NANYANG TECHNOLOGICAL UNIVERSITY

SINGAPORE

IE4476 Image Processing

Project Report

Automatic Retinal Vessel Segmentation

Student Name:	Tran Anh Quan
Matric No:	U1920261L
Academic Year:	2022/2023
Semester:	1

Submission Date: 14 Nov 2022

Table of Contents

1. Introduction

Automatic retinal vessel segmentation is a crucial but challenging part of computer-aided diagnosis of ophthalmology diseases [1], [2]. Several approaches have been proposed to solve the task of retinal vessel segmentation, including machine learning and deep-learning-based methods, morphological image processing methods, and adaptive thresholding methods, etc. [2].

In this project, an automatic retinal vessel segmentation system was developed using Python. Two techniques for solving this task were explored. The first one is an unsupervised method using adaptive thresholding. The second one is a supervised deep-learning-based method which utilizes U-Net [3] and the DRIVE dataset [8]. The results from both methods will be evaluated and analyzed.

2. Methods

2.1. Adaptive Thresholding Approach

This approach adopts a 3-stage pipeline **(Figure 1)**. The code for this method is attached to **Appendix 1**.

At the preprocessing stage, the green channel is extracted from the input RGB image. Contrast Limited Adaptive Histogram Equalization (CLAHE) is then applied to the extracted grayscale image to improve its contrast. The equalized image then goes through a noise removal process.

At the segmentation stage, an adaptive Gaussian thresholding is applied to the preprocessed image to extract the retinal vessels. The output of this stage is a binary image.

At the post-processing stage, connected component analysis is conducted to remove the outer fundus border of segmented image. The binary image is then further enhanced by a morphological closing operation.

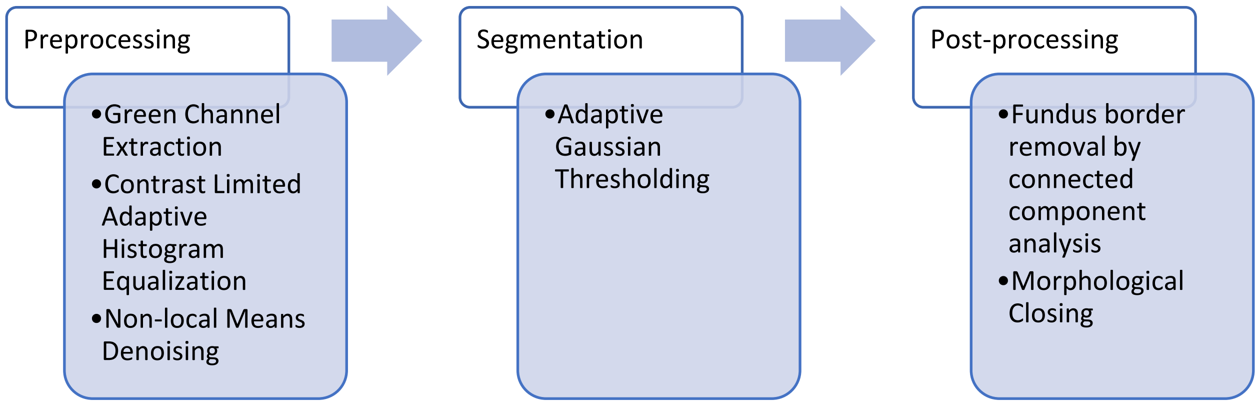

Preprocessing
•Green Channel Extraction
•Contrast Limited Adaptive Histogram Equalization
•Non-local Means Denoising
Segmentation
•Adaptive Gaussian Thresholding
Post-processing
•Fundus border removal by connected component analysis
•Morphological Closing

Figure 1: Retinal vessel segmentation pipeline with adaptive thresholding

The sample retinal image for illustration is the training image 24 **(Figure 2)** from the DRIVE retinal vessel segmentation dataset [7].

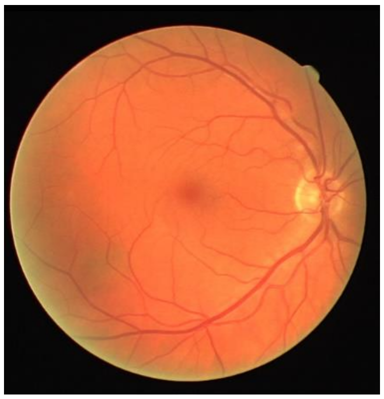


Figure 2: Sample image for illustration

2.1.1. Green Channel Extraction

The green color channel is used for this task. Among the 3 color channels, the green channel shows the most visible blood vessel **(Figure 3)**, hence is the most suitable one for the task.

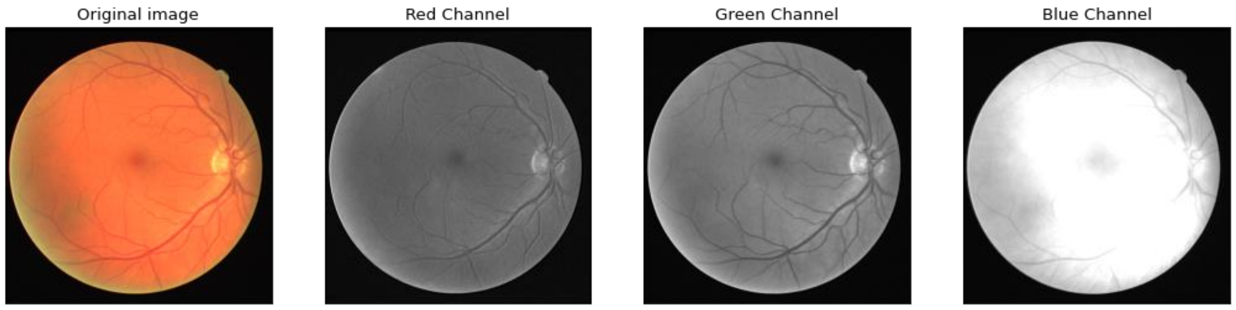


Figure 3: Original RGB image and grayscale images of each channel

2.1.2. Contrast Limited Adaptive Histogram Equalization (CLAHE)

Histogram equalization is often applied into image enhancement to increase the image's contrast. Adaptive histogram equalization (AHE) is an improvement on the original histogram equalization algorithm to better enhance the regions with significantly lower or higher intensity than most of the image. In AHE, multiple histograms are computed for each section of the image, and histogram equalization is done separately for each of the computed histogram [4].

