Computer Vision Course: Mini Project #4

Morphological Preprocessing for Retinal Vessels Locating

Seyed Mohammad Taha Tabatabaei

Deadline: Dec 11, 2023

Report date: Dec 26, 2023

Professor: Dr. Pourreza

Abstract

This article aims to evaluate the performance of Retinal-VesselNet, which is a neural network with U-net architecture, on two widely used retinal image datasets - Digital Retinal Images for Vessel Extraction (DRIVE) and STructured Analysis of the Retina (STARE). The article also explores two separate morphological preprocessing methods to improve vessel detection in retinal images. Detailed sensitivity and specificity metrics are used to compare the performance of the model and the preprocessing methods. The results suggest that while the model can accurately detect major vessels and their subtrees, there is still room for improvement in detecting all vessels. The article provides detailed tables, images, and code to facilitate further research on this topic.

Methods

1. Evaluate Retinal-VesselNet with DRIVE and STARE

Retinal-VesselNet is a Neural network with U-net architecture. We aimed to evaluate this tool with two famous retinal image datasets, Digital Retinal Images for Vessel Extraction (DRIVE) and STructured Analysis of the Retina (STARE). You can access DRIVE from here and STARE from here.

I have trained the Model provided by this repository[1] on github with images from DRIVE, using google Colab and with GPU enabled. The training loss after 150 epochs was 0.0304 and the validation loss was 0.1657.

Sensitivity and Specificity

To compare two datasets, we use sensitivity and specificity metrics. After training the model, I used the script *TestAndEvaluation.ipynb* in the mentioned repository to test datasets. For the STARE dataset, masks were unavailable, so I generated the masks by using the script you can find in appendix [1]. I assume that in the black background, all the pixels' values in each channel must be below a certain threshold. You can see the thresholds in image [1]. You can see the masks for each channel in image [2] and the final mask in image [3].

2. Morphological preprocessing

Method one

As I explored different sources, I found the References [1] with an interesting preprocessing procedure. It uses a Top Hat transform to enhance vessels. As the paper suggests, using a linear structuring element with a length near the diameter value of the largest vessel can be a good choice. Despite that, the paper indicates that because there will be no change in vessel shape if the vessel and structuring element are in parallel, we should convolve the element in various directions. In the end, the article says in order to suppress the unwanted noise generated during applying the transform,

[1] https://github.com/DeepTrial/Retina-VesselNet

we must apply a Gaussian filter. With respect to all the notes above, you can see the first preprocessing method script in appendix [2]. You can see a sample image before [4] and after [5] preprocessing.

It is important to note that the main repository had a default preprocessing pipeline, so in both new preprocessing methods I have introduced in this report, I used my preprocessing method in addition to the default preprocessing.

Method two

For the second preprocessing method, I tried to mix what I had learned after reading a number of papers. So, the idea I tried in this section is somehow novel, as I did not find it explicitly in any papers I read. The idea is based on the fact that Bottom Hat transform, gives us the vessel, and removes the background. So, my procedure has the following steps. 1) apply the Bottom Hat transform to the image. 2) dilate the result to get more strong vessels (thicker vessels). 3) subtract the result from the main image. The result of the first step is vessels in white and background in black. In the second step, dilation extends the white area, which is our desired vessels. Finally, subtracting helps us to have vessels in the main image thicker but in darker pixels. See the sample image from DRIVE in [6], after the first step in [7], after the second step in [8], and the final step in [9]. It is worth mentioning that we apply this procedure only on the green channel of the RGB input image. It is because applying this on other channels will result in a noisy result, especially the blue channel. See the code for this method in appendix [3]. I added all other references I read in the References section.

Discussion

1. Evaluate Retinal-VesselNet with DRIVE and STARE

Sensitivity and Specificity

The results are shown in the table [1]. About specificity, we can find that the model does not have a hard job at finding pixels that are not a vessel, as specificity checks the TNR (true negative over all negative).

On the other hand, the model cannot perform well in finding all the vessels. From the outputs result, the model can find major vessels and their subtree, but 77% on DRIVE is not so satisfying.

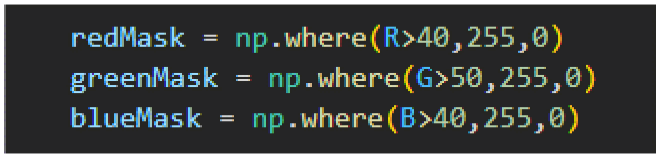
You can see a predicted vessel tree on a DRIVE dataset sample in image [10] and the original RGB photo in image [11]. The relevant images for STARE are [12] and [13]. See the confusion matrix for DRIVE in [14] and for STARE in [15].

2. Morphological preprocessing

You can see the sensitivity and specificity for both DRIVE and STARE images with first and second preprocessing methods in table [2]. The relevant confusion matrix for first method in DRIVE is on image [16], for second method in DRIVE in image [17], for first method in STARE in image [18] and for second method in STARE in image [19]. It shows that my own preprocessing did a good job on DRIVE dataset.

