Retinal vessel segmentation

Berzoini Raffaele

I. Introduction

Retinal vessels can be a very accurate indicator for many diagnoses such as diabetes, hypertension, and arteriosclerosis. Being able to detect correctly the retinal vessels can help to perform accurate measurement of all the features that permit to make diagnoses. In this report, I am going to describe an automated method to locate vessels in images of the ocular fundus. All the pictures were taken by a TopCon TRV-50 fundus camera at 35° field of view. All the images were provided by the STARE project [1].
There were two main problems during the segmentation: retinopathies and the fact that a global threshold would not have worked well on the images.
Applying a global threshold, as shown in Fig. 1, causes a correct classification only in certain areas if the background is uneven.

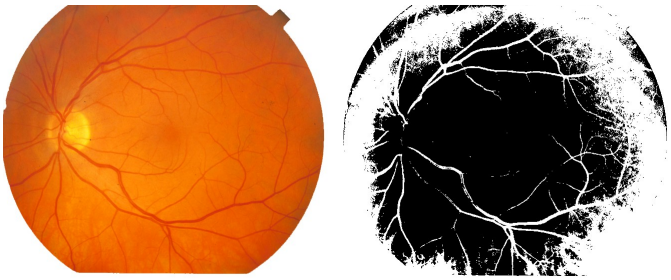

Figure 1. Example of a global threshold application.

Therefore I decided to adopt an adaptive-threshold technique, simplified and computationally less expensive compared to [2] and [3].
Another not-under valuable problem is the one due to retinopathies such as diabetic retinopathy or retinitis, as shown in Fig. 2. They could lead to the wrong classification of ellipse-shaped areas and also to a lot of false vessels in regions where the background is highly uneven.

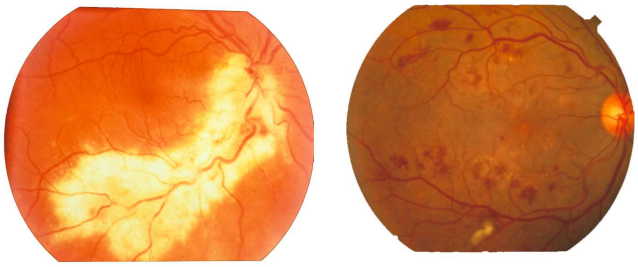

Figure 2. Left: retinitis; Right: diabetic retinopathy

This method aims to work automatically on every image, however, a group of three parameters can be adjusted in order to obtain the best result.

II. Algorithm

The algorithm is divided into 3 parts:

1) Green channel extraction.
2) Adaptive threshold application.
3) Noise reduction.

A. Green channel extraction

A BGR image is composed of three channels. In this case, all the images are 24-bit, so each channel has 8 bits, for red, green, and blue. Every pixel is represented by three values between 0 and 255. After the extraction of the three channels, it can be noticed how the red channel is saturated, the blue one is empty, while the green channel is the best one to operate on.
The code of this function can be found under the *extractChannel(src, channel)* signature, where *src* is a NumPy array where the pixel values are stored and *channel* is a String indicating which channel has to be extracted.

B. Adaptive threshold application

Among a fair number of methods, three were the best performing:

- Adaptive gaussian thresholding.
- Adaptive mean thresholding.
- Otzu's binarization.

The first method returns a threshold value as a Gaussian-weighted sum of the pixel neighborhood values. The dimension of the neighborhood is initialized at the beginning and remains constant for each pixel analyzed. The Gaussian-weighted sum is calculated for every pixel. After the calculation, a constant is subtracted before returning the value of the local threshold. This subtraction is performed in order to homogenize the analysis along contiguous pixels.

The second method operates in the same way as the previous one. The only difference is that the Gaussian-weighted sum is substituted by the calculation of the average of the pixels belonging to the actual neighborhood. As in the previous method, a constant is subtracted for the reason explained above.