However, AHE is prone to amplifying noise. CLAHE is thus proposed by Pizer et al. [4] to address this problem. The algorithm introduces a clip limit to the contrast amplification. The part of the histogram that is above the clip limit is redistributed equally among all histogram bins, in order to renormalize the histogram **(Figure 4)**. By that, the CLAHE algorithm effectively limits the slope of the mapping function **(Figure 5)**, hence reduces the effect of noise amplification while still preserves the contrast enhancement advantage of AHE.

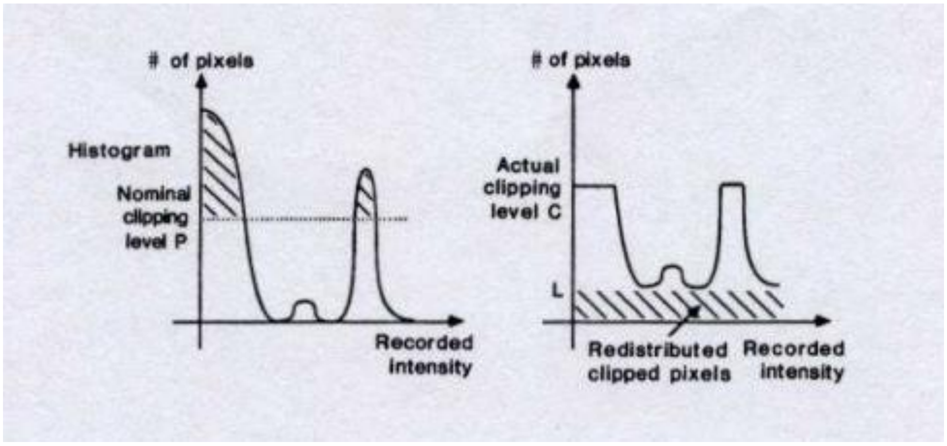


Figure 4: Original histogram (left) and clipped then renormalized histogram (right) [4]

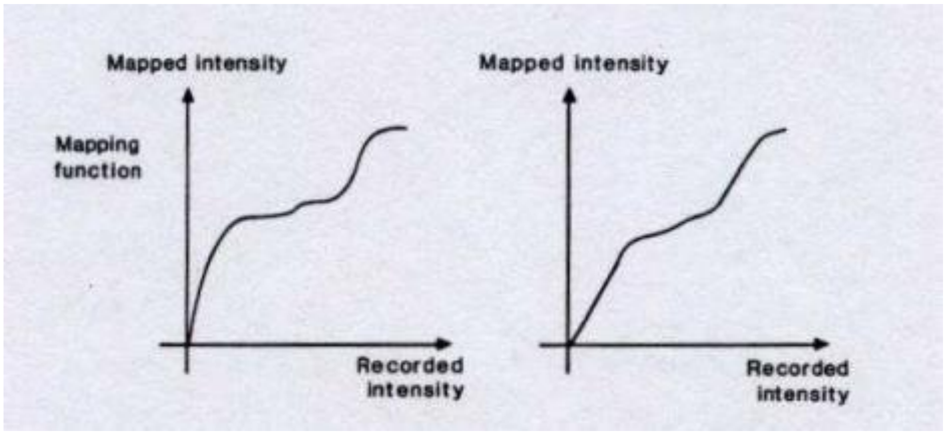


Figure 5: Mapping function for original histogram (left) and for clipped then renormalized histogram (right) [4]

The CLAHE algorithm is implemented by the Python's OpenCV library by the function *cv2.createCLAHE(clipLimit, tileGridSize)* [5], where *clipLimit* is the threshold for contrast limiting, and *tileGridSize* is the size of the grids that the image is divided into for adaptive histogram equalization. For my project, the clip limit is set at 4.0, while each divided region is a square of 8x8 pixels **(Figure 6)**.

```
# create a CLAHE object
clahe = cv2.createCLAHE(clipLimit=4.0, tileGridSize=(8,8))
# apply CLAHE into the Green channel
img_equalized = clahe.apply(img_green)
```

Figure 6: Code for applying CLAHE to the green channel image

The green channel of the sample image after CLAHE is shown in **Figure 7**. It can be seen that the contrast is better, and the vessels are more visible.

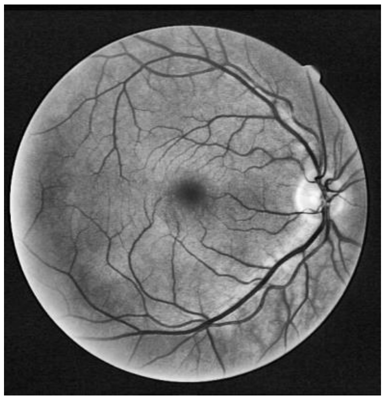


Figure 7: Green channel of the sample image after CLAHE is applied

2.1.3. Non-local Means Denoising

Although limited, noise still appears in the CLAHE-equalized image **(Figure 7)**, hence further noise removal is required. The noise removal is done by the non-local means denoising algorithm [6]. Non-local mean filters map a pixel's intensity to a weighted average of a selection of non-local pixels which are in a similar environment. The level of similarity between the neighborhood of the pixel of interest and those of the chosen pixels determines the weights for the weighted average [6]. Non-local means denoising is able to keep texture information that could be blurred by other denoising algorithms.

The non-local means denoising algorithm is implemented in OpenCV by the function *cv2.fastNlMeansDenoising(image, output, h, templateWindowSize, searchWindowSize)* [7], where *h* is the filter strength coefficient (the higher this coefficient, the more blur the image is), *templateWindowSize* is the size of the template patch that is used to compute weights, and *searchWindowSize* is the size of the research window. For my project, the filter strength is set at 17, while the template patch size is 7 and the research window size is 11 **(Figure 8)**.

```
# denoise the equalized image
noiseless_image_bw = cv2.fastNlMeansDenoising(img_equalized, None, 17, 7,
11)
```

Figure 8: Code for applying Non-local Means Denoising to the equalized image

The green channel of the sample image after CLAHE and denoising is shown in **Figure 9**.

