Mekiran Srividhyadharsan Student ID: 4248976

Counting Cell Nuclei – G52IIP Coursework

Introduction

By running the MATLAB code provided, this one program can take any of the three original confocal laser microscopic images and output binary images plus the number of cell nuclei in each image. In this report, I will discuss the steps I took; highlighting alternate options and go through the strengths and weaknesses.

Colour Space Conversion

The first main challenge was deciding on which colour space to work with. Colour space relates numbers to colours in images and is a three-dimensional object which contains all the colour combinations. In order to better visualise more than one colour space at a time, colour spaces are often repsented using two-dimensional slices from their full three-dimensional shape. There are many colour spaces and each have their own strengths and weaknesses. The most common type is Red-Green-Blue (RGB), used in image acqusition but does not always need to be used with image processing. MATLAB has built in functions to convert images from one colour space to another but you can also write code to specify changes in the scaling sensitivity, and allow for greater accuarcy.

While I initally though HSV would be the best, as this colour space is used the most for colour and this task dealt with two colours containg two sets of data, going back to the RGB image, I realised that each colour plane in the RGB image (red,green, blue) is a single valued (char) image that I can visually display. I chose this method in my proram because it allows me to separate my cell nuclei which is in the green colour plane, from my cell wall which is in the red colour plane, allowing me to focus the rest of the image processing exactly on what the task required, counting cell nuclei. Taking StackNinja1 as an example , the figures below describes what I mean. I could isolate the colour plane into one variable (G) and continue the image processing focusing on the cell nuclei.

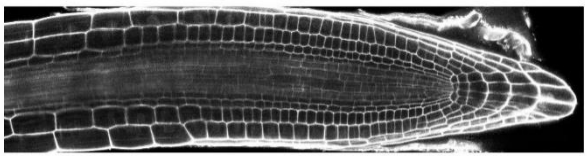

Figure 1: StackNinja1‘s Red Colour Plane

Figure 2: StackNinja1‘s Green Colour Plane

In terms of strengths and weakness of this particular implementation, the strength is that it uses the original image so no scaling or conversion has to be done, which means that there is less complicated code and maths to deal with. The focus of the next image processing task is focused solely on the cell nuclei. The only weakness of this approach is that it is heavily reliant on the image quality of the confocal microscope images and the staining. In theory, each of the cells must have one nuclei each but that doesn’t seem the case from these images. Also, no conversion is happening here in term of brightness of the image and some of the original images do have darker and lighter regions which could potentially be fixed by HSV in terms of saturation or brightness value.

Noise Reduction

Looking at figure 3, we can see that there is some noise in this image. For the rest of the noise reduction analysis, I decided to focus on a section which is shown below for convenience and to better showcase the removal of the noise.