The third method, Otzu's binarization, is a little bit more complex. Otsu's thresholding method involves iterating through all the possible threshold values and calculating a measure of spread for the pixel levels on each side of the

threshold, i.e. the pixels that either fall in foreground or background. The aim is to find the threshold value where the sum of foreground and background spreads is at its minimum. At each possible threshold the 'Within-Class Variance' is calculated:

$$WCV = W_b * {\sigma_b}^2 + W_f * {\sigma_f}^2 \quad (1)$$

where W_b is the weighted sum of the pixels belonging to the background, W_f of the pixels belonging to the foreground, and σ^2 is the variance. In the end, the threshold chosen is the one with the lowest WCV. The details of this method can be found in [4].

After having compared all the three methods the best results were obtained with the adaptive mean thresholding. The code is located in the *adaptiveThreshold(gray, mask_dim, constant)* function: *gray* is the NumPy array storing the values of the gray-scale image after the II-A; *mask_dim* is an int value that indicates the dimension of the neighborhood to analyze and *constant* is the int value subtracted after each calculation of the average of the intensity of the pixels. The function works as follows:

1) It is checked if *mask_dim* is odd, if not, it is properly corrected.
2) The *ext* int variable is set as $ext = \frac{mask\_dim}{2}$.
3) The margins are reproduced with the *cv2.copyMakeBorder()* function from the OpenCV package, in order to calculate the threshold also on the edges of the image.
4) A double *for()* loop is performed on the image in order to scan all the pixels.
5) At each pixel his neighborhood is extracted and stored in the *mask* variable.
6) The average of the neighborhood is computed via the *numpy.mean()* function and the *constant* value is subtracted to obtain the local threshold.
7) In the end the pixel is classified as background or as foreground depending on whether it is greater or smaller than the threshold[1].

In the *adaptiveThreshold()* function just described there are two parameters that influence the output: *mask_dim* and *constant*

- *mask_dim* is the side of the squared neighborhood considered for the calculation of the average intensity. With the same *constant* value, a higher *mask_dim* implies a higher sensitivity to signal and, as a consequence, to noise, as shown in Fig. 3. Moreover a higher *mask_dim* has worse results with images with uneven backgrounds.
- *constant* is the value subtracted after the computation of the average. With the same *mask_dim* a higher *constant* value leads to a smaller sensitivity to noise but also to signal, as shown in Fig. 4. With uneven background, a higher constant is to prefer.

[1]Note that at this point of the algorithm and until the end of all the procedure the pixels labeled as vessels are set to 0, while the background is set to 255

C. Noise reduction

In this final part of the algorithm, the ideal goal is to remove all the pixels wrongly classified. To do so, I implemented two different noise reduction functions. They are separated because one of them needs the third parameter mentioned in I.

1) Noise reduction 1: The first function is *noiseReduction(image, size)*, where *image* is the Output of II-B and *size* is an int variable indicating the minimum dimension (in pixels) that a certain area has to have to not be classified as background. This function iterate through all the pixels of the image, if a pixel is labeled as vessel the *floodFill()*[2] function is applied starting from that pixel. Then if *floodFill()* returns a size smaller than the *noise_size* set in *main()* that area is labeled as background.

2) Noise reduction 2: The second function is *noiseReduction2(image)* where *image* is the resulting image after the first noise reduction. In this part, seeing as how retinopathies often lead to the wrong classification of ellipse-shaped areas, a set of circles and ellipses is generated through the *generateEllipse()* function. Then each of these shapes is compared with the image through the *shapeComparison()* function which extracts a mask in the same way *adaptiveThreshold()* does. The mask is compared with the shape and, if there is a correspondence higher than *tolerance*, it means that the mask analyzed contains noise so the ellipse-shaped area contained in the mask is labeled as background.

