Assignment II - Global and Local Thresholding Methods

Renata Falguera Gonçalves - RA: 262889

[1]Institute of Computing – Campinas State University (Unicamp)

Abstract. *The purpose of this assignment is to perform experiments with Global and Local Thresholding Methods on Monochromatic images.*

1. Introduction

Generally, segmentation refers to the process of dividing a digital image into multiple regions (set of pixels) or objects, for the purpose of simplifying and / or changing the representation of an image for easier analysis. This division can be made using two different concepts: the similarity and the discontinuity concept.

The first one is based on methods that try to group points of the image that present similar values for a given set of characteristics. In the second one, the methods aim to partition the image based on abrupt gray levels changes, characterized by the presence of isolated points, lines or edges in the image.

One of the simplest methods in the discontinuity concept, which was treated in this assignment, is the *thresholding* technique.

Thresholding consists of classifying the pixels of an image according to the specification of one or more thresholds. Global thresholding methods determine a single threshold value for the entire image. In contrast, local adaptive thresholding methods calculate a threshold for each pixel of the image based on information contained within a pixel neighborhood.

In either case, if a p(x,y) pixel in the input image has a value higher than the threshold, then the p(x,y) pixel is classified as an object (black), otherwise it is labeled as a background (white).

In this assignment, 8 methods of thresholding were developed, and they are explained in Sections 4.4 and 4.5:

- Global Method;
- Bernsen Method;
- Niblack Method;
- Sauvola and Pietaksinen Method;
- Phansalskar, More and Sabale Method;
- Contrast Method;
- Average Method;
- Median Method;

2. Assignment Structure

All archives used for this assignment have been saved to a zip file named "***Assignment_II***", which contains:

- Main and Methods scripts (being 9 scripts in total);
- Images for test and evaluation of the algorithm;
- Output Folder;

3. The Program

Due to the large number of libraries provided, the program was developed with the newest version of Python, 3.6.8. For better image processing and easy data manipulation, it was decided to use the Numpy (version: 1.17.1), Math, ArgParse, OpenCV (version: 4.1.1) and MatplotLib libraries.

3.1. How to Execute

The user must download the ***Assigment_II.zip*** file and extract all the archives in a folder of their choice. As said in Section 2, there are nine different scripts:

- ***main.py*** (Main Program);
- ***globalmethod.py*** (Global Method Script);
- ***bernsenmethod.py*** (Bernsen Method Script);
- ***niblackmethod.py*** (Niblack Method Script);
- ***sauvola_pietaksinen_method.py*** (Sauvola and Pietaksinen Method Script);
- ***phansalskar_more_sabale_method.py*** (Phansalskar, More and Sabale Method Script);
- ***contrastmethod.py*** (Contrast Method Script);
- ***averagemethod.py*** (Average Method Script);
- ***medianmethod.py*** (Median Method Script);

The program can be run through the ***main.py*** script.

The script takes as its argument the algorithm to be executed and the name of the input image. An example execution is shown below:

```
python3 main.py -img baboon.pgm -hist baboon.png
```

3.2. Input and Output

The program inputs are monocromatic images in the **PGM** format informed in the statement of the assignmnet (*http://www.ic.unicamp.br/helio/imagens_pgm/*). The outputs are monochromatic images (**PGM** format) and its histogram (**PNG** format), after the thresholding process done at the input images, which are saved into the Output folder (as shown in Section 2).

4. The Problem Solution

As said in Section 3.1, the main script is the base of execution of the program. IIt imports the other 8 scripts (Figure 1):

- *globalmethod.py* script, with the nominator "*global_method*";
- *bernsenmethod.py* script, with the nominator "*bernsen_method*";
- *niblackmethod.py* script, with the nominator "*niblack_method*";
- *sauvola_pietaksinen_method.py* script, with the nominator "*svp_method*";
- *phansalskar_more_sabale_method.py* script, with the nominator "*pms_method*";
- *contrastmethod.py* script, with the nominator "*contrast_method*";
- *averagemethod.py* script, with the nominator "*average_method*";
- *medianmethod.py* script, with the nominator "*median_method*";

The *main* program has 3 execution steps:

1. Passing and obtaining the image path argument given at the command as shown in Section 3.1;
2. Opening and reading of the monochromatic image with the **PGM** format;
3. Accessing the others scripts to apply each method;
4. Save the output images in folders;

```
import globalmethod as global_method
import bernsenmethod as barnsen_method
import niblaclmethod as niblack_method
import sauvola_pietaksinen_method as svp_method
import phansalskar_more_sabale_method as pms_method
import more_sabale_method as ms_method
import contrastmethod as contrast_method
import averagemethod as average_method
import medianmethod as median_method
```

Figure 1. Importing all the 8 Thresholding Methods scripts.

4.1. Passing and obtaining the image path argument

In this first part, it was used all the classes and functions contained at the ArgParse library. As seen in Figure 2, the first step when using *argparse* is to create a parser object and tell it what arguments to expect (Figure 3). The parser can then be used to process the command line arguments when your program runs.

```
    parser = argparse.ArgumentParser()
```

Figure 2. Setting up a Parser.

Once all of the arguments are defined, you can parse the command line by passing a sequence of argument strings to *parse_args()* (Figure 3). By default, the arguments are taken from *sys.argv[1:]*, but you can also pass your own list. The options are processed using the GNU/POSIX syntax, so option and argument values can be mixed in the sequence.

The return value of *parse_args()* is a namespace containing the arguments to the command. The object holds the argument values as attributes, so if your argument destination is "myoption", you access the value as *args.myoption*.

In this specific problem, the argument is given by "img_address", and "hist_address", so we access it using *args.img_address* and *args.hist_address* and save at the *filename_input* and *hist_name* variable (Figure 3).

4.2. Opening and Reading

In this second part, it was used the function *cv2.imread()* from the OpenCv library. This function has two parameters: the first one is the image path; the second one is a flag that can assume these 3 values:

- -1 : For loading the image as such including alpha channel;

```
#Input
parser.add_argument('-img',
          '--img_address',
          required = True,
          help='Selected␣Image')

parser.add_argument('-hist',
          '--hist_address',
          required = True,
          help='Selected␣Image')

args = parser.parse_args()

filename_input = args.img_address
hist_name = args.hist_address
```

Figure 3. Setting up a Parser, defining the arguments and saving the passing argument.

- 0 : For loading the image in gray scale mode;
- 1 : For loading a color image. Any transparency of image will be neglected. It is the default flag.

Since it was desired to access the monochromatic image keeping the structure given by its **PGM** format, it was applied the -1 flag and saved the image file in the variable called *imageMono*, as seen in the Figure 4.

```
   imageMono = cv2.imread(filename_input,-1)
```

Figure 4. Opening and reading of the monochromatic image.

