Report - Exercise 01

Medical Image Computing

MT-M-3-ILV-IM2

Master Medical Technologies

Semester 3

Lecturer: Marco Augustin

Group: MA-MED-21-VZ

Authors: Peter Rott, Filip Slatković, Markus Wernard

23rd November 2022

Contents

1 OCT image processing framework

Figure 1.1: Raw OCT image

1.1 Histogram

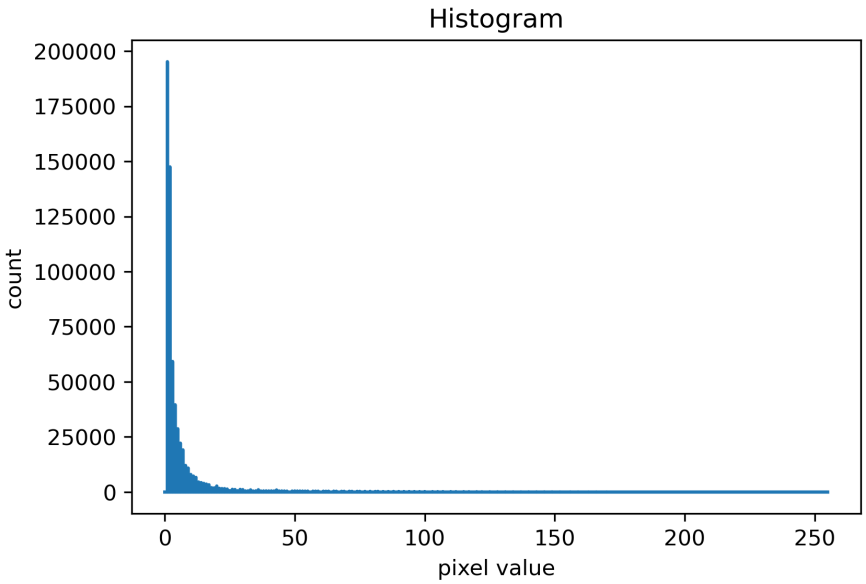

Figure 1.2: Histogram of the raw OTC image with a pixel value range from 0 to 255

1.2 Intensity Transformations

For the transformation of the image there were two methods compared. The first method is the log transformation, here all pixel values getting replaced with its logarithmic values. Figure 1.3 shows the result on the Raw OTC image in Figure 1.1. The Gamma transformation is selected as the second method of image transformation with its results presented in 1.4. Both methods can be used to highlight different areas of the eye. However, the different areas are less visible in the gamma transformation. Therefore, the log transformed image was used for further image processing.