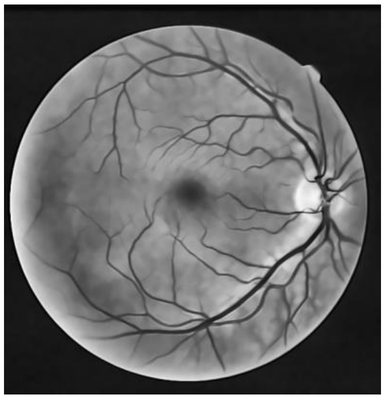


Figure 9: Green channel of the sample image after CLAHE and denoising

2.1.4. Adaptive Gaussian Thresholding

Thresholding is a common way to segment the image based on its intensity. However, global thresholding does not work well if different areas of the image have different illumination conditions, such as what we see in **Figure 9**. The solution for this is to use different thresholds for each pixel based on the region surrounding it, which is called adaptive thresholding. In adaptive Gaussian thresholding, the threshold is calculated by the weighted sum of neighborhood intensities where the weights follow a Gaussian window **(1) (Figure 10)**, minus a constant C [10].

The Gaussian window is a bell-curve shaped window which has the formula

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{1}$$

where (x, y) is the 2D coordinate, and σ denotes standard deviation.

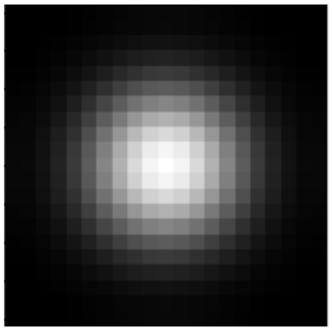


Figure 10: A 21x21 Gaussian window

The adaptive Gaussian thresholding is implemented in OpenCV by the function *cv2.adaptiveThreshold(image, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, thresholdType, blockSize, C)* [10], where *thresholdType* is the threshold type, *blockSize* is the size of the neighborhood for threshold calculation, and *C* is the constant for subtraction. For my project, the threshold type is set to be cv2.THRESH_BINARY_INV (i.e., to keep the pixel values under the threshold), the neighborhood is a 21x21 square centered at the target pixel, and the value for the subtraction constant is 6 **(Figure 11)**.

```
# apply adaptive gaussian thresholding
thresholded = cv2.adaptiveThreshold(noiseless_image_bw,255,cv2.ADAPTIVE_TH
RESH_GAUSSIAN_C,cv2.THRESH_BINARY_INV,21,6)
```

Figure 11: Code for applying Adaptive Gaussian Thresholding

The output after the thresholding step is shown in **Figure 12**. It can be seen that the blood vessels are well segmented.

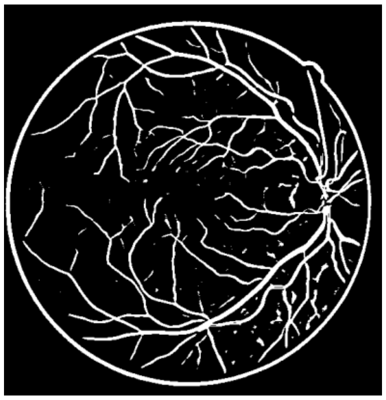


Figure 12: The sample image after the thresholding step

2.1.5. Fundus border removal by Connected Component Analysis

As seen in **Figure 12**, the image after segmentation does not only contain the blood vessels, but the outer border of the fundus also. It is because after the histogram equalization step, this border is suppressed by the inner fundus neighborhood, which has much more intensity. To remove this border, connected component analysis is adopted. Connected component analysis labels the connected regions in the binary image. By doing this, we will be able to extract the fundus border, and then remove the border by a bitwise XOR operation with the input image.

There is only one result for connected component analysis, but the difference between algorithms is their space and time complexity. The algorithm for connected component analysis in this project is the Spaghetti labelling algorithm [11]. It is implemented by default by the OpenCV function *cv2.connectedComponents(binaryImage)* [12]. The function outputs the number of connected components in the binary image and a labelled image. As the outer-most connected component (except the black background), the fundus border should be labelled “1” by the function. Hence, we can extract the border by the code in **Figure 13**.

```
# connected component analysis
num_labels, labels_im = cv2.connectedComponents(thresholded)

# extracting the fundus border at label 1
mask = (labels_im==1).astype(np.uint8)
mask = mask*255

# remove the fundus border by an XOR operation
removed = cv2.bitwise_xor(thresholded, mask)
```

Figure 13: Code for removing the outer fundus border

The extracted fundus border and the segmented image after removing the border is shown in **figure 14**.

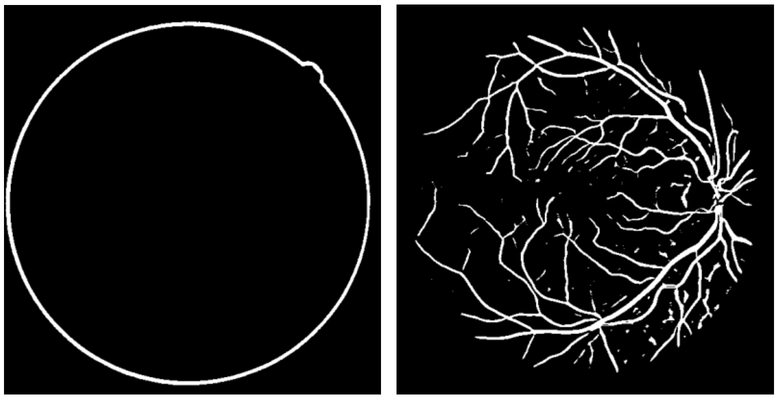


Figure 14: Extracted border (left) and segmented image after removing the border (right)

2.1.6. Morphological Closing

The final step is to apply a morphological closing to remove noise in the binary image. Morphological closing is a dilation followed by an erosion operation **(2)** [13]. Morphological closing fills small black holes in the input image, which could be useful in this case to fill the missing points in the blood vessel. The formula for morphological closing of a set A by a structuring element B is shown as follow.

$$A \bullet B = (A \oplus B) \ominus B \tag{2}$$

For the morphological closing in this project, the structuring element used is as shown below.

$$B = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

The morphological closing operation is implemented in OpenCV by the function *cv2.morphologyEx(binaryImage*, *cv2.MORPH_CLOSE*, *kernel)* [10], where *kernel* is the structuring element. The code for morphological closing is as shown in **Figure 15**.

```
kernel = np.array([[1,0,1],
                   [0,1,0],
                   [1,0,1]]).astype(np.uint8)
#morphological closing
closed = cv2.morphologyEx(removed, cv2.MORPH_CLOSE, kernel)
```

Figure 15: Code for morphological closing

We then get the final output as shown in **Figure 16**.