After this, *noiseReduction()* is performed again in order to delete all the little areas left by the removal executed by *noiseReduction2()*

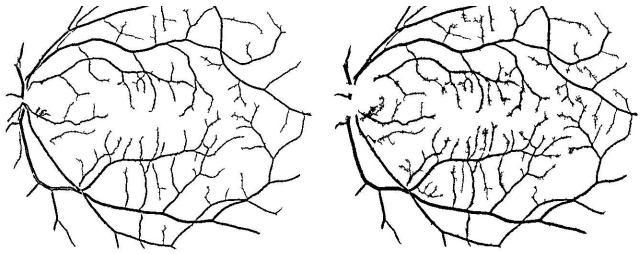

Figure 3. Left: mask = 11; Right: mask = 21

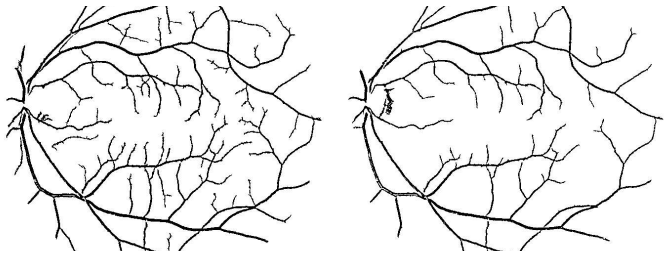

Figure 4. Left: constant = 2; Right: constant = 4

[2]The *floodFill()* function fills with a certain value all the pixels that are connected to the starting one and have their values included in a certain range around the value of the first pixel

III. Results

Since this algorithm aims to be an automated method to segment retinal vessels I'm going to report the results performed on all the images with the same three parameters, so none of the three parameters were adjusted to get the best result for each image. To evaluate the performance of the algorithm twenty images have been compared with the ground-truth. The ground-truth is set by hand-labeled images executed by two experts from [1]. The second expert provided a less conservative set of hand-labeled images classifying around 40% more pixels as vessels than the first expert. The column "Parameters" indicates the three values given, respectively, to *mask_dim*, *constant*, and *noise_size*.

Table I
IoU and F1 scores obtained with different sets of parameters

Parameters	Expert	Avg F1	StD F1	Avg IoU	StD IoU
21-2-100	1	69.20%	8.13%	53.44%	8.70%
21-2-100	2	66.87%	9.23%	50.90%	9.65%
11-2-100	1	70.39%	9.35%	55.07%	10.55%
11-2-100	2	63.12%	8.89%	46.71%	9.24%
21-4-400	1	69.26%	9.88%	53.81%	10.99%
21-4-400	2	61.63%	9.11%	45.14%	9.21%
11-1-400	1	69.78%	7.14%	54.03%	8.04%
11-1-400	2	65.60%	7.57%	49.26%	8.02%

The F1, or dice, score is defined as 2×the Area of Overlap divided by the total number of pixels in both images.
The IoU score (Intersection-over-Union) is defined as the area of overlap between the predicted segmentation and the ground truth divided by the area of union between the predicted segmentation and the ground truth.
Given the following definitions:

- True Positive (TP): real vessel labeled as vessel.
- True Negative (TN): real background labeled as background.
- False Positive (FP): real background labeled as vessel.
- False Negative (FN): real vessel labeled as background.

The scores are:

$$F1 := \frac{2 \times TP}{2 \times TP + FP + FN} \tag{2}$$

$$IoU := \frac{TP}{TP + FP + FN} \tag{3}$$

They are positively correlated, meaning if one says model A is better than model B at segmenting an image, then the other will say the same. They both range from 0 to 1 (or 0% to 100%), with 1 signifying the greatest similarity between predicted and truth.

It's better to evaluate the performances of the algorithm with IoU and F1 instead of sensitivity, specificity, and accuracy because in these binary images one of the two classes, that is the background, includes about 90% of the pixels. This great imbalance between the two classes returns misleading results, like accuracy and specificity that are always above 90% and in most cases above 95%. So it's difficult to find a threshold to divide bad and good results, e.g. classifying everything as background we can obtain an average accuracy of 90%, at the opposite IoU and F1 return results equal to 0.00%.