Log Transformation

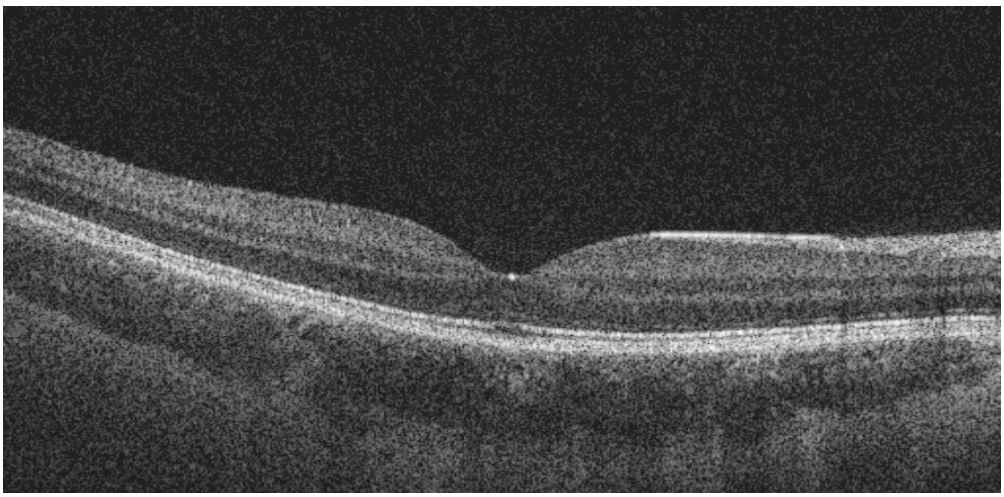

Figure 1.3: Result of the log transformation for the OTC raw image

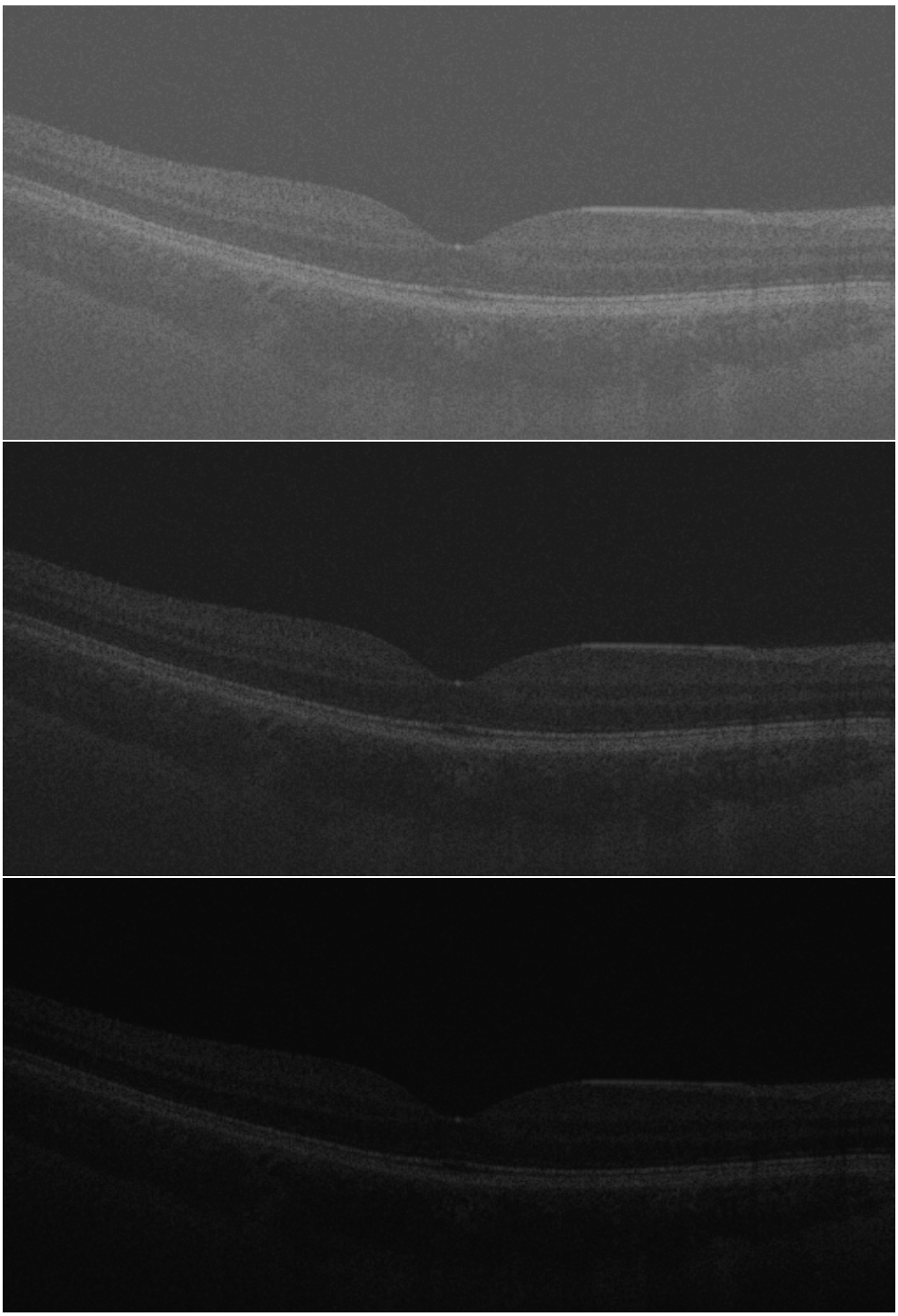

Figure 1.4: Result of the gamma transformations for the OCT raw image; top: Gamma = 0.1, middle: Gamma = 0.2, bottom: Gamma = 0.3

1.3 Filter & Normalize Image

In this Section the transformed image gets further filtered and afterwards normalized. For the image filtering an Average filter and the Gaussian filter were applied. The results are displayed in figures 1.5 and 1.6. The Filters where implemented in OpenCv with the functions $cv2.filter2D$ and $cv2.GaussianBlur$. It gets visible that with a larger kernel size of the filter, the image noise decreases, however the image also becomes blurrier. Therefore, the Gaussian Filter with a kernel size of [5x5] was chosen for further processing of the image. The next step is the normalization of the filtered Image to a range from 0 to 1. Here fore the function $cv2.normalize$ is used. Furthermore the image needed to be transformed into the data type float32. The effect of the normalization is a further reduction of the image noise as well as a better representation of the edges due to a higher contrast. The normalized image is shown in figure 1.7.