4.3. Script Accessing

Since the images will be saved in a folder, we use the *cv2.imwrite()* function. It has two parameters: the first is the path where the image will be saved; the second is the image after application of all the eight Thresholding Methods (Figure 5).

```
 cv2.imwrite("output/Global_Method_" + filename_input,
    global_method.global_thresholding(imageMono,
    filename_input))
```

Figure 5. An example of the main script accessing the *globalmethod.py* script and saving the output image.

In this third step, it was applied each Thresholding Technique to the images, by accessing each function from each method script. For each technique it was created the following functions:For each technique the following functions were created:

- Global_Thresholding(*image, hist_name*);
- Bernsen_Thresholding (*image, hist_name*);

- Niblack_Thresholding(*image, hist_name*);
- Sauvola_Pietaksinen_Thresholding (*image, hist_name*);
- Phansalskar_More_Sabale_Thresholding (*image, hist_name*);
- Contrast_Thresholding(*image, hist_name*);
- Average_Thresholding (*image, hist_name*);
- Median_Thresholding (*image, hist_name*);

As seen above, each one of the functions have the same parameters (*image, file_name*). The *image* parameter receives the image file and the *file_name* parameter receives the image file name, which is used in the image histogram nomenclature.

For calculating the image histogram after applying the thresholding, it was use the *plt.hist()* and the *cv2.calcHist()* functions provided by the OpenCv and MathPlotLib.pyplot libraries, respectively.

The function *cv2.calcHist()* calculates the histogram of one or more arrays, receiving as parameters: image, number of source images, list of dimensions channels used to compute the histogram, histogram size, histogram boundaries. The *plt.hist()* function, plots a graphic of the histogram calculated, and *plt.savefig()* saves it into a folder.

An example of the algorithm developed is shown in the Figure 6.

```
hist = cv2.calcHist([imgg],[0],None,[256],[0,256])
plt.hist(imgg.ravel(),256,[0,256])
plt.title('Contrast␣Method␣Histogram')
plt.savefig('output/Contrast/Histogram/Histogram_' +
    hist_name)
```

Figure 6. An example of the calculation and saving process of the histogram in the Contrast Method.

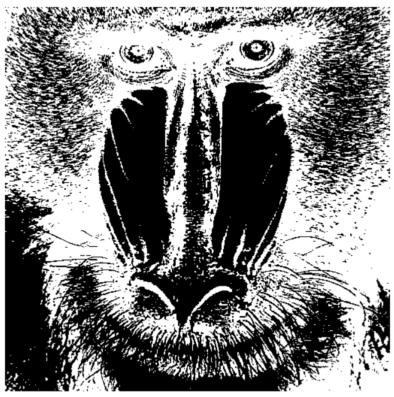
4.4. Global Thresholding Method

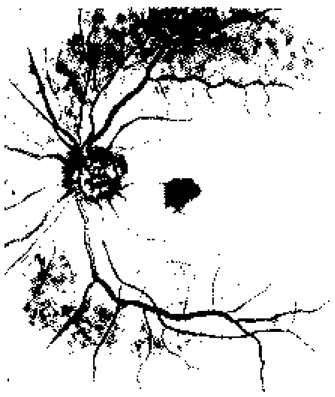
As said in the Section 1, the global thresholding determines a single threshold value for the entire image. Therefore, it was used a threshold *T* valuing **128** (middle value of the gray level scale of 0 to 255) for target the image.

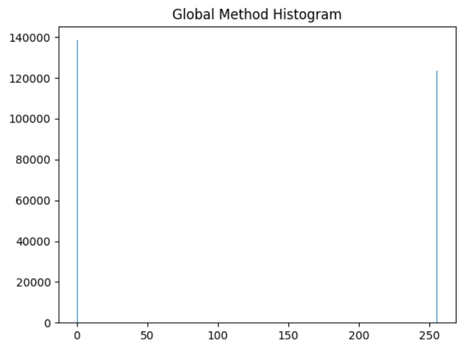
For each pixel *p(x,y)*, it is evaluated if the pixel intensity is bigger or lower than *T*. If the value is bigger, the pixel is considered as a point of an object of the image, and it receives the value **255**. Otherwise, it is considered as a point of the background, and receives the **0** value (Figure).

$$T = \begin{cases} 0 & \text{if } p(x,y) \text{ is bigger than T} \\ 255 & \text{if } p(x,y) \text{ is lower or equal than T} \end{cases}$$

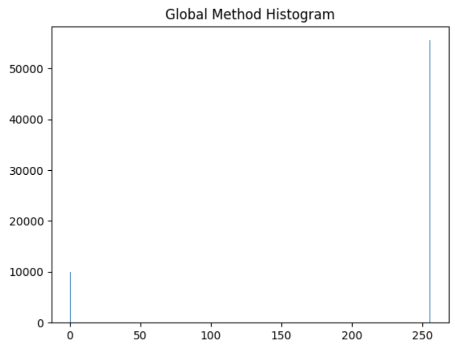
At the end of the method application, it is returned a binary thresholded image called "*img_thresholded*" and is given its histogram. Some results using the Baboon and Retina image are shown in the Figure 7 and the algorithm is shown in the Figure 8.


(a)


(b)



(c)



(d)

Figure 7. Results given by applying the Global Method: (a) Baboon image; (b) Retina image; (c) Baboon Histogram; and, (d) Retina Histogram.

```python
 def Global_Thresholding(image,file_name):
  img_thresholded = image.copy()
  T = 128

#Access each pixel (x,y) of the image an than applying
    the thresholding
  for x in range(imgg.shape[0]):
    for y in range(imgg.shape[1]):

      #Apply the Thresholding to divide the image
      if(imgg[x,y] <= T):
        imgg[x,y] = 255
      else:
        imgg[x,y] = 0

  #Image Histogram
  #Calculate image histogram
  hist = cv2.calcHist([imgg],[0],None,[256],[0,256])

  #PLot the histogram grafic
  plt.hist(imgg.ravel(),256,[0,256])
  plt.title('Global␣Method␣Histogram')
  plt.savefig('output/Global/Histogram/Histogram_' +
      hist_name)

  return img_thresholded
```

Figure 8. The Global Method algorithm.

4.5. Local Thresholding Methods

As said in the Section 1, local adaptive thresholding methods calculate a threshold for each pixel of the image based on information contained in a pixel neighborhood. In this assignment, it was decided to use a 3 x 3 pixels size neighborhood as seen in Table 1.