Results

Images:



```
redMask = np.where(R>40,255,0)
greenMask = np.where(G>50,255,0)
blueMask = np.where(B>40,255,0)
```

Figure 1: Mask threshold

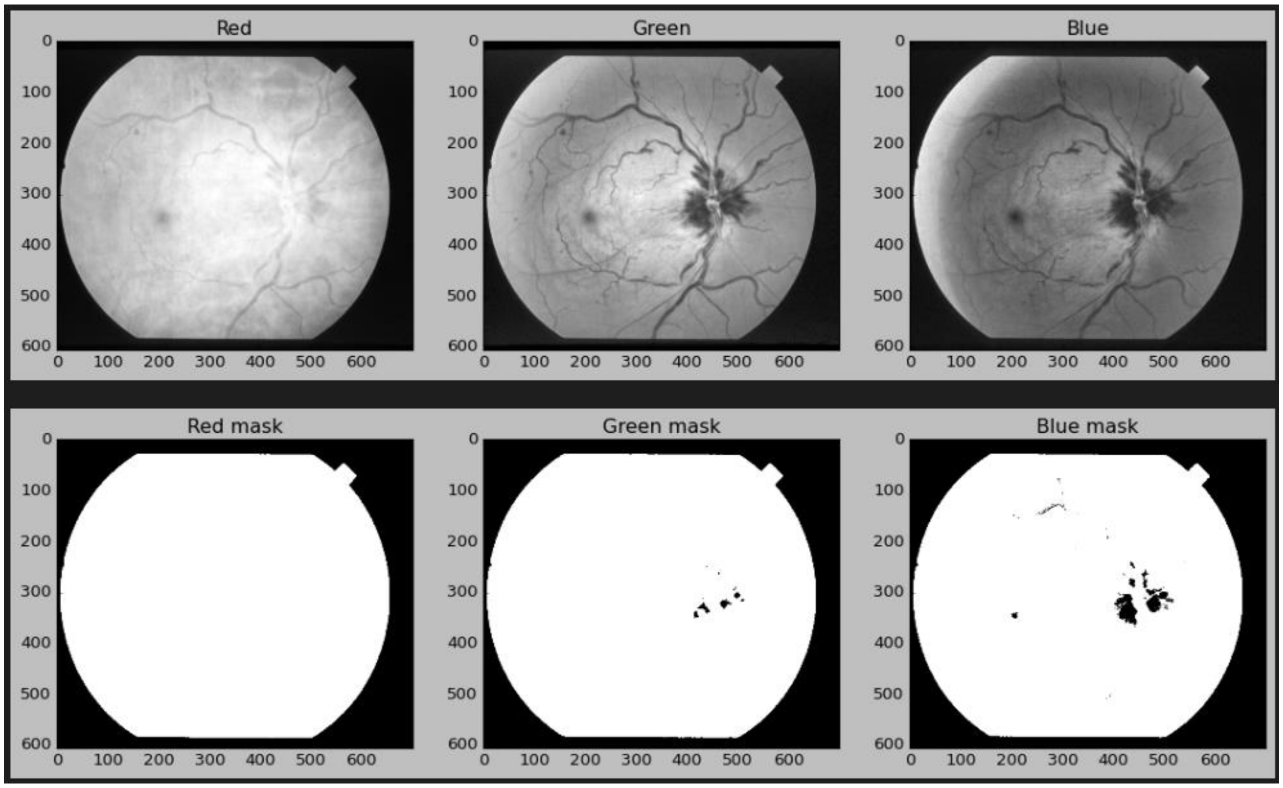


Figure 2: RGB channels for a sample image and it's masks

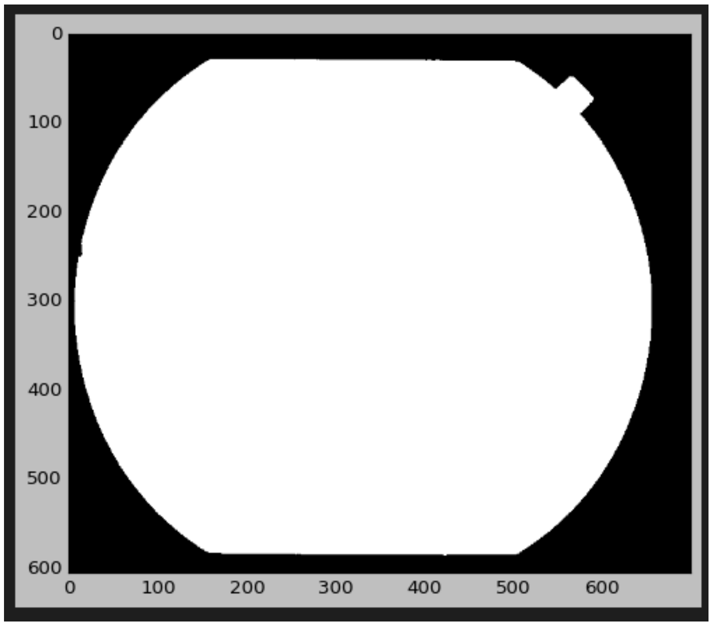


Figure 3: Final mask

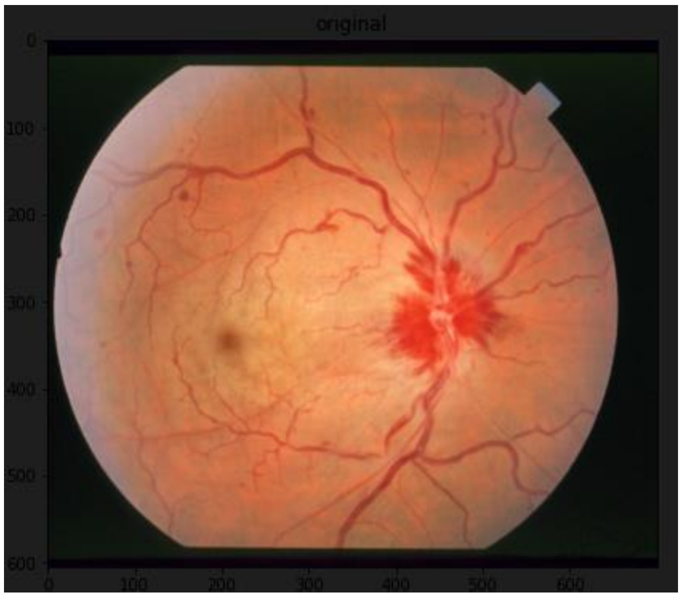


Figure 4: Before Top Hat preprocessing

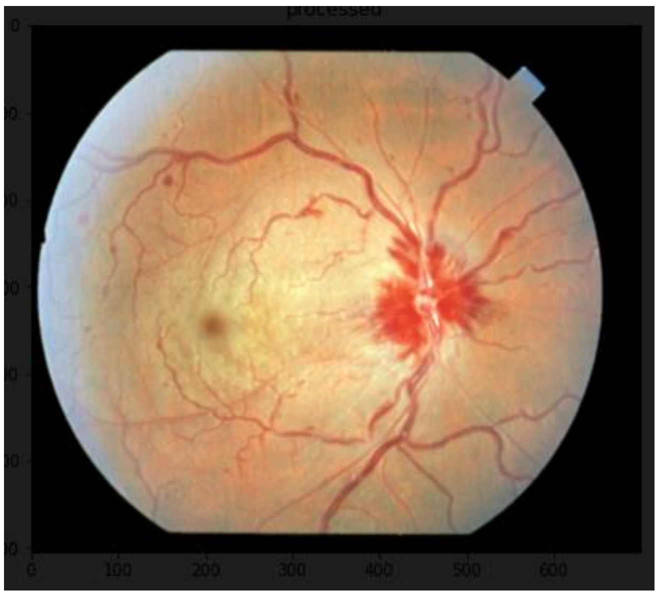


Figure 5: After Top Hat preprocessing

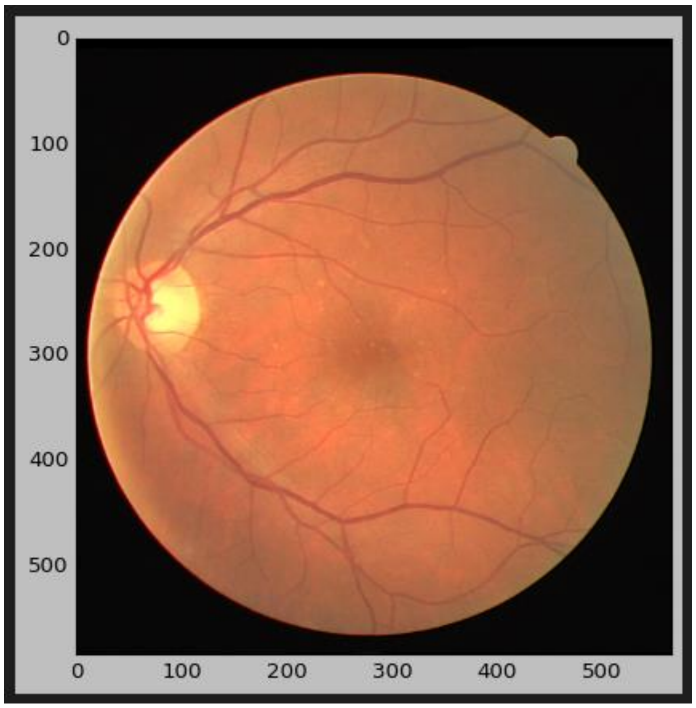


Figure 6: Sample image before 2nd preprocessing method

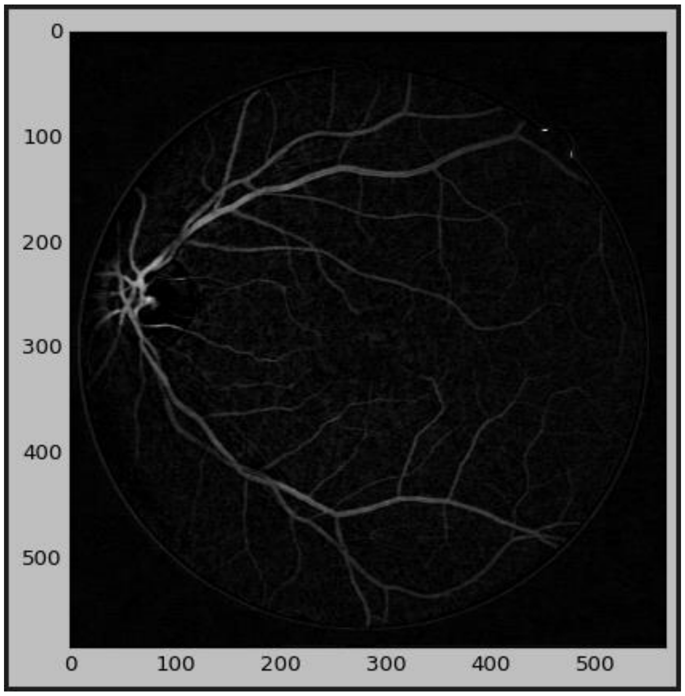