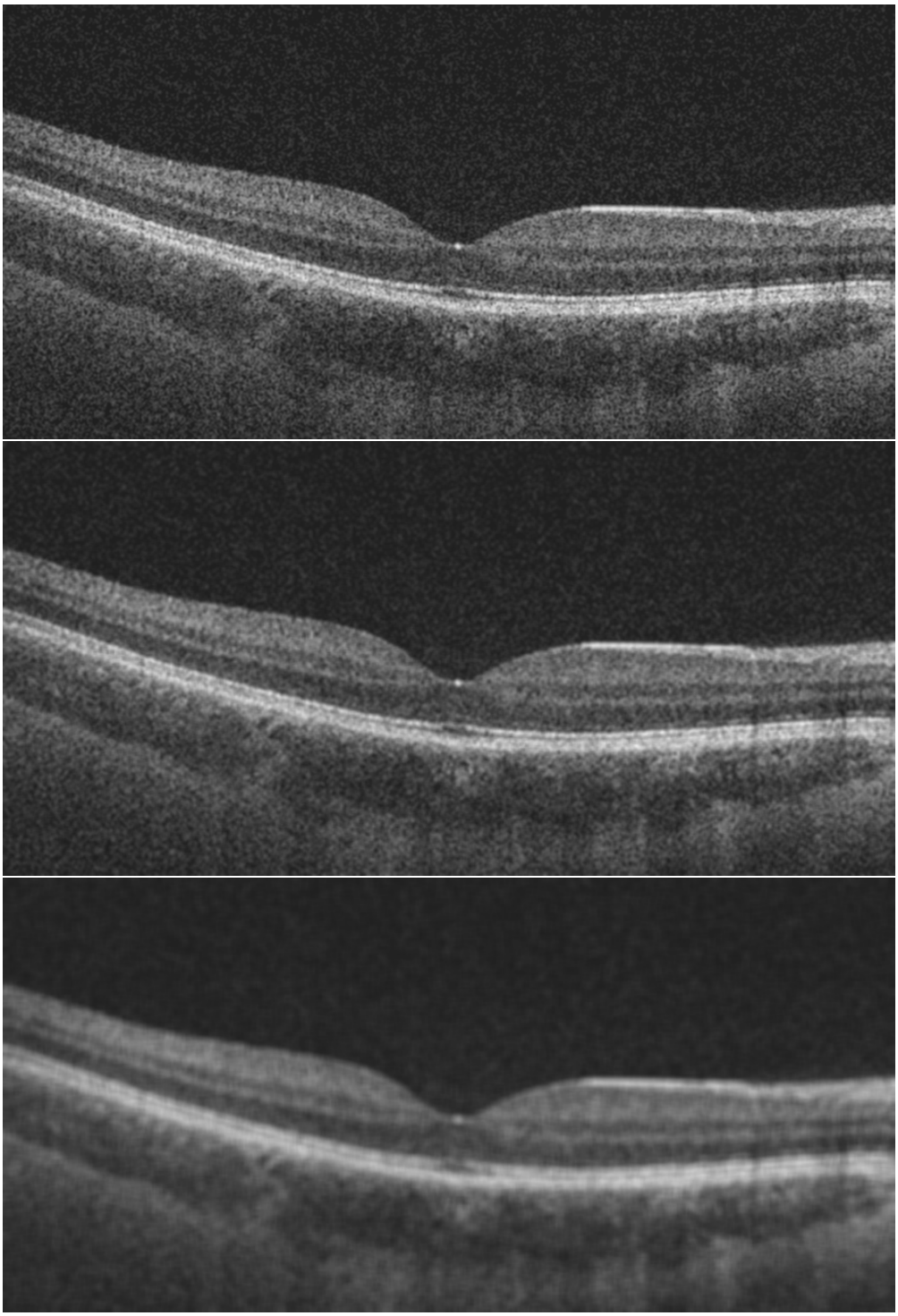

Figure 1.5: Average filtered image with kernel sizes of [3x3] (top),[5x5] (middle) and [11x11] (bottom)

