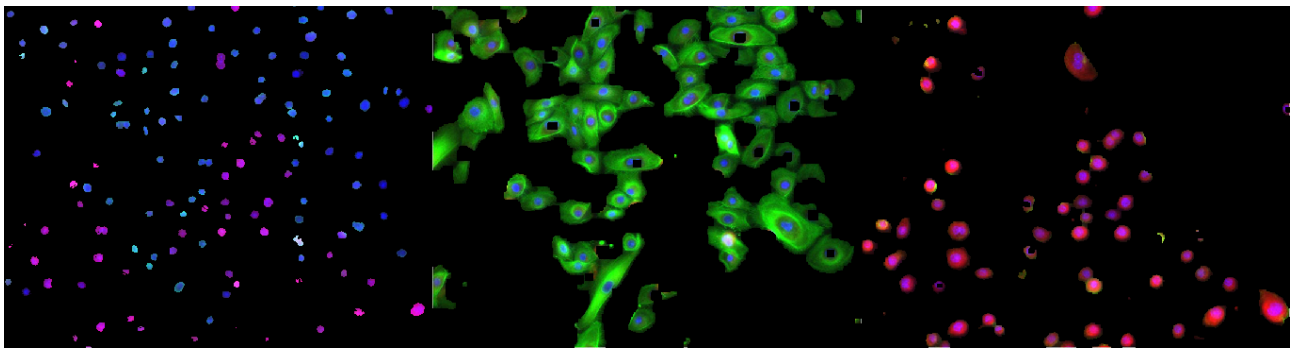


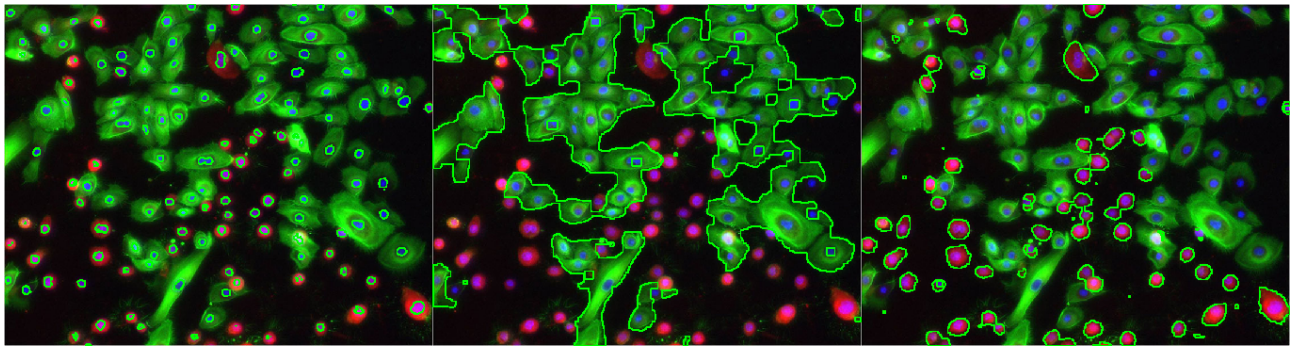
Blue Object: 20.92% Coverage
Green Object: 0.58% Coverage
Red Object: 2.08% Coverage

Finally:

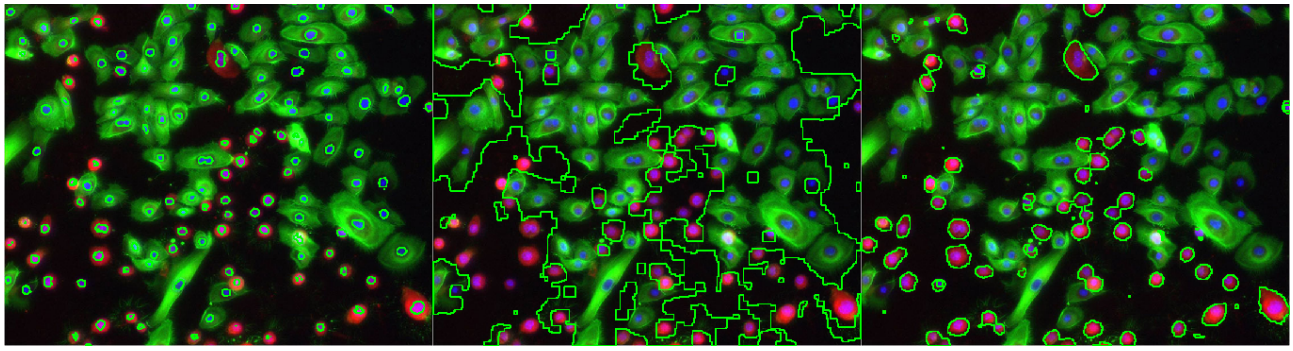
python ColorClassification.py lg-ksRLq1280.jpg -C 30 30 20 -k1 0 6 3 -k2 0 13 8