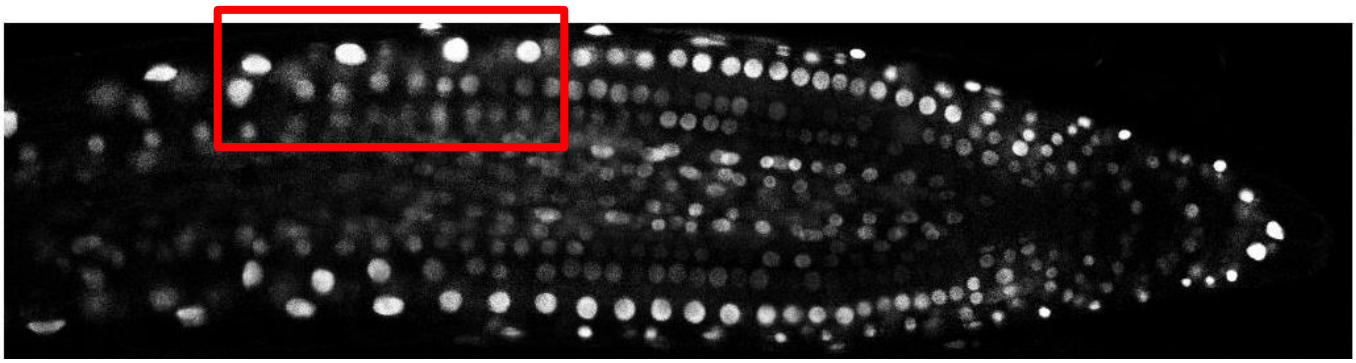

Figure 3: StackNinja1's Green Colour Plane original

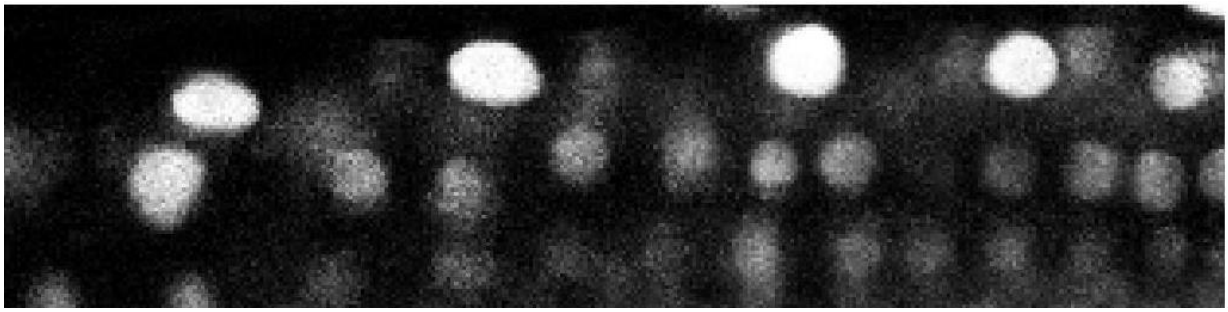

Figure 4: StackNinja1's Green Colour Plane original – focused spot for noise reduction analysis

Noise refers to small errors in pixel values during the image acquisition process, causing a false reflection on the intensities of the real image. Initially, I focused on Gaussian Noise smoothing to remove the additive noise. Gaussian smoothing works by using the gaussian filter to set each pixel to the mean of the surrounding pixels, which will reduce any local variations. There is a MATLAB function called 'imgaussfilt' that is used for 2D Gaussian filtering of images which normally has a standard deviation of 0.5 but you can specify the standard deviation (sigma). To change the mask size, you can write code to manually create Gaussian Filters using 'fspecial' where you can choose the mask size (size of the filter like 3x3 or 5x5). You then use the 'imfilter' command to apply the filter to the image. However, since MATLAB default size is 3x3 and the changes are so small either way, I decided to just use 'imgaussfilt'. I did decide on a sigma value of 1 over the normal 0.5 or above. When it was 2 or above, I felt that the overall image was smoothened out too much and detail was being lost, so I kept it on 1.

I then compared the linear filtering with median filtering as shown in figure 5. Overall, while again there is slight difference, median filtering sets the average value to median value instead of mean. In theory, the median filter is used for salt and pepper noise; is thought be better at edge preserving and is less sensitive to outliers allowing for the sharpness of the original image to be maintained. I did also implement adaptive filtering using the 'wiener2' command but I thought median filtering looked the best as shown in figure 5.

The strength of median filtering is that the additive noise has been removed and there isn't a loss of detail caused by the smoothening effect of the gaussian filters. The weakness of this method is that you could technically do more. There is a way to do adaptive median filtering but that required a lot of code to write and test and although that could arguably be the best method for noise removal, considering the results we are achieving, that seems unnecessary as image quality is quite good with what has already been implemented. I doubt that there would be a significant change regardless.