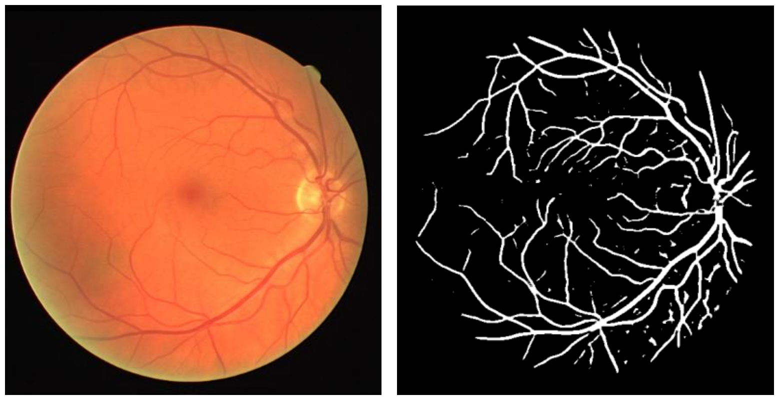


Figure 16: Input (left) and output (right) of my automatic retinal vessel segmentation system based on adaptive thresholding

2.2. Deep-Learning-based Approach

In this part of the project, I would like to explore a state-of-the-art method for retinal image segmentation using U-Net [3] neural network. My implementation of U-Net is based on the work of Yao et al. [9] **(Appendix 2)**. I made some modifications to their original codes to fit the purpose of this project.

The pipeline for training and inference of the deep-learning-based retinal vessel segmentation network is described in **Figure 17** and **Figure 18**, respectively.

In the preprocessing stage of both training and inference pipeline, the input image is normalized and applied CLAHE in all 3 color channels. The output then goes through the gamma correction process.

In the training stage of the training pipeline, the dataset is split into training and validation sets. Then several training and validation patches are created by extracting smaller regions in the images of the training and validation sets. The network is then trained for some epochs and is evaluated on the validation set.

In the inference stage of the inference pipeline, the image for inference is divided into patches and each of them is fed through the network separately. The outputs are then concatenated to construct the final output binary image in the postprocessing stage.

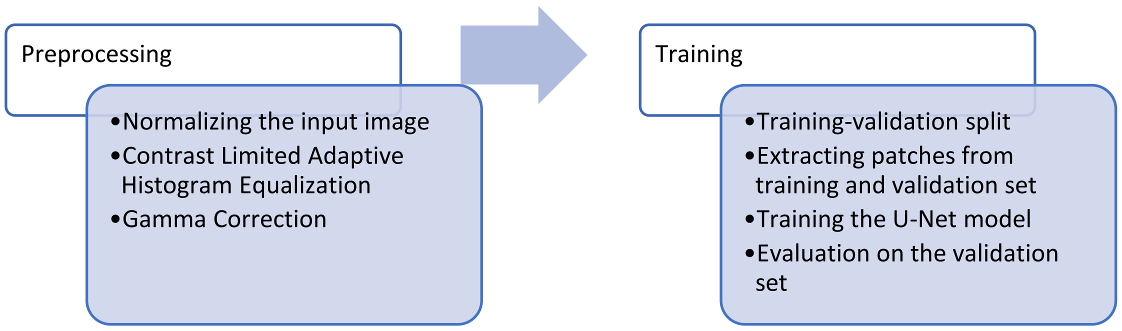

Preprocessing
•Normalizing the input image
•Contrast Limited Adaptive Histogram Equalization
•Gamma Correction
Training
•Training-validation split
•Extracting patches from training and validation set
•Training the U-Net model
•Evaluation on the validation set

Figure 17: Training pipeline for the segmentation network

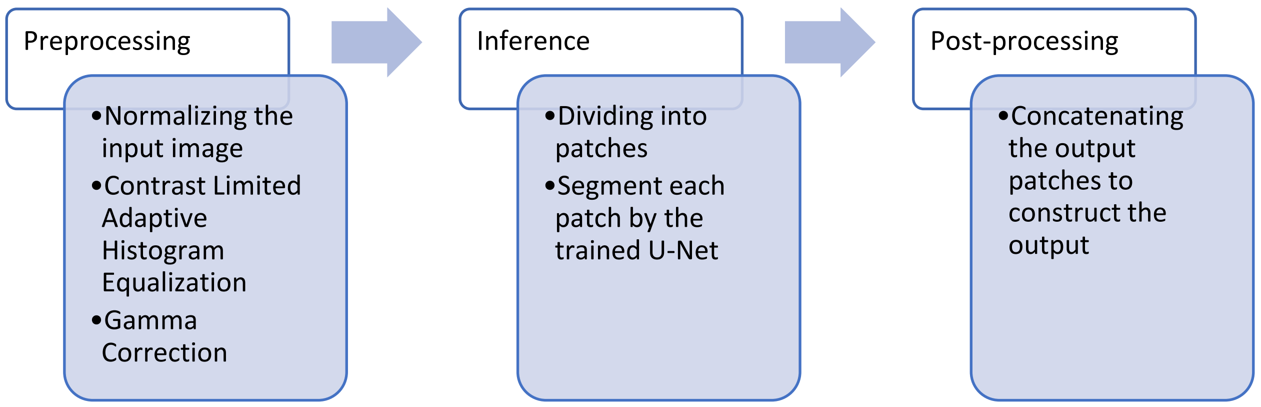

Preprocessing
•Normalizing the input image
•Contrast Limited Adaptive Histogram Equalization
•Gamma Correction
Inference
•Dividing into patches
•Segment each patch by the trained U-Net
Post-processing
•Concatenating the output patches to construct the output

Figure 18: Inference pipeline for the segmentation network

2.2.1. Data Preprocessing

The input data is first normalized to improve its contrast by bringing its intensity range to a normal distribution. The normalization is done separately in each channel by the operation described below.

$$I_{norm}(x, y) = \frac{I(x, y) - \text{mean}(I)}{\text{std}(I)} \tag{3}$$

The normalized image is then applied CLAHE [4] to further improve the contrast in each small region. CLAHE is applied separately to each color channel.