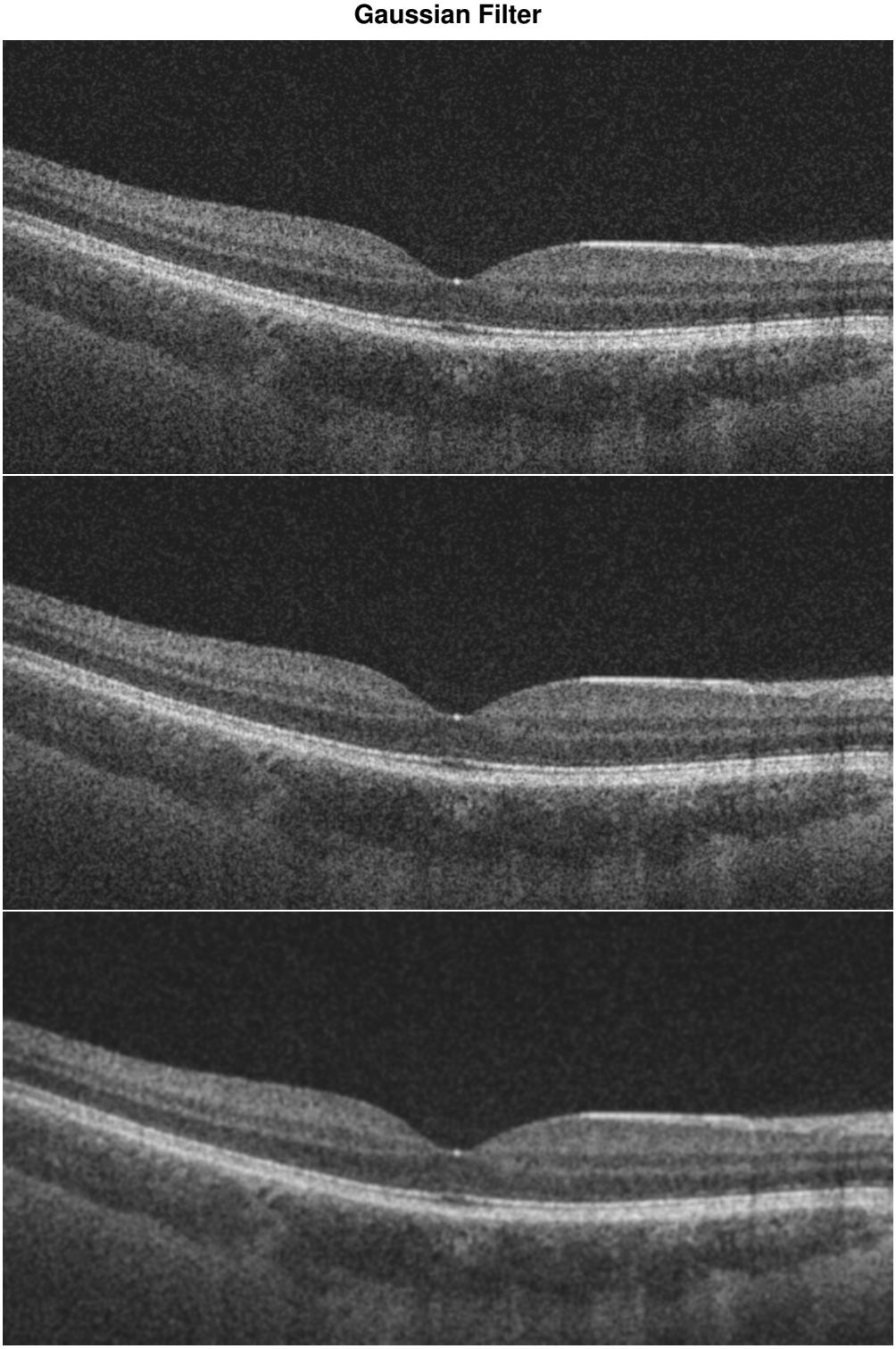

Figure 1.6: Gaussian filtered image with kernel sizes of [3x3] (top),[5x5] (middle) and [11x11] (bottom)

Normalized filtered image

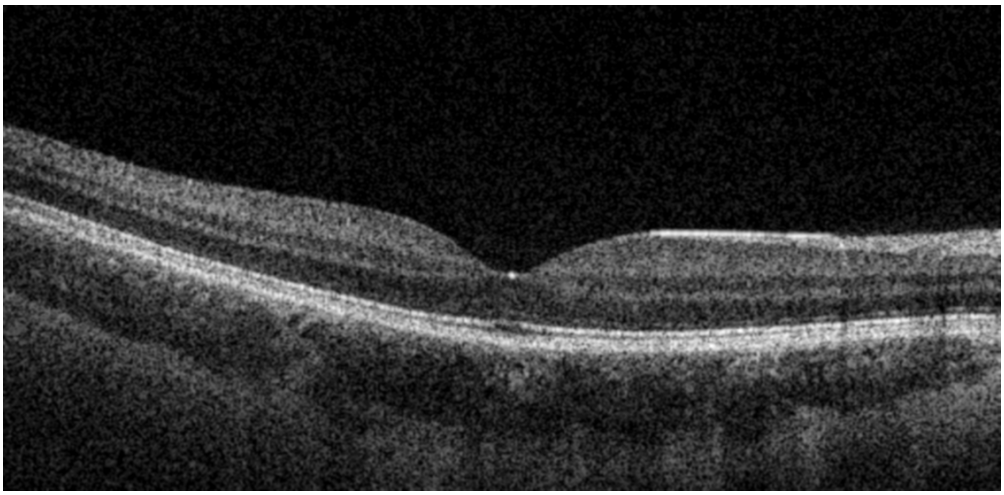

Figure 1.7: Normalized OCT image with a Bit range between 0 and 1 after a log transformation and Gaussian filtering with kernel size of [5x5]

1.4 Neighborhood Analysis

In this section the normalized filtered image is used to asses the Neighborhood with the size of 21x21 pixels of the three different pixels. To get a better understanding where the selected area of the pixels is in the image the $cv2.rectangle$ function is used. These parts of the image are then cropped out and then the histograms are calculated. The cropped areas and the values of the pixels in these are are presented in figure 1.8 and 1.9.