For saving the neighborhood values it was used a variable called "*neighborhood*", as shown in the figure 9.

p(x-1, y -1)	p(x, y -1)	p(x+1, y -1)
p(x-1, y)	p(x,y)	p(x+1, y)
p(x-1 y +1)	p(x, y +1)	p(x-+1, y+1)

Table 1. 3 X 3 pixels size neighborhood, centered in *(x,y).*

```python
neighborhood = np.array([image_copy[x,y],image_copy[x,y
    -1], image_copy[x,y+1], image_copy[x-1,y-1],
    image_copy[x+1,y+1], image_copy[x+1,y], image_copy[x
    -1,y], image_copy[x-1,y+1], image_copy[x+1,y-1]])
```

Figure 9. Saving the values of a *n x n* neighborhood of a pixel *p(x,y)*

For the methods that needed to calculate the local average and standard deviation of a specific pixel neighborhood to generate the threshold *T*, it was made the use of the *np.mean()* and *np.std()* functions, respectively, provided by the Numpy Library (Figure 10).

```
local_average = np.mean(neighborhood)
standard_deviation = np.std(neighborhood)
```

Figure 10. Calculation example of the local average and standard deviation at the Niblack Method.

4.5.1. Bernsen Thresholding Method

In the Bernsen Thresholding Method a threshold *T* is calculated for each pixel *p(x,y)* of the image as:

$$T(x,y) = (z_{max} + z_{min})/2$$

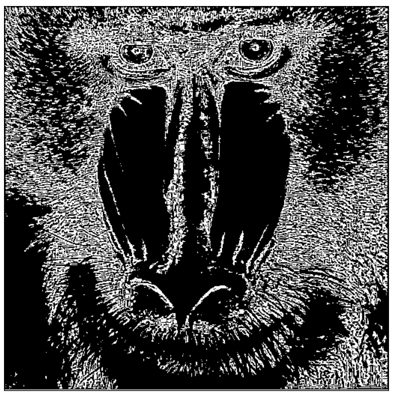
where ***zmax*** and ***zmin*** are the maximum and minimum gray level values, respectively, of an *n x n* neighborhood centered in *(x,y)*.

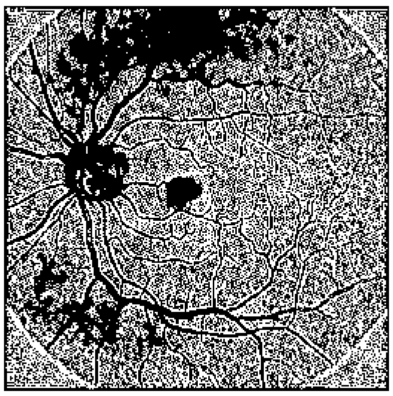
After the array was ordered in crescent order, since it is easier to obtain the maximum and minimum gray level value of the neighborhood this way, it is evaluated if the pixel intensity it is bigger or lower than *T*.

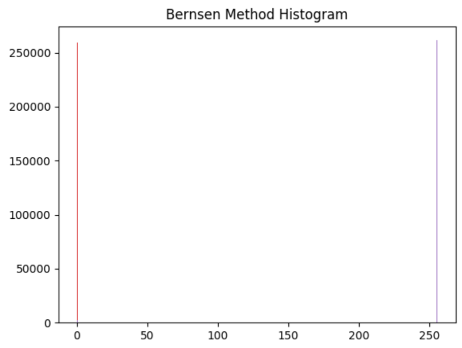
If the value is bigger than *T*, the pixel is considered a point of an object of the image, and it receives the value **255**. Otherwise, it is considered a point of the background, and receives the **0** value.

$$T(x,y) = \begin{cases} 0 & \text{if } p(x,y) \text{ is bigger than T} \\ 255 & \text{if } p(x,y) \text{ is lower or equal than T} \end{cases}$$

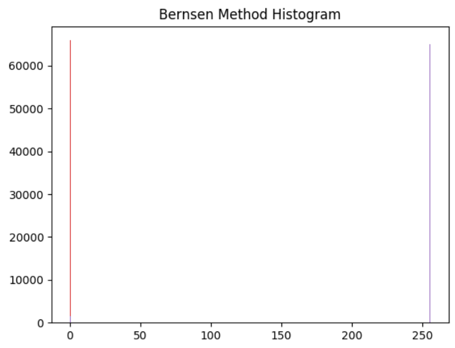
At the end of the method application, it is created a binary thresholded image and is given its histogram. Some results using the Baboon and Retina image are shown in the Figure 11 and the algorithm is shown in the Figure 12.


(a)


(b)



(c)



(d)

Figure 11. Results given by applying the Bernsen Method: (a) Baboon image; (b) Retina image; (c) Baboon Histogram; and, (d) Retina Histogram.

```python
def Bernsen_Thresholding(image,file_name):
  image_copy = image.copy()
  image_copy = np.pad(image, (1,1), 'constant')
  img_thresholded = image_copy.copy()

  for x in range(1,image_copy.shape[0]-1):
    for y in range(1,image_copy.shape[1]-1):

      neighborhood = np.array([image_copy[x,y],
          image_copy[x,y-1], image_copy[x,y+1],
          image_copy[x-1,y-1], image_copy[x+1,y+1],
          image_copy[x+1,y], image_copy[x-1,y],
          image_copy[x-1,y+1], image_copy[x+1,y-1]])

      neighborhood = np.sort(neighborhood)

      zmin = neighborhood[0]
      zmax = neighborhood[len(neighborhood) - 1]

      T = (int(zmax) + int(zmin))/2

      if(image_copy[x,y] <= T):
        img_thresholded[x,y] = 255
      else:
        img_thresholded[x,y] = 0

  #Image Histogram
  #Calculate image histogram
  hist = cv2.calcHist([img_thresholded],[0],None
      ,[256],[0,256])

  #PLot the histogram grafic
  plt.hist(img_thresholded.ravel(),256,[0,256])
  plt.title('Bernsen␣Method␣Histogram')
  plt.savefig('output/Bernsen/Histogram/Histogram_' +
      hist_name)

  return img_thresholded
```

Figure 12. The Bernsen Method algorithm.

4.5.2. Niblack Thresholding Method

In the Niblack Thresholding Method a threshold *T* is calculated for each pixel *p(x,y)* of the image as:

$$T(x, y) = \mu(x, y) + k\sigma(x, y)$$

where **u(x,y)** and **o(x,y)** are the local average and the standard deviation, respectively, in a *n x n* neighborhood centered in *(x,y)*. The **k** value is 0.8, and it is used to adjust the edge fraction of the object to be considered as part of the object.

After the local average and standard deviation were calculated, the threshold *T* was computed and it is evaluated if the pixel intensity is bigger or lower than *T*.

If the value is bigger than *T*, the pixel is considered a point of an object of the image, and it receives the value **255**. Otherwise, it is considered a point of the background, and receives the **0** value.

$$T(x,y) = \begin{cases} 0 & \text{if } p(x,y) \text{ is bigger than T} \\ 255 & \text{if } p(x,y) \text{ is lower or equal than T} \end{cases}$$

Some results using the Baboon and Retina image are shown in the Figure 13 and the algorithm is shown in the Figure 12.