Figure 7: After 1st step

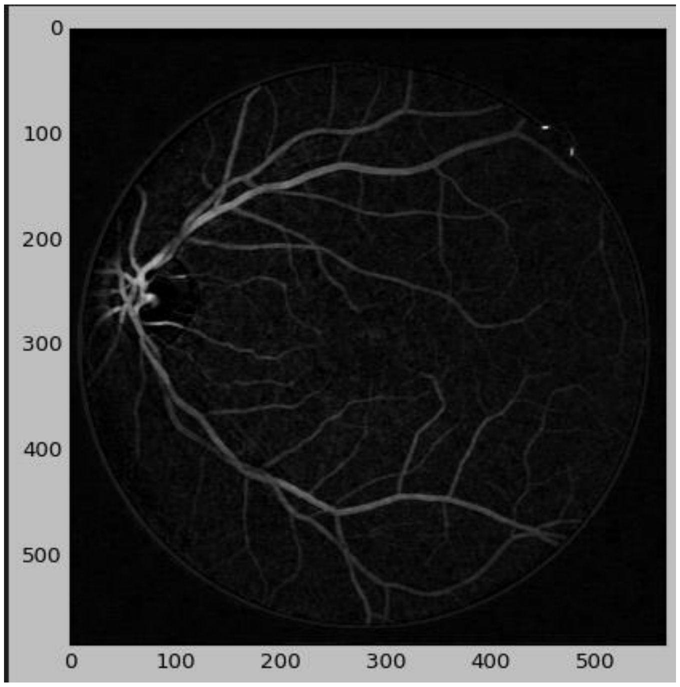


Figure 8: After 2nd setep

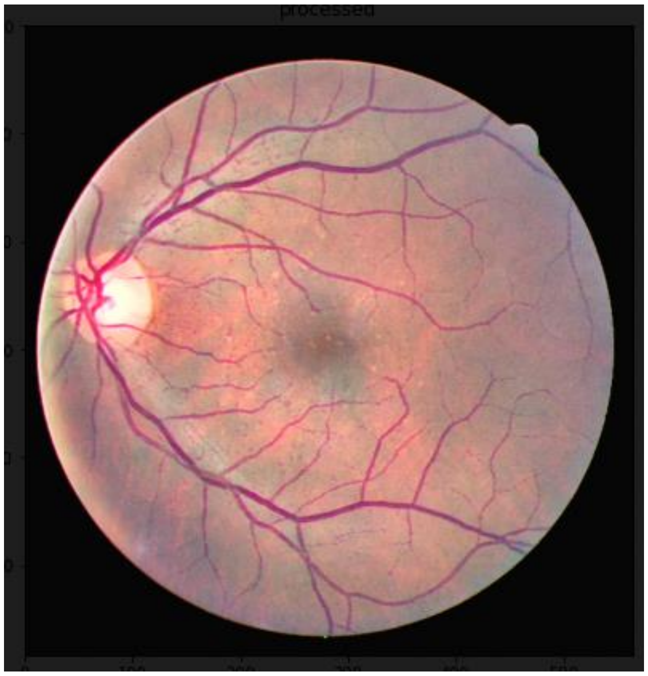


Figure 9: After 3rd step

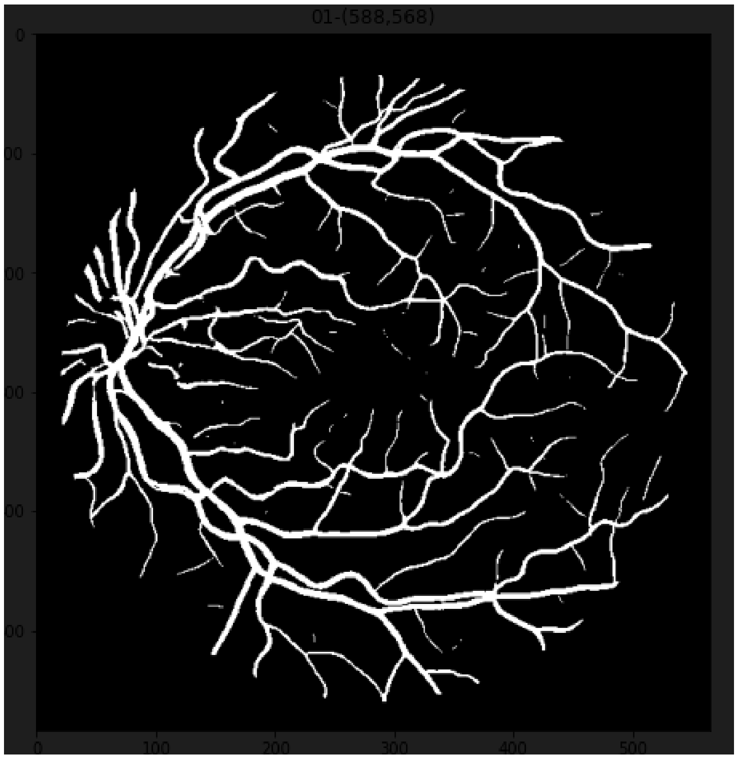


Figure 10: A represented vessel tree by model from DRIVE

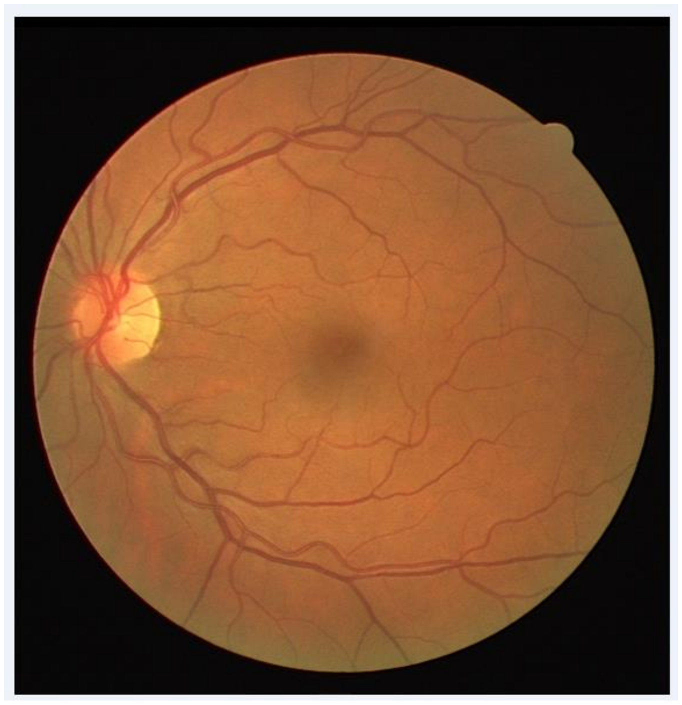


Figure 11: A sample of DRIVE

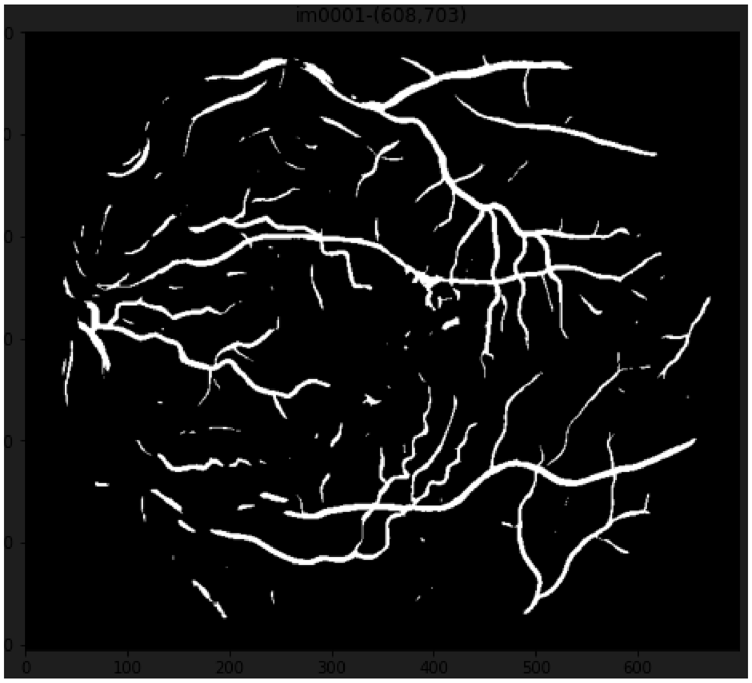


Figure 12: A represented vessel tree by model from STARE

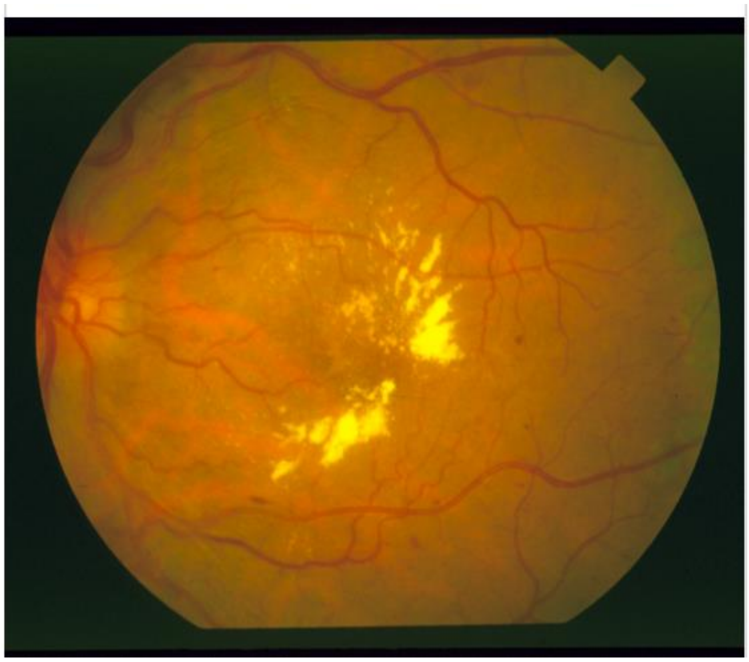


Figure 13: A sample from STARE

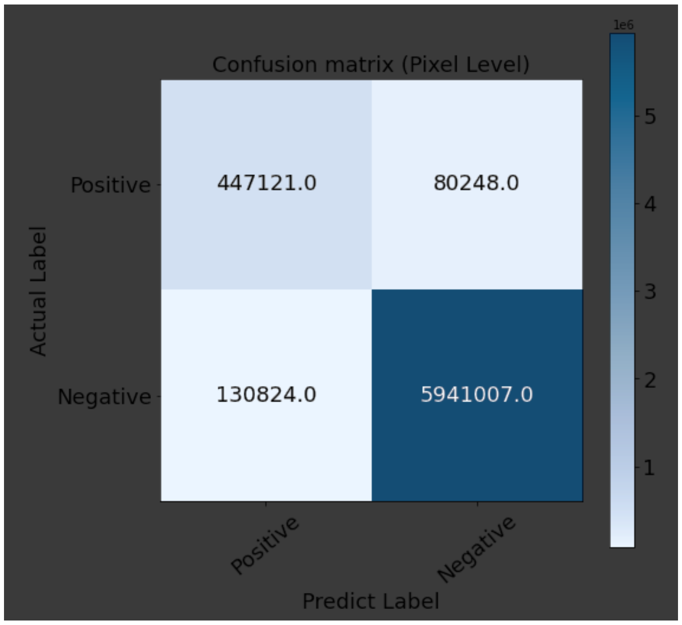


Figure 14: Matrix for default model with DRIVE

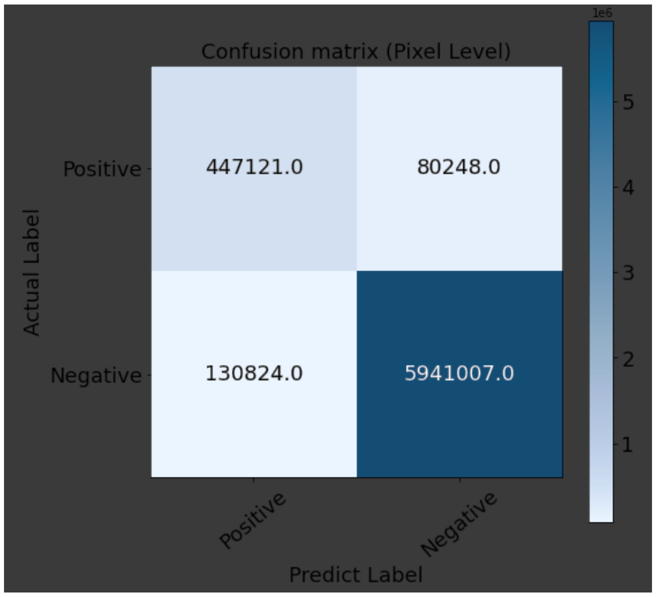