The results on the green cells aren't 100% satisfactory. We can see that several shadowed cells are missed, but trying to reduce the Green intensity threshold (C) in order to include them also adds a lot of noise:



I also experimented with the HSV color space trying to increase the brightness (not included in current code) but this also added noise. There are techniques to remove shadows which may have helped our case. These are however advanced techniques so I did not delve into them: https://web.engr.illinois.edu/~dhoiem/publications/cvpr11_shadow.pdf

The nuclei on the other hand (blue) are recognized efficiently. The red cells are also detected without any cell being missed, despite some slight noise. Using the open operation we manage to close most holes that are left from the blue nuclei.

So, with the original parameters the surface coverage is :

Blue object: 3.75% Coverage

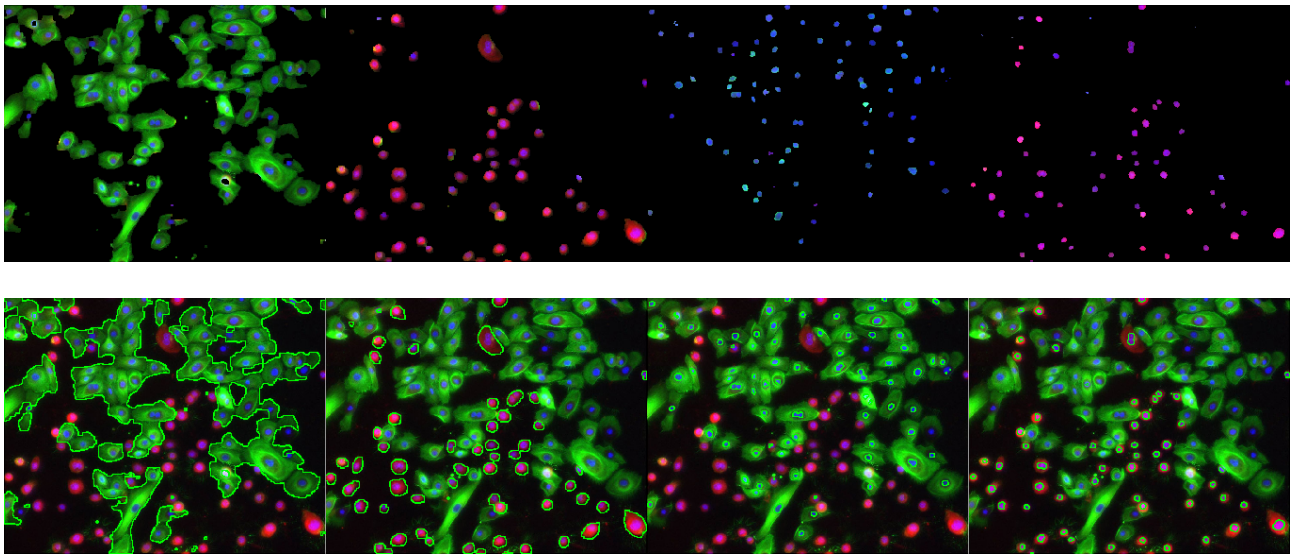
Green object: 38.38% Coverage

Red object: 7.84% Coverage

Finally, in order to handle the special case of analyzing nuclei surface compared to the separate cells we need to distinguish the two types of nuclei from the blue channel. In order to do that I separated the two nuclei by setting two distinct accepted ranges of color. First we define a copy of the blue image as RedNucleus = img_b.copy(). Then we zero out pixels that are lower than 80 on the red channel, lower than 100 on the blue channel and higher than 100 on the green channel. The rest of the pixels are set to 250 in order for the final image to be binary. Similarly for the BlueNucleus starts out as a copy of the blue image and then pixels with red > 100, b < 100 and green > 20 are zeroed out. In order to arrive at these values I zoomed in the image to better understand which range better fits our needs. Afterwards the original GreenMask is combined with the BlueNucleus to be sure that we have the whole green cells (same for RedMask and RedNucleus). In order for the user to activate this mode all he needs to do is type -cells.

The kernels in this case correspond to [Green Cell, Red Cell, Nuclei]. Note that there is no 4th kernel argument even tho we basically have 4 images in this case. Simply the last kernel argument is used in both nuclei images.

python ColorClassification.py lg-ksRLq1280.jpg -C 30 30 20 -k1 5 6 3 -k2 7 6 3 -cells



The blue nuclei cover 5.91 percent of the total green cell area
The purple nuclei cover 23.02 percent of the total red cell area