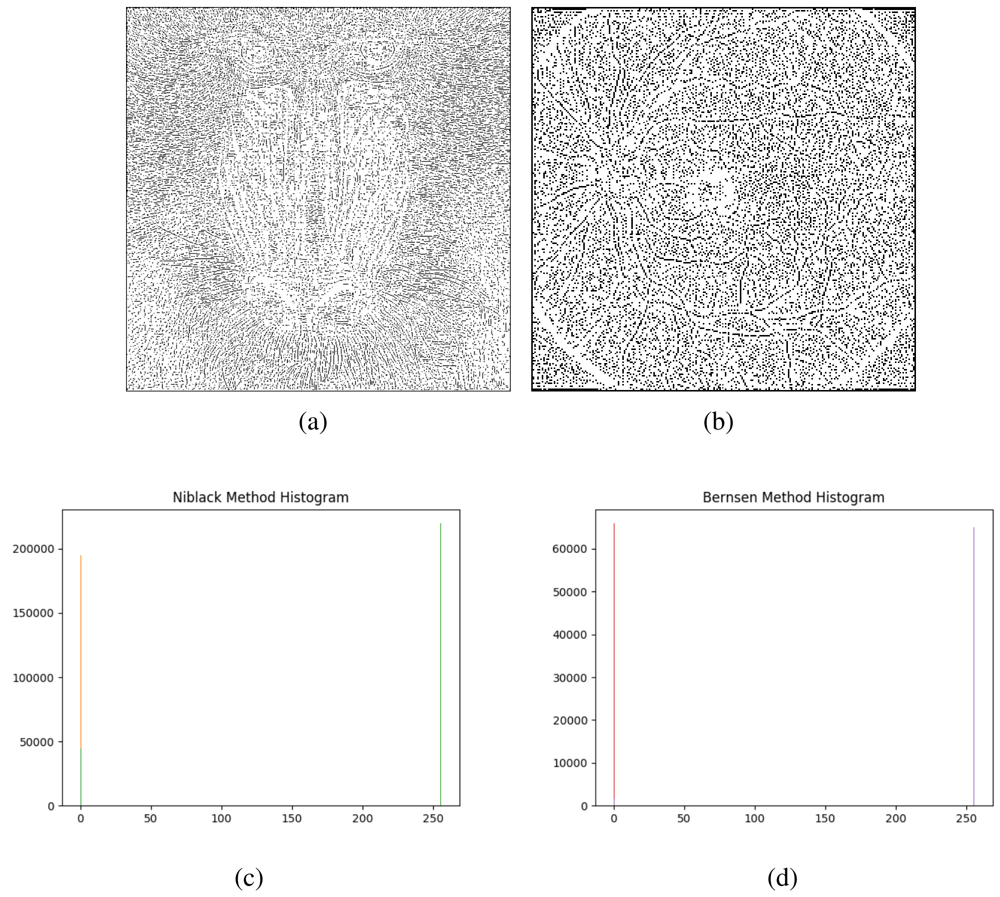


Figure 13. Results given by applying the Niblack Method: (a) Baboon image; (b) Retina image; (c) Baboon Histogram; and, (d) Retina Histogram.

```python
def Niblack_Thresholding(image,file_name):

  image_copy = image.copy()
  image_copy = np.pad(image, (1,1), 'constant')
  img_thresholded = image_copy.copy()
  k = 0.8

  for x in range(1,image_copy.shape[0]-1):
    for y in range(1,image_copy.shape[1]-1):

      neighborhood = np.array([image_copy[x,y],
          image_copy[x,y-1], image_copy[x,y+1],
          image_copy[x-1,y-1], image_copy[x+1,y+1],
          image_copy[x+1,y], image_copy[x-1,y],
          image_copy[x-1,y+1], image_copy[x+1,y-1]])

      local_average = np.mean(neighborhood)
      standard_deviation = np.std(neighborhood)

      T = local_average + (k *standard_deviation)

      if(image_copy[x,y] <= T):
        img_thresholded[x,y] = 255
      else:
        img_thresholded[x,y] = 0

  #Image Histogram
  #Calculate image histogram
  hist = cv2.calcHist([img_thresholded],[0],None
      ,[256],[0,256])

  #PLot the histogram grafic
  plt.hist(img_thresholded.ravel(),256,[0,256])
  plt.title('Niblack␣Method␣Histogram')
  plt.savefig('output/Niblack/Histogram/Histogram_'+
      hist_name)

  return img_thresholded
```

Figure 14. The Niblack Method algorithm.

4.5.3. Sauvola and Pietaksinen Thresholding Method

In the Sauvola and Pietaksinen Thresholding Method a threshold *T* is calculated for each pixel *p(x,y)* of the image as:

$$T(x,y) = \mu(x,y) + [1 + k((\sigma(x,y)/R) - 1]$$

where **u(x,y)** and **o(x,y)** are the local average and the standard deviation, respectively, in a *n x n* neighborhood centered in *(x,y)*. The **k** and **R** values are 0.5 and 128, respectively.

After the local average and standard deviation were calculated, the threshold T was computed and it is evaluated if the pixel intensity is bigger or lower than T.

If the value is bigger than T, the pixel is considered a point of an object of the image, and it receives the value **255**. Otherwise, it is considered a point of the background, and receives the **0** value.

$$T(x,y) = \begin{cases} 0 & \text{if } p(x,y) \text{ is bigger than T} \\ 255 & \text{if } p(x,y) \text{ is lower or equal than T} \end{cases}$$

Some results using the Baboon and Retina image are shown in the Figure 15 and the algorithm is shown in the Figure 16.