Gamma correction is the last step of preprocessing, where the luminance of the image is adjusted [13]. In this step, each pixel value V_{in} in the input image is mapped to an adjusted value V_{out} by the following operation

$$V_{out} = \left(\frac{V_{in}}{255}\right)^{\gamma} \times 255 \tag{4}$$

where the choice of γ (gamma) depends on how we want to adjust the luminance of the image **(Figure 19)**.

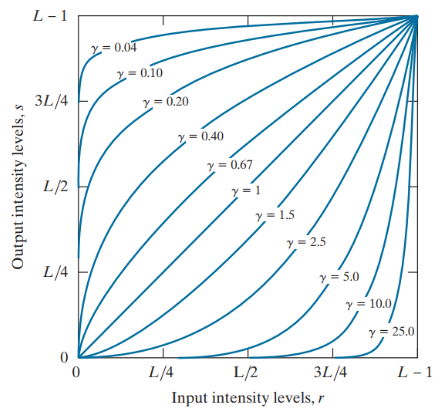


Figure 19: Input-output intensity curve for different values of gamma [13]

As the image is darken after the CLAHE step, γ is then chosen to be smaller than 1 (i.e., encoding gamma) for better lightness. The γ chosen is 0.833.

How the input image is transformed after each preprocessing step is shown in **Figure 20**.

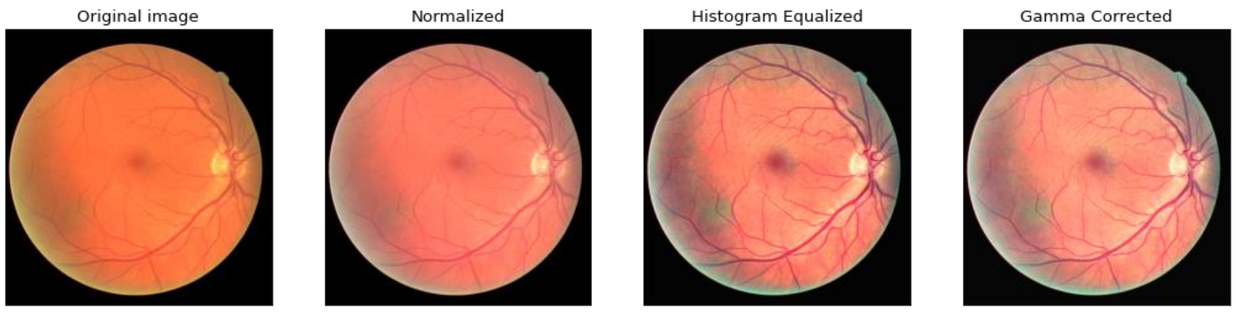


Figure 20: The sample input image after each preprocessing step

2.2.2. Dataset and Train-Validation split

My deep-learning-based method is a supervised approach with the help of the DRIVE dataset [8]. The dataset contains 20 fundus images for the training set, which are labelled, and 20 unlabeled images for the test set. In this project, only the training set is used. It is split into 16 training images and 4 images for validation. The training-validation split is manual, as I chose images 24, 26, 32, 34 for validation. The images 24 and 26 **(Figure 21)** are chosen for validation as the image 24 is the image that I need to report the accuracy, while image 26 (***image 25_test.tif*** as named in the course site) is the image that I am supposed to run my model on and submit the result. Hence, to be fair, I will not use images 24 and 26 for training.

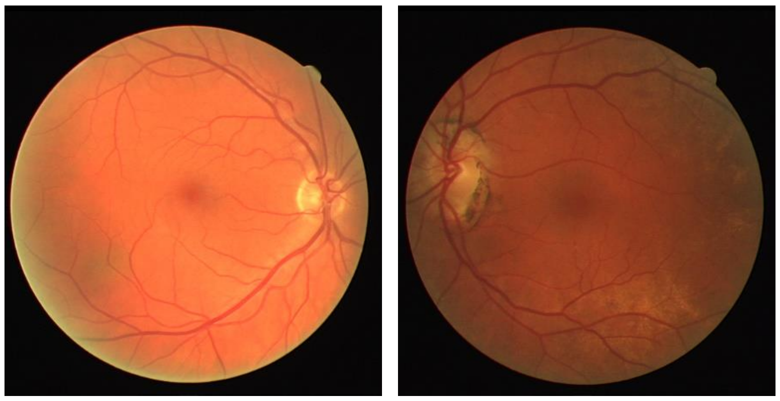


Figure 21: Image 24 (left) and image 26 (right) of the training set of DRIVE, which are used for validation

2.2.3. Creating patches of image

When doing training of the network, for the neural network to better focus on each region of the fundus image and to reduce the number of network parameters, small patches will be drawn randomly from the training and the validation set. For each image, 1000 patches of size 48x48 are extracted, and the algorithm ensures that there are at least 25 pixels of blood vessel in the patch.

Hence, at the end we will have 16000 patches for training and 4000 for validation. Some samples of extracted patch and its ground truth blood vessel is shown in **Figure 22**.

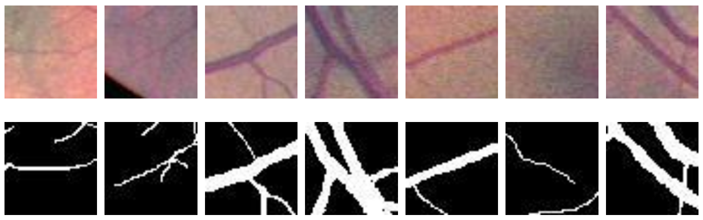


Figure 22: Some samples of extracted patch (upper) and its ground truth blood vessel (lower)

Since the neural network works on patches of 48x48 pixels, when doing inference, the image should be divided into patches of the same size. If the image's width or height is not divisible by 48, it should be zero-padded to be able to be divided into 48x48 patches. For example, if the original image size is 584x565, it should be zero-padded to create a 624x624 image. After each patch of the image has been segmented, they will be concatenated to construct the whole vessel image. Then the outer pixels which were padded in previously should be removed to form the final output.