Cropped sections of the normalized image

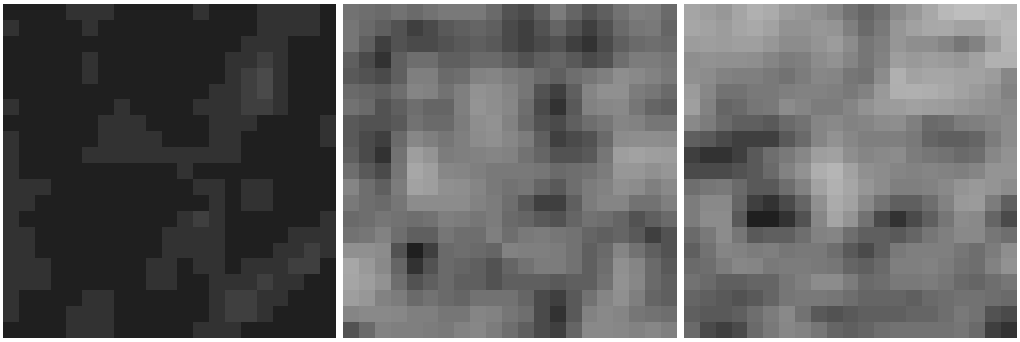

Figure 1.8

First crop of Pixel 160:160, figure 1.8 (left)): Standard deviation: 0.02685174, Mean value: 0.04229191

Second crop of Pixel 288:288, figure 1.8 (middle): Standard deviation: 0.06966738, Mean value: 0.24162757

Third crop of Pixel 519:519, figure 1.8 (right): Standard deviation: 0.05398345, Mean value: 0.17655321