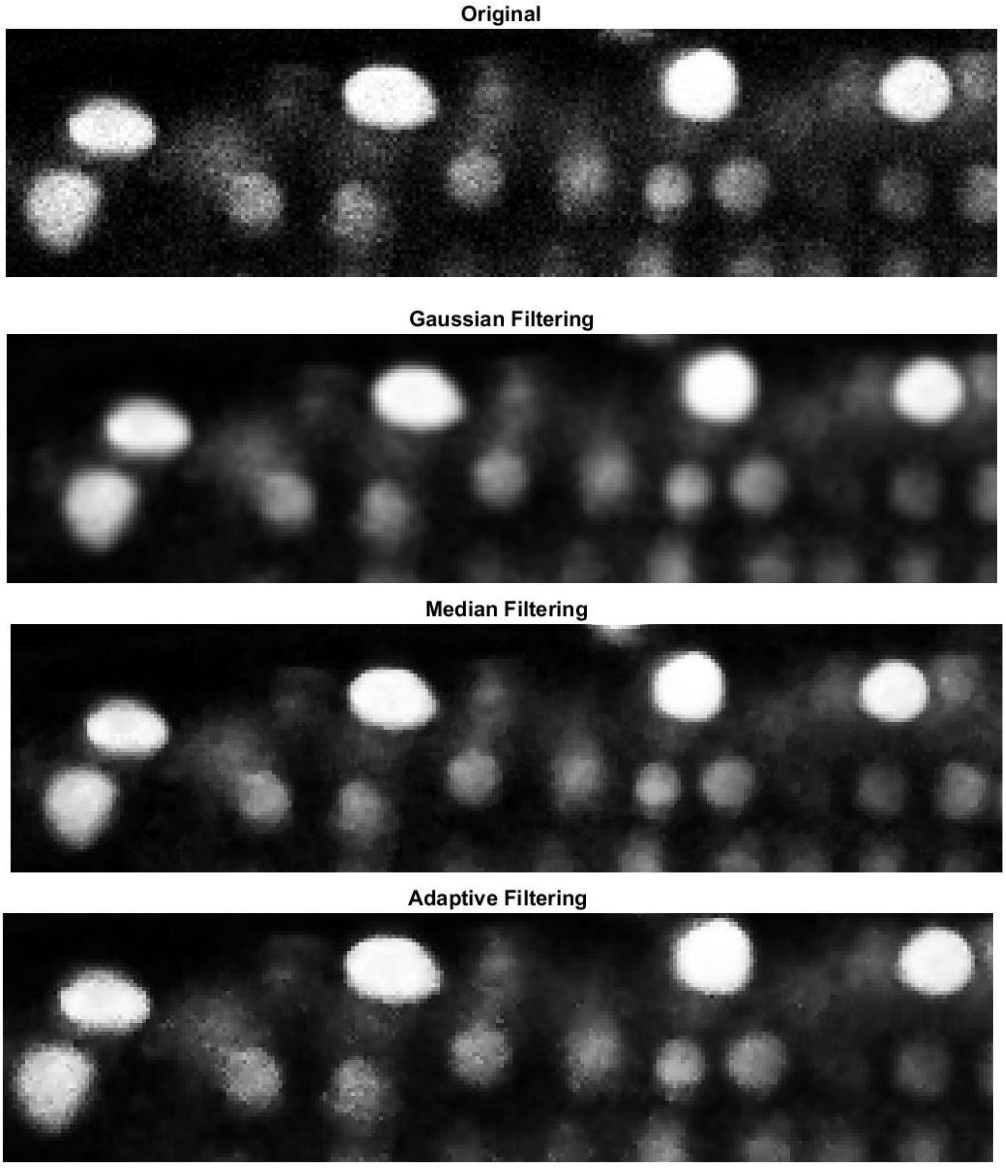

Figure 5: Focused spot for noise reduction analysis, original (top) compared to Gaussian filter with a standard deviation of 1 (second), median filtering (third) and adaptive filtering (bottom)

Thresholding/Segmentation and Binary image processing

I have decided to combine these two sections as they work in tangent with each other. Image thresholding is when you partition an image into the foreground and background by assigning an intensity value T (threshold) for each pixel. There are two types of thresholding, global and local, and there are two corresponding commands, 'graythresh' and 'adaptthresh'. In my program, I started with 'graythresh' which carries out global thresholding using the Otsu's method. You can also get an EM (effectiveness metric) which is a value in the range of 0 to 1 that represents the effectiveness of the thresholding procedure. With the 'imbinarize' command you can create a binary image by replacing all values with either 1s or 0s using a globally determined method. By default, a 256-binary image histogram is used to compute Otsu's threshold. The 'strel' command creates a flat morphological structuring element. Since we are looking at cell nuclei, the shape of the cell structuring element should be a disk. We can set the radius of the structuring element and the number of the lines (has to be even). If the radius is too high, very little cells would appear. Finally, I used both the 'imerode' command with the structuring element to erode the image and the 'imdilate' command to dilate the image. The weakness of this particular approach is that the global thresholding command was clearer and had a finer image quality. However, it was not showing all the cells, so I had to test and finetune the local adaptive thresholding method so that it was clear enough to be used. Figure 6 shows the comparison of the final output.

