

Assignment 2: Image Processing & Analysis

1 A Morphological Image Processing Pipeline for Medical Images

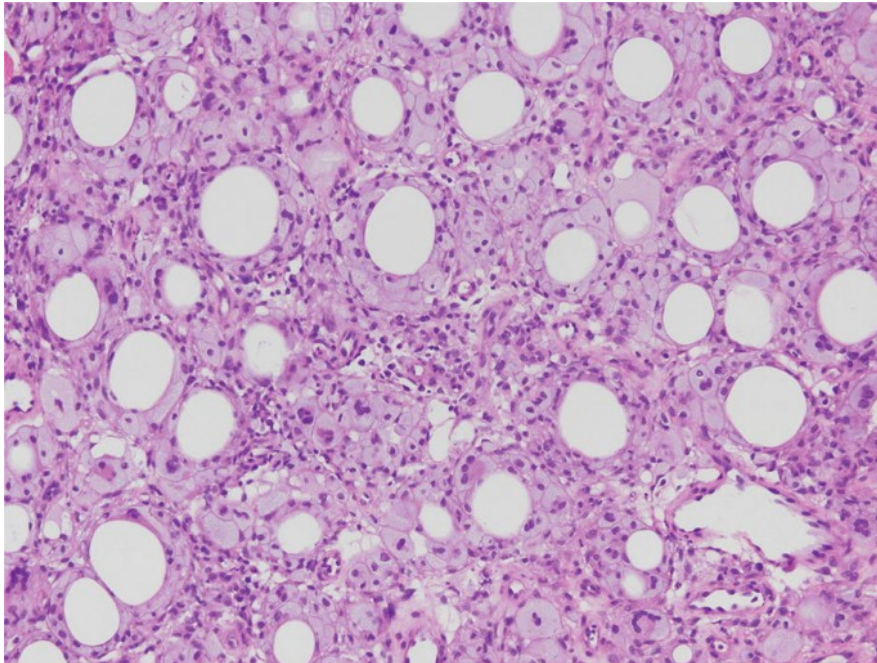


Figure 1: Original Skin Biopsy Image

1.1 Conversion to A Single-Channel Image

```
1 # Task 1: A Morphological image processing pipeline for medical images
2 # Task 1.1: Conversion to a single channel image
3 import cv2
4
5 # read in original image (in BGR format)
6 image = cv2.imread("../Task1.jpg")
7
8 # convert to greyscale
9 greyscale = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
10 cv2.imwrite("./output/greyscale.jpg", greyscale)
11
12 # convert to blue channel only
13 b_channel = image.copy()
14 b_channel[:, :, 1] = 0
15 b_channel[:, :, 2] = 0
16 cv2.imwrite("./output/b_channel.jpg", b_channel)
17
18 # convert blue channel to greyscale
19 b_channel_greyscale = cv2.cvtColor(b_channel, cv2.COLOR_BGR2GRAY)
20 b_channel_greyscale_contrast = b_channel_greyscale.std()
21 cv2.imwrite("./output/b_channel_greyscale.jpg", b_channel_greyscale)
22
23 # convert to green channel only
24 g_channel = image.copy()
25 g_channel[:, :, 0] = 0
```

```

26 g_channel[:, :, 2] = 0
27 cv2.imwrite("./output/g_channel.jpg", g_channel)
28
29 # convert green channel to greyscale
30 g_channel_greyscale = cv2.cvtColor(g_channel, cv2.COLOR_BGR2GRAY)
31 g_channel_greyscale_contrast = g_channel_greyscale.std()
32 cv2.imwrite("./output/g_channel_greyscale.jpg", g_channel_greyscale)
33
34 # convert to red channel only
35 r_channel = image.copy()
36 r_channel[:, :, 0] = 0
37 r_channel[:, :, 1] = 0
38 cv2.imwrite("./output/r_channel.jpg", r_channel)
39
40 # convert red channel to greyscale
41 r_channel_greyscale = cv2.cvtColor(r_channel, cv2.COLOR_BGR2GRAY)
42 r_channel_greyscale_contrast = r_channel_greyscale.std()
43 cv2.imwrite("./output/r_channel_greyscale.jpg", g_channel_greyscale)
44
45 # assess objectively which allows most contrast
46 print("Blue Channel Greyscale Contrast: " + str(b_channel_greyscale_contrast))
47 print("Green Channel Greyscale Contrast: " + str(g_channel_greyscale_contrast))
48 print("Red Channel Greyscale Contrast: " + str(r_channel_greyscale_contrast))