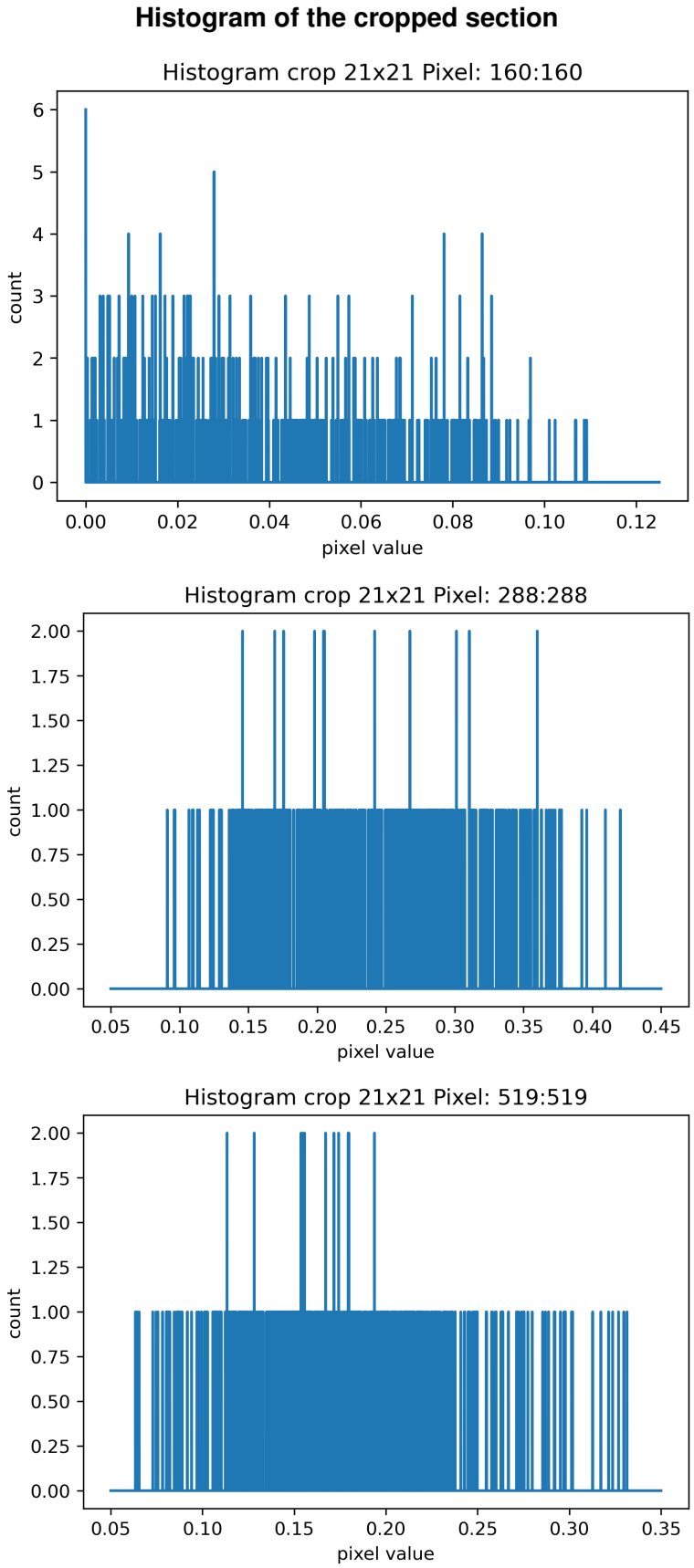

Figure 1.9: Showing the histograms of the different cropped files

1.5 Thresholding

In order to split up the image into retina and vitreous part a binary threshold T is defined. The result is visualized in 1.10. It is implemented with the $cv2.throshold$ function, where the thresh is set to 3728 and the max value to 65535. The thresh is selected since it gives a good distinction of retina and vitreous part of the image, second number is the maximal value of a pixel when using uint16 datatype.

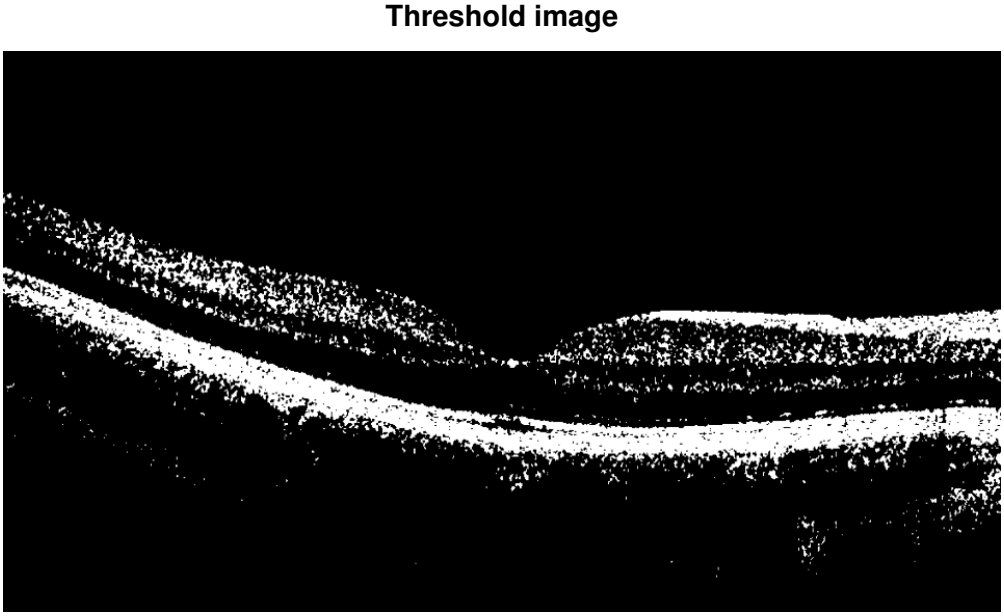

Figure 1.10: Binary threshold was used to make a difference between retina and vitreous

2 Edge detection

2.1 Sobel Kernel

In the beginning the Sobel Kernel method is implemented to detect horizontal and vertical edges. This is done by using the OpenCv function $cv2.Sobel$. It is important to define the right data type for this function. Since the normalized 32float image is used the data type needs to be set to 32float. The results are presented in figure 2.1.