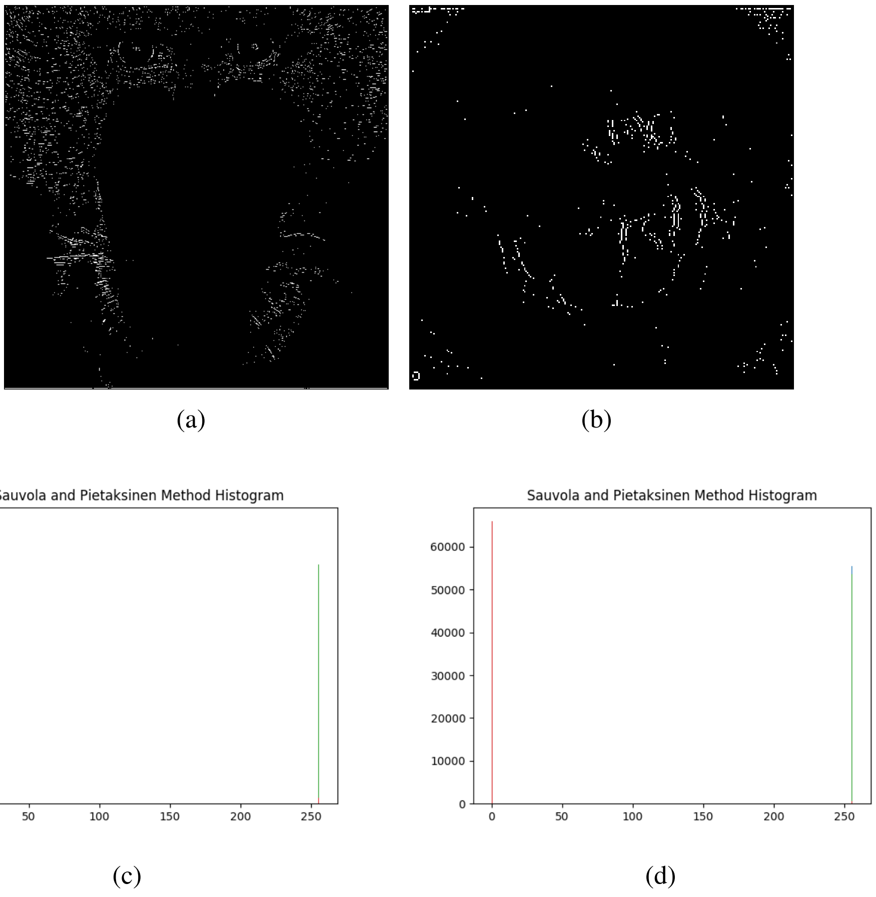


Figure 15. Results given by applying the Sauvola and Pietaksinen Method: (a) Baboon image; (b) Retina image; (c) Baboon Histogram; and, (d) Retina Histogram.

```python
def Sauvola_Pietaksinen_Thresholding(image,file_name):
  image_copy = image.copy()
  image_copy = np.pad(image, (1,1), 'constant')
  img_thresholded = image_copy.copy()

  k = 0.5
  R = 128

  for x in range(1,image_copy.shape[0]-1):
    for y in range(1,image_copy.shape[1]-1):

      neighborhood = np.array([image_copy[x,y],
          image_copy[x,y-1], image_copy[x,y+1],
          image_copy[x-1,y-1], image_copy[x+1,y+1],
          image_copy[x+1,y], image_copy[x-1,y],
          image_copy[x-1,y+1], image_copy[x+1,y-1]])

      local_average = np.mean(neighborhood)
      standard_deviation = np.std(neighborhood)

      T = local_average * ( 1 + (k* ((standard_deviation
          /R) - 1)))

      if(image_copy[x,y] <= T):
        img_thresholded[x,y] = 255
      else:
        img_thresholded[x,y] = 0

  #Image Histogram
  #Calculate image histogram
  hist = cv2.calcHist([img_thresholded],[0],None
      ,[256],[0,256])

  #PLot the histogram grafic
  plt.hist(img_thresholded.ravel(),256,[0,256])
  plt.title('Sauvola␣and␣Pietaksinen␣Method␣Histogram')
  plt.savefig('output/SauvolaPietaksinen/Histogram/
      Histogram_' + hist_name)

  return img_thresholded
```

Figure 16. The Sauvola and Pietaksinen Method algorithm.

4.5.4. Phansalskar, More and Sabale Thresholding Method

In the Phansalskar, More and Sabale Thresholding Method a threshold *T* is calculated for each pixel *p(x,y)* of the image as:

$$T(x,y) = \mu(x,y) + [1 + (p\exp(-q * \mu(x,y)) + k((\sigma(x,y)/R) - 1]$$

where **u(x,y)** and **o(x,y)** are the local average and the standard deviation, respectively, in a *n x n* neighborhood centered in *(x,y)*. The **k**, **R**, **p** and **q** values are 0.25, 0.5, 2 and 10, respectively.

After the local average and standard deviation were calculated, the threshold *T* was computed and it is evaluated if the pixel intensity is bigger or lower than *T*.

If the value is bigger than *T*, the pixel is considered a point of an object of the image, and it receives the value **255**. Otherwise, it is considered a point of the background, and receives the **0** value.

$$T(x,y) = \begin{cases} 0 & \text{if } p(x,y) \text{ is bigger than T} \\ 255 & \text{if } p(x,y) \text{ is lower or equal than T} \end{cases}$$

Some results using the Baboon and Retina image are shown in the Figure 17 and the algorithm is shown in the Figure 18.