```

Listing 1: 1_single_channel_conversion.py

Since the image has predominant hues of pink-purple, we would expect the green-channel-only image to be the one that yields the highest contrast, as pink & purple colours are made up primarily by the blue & red channels: the dominance of these channels results in little variance in intensity within these channels, and therefore green will have the highest intensity variance. This is proven true by the text output of the above code, where the standard deviation of the greyscale image based off the green channel alone is by far the highest:

```

[andrew@arch] ~/currsem/CT404: Graphics & Image Processing/assignments/assignment2/code/task1 ? (master)
% python 1_single_channel_conversion.py ✓ 1s
Blue Channel Greyscale Contrast: 2.4259424019744213
Green Channel Greyscale Contrast: 23.4691181827275
Red Channel Greyscale Contrast: 7.1678348221775
[andrew@arch] ~/currsem/CT404: Graphics & Image Processing/assignments/assignment2/code/task1 ? (master)
% | ✓

```

Figure 2: Output of 1_single_channel_conversion.py

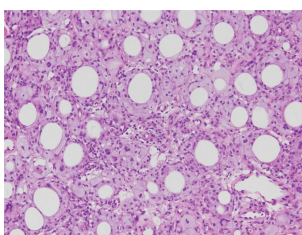


Figure 3: Original image

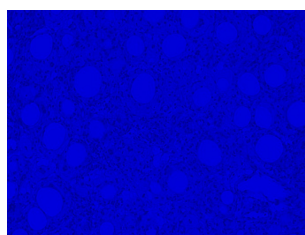


Figure 5: B-Channel

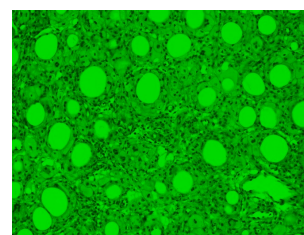


Figure 7: G-Channel

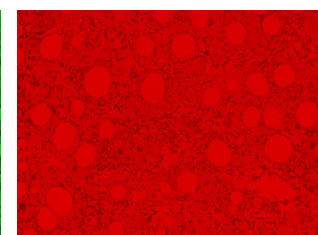


Figure 9: R-Channel

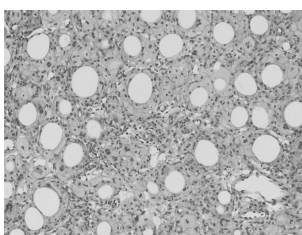


Figure 4: Greyscale original

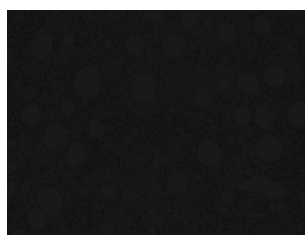


Figure 6: B-Greyscale

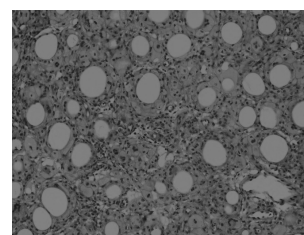


Figure 8: G-Greyscale

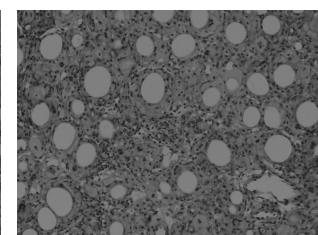


Figure 10: R-Greyscale

My selected single-channel image is the greyscale version of the green-channel-only image, as it yields the greatest contrast:

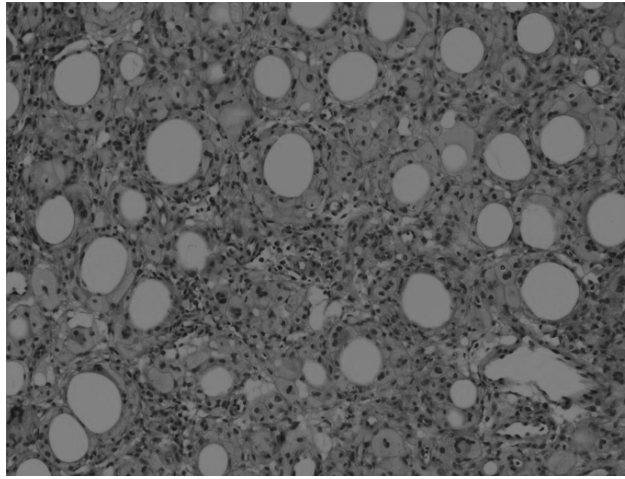


Figure 11: Selected single-channel image: greyscale green-channel-only

1.2 Image Enhancement

```

1 # Task 1.2: Image Enhancement
2 import cv2
3
4 # read in chosen single-channel greyscale image
5 image = cv2.imread("./output/g_channel_greyscale.jpg", cv2.IMREAD_GRAYSCALE)
6
7 # apply histogram equalisation
8 equalised_image = cv2.equalizeHist(image)
9 equalised_image_contrast = equalised_image.std()
10 cv2.imwrite("./output/histogram_equalised.jpg", equalised_image)
11
12 # apply contrast stretching
13 stretched_image = cv2.normalize(image, None, 0, 255, cv2.NORM_MINMAX)
14 stretched_image_contrast = stretched_image.std()
15 cv2.imwrite("./output/contrast_stretched.jpg", stretched_image)
16
17 print("Histogram Equalisation Contrast: " + str(equalised_image_contrast))
18 print("Contrast Stretching Contrast: " + str(stretched_image_contrast))

```

Listing 2: 2_image_enhancement.py

```

[andrew@arch] ~/cursem/CT404: Graphics & Image Processing/assignments/assignment2/code/task1 P (master)
% python 2_image_enhancement.py ✓ 17s
Histogram Equalisation Contrast: 74.39671562396384
Contrast Stretching Contrast: 49.452865136563894
[andrew@arch] ~/cursem/CT404: Graphics & Image Processing/assignments/assignment2/code/task1 P (master)
% 

```

Figure 12: Output of 2_image_enhancement.py

I chose to use the histogram equalisation technique as it gave the best contrast, as seen from the calculated standard deviation in contrast above and in the output images below.

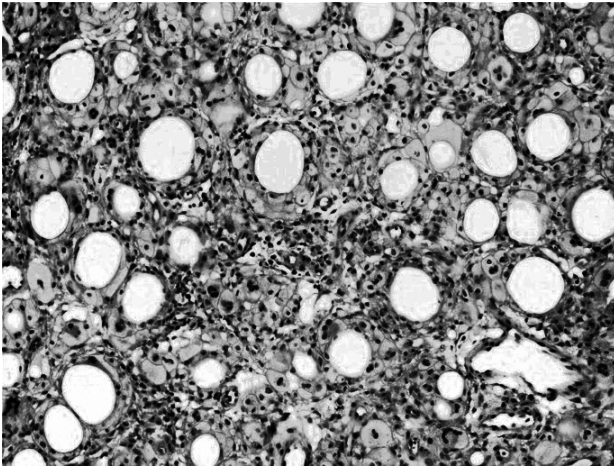


Figure 13: Histogram-equalised image

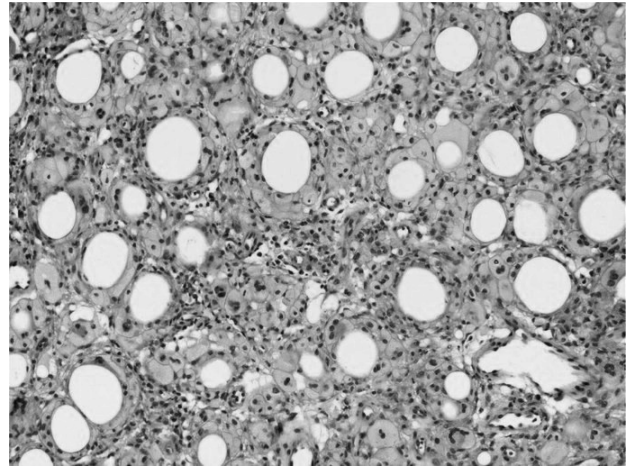


Figure 14: Contrast-stretched image

1.3 Thresholding

```

1 # Task 1.3: Thresholding
2 import cv2
3
4 # read in chosen enhanced image
5 image = cv2.imread("./output/histogram_equalised.jpg", cv2.IMREAD_GRAYSCALE)
6
7 # perform otsu thresholding to find the optimal threshold
8 threshold_value, otsu_thresholded = cv2.threshold(image, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
9 cv2.imwrite("./output/otsu.jpg", otsu_thresholded)
10
11 print("Threshold value used: " + str(threshold_value))

```

Listing 3: 3_thresholding.py

```

[andrew@arch] ~/cursem/CT404: Graphics & Image Processing/assignments/assignment2/code/task1 P (master) +
% python 3 thresholding.py
Threshold value used: 129.0
[andrew@arch] ~/cursem/CT404: Graphics & Image Processing/assignments/assignment2/code/task1 P (master) +
% |

```

Figure 15: Output of 3_thresholding.py

I used Otsu's algorithm to find the optimal threshold value that best separated the foreground (objects of interest) from the background. As can be seen from the above output, the optimal value chosen was 129.