I reported in Table I the results of IoU and F1 over four different sets of parameters. The results obtained with the second expert are worse, this means that my classifier tends to not be able to locate the smallest vessels. In Fig. 5 a visualization of a result for each of the four sets of parameters compared to the hand-labeled image from the expert two, which is the ground-truth that returns worse IoU and F1 scores after comparison.

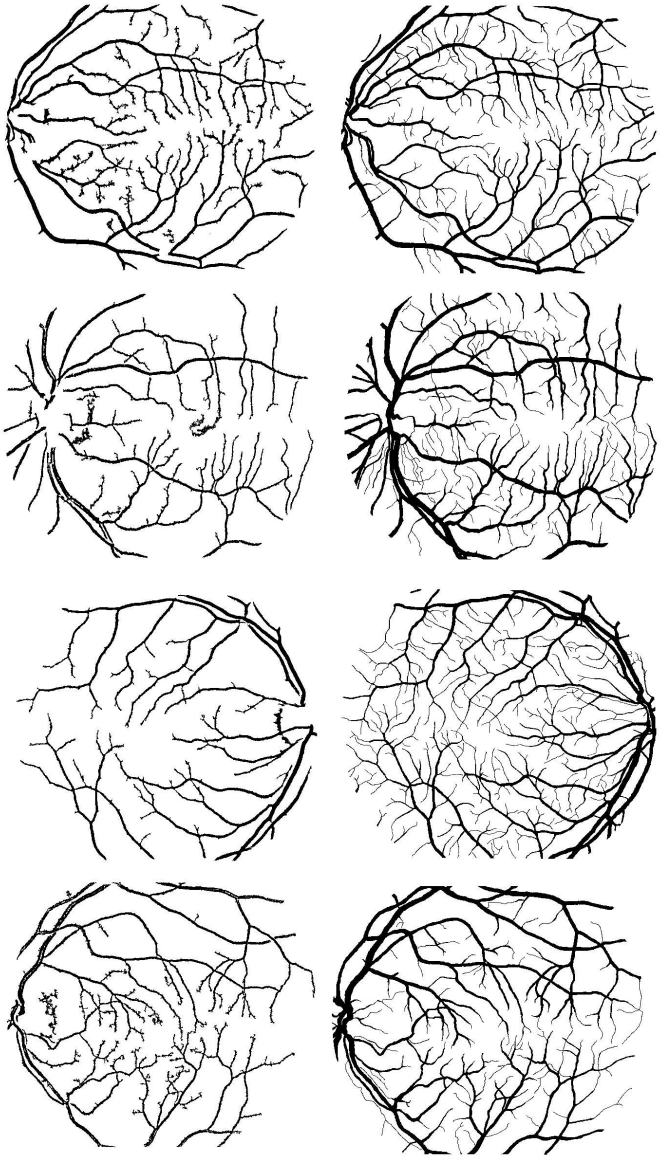

Figure 5. Left: Results from adaptive thresholding, Right: Hand-labeled vessels

Observing the results reported in Table I, it is possible to verify the influence of *mask_dim* and *constant* explained in II-B. Reducing *mask_dim* (from 21-2-100 to 11-2-100), the algorithm is less sensitive and, in particular with the ground-truth of expert two, the F1 and IoU scores get worse because of a minor amount of small vessels detection. With expert one, there is a little increment of the scores because, with smaller *mask_dim*, there is also less sensitivity to noise. About *constant* (moving from 11-2-100 to 11-1-400 one), the scores for expert two increase because a smaller *constant* brings the algorithm to be more sensitive while with expert one there is a small reduction due to a noise increment although a more aggressive noise reduction (*noise_size* from 100 to 400). The opposite effect of *constant* can be observed moving from the "21-2-100" test to the "21-4-100" one (always comparing the scores obtained with the ground-truth from expert two).