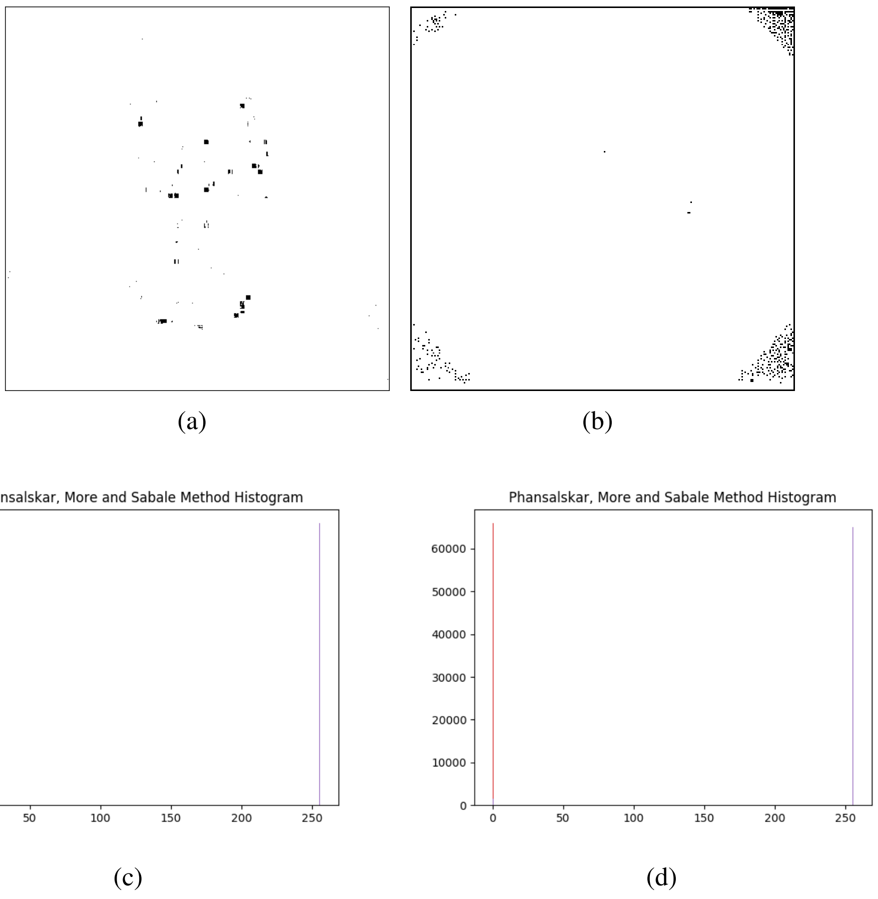


Figure 17. Results given by applying the Phansalskar, More and Sabale Method: (a) Baboon image; (b) Retina image; (c) Baboon Histogram; and, (d) Retina Histogram.

```
def Phansalskar_More_Sabale_Thresholding(image):
  image_copy = image.copy()
  image_copy = np.pad(image, (1,1), 'constant')
  img_thresholded = image_copy.copy()

  k = 0.25
  R = 0.5
  p = 2
  q = 10

  for x in range(1,image_copy.shape[0]-1):
    for y in range(1,image_copy.shape[1]-1):

      neighborhood = np.array([image_copy[x,y],
          image_copy[x,y-1], image_copy[x,y+1],
          image_copy[x-1,y-1], image_copy[x+1,y+1],
          image_copy[x+1,y], image_copy[x-1,y],
          image_copy[x-1,y+1], image_copy[x+1,y-1]])

      local_average = np.mean(neighborhood)
      standard_deviation = np.std(neighborhood)

      exponecial_value = mp.exp((-q * local_average))

      T = local_average * ( 1 + (p * exponecial_value) +
           (k* ((standard_deviation/R) - 1)))

      if(image_copy[x,y] <= T):
          img_thresholded[x,y] = 255
      else:
          img_thresholded[x,y] = 0

    #Image Histogram
  #Calculate image histogram
  hist = cv2.calcHist([img_thresholded],[0],None
      ,[256],[0,256])

  #PLot the histogram grafic
  plt.hist(img_thresholded.ravel(),256,[0,256])
  plt.title('Phansalskar,␣More␣and␣Sabale␣Method␣
      Histogram')
  plt.savefig('output/PhansalskarMoreSabale/Histogram/
      Histogram_' + hist_name)

  return img_thresholded
```

Figure 18. The Phansalskar, More and Sabale Method algorithm.

4.5.5. Median Thresholding Method

In the Median Thresholding Method a threshold *T* is calculated for each pixel *p(x,y)* of the image where **T (x, y)** is the median value of the local intensity distribution in a *n x n* neighborhood centered in *(x,y)*.

After the array was ordered in crescent order, to easily obtain the median value, where its position is obtained by:

$$Median\_position) = (n * n - 1)/2$$

The threshold *T* is computed and it is evaluated if the pixel intensity it is bigger or lower than *T*. If the value is bigger than *T*, the pixel is considered a point of an object of the image, and it receives the value **255**. Otherwise, it is considered a point of the background, and receives the **0** value.

$$T(x,y) = \begin{cases} 0 & \text{if } p(x,y) \text{ is bigger than T} \\ 255 & \text{if } p(x,y) \text{ is lower or equal than T} \end{cases}$$

Some results using the Baboon and Retina image are shown in the Figure 19 and the algorithm is shown in the Figure 20.

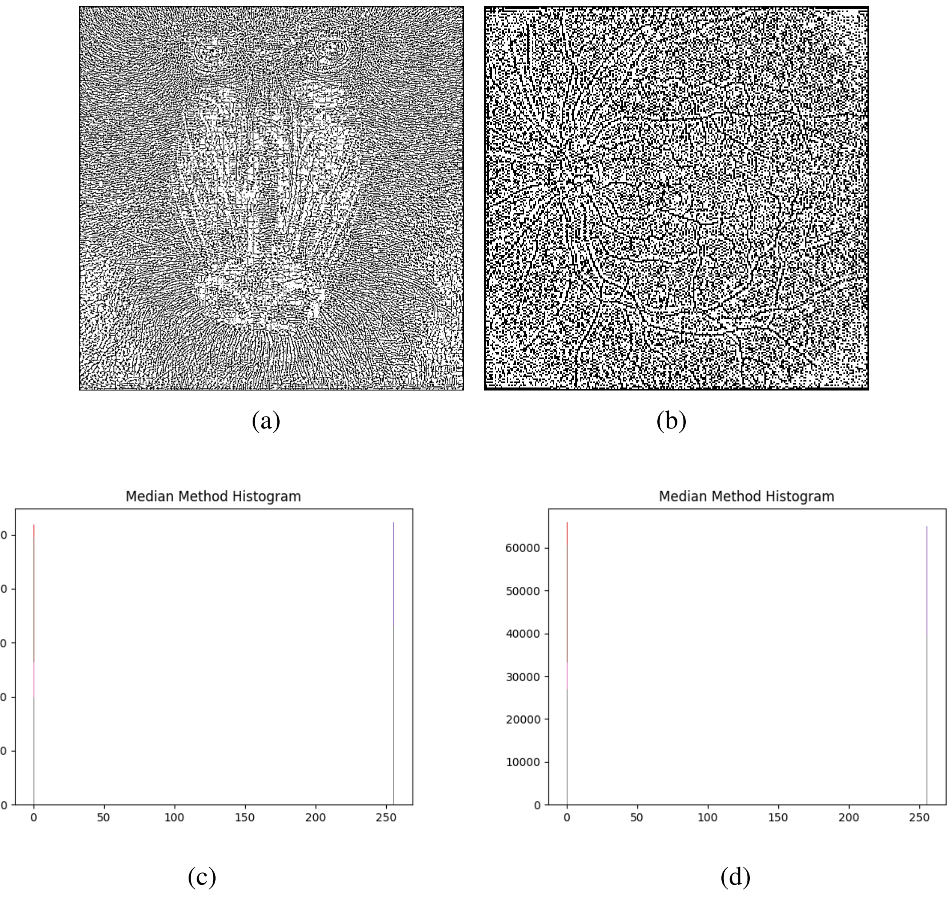


Figure 19. Results given by applying the Median Method: (a) Baboon image; (b) Retina image; (c) Baboon Histogram; and, (d) Retina Histogram.

```
def Median_Thresholding(image,file_name):
  image_copy = image.copy()
  image_copy = np.pad(image, (1,1), 'constant')
  img_thresholded = image_copy.copy()

  for x in range(1,image_copy.shape[0]-1):
    for y in range(1,image_copy.shape[1]-1):

      neighborhood = np.array([image_copy[x,y],
          image_copy[x,y-1], image_copy[x,y+1],
          image_copy[x-1,y-1], image_copy[x+1,y+1],
          image_copy[x+1,y], image_copy[x-1,y],
          image_copy[x-1,y+1], image_copy[x+1,y-1]])

      neighborhood = np.sort(neighborhood)
      median_value = neighborhood[mp.floor((len(
          neighborhood) - 1)/2)]

      T = median_value

      if(image_copy[x,y] <= T):
        img_thresholded[x,y] = 255
      else:
        img_thresholded[x,y] = 0

  #Image Histogram
  #Calculate image histogram
  hist = cv2.calcHist([img_thresholded],[0],None
      ,[256],[0,256])

  #PLot the histogram grafic
  plt.hist(img_thresholded.ravel(),256,[0,256])
  plt.title('Median␣Method␣Histogram')
  plt.savefig('output/Median/Histogram/Histogram_' +
      hist_name)

  return img_thresholded
```

Figure 20. The Median Method algorithm.