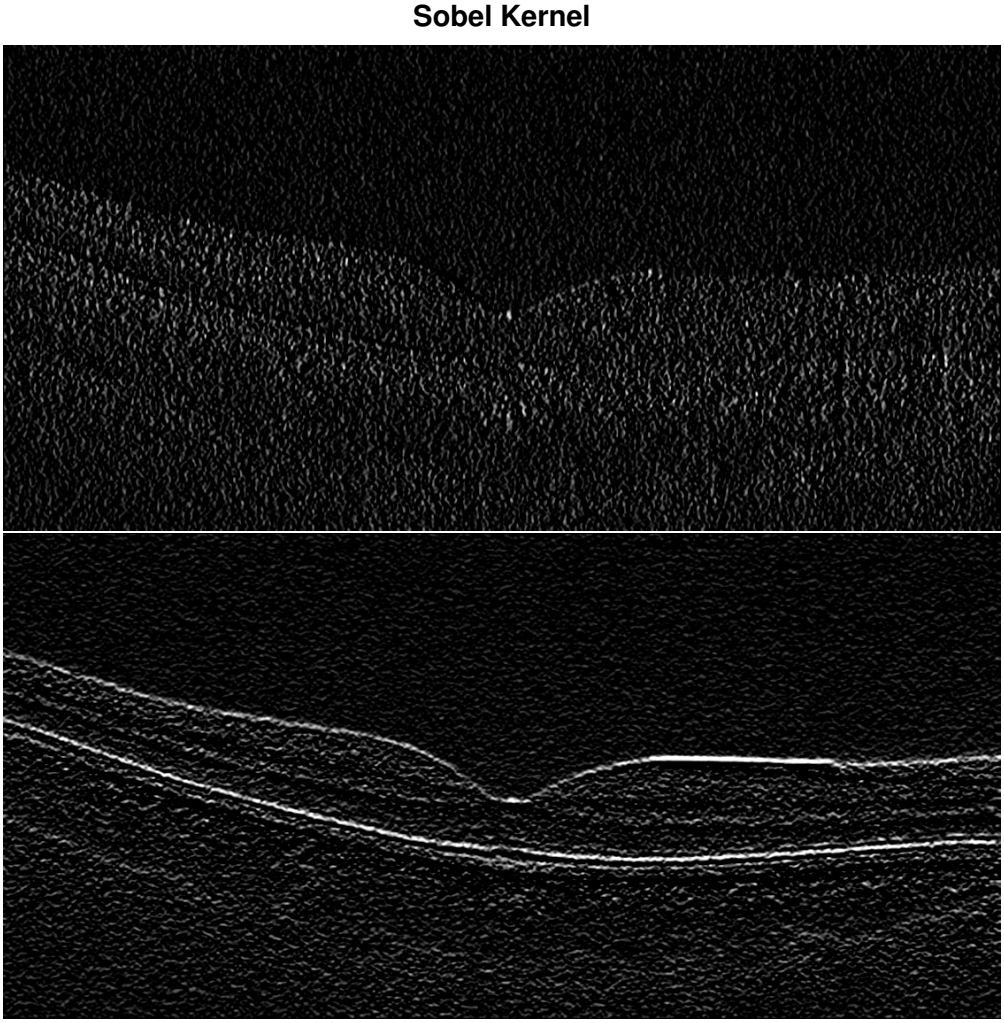

Figure 2.1: Sobel kernel vertical (top) and horizontal (bottom) edge detection

In both images, the transition from retina to vitreous is noticeable. Nevertheless, the

edges are better recognizable in the vertical Sobel Kernel.

2.2 Image Gradient and Threshold

Based on these results of the Sobel kernel, the image gradient can now be calculated, the outcome is shown in figure 2.2. For these process the image is transformed into a uint16 data type.

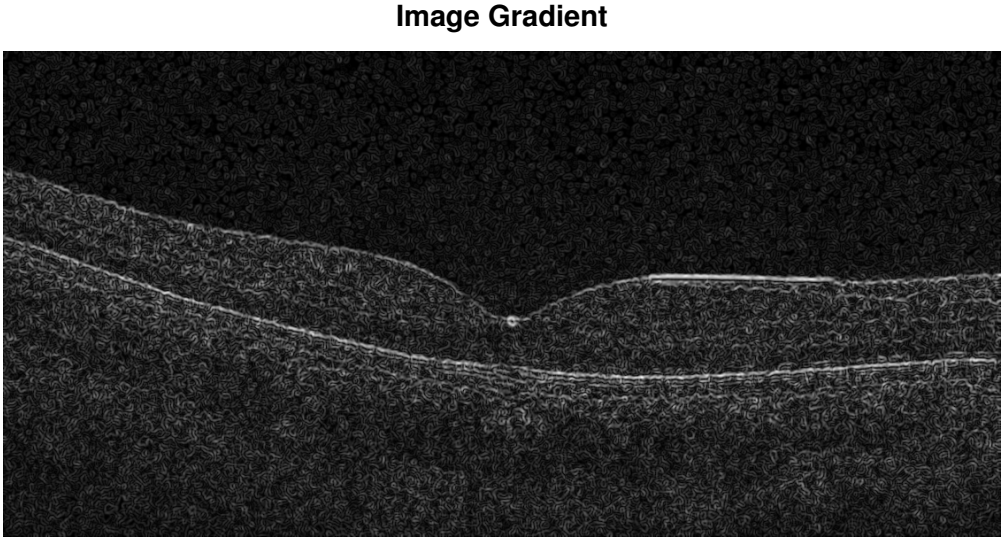

Figure 2.2: Image Gradient of the Sobel kernel

In the end a threshold is specified to segment the most prominent boundaries in the image. In figure 2.3 the image is displayed with a threshold of 25000. If the threshold is set smaller, the amount of pixels which are showing the transition from retina to choroid becomes larger. If it is set higher, many pixels are lost on the edge between retina and vitreous.