2.2.4. The U-Net Architecture

The network architecture is U-Net [3], a well-known network for image segmentation. The network takes a fundus image as input, and output a segmented binary image by classifying each pixel as either part of the vessel or just the background. U-Net consists of a down-sampling path and an up-sampling part **(Figure 23)**. In the down-sampling path, several subsequent convolutional layers are applied to get the feature maps of the input image. In the up-sampling path, deconvolution layers are used to expand the feature map back to the original size, hence create the final binary output. Previous high-resolution features in the down-sampling path are concatenated with the features in the up-sampling path through skip-connection.

In the project, U-Net is implemented in Python using the TensorFlow library.

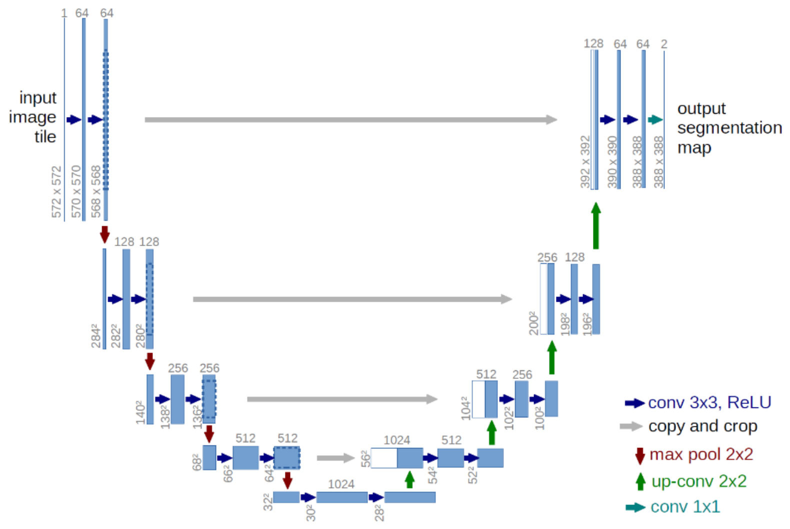


Figure 23: U-Net architecture [3]

2.2.5. *Training strategy*

The network is trained for 150 epochs. The loss function to be minimized by the network is the binary cross entropy loss function **(5)**, which is defined as follows

$$J = \frac{1}{N}\sum_{i=1}^{N}[y_i \log \hat{y}_i + (1 - y_i)\log(1 - \hat{y}_i)] \tag{5}$$

where y_i is the ground truth value of the pixel, $\hat{y}_i$ is the predicted value of the pixel given by the model, and N is the total number of pixels in every training patches.

The learning rate is set constant at 0.0003, and the batch size is 64.

3. Performance Metrics

The result for this task is evaluated mainly by accuracy, which shows the ratio of pixels of the final output that match exactly the pixels of the ground truth image. Accuracy is calculated by equation **(6)** below.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{6}$$

where TP denotes the number of true positives (i.e., pixels that are correctly classified as part of the vessel), TN denotes the number of true negatives (i.e., pixels that are correctly classified as background), FP denotes the number of false positives (i.e., pixels that are incorrectly classified as part of the vessel), and FN denotes the number of false negatives (i.e., pixels that are incorrectly classified as background).

However, accuracy may not be the best metric for this task, as the ratio of pixels that are part of the vessel is very low (about 10%). Hence, an output with high accuracy is not always what is desired. For example, the accuracy could be about 90% by just output a black image, with no pixel labelled as the blood vessel. Therefore, the F1-score **(7)** and the Matthew correlation coefficient (MCC) **(8),** as used in [1], are also adopted besides accuracy. Their formulae are as follow.

$$F1 = \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}} = \frac{2}{\frac{TP + FP}{TP} + \frac{TP + FN}{TP}} \tag{7}$$

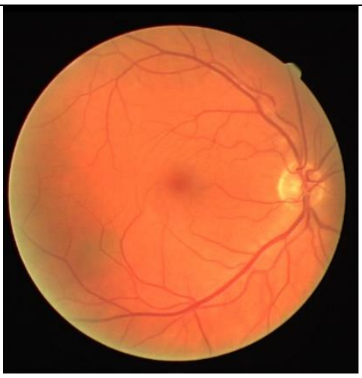
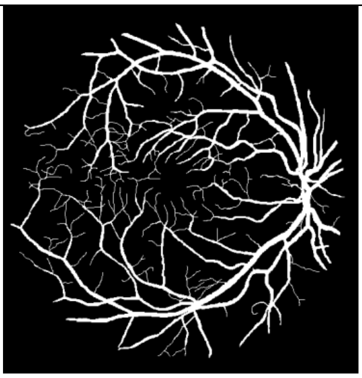
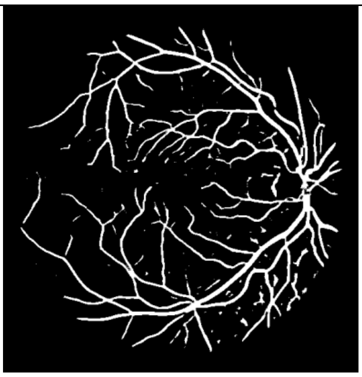
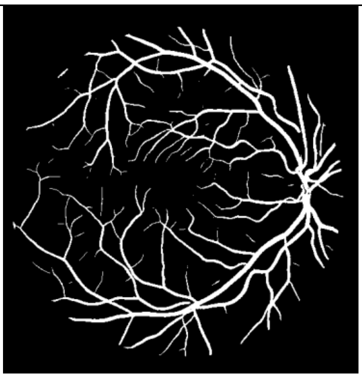
$$MCC = \frac{TN \times TP - FN \times FP}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{8}$$

4. Results and Discussion

4.1. Results

Firstly, I would like to show the results for the training image 24 (***24_training.tif***) following each of the methods in **Table 1**.