4.5.6. Average Thresholding Method

In the Average Thresholding Method a threshold *T* is calculated for each pixel *p(x,y)* of the image as:

$$T(x, y) = \mu(x, y)$$

where *u(x,y)* is the local average value in a *n x n* neighborhood centered in *(x,y)*.

After the local average and standard deviation were calculated, the threshold *T* was computed and it is evaluated if the pixel intensity is bigger or lower than *T*.

If the value is bigger, the pixel is considered a point of an object of the image, and it receives the value **255**. Otherwise, it is considered a point of the background, and receives the **0** value.

$$T(x,y) = \begin{cases} 0 & \text{if } p(x,y) \text{ is bigger than T} \\ 255 & \text{if } p(x,y) \text{ is lower or equal than T} \end{cases}$$

Some results using the Baboon and Retina image are shown in the Figure 21 and the algorithm is shown in the Figure 22.

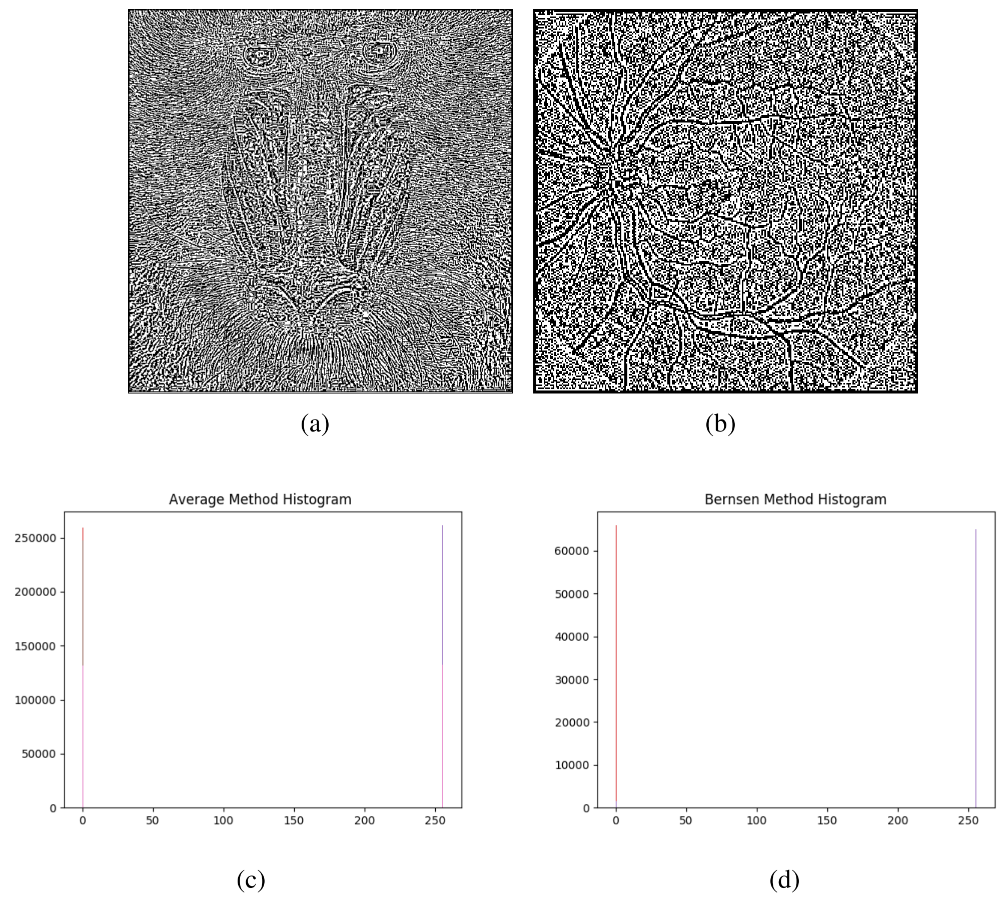


Figure 21. Results given by applying the Average Method: (a) Baboon image; (b) Retina image; (c) Baboon Histogram; and, (d) Retina Histogram.

```
def Median_Thresholding(image,file_name):
  image_copy = image.copy()
  image_copy = np.pad(image, (1,1), 'constant')
  img_thresholded = image_copy.copy()

  for x in range(1,image_copy.shape[0]-1):
    for y in range(1,image_copy.shape[1]-1):

      neighborhood = np.array([image_copy[x,y],
          image_copy[x,y-1], image_copy[x,y+1],
          image_copy[x-1,y-1], image_copy[x+1,y+1],
          image_copy[x+1,y], image_copy[x-1,y],
          image_copy[x-1,y+1], image_copy[x+1,y-1]])

      local_average = np.mean(neighborhood)

      T = local_average

      if(image_copy[x,y] <= T):
        img_thresholded[x,y] = 255
      else:
        img_thresholded[x,y] = 0


  #Image Histogram
  #Calculate image histogram
  hist = cv2.calcHist([img_thresholded],[0],None
      ,[256],[0,256])

  #PLot the histogram grafic
  plt.hist(img_thresholded.ravel(),256,[0,256])
  plt.title('Average␣Method␣Histogram␣')
  plt.savefig('output/Average/Histogram/Histogram_' +
      hist_name)

  return img_thresholded
```

Figure 22. The Average Method algorithm.

4.5.7. Contrast Thresholding Method

In the Contrast Thresholding Method was calculated the local maximum and minimum gray level value for each pixel *p(x,y)* of the image in a *n x n* neighborhood.

The local maximum and minimum were computed and a evaluation was made with the pixel and the local maximum and minimum intensities. Therefore, it was calculated the difference between the pixel *p(x,y)* with the maximum and minimum value, and then made a comparison with both results (Figure 23).