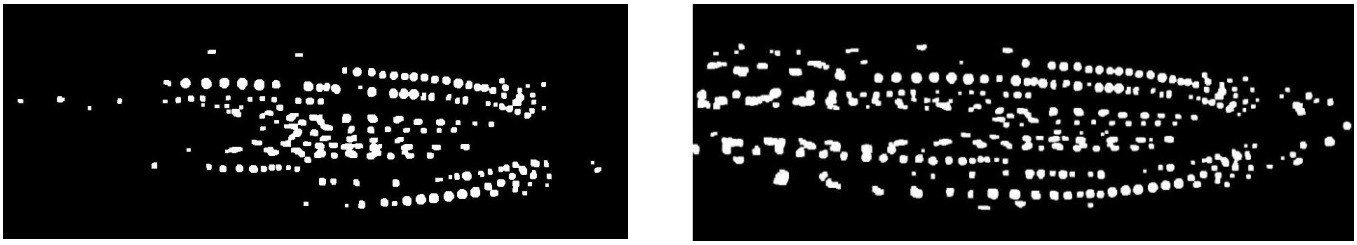

Figure 6: Final Image Output changing the type of threshold used. Left image is global which had a count of 155, right image is local adaptive threshold which had a count of 228

For local adaptive thresholding, I used the 'adaptthresh' command and the 'imbinarize' command. I could control sensitivity and I ending up setting it 0.1 for the best results, as when it was too high (0.9) less cells appear, and the quality of the image decreased overall due to cell overlapping. I ended up on a radius of 3 and 6 line structuring elements for the 'strel' command, as I thought this would produce the best version of the image after erosion and dilation. Using either 'bwconncomp' or 'regionprops' and specifying 'centroid', I was able to get a number on the nuclei present in the corresponding image that I stored in a variable called 'count'. For bwconncomp the number is the value after NumObjects; for regionprops the number of nuclei is displayed after running the variable containing the regionprops and before the x1 (e.g. count: 228 x 1 - 228 nuclei). Both can also be found in the workspace in MATLAB. Originally, I used the 'imcomponent' command to switch the black and white pixels since I thought it would make the image easier to look at and to count the cells. While there were no issues with global threshold, when I used it with local adaptive thresholding, there were some pixels that didn't change that was outside of the cell and the regionprops/bwconncomp did not work, so I decided to remove that line of code.

Final Results and Evaluation

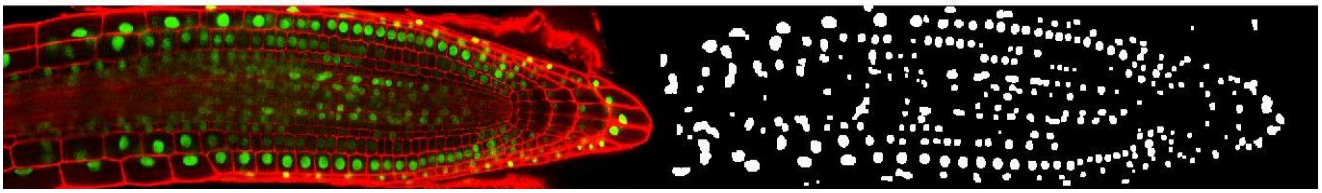

Figure 7: StackNinja1.bmp processed output has a count of 250.

.