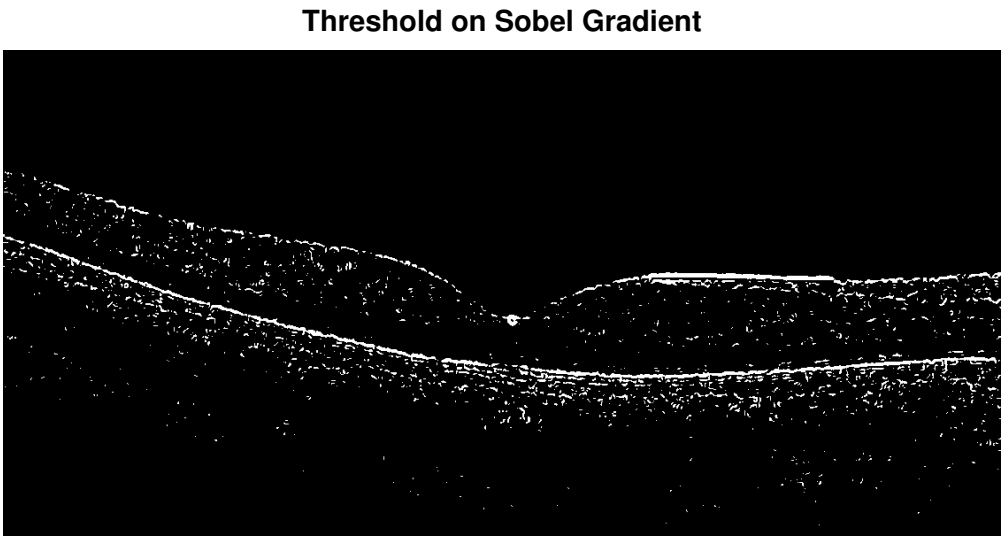

Figure 2.3: Images with Threshold of 25000

2.3 Canny Edge Detection

As additional task the Canny Edge Detection is implemented with the $cv2.Canny$ function. To use these function the input image needs to be of the data type unit8. The result is illustrated in figure 2.4. The edges are very finely recognizable and the threshold of 220 is significantly higher set in relation to the previous task. The lower edge is almost continuously detected, while the upper boundary has some gaps. In total this method shows the edges more detailed with less pixels.

Figure 2.4: Canny edge detection of the normalized Image

3 Image segmentation

3.1 Otsu Thresholding Algorithm

The Otsu Thresholding Algorithm implemented in OpenCV returns the calculated Otsu threshold and the processed image on an uint8 or uint16 input image. As the filtered and normalized image of task 2.3. is of data type float32 it needs to be converted first.

The image was converted into data type uint8 to be processed by the Otsu Algorithm, what leads to the outcome depicted in figure 3.1. The Otsu threshold has been returned with 58.0 of 255 (uint8).

Figure 3.1: Filtered and Normalized Image processed by Otsu Thresholding Algorithm

Edges can be improved by applying an additional Gauss filter before thresholding. In this case a Gauss filter with a kernel size of $[11\text{x}11]$ and a standard deviation of 5 has been applied and leads to a Otsu Thresholding depicted in figure 3.2. The threshold was returned with 55.0 of 255 (uint8).

Figure 3.2: Filtered and Normalized Image processed by Otsu Thresholding Algorithm with additional Gauss filter applied

3.2 Additional Segmentation Algorithm

For the Additional Segmentation Algorithm the "Triangle Thresholding Algorithm" has been chosen. In the OpenCV documentation it is described as an "algorithm to choose the optimal threshold value". It also returns the calculated Triangle threshold and the processed image on an int8 input image.

The filtered and normalized image is converted into data type int8 to be processed by the Triangle Thresholding Algorithm, this leads to the outcome depicted in figure 3.3 The Triangle threshold is returned with 25.0 of 255 (uint8) in this case.

Figure 3.3: Filtered and Normalized Image processed by Triangle Thresholding Algorithm

Edges can be improved by applying an additional Gauss filter before thresholding. In this case a Gauss filter with a kernel size of $[11\text{x}11]$ and a standard deviation of 5 has

been applied and leads to a Triangle Thresholding depicted in figure 3.4. The threshold was returned with 18.0 of 255 (uint8).

Figure 3.4: Filtered and Normalized Image processed by Triangle Thresholding Algorithm with additional Gauss filter applied

3.3 Segmentation Evaluation

The chosen kernel size is a compromise of maximum edge highlighting and minimum information loss. By further increasing the kernel size the surface of the retina would be smoothed as well and the information of interest would be lost. Compared to the Otsu Thresholding Algorithm, the Triangle Thresholding Algorithm is more efficient in segmenting the Vitreous and the Retina.

List of Figures