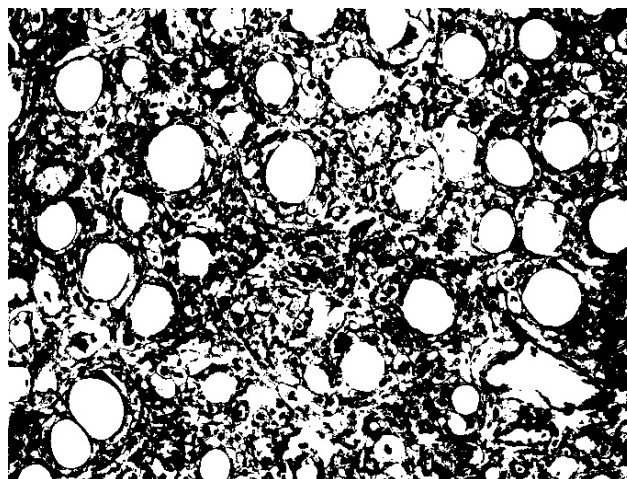


Figure 16: Image with Otsu thresholding

1.4 Noise Removal

```
1 # Task 1.4: Noise Removal
2 import cv2
3
4 # read in thresholded image
5 image = cv2.imread("./output/otsu.jpg", cv2.IMREAD_GRAYSCALE)
6
7 # try several different sizes of structuring element (must be odd)
8 for kernel_size in range(1, 16, 2):
9     # define a disk-shaped structuring element
10    structuring_element = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (kernel_size, kernel_size))
11
12    # apply morphological opening to remove noise
13    opened_image = cv2.morphologyEx(image, cv2.MORPH_OPEN, structuring_element)
14    cv2.imwrite(f"./output/kernel_size_{kernel_size}.jpg", opened_image)
```

Listing 4: 4_noise_removal.py

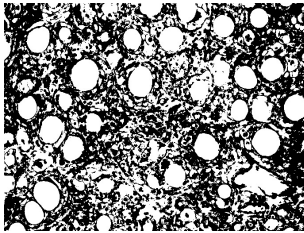


Figure 17: kernel_size = 1

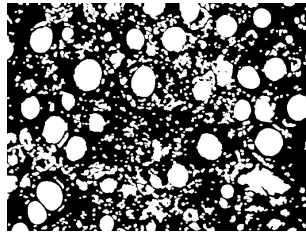


Figure 19: kernel_size = 5

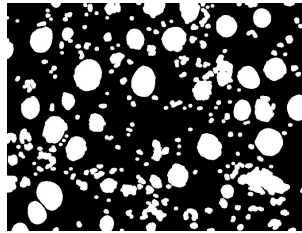


Figure 21: kernel_size = 9

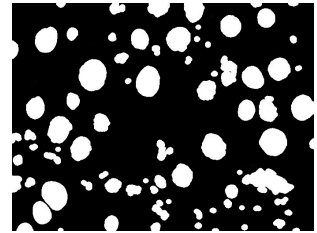


Figure 23: kernel_size = 13

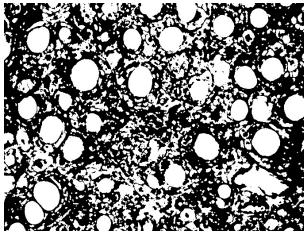


Figure 18: kernel_size = 3

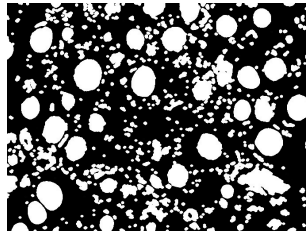


Figure 20: kernel_size = 7

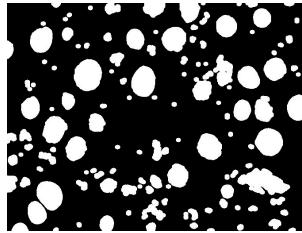


Figure 22: kernel_size = 11

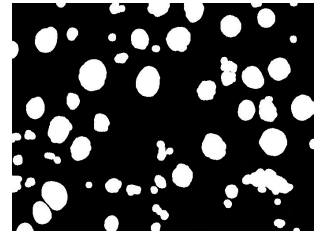


Figure 24: kernel_size = 15