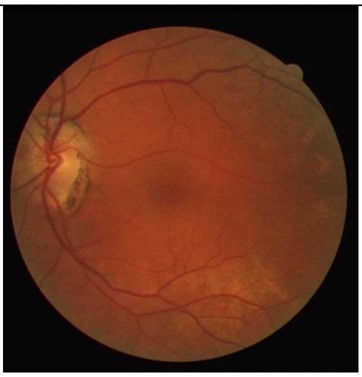
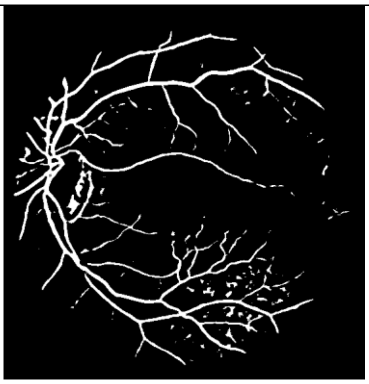
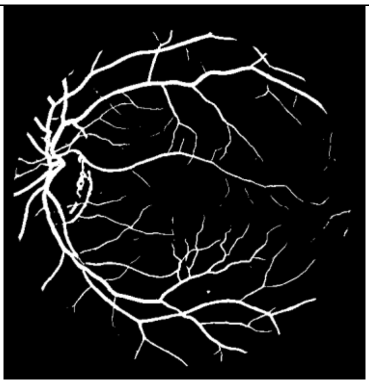
Table 1: Results for the sample image *24_training.tif*

Original Image	**Ground Truth**
	
Output by Adaptive Thresholding	**Output by U-Net**
 **Accuracy:** 95.212% **F1-score:** 76.705% **MCC:** 74.968%	 **Accuracy:** 96.107% **F1-score:** 81.385% **MCC:** 79.907%

Secondly, I would like to show the results for the training image 26 (*26_training.tif* in the DRIVE dataset or ***25_test.tif*** in the course site) that was assigned to me **(Table 2)**. As the output from the deep-learning-based method looks better, I will use it for my submission (file ***25_test_segmented.gif***).

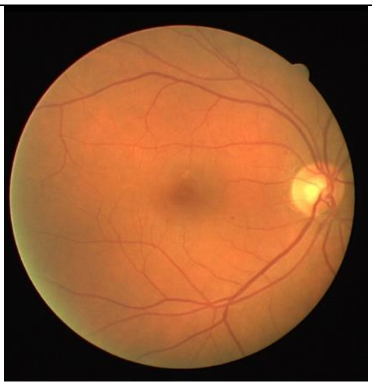
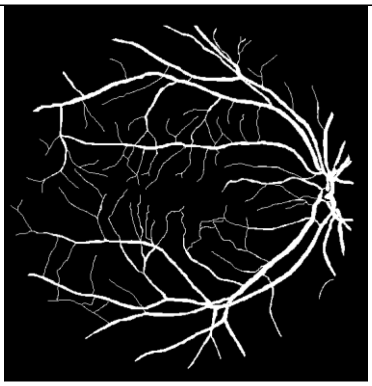
Table 2: Results for the assigned image *25_test.tif*

Original Image	

	
Output by Adaptive Thresholding	**Output by U-Net**
	

Lastly, I would like to show the results on other images in the validation set as well, which are images 32 **(Table 3)** and 34 **(Table 4)**.

Table 3: Results for the image *32_training.tif*

Original Image	**Ground Truth**
	
Output by Adaptive Thresholding	**Output by U-Net**

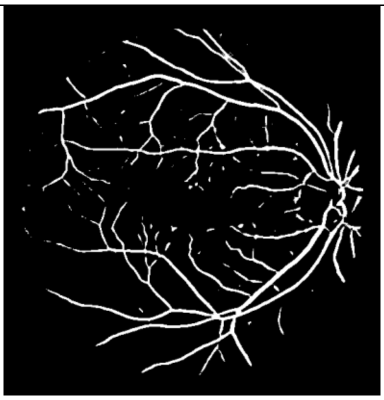
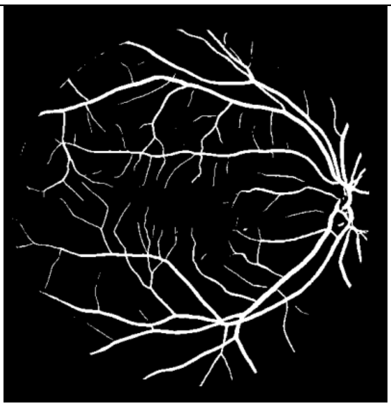
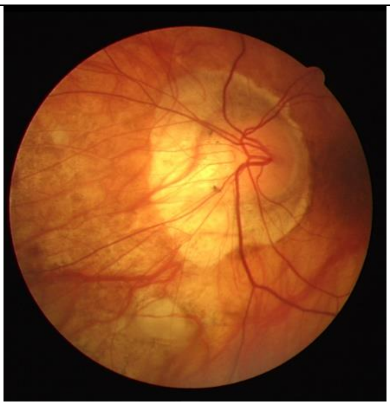
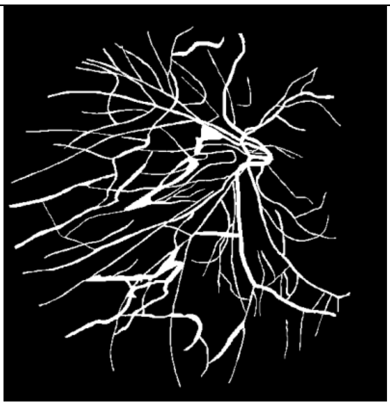
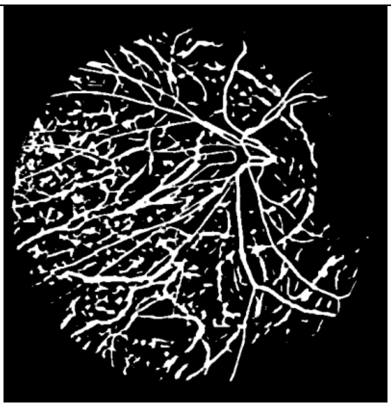
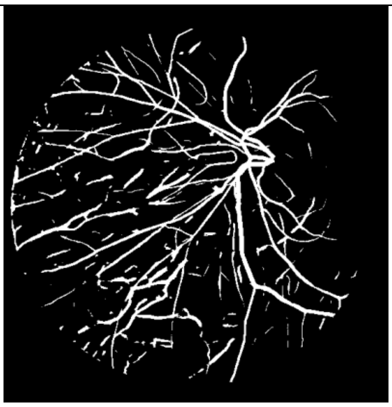
 **Accuracy:** 96.359% **F1-score:** 75.893% **MCC:** 74.310%	 **Accuracy:** 96.884% **F1-score:** 80.196% **MCC:** 78.635%

