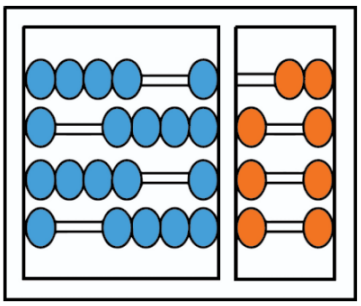


Instituto de Computação

UNIVERSIDADE ESTADUAL DE CAMPINAS

Introdução ao Processamento Digital de Imagem (MO443)

Professor: Hélio Pedrini

Relatório - Trabalho 2

Marcella de Sant' Ana RA: 223588

29/09/2022

1. Introdução e Objetivos

A área de processamento de imagens envolve diversas técnicas com a finalidade de prover mais informações para o usuário visualizador, com muitas aplicações na área de Medicina, Biologia, Cartografia, entre outras.

Tendo em vista isso, o objetivo deste relatório é exibir discussões referentes à aplicação de algumas técnicas de limiarização, as quais transformam a imagem em sua forma binarizada, com o intuito de se aproximar da imagem original, do que a imagem original traz de informação, onde cada técnica possui seus parâmetros específicos que causam impacto no retorno da informação.

2. Materiais e Métodos

Os materiais utilizados produzir as imagens com determinados processamentos são elencados a seguir:

- Python 3.9.0 - Software de programação
  - Bibliotecas:
    - opencv-python (cv2)
    - numpy
    - matplotlib.pyplot
- Visual Studio Code - Editor de código
- Imagens utilizadas: **baboon.pgm**, **sonnet.pgm** e **wedge.pgm**

Além disso, é apresentado abaixo, na Figura 1, como foram organizados os resultados relacionados a cada método, bem como se estruturou a execução para obtenção dos mesmos:

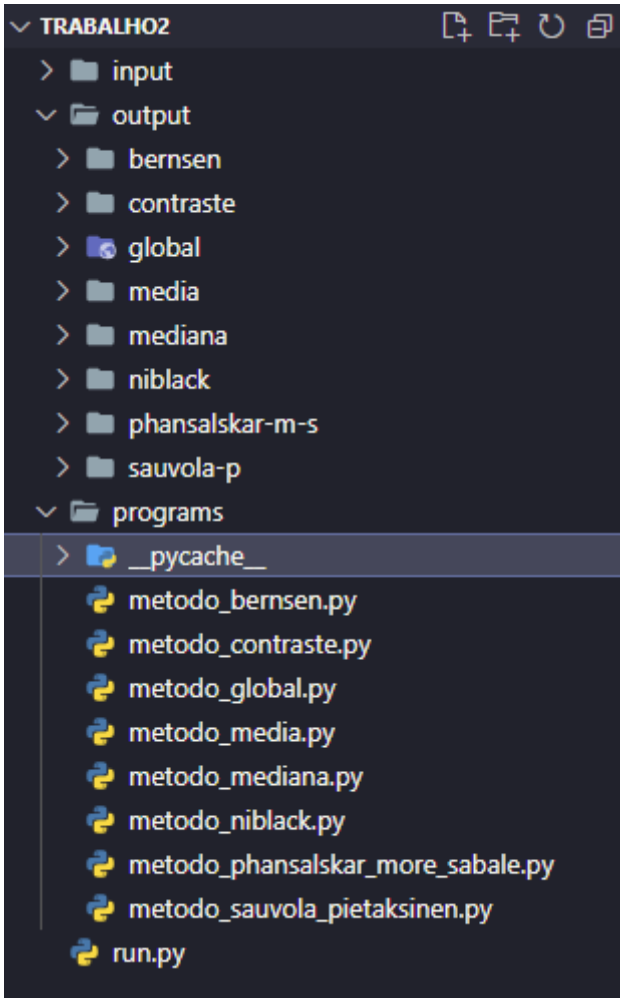


Figura 1 - Estrutura dos programas desenvolvidos

Seguindo o esquema de pasta da Figura 1, tem-se um conjunto de imagens a serem lidas, na pasta *input* (referente aos respectivos métodos). E, a obtenção das respostas referentes a cada método se dá pela execução do *script* "**run.py**", ou seja, "**python run.py**". Ao executá-lo, espera-se que sejam geradas imagens em formato "png" na pasta *output*, a qual está subdividida por método; as imagens serão armazenadas referente ao seu método correspondente. Por fim, a pasta *programs* contém a lógica desenvolvida para cada método proposto; o arquivo README.md contém as instruções para execução do *script* "**run.py**", bem como o requirements.txt, os pacotes necessários.