Figure 15: Matrix for default model with STARE

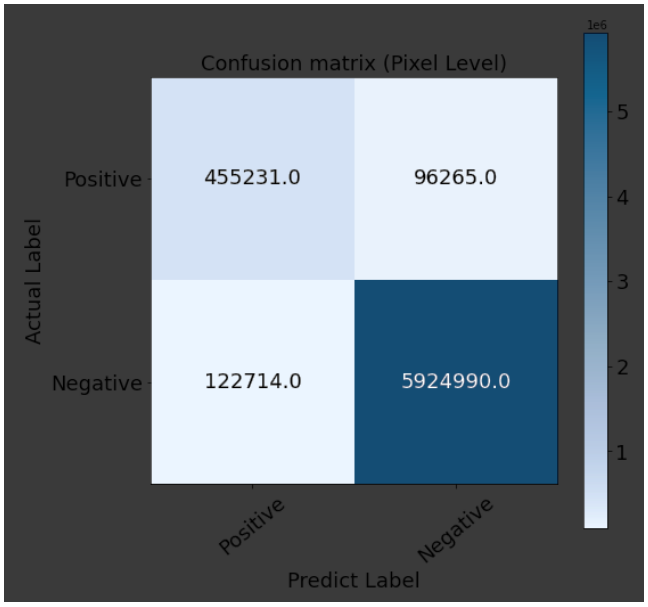


Figure 16: Method 1 with DRIVE

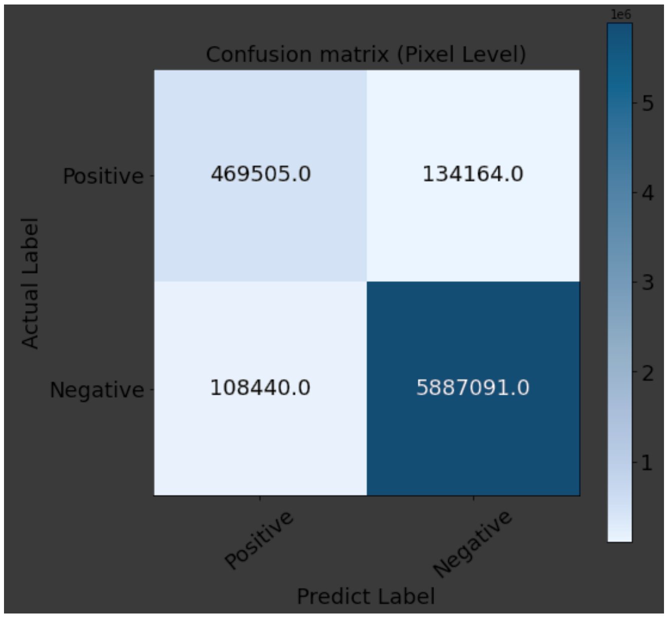


Figure 17: Method 2 with DRIVE

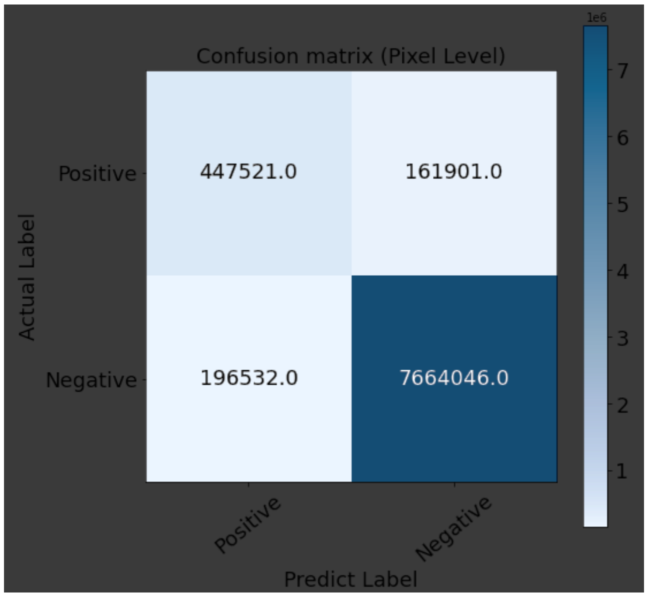


Figure 18: Method 1 with STARE

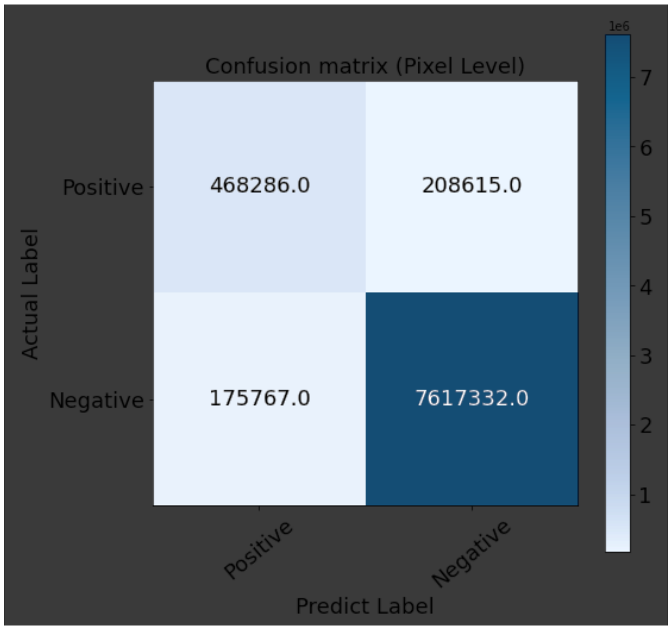


Figure 19: Method 2 with STARE

Tables:

Table 1: Sensitivity & Specificity in default model

	DRIVE	*STARE*
Sensitivity	0.7760	0.7383
Specificity	0.9866	0.9799

Table 2: Preprocessing methods comparison

	DRIVE method 1	*DRIVE method 2*	*STARE method 1*	*STARE method 2*
Sensitivity	0.7898	0.8142	0.6852	0.7155