IV. Conclusions

The described method can detect about 70% of the vessels. However, based on the image analyzed, some noise is more difficult to detect and this leads to a higher false-positive rate. A more detailed set of shapes for noise detection could help to reduce this rate.

Another important aspect of this approach is that it produces contiguous segments and not isolated pixels like in other thresholding methods, as in [5] where this happens despite the global threshold is applied after having pre-processed the image. Contiguous segments are important for the correct measurement of the vessel's properties. The aim of this method is to work well on the totality of the image, despite the presence of uneven background or retinopathies manifestations.

The algorithm takes approximately from 2 to 6 minutes for a 605×700 pixels image of the ocular fundus and this, with ideal results, could permit to scan a great number of patients in order to improve the ability to detect retinopathies thanks to an objective way to measure retinal vessel properties. Moreover the time needed by the algorithm is less compared to the hand-labeling procedure that can take several hours to be performed.

I would like also to report the two results, in Fig. 6, obtained from the images of Fig. 2. It can be seen how the algorithm has been able to avoid the wrong classification on the outline of the inflammation caused by retinitis and to remove the noise caused by the hemorrhages of diabetic retinopathy.

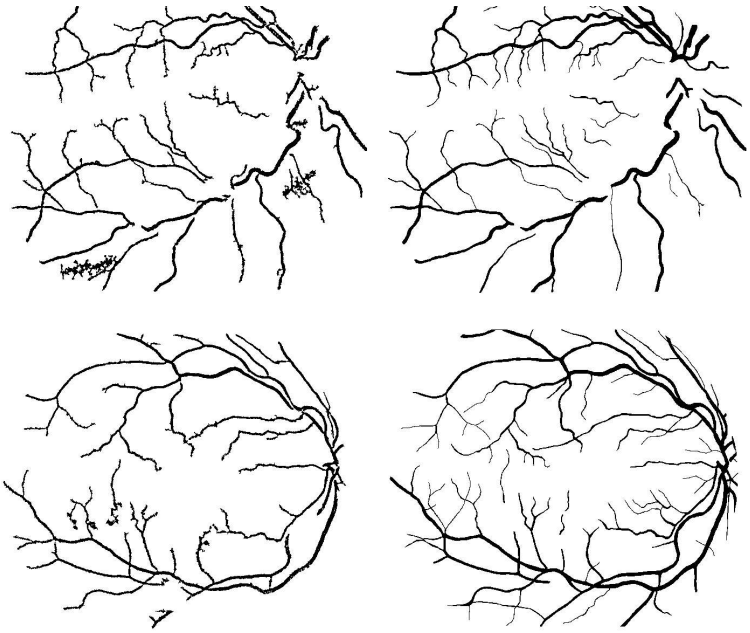

Figure 6. Above: ocular fundus affected by retinitis. Below: ocular fundus affected by diabetic retinopathy. On the right the ground-truth from expert one.

References

[1] M. Goldbaum, *The STARE Project*, 1975 (accessed October, 2020). [Online]. Available: https://cecas.clemson.edu/~ahoover/stare/

[2] A. D. Hoover, V. Kouznetsova, and M. Goldbaum, "Locating blood vessels in retinal images by piecewise threshold probing of a matched filter response," *IEEE Transactions on Medical Imaging*, vol. 19, no. 3, pp. 203–210, 2000.

[3] F. H. Chan, F. K. Lam, and H. Zhu, "Adaptive thresholding by variational method," *IEEE Transactions on Image Processing*, vol. 7, no. 3, pp. 468–473, 1998.

[4] N. Otsu, "A threshold selection method from gray-level histograms [j]," *Automatica*, vol. 11, no. 285-296, pp. 23–27, 1975.

[5] S. Chaudhuri, S. Chatterjee, N. Katz, M. Nelson, and M. Goldbaum, "Detection of blood vessels in retinal images using two-dimensional matched filters," *IEEE Transactions on Medical Imaging*, vol. 8, no. 3, pp. 263–269, 1989.