If the difference between the pixel *p(x,y)* and the maximum value is bigger than the difference between the pixel *p(x,y)* and the minimum value, it means that *p(x,y)* is closer with the maximum value, and it is considered a point of an object of the image, receiving the value **255**. Otherwise, if it is closer with the minimum value, it is considered a point

```
if (abs((int(image_copy[x,y]) - min_value)) <= abs((int(
   image_copy[x,y]) - max_value))):
      img_thresholded[x,y] = 0
elif(abs((int(image_copy[x,y]) - min_value)) > abs((int(
   image_copy[x,y]) - max_value))):
        img_thresholded[x,y] = 255
```

Figure 23. The Median Method algorithm.

of the background, receiving the **0** value.

Some results using the Baboon and Retina image are shown in the Figure 24 and the algorithm is shown in the Figure 25.

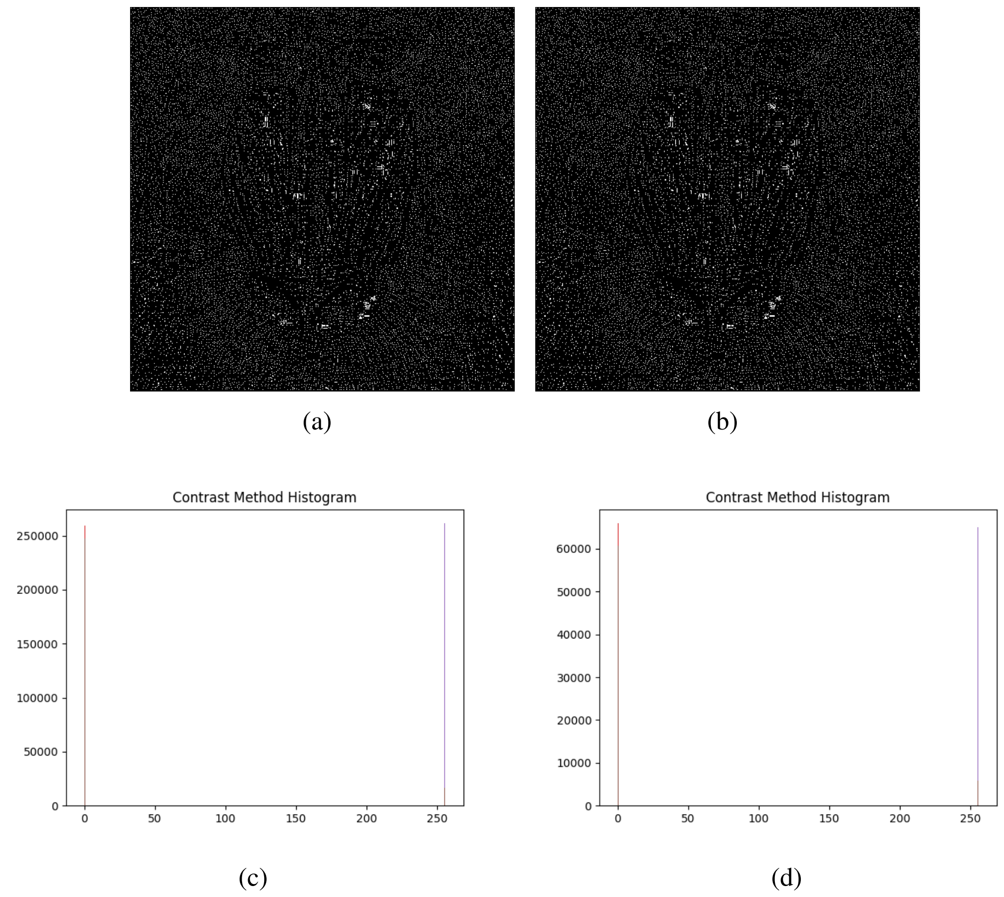


Figure 24. Results given by applying the Contrast Method: (a) Baboon image; (b) Retina image; (c) Baboon Histogram; and, (d) Retina Histogram.

```
def Contrast_Thresholding(image,file_name):
  image_copy = image.copy()
  image_copy = np.pad(image, (1,1), 'constant')
  img_thresholded = image_copy.copy()

  for x in range(1,image_copy.shape[0]-1):
    for y in range(1,image_copy.shape[1]-1):

      neighborhood = np.array([image_copy[x,y],
          image_copy[x,y-1], image_copy[x,y+1],
          image_copy[x-1,y-1], image_copy[x+1,y+1],
          image_copy[x+1,y], image_copy[x-1,y],
          image_copy[x-1,y+1], image_copy[x+1,y-1]])

      neighborhood = np.sort(neighborhood)

      min_value = neighborhood[0]
      max_value = neighborhood[len(neighborhood) - 1]

      if(abs((int(image_copy[x,y]) - min_value)) <= abs
          ((int(image_copy[x,y]) - max_value))):
        img_thresholded[x,y] = 0
      elif(abs((int(image_copy[x,y]) - min_value)) > abs
          ((int(image_copy[x,y]) - max_value))):
        img_thresholded[x,y] = 255

  #Image Histogram
  #Calculate image histogram
  hist = cv2.calcHist([img_thresholded],[0],None
      ,[256],[0,256])

  #PLot the histogram grafic
  plt.hist(img_thresholded.ravel(),256,[0,256])
  plt.title('Contrast␣Method␣Histogram')
  plt.savefig('output/Contrast/Histogram/Histogram_' +
      hist_name)

  return img_thresholded
```

Figure 25. The Average Method algorithm.

5. Conclusion

After performing all the tests using a small amount of Monochromatic images, it was analysed: how different were the results images generated from each technique; and the difference between the histogram result obtained and the black pixel fraction of each method.

After applying the different techniques to a significant set of images, it was verified that the Niblack and the Median methods had the best results above all. The first one calculated the new pixel value based on the local average and standard deviation of a *n x n* pixels neighborhood, handling better with the features of each region, noises, and maintaining a reasonable level of details. The second one calculated the new pixel value based on the median value of a *n x n* pixels neighborhood, dealing better with the image noises. Noises normally stay in the extremes, are points of a very high or low gray level value.

Analysing the outputs images histograms and black pixels fraction, it could be seen that most of the methods keep a close proportion between the object and background pixels, that is, a similar black pixels fraction.

The affirmation above can be proved by observing the black pixels fraction, for example, of the Wedge and the Baboon shown below. It can be seen that the Global, Bernsen and Niblack have the same proportion *A*, and the others have the same *B* proportion, as seen:

- Global Method: Wedge with 41,28%, and Baboon with 53,40%
- Bernsen Method: Wedge with 41,28% and Baboon with 53,40%
- Niblack Method: Wedge41,28% and Baboon with 53,40%
- Sauvola and Pietaksinen Method: 92,03% and Baboon with 99,18%
- Phansalskar, More and Sabale Method: 92,03% and Baboon with 99,18%
- Contrast Method: 92,03% and Baboon with 99,18%
- Average Method: 92,03% and Baboon with 99,18%
- Median Method: 92,03% and Baboon with 99,18%