Specificity	0.9840	0.9777	0.9792	0.9732

Appendix

[1] : Generating mask for images in STARE dataset

```
from glob import glob
import matplotlib.pyplot as plt
import numpy as np
import os
dataset_path = "E:\\taha\\code\\Retina-VesselNet\\STARE-labeled\\"
test_image_dir = dataset_path + "stare-images\\"
test_mask_dir = dataset_path + "mask\\"

test_image_path_list=glob(test_image_dir+"*.ppm")
print(test_image_path_list[0])
print("number of testing images:",len(test_image_path_list))


if not os.path.exists(test_mask_dir):
    os.mkdir(test_mask_dir)

for i in range(len(test_image_path_list)):
    image_path=test_image_path_list[i]
    image_name=image_path.split("\\")[-1].split(".")[0]
    print(image_name)

    image=plt.imread(image_path)
    original_shape=image.shape
    print(original_shape)

    plt.imshow(image)
    plt.show()

    R = image[:,:,0]
    G = image[:,:,1]
    B = image[:,:,2]

    fig = plt.figure(figsize=(16,16))

    fig.add_subplot(1,3,1)
    plt.imshow(R,cmap=plt.cm.gray)
    plt.title("Red")

    fig.add_subplot(1,3,2)
    plt.imshow(G,cmap=plt.cm.gray)
    plt.title("Green")

    fig.add_subplot(1,3,3)
    plt.imshow(B,cmap=plt.cm.gray)
    plt.title("Blue")

    plt.show()

    redMask = np.where(R>40,255,0)
    greenMask = np.where(G>50,255,0)
    blueMask = np.where(B>40,255,0)

    fig = plt.figure(figsize=(16,16))

    fig.add_subplot(1,3,1)
    plt.imshow(redMask,cmap=plt.cm.gray)
    plt.title("Red")

    fig.add_subplot(1,3,2)
    plt.imshow(greenMask,cmap=plt.cm.gray)
    plt.title("Green")

    fig.add_subplot(1,3,3)
    plt.imshow(blueMask,cmap=plt.cm.gray)
    plt.title("Blue")

    plt.show()

    mask = np.where( (redMask<1)  & (greenMask<1) & (blueMask<1), 0,250 )

    print(mask.shape)

    plt.imshow(mask,cmap=plt.cm.gray)
    plt.show()

    plt.imsave(f"{test_mask_dir+image_name}_mask.gif",mask,cmap=plt.cm.gray
)
```