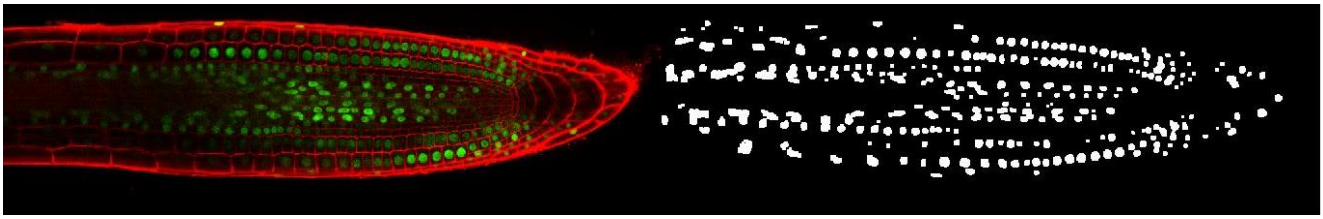

Figure 8: StackNinja3.bmp processed output has a count of 228.

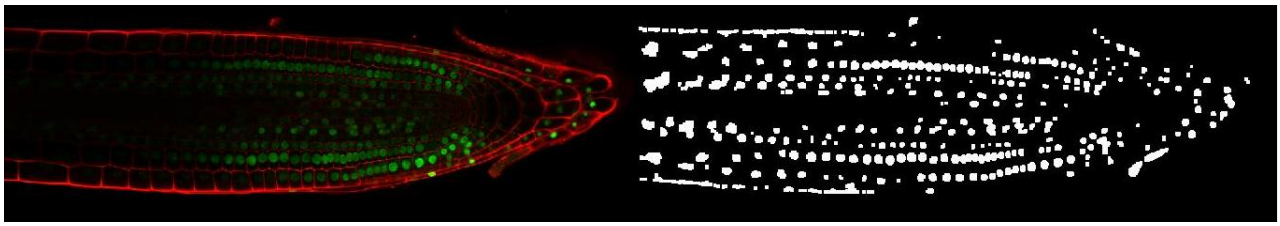

Figure 9: StackNinja3.bmp processed output has a count of 280.

The strengths that this method has is that a lot of cell nuclei are clearly visible after the image has been processed. Using the 'regionprops'/'bwconncomps', you can get a number, which means that you do not have to count cells which can be quite a time-consuming and human error-prone process. If you did want to count, the cells are much easier to count in the binary images than in the originals as shown in figures 7, 8 and 9. Not only have the cells been isolated out (i.e. no cell wall) but the original image has dark areas that are hard to see, but with this output, we are showing that there are more cells which were hidden in the original image. Secondly, the borders between cells (particularly in the middle) are hard to make out in the original, but due to steps implemented in the program, cell borders of most of the cells (particularly in the middle, top and bottom) are visualised greater in the binary image, which allows for more accurate counting.

Furthermore, compared to global threshold, more cells are shown throughout the image. The code is not particularly hard to implement and changes to various aspects of the code i.e. sensitivity of the structuring element, can be easily made for particularly challenging images. In fact, after looking at these images, scientists might be able to set a precedent when it comes to cell size in terms of the radius, which could help decide what cells to count or discount. The code allows for this to be a simple number change on the 'strel' command. To load different images, you change two lines of codes right at the top of the program.

The weaknesses of this method are that some of the cells have overlapped and cell borders have not been clearly defined. Since, I decided to use local adaptive thresholding, I lost the chance to use the effectiveness metric when it comes to thresholding, which means that it did become subjective guessing again. Looking at figures 7, 8 and 9, problems arises more to the far right and left of the image, and particularly with figure 9, the root tip and the far left of the image both have some of their border displayed. This does not show in the median filtering, which means that the thresholding and binary processing causes this to appear, although many cells are only visible after they go through the final steps. Since the code involves a lot of simple commands that uses the image processing toolbox, potentially if you manually enter more complicated code after doing the maths, the image can be finetuned so that there is no cell overlap and some of the border in figure 3 can be removed.

Conclusion

The main challenges were to try to get as many cell nuclei as possible, to have a program that worked fairly well for all three images and to subjectively decide which change to the code is ideal overall. I am quite happy with the images produced and that I got a count as I do think counting each individual cell would be too inaccurate and subjective of a process. Overall, I argue that the method has successfully accomplished the task, especially since it is impossible to get a perfect image with image processing.