Table 4: Results for the image *34_training.tif*

Original Image	Ground Truth
	
Output by Adaptive Thresholding	**Output by U-Net**
 **Accuracy:** 89.489%	 **Accuracy:** 94.077%

F1-score: 57.880% **MCC:** 54.414%	**F1-score:** 68.011% **MCC:** 65.086%

4.2. Discussion

It can be seen that the U-Net based method outperforms the adaptive thresholding method in all cases. Although the accuracy is only improved by about 1%, the F1-score and MCC are significantly improved, by about 5%. This is because there are more parameters in the U-Net based methods, and those parameters are all learned by the machine. Hence, this method has more complexity and is able to detect vessels that are small or hard to see. On the other hand, the adaptive thresholding method have fewer parameters and all of them are hand-crafted by human, therefore it does not have the sufficient complexity to detect the small vessels. We can also see that in most case, the adaptive thresholding approach is not able to completely remove the bright optic disk of the fundus, while the deep learning approach is able to.

The adaptive thresholding method is also more prone to the noise in the fundus background, like what we see in **Table 4**. In this case, the adaptive thresholding method cannot give a decent output. However, the kind of image like ***34_training.tif*** **(Table 4)** creates a very challenging task for any automatic retinal vessel segmentation system. The noise in the image also negatively affects the performance of the U-Net based model, though to a smaller extent than it affects the adaptive-thresholding model.

An advantage of the adaptive-thresholding model to the U-Net based model is its faster inference time. This is because the deep neural network of U-Net takes a lot of time and space complexity.

5. Conclusion

In this project, I have explored two approaches to the task of automatic retinal vessel segmentation. Both give a decent output, but the deep-learning based one stands out to give better performance on evaluation metrics and better robustness compared to the adaptive-thresholding one. That explains why researchers in this field have become more interested in machine-learning and deep-learning-based vessel segmentation. However, both models did not give a perfect result, and further research should focus on how to address the image noise issue to improve the overall model performance.

Appendix 1: Code for the Adaptive Thresholding method

The code was implemented in Python on Google Colab. The link to the IPYNB file is as shown here.

https://drive.google.com/file/d/1PXeG3YJZv2h7uuTUeAakS95du8nJ-bbj/view?usp=sharing

A PDF and a HTML file of the code, named ***AdaptiveThreshold_code.pdf*** and ***AdaptiveThreshold_code.html*** respectively is attached to my submission on NTULearn. The HTML file may be better for viewing the notebook.

Appendix 2: Code for the Deep Learning based method

The code was implemented in Python on Google Colab. The link to the IPYNB file is as shown here.

https://drive.google.com/file/d/1qLWnNGrb5ynHqv0zY8qbJvVNONFR6Lnj/view?usp=share_link

A PDF and a HTML file of the code, named ***UNet_code.pdf*** and ***UNet_code.html*** respectively is attached to my submission on NTULearn. The HTML file may be better for viewing the notebook.

References

[1] X. Wang, X. Jiang, J. Ren, “Blood Vessel Segmentation from Fundus Image by a Cascade Classification Framework,” *Pattern Recognition*, vol. 88, pp. 331-341, Apr 2019.

[2] M. R. Mookiah, S. Hogg, T. J. MacGillivray, V. Prathiba, R. Pradeepa, V. Mohan, R. M. Anjana, A. S. Doney, C. N. A. Palmer, and E. Trucco, “A review of machine learning methods for retinal blood vessel segmentation and artery/vein classification,” *Medical Image Analysis*, vol. 68, p. 101905, Feb. 2021.

[3] O. Ronneberger, P. Fischer, and T. Brox, “U-Net: Convolutional Networks for Biomedical Image Segmentation,” In International Conference on Medical image computing and computer-assisted intervention, 2015, pp. 234–241.

[4] S. M. Pizer, E. P. Amburn, J. D. Austin, R. Cromartie, A. Geselowitz, T. Greer, B. T. H. Romeny, J. B. Zimmerman, and K. Zuiderveld, “Adaptive Histogram Equalization and its variations,” Computer Vision, Graphics, and Image Processing, vol. 39, no. 3, pp. 355–368, 1987.

[5] “Histograms,” *OpenCV*. [Online]. Available: https://docs.opencv.org/4.x/d6/dc7/group__imgproc__hist.html. [Accessed: 14-Nov-2022].

[6] A. Buades, B. Coll, and J.-M. Morel, “Non-local means denoising,” Image Processing On Line, vol. 1, pp. 208–212, 2011.

[7] “Denoising,” *OpenCV*. [Online]. Available: https://docs.opencv.org/4.x/d1/d79/group__photo__denoise.html. [Accessed: 14-Nov-2022].

[8] “DRIVE: Digital Retinal Images for Vessel Extraction,” *DRIVE Grand Challenge*. [Online]. Available: https://drive.grand-challenge.org/. [Accessed: 14-Nov-2022].

[9] Z. Yao, K. He, H. Zhou, Z. Zhang, G. Zhu, C. Xing, J. Zhang, Z. Zhang, B. Shao, Y. Tao, X. Sun, Y. Hou, M. Duan, S. Liu, L. Huang, and F. Zhou, “Eye3DVas: Three-dimensional reconstruction of retinal vascular structures by integrating fundus image features,” *Frontiers in Optics / Laser Science*, 2020.

[10] “Miscellaneous Image Transformations,” *OpenCV*. [Online]. Available: https://docs.opencv.org/4.x/d7/d1b/group__imgproc__misc.html. [Accessed: 14-Nov-2022].

[11] F. Bolelli, S. Allegretti, L. Baraldi, and C. Grana, “Spaghetti labeling: Directed acyclic graphs for block-based connected components labeling,” *IEEE Transactions on Image Processing*, vol. 29, pp. 1999–2012, 2020.

[12] “Structural Analysis and Shape Descriptors,” *OpenCV*. [Online]. Available: https://docs.opencv.org/3.4/d3/dc0/group__imgproc__shape.html. [Accessed: 14-Nov-2022].

[13] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 4th., New York, NY: Pearson, 2018.

[14] “Image Filtering,” *OpenCV*. [Online]. Available: https://docs.opencv.org/3.4/d4/d86/group__imgproc__filter.html. [Accessed: 14-Nov-2022].