[2] : Top Hat preprocessing script based

```
def morpho(img):
  (R,G,B) = cv2.split(img)

  # Calculate the maximum response of 12 directions
  opening_kernel = cv2.getStructuringElement(cv2.MORPH_RECT,
(opening_kernel_size, 1)).astype(np.uint8)  # Ensure the data type is uint8

  topsG = []
  topsR = []
  topsB = []
  for angle in range(0, 180, 15):
      mat = cv2.getRotationMatrix2D((opening_kernel_size // 2, 0), angle,
1)
      topsR.append(cv2.morphologyEx(R, cv2.MORPH_TOPHAT, opening_kernel,
mat))
      topsG.append(cv2.morphologyEx(G, cv2.MORPH_TOPHAT, opening_kernel,
mat))
      topsB.append(cv2.morphologyEx(B, cv2.MORPH_TOPHAT, opening_kernel,
mat))

  R1 = np.max(topsR, axis=0)
  G1 = np.max(topsG, axis=0)
  B1 = np.max(topsB, axis=0)

  R7 = R + R1
  G7 = G + G1
  B7 = B + B1

  R7 = cv2.GaussianBlur(R7, (5, 5), 0)
  G7 = cv2.GaussianBlur(G7, (3, 3), 0)
  B7 = cv2.GaussianBlur(B7, (3, 3), 0)

  img2 = cv2.merge([R,G7,B])

  return img2
```

[3]: Second method preprocessing based on Bottom Hat

```
def morpho(img):
  (R,G,B) = cv2.split(img)

  size =7
  kernel = np.ones((size, size), np.uint8)

  R1 = cv2.morphologyEx(R, cv2.MORPH_BLACKHAT, kernel)
  R1 = np.where(R1>5,R1,0)
  R2 = R - R1

  G1 = cv2.morphologyEx(G, cv2.MORPH_BLACKHAT, kernel)
  G1 = np.where(G1>10,G1,0)
  G2 = G - G1

  B1 = cv2.morphologyEx(B, cv2.MORPH_BLACKHAT, kernel)
  B1 = np.where(B1>10,B1,0)
  B2 = B - B1

  size =2
  kernel = np.ones((size, size), np.uint8)

  R8 = cv2.dilate(R1,kernel)
  G8 = cv2.dilate(G1,kernel)
  B8 = cv2.dilate(B1,kernel)

  R9 = R - R8
  G9 = G - G8
  B9 = B - B8

  img2 = cv2.merge([R,G9,B])

  return img2
```

References

[1] : Gehad Hassan, Nashwa El-Bendary, Aboul Ella Hassanien, Ali Fahmy, Shoeb Abullah M., Vaclav Snasel. “Retinal Blood Vessel Segmentation Approach Based on Mathematical Morphology”. Procedia Computer Science, Volume 65, 2015, Pages 612-622, ISSN 1877-0509.

[2]: M. Elena Martinez-Perez, Alun D. Hughes, Simon A. Thom, Anil A. Bharath, Kim H. Parker. “Segmentation of blood vessels from red-free and fluorescein retinal images”. Medical Image Analysis, Volume 11, Issue 1, 2007, Pages 47-61, ISSN 1361-8415.

[3]: Bob Zhang, Lin Zhang, Lei Zhang, Fakhri Karray. “Retinal vessel extraction by matched filter with first-order derivative of Gaussian”. Computers in Biology and Medicine, Volume 40, Issue 4, 2010, Pages 438-445, ISSN 0010-4825.

[4]: Elaheh Imani, Malihe Javidi, Hamid-Reza Pourreza.”Improvement of retinal blood vessel detection using morphological component analysis”. Computer Methods and Programs in Biomedicine, Volume 118, Issue 3, 2015, Pages 263-279, ISSN 0169-2607.

[5]: Soomro, T.A.; Ali, A.; Jandan, N.A.; Afifi, A.J.; Irfan, M.; Alqhtani, S.; Glowacz, A.; Alqahtani, A.; Tadeusiewicz, R.; Kantoch, E.; Zheng, L. Impact of Novel Image Preprocessing Techniques on Retinal Vessel Segmentation. Electronics 2021, 10, 2297. https://doi.org/10.3390/electronics10182297