3. Resultados e Discussões

Primeiramente, serão apresentados os histogramas quantizando os níveis de cinza para cada imagem de entrada:

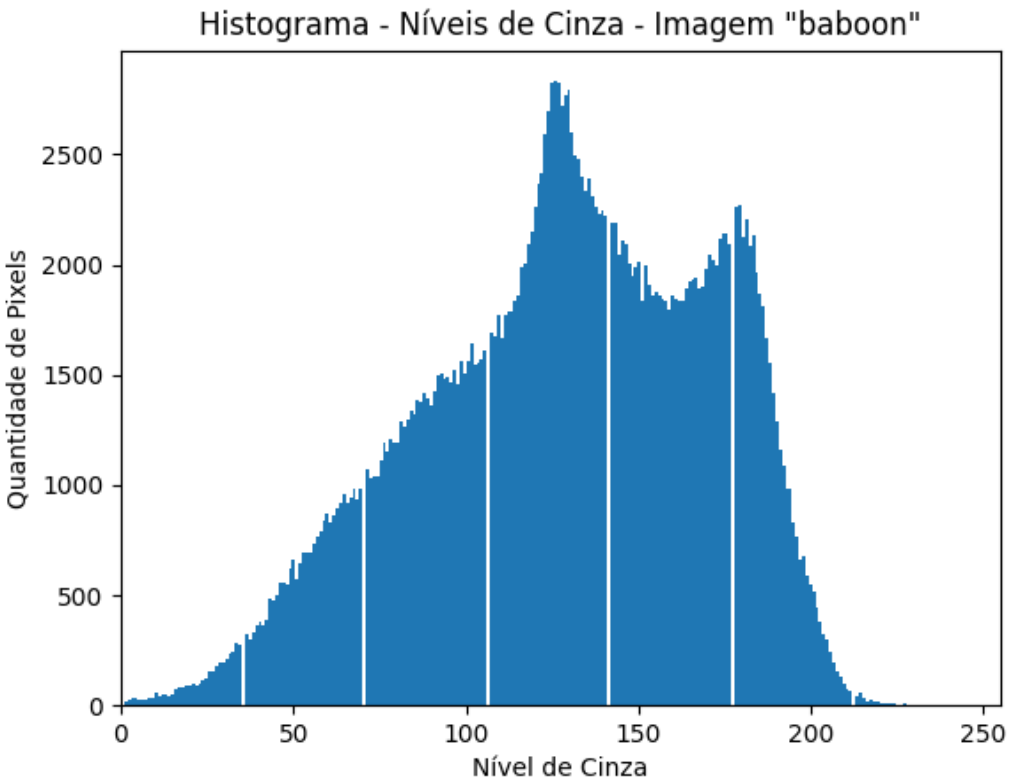


Figura 2 - Histograma de níveis de cinza para a imagem *baboon*

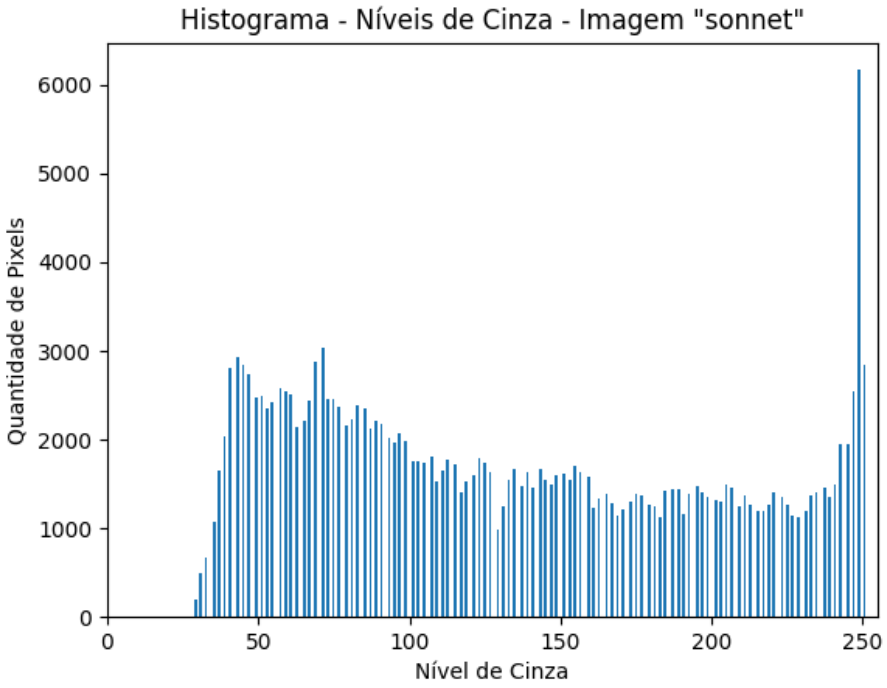


Figura 3 - Histograma de níveis de cinza para a imagem *baboon*

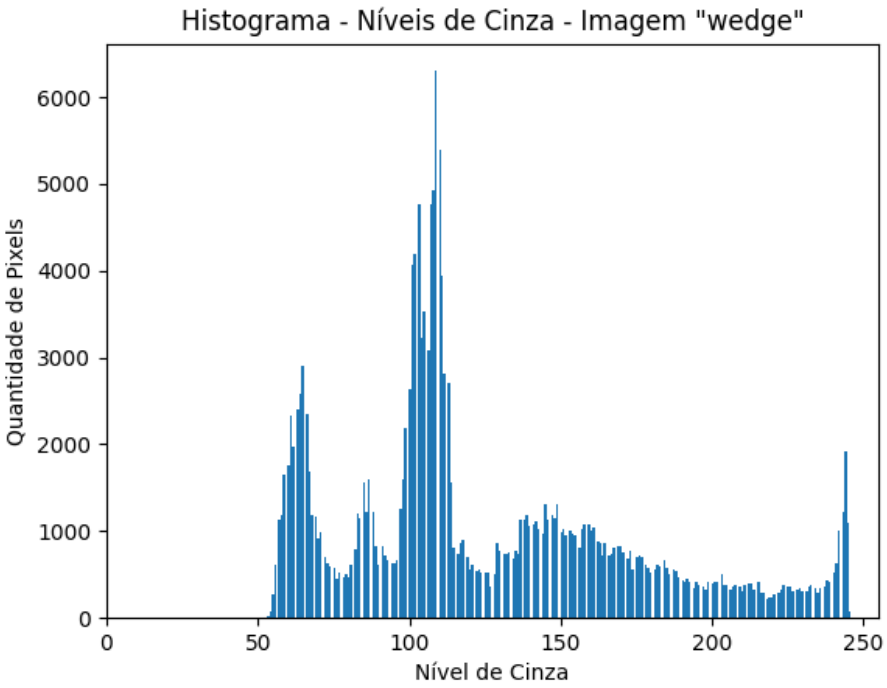


Figura 4 - Histograma de níveis de cinza para a imagem *baboon*

A seguir, serão demonstrados como cada método foi implementado acompanhado de seus respectivos resultados.

3.1 Método Global

Este método tinha como objetivo central considerar um limiar global e aplicar ele em cada pixel: se o valor de intensidade do um pixel fosse maior do que um limiar **T**, o pixel seria considerado como pertencente ao objeto (ou seja, o valor do pixel seria substituído por zero); caso contrário, seria considerado como fundo (ou seja, o valor do pixel seria substituído por 255). A seguir, é apresentado na Figura 5 como foi implementado cada um dos processamentos anteriores:

```
img = cv.imread(f'input/{image_name}.pgm', cv.IMREAD_GRAYSCALE)
cv.imwrite(f'output/{image_name}_original.png', np.array(img))
a = np.array(img)
a = a.astype(int)
rows = len(a)
collumns = len(a[0])

limiar = T
expression = img > limiar if invert_background else img < limiar

img[expression] = 0
img[img > 0] = 255
```

Figura 5 - Implementação do Método Global

Primeiramente, é informado um valor de limiar **T** desejado (por exemplo, 100) e foi criado uma variável chamada *invert_background* pois em algumas imagens que foram testadas, fazia mais sentido o negativo do resultado obtido. Considerando que a princípio, não é preciso inverter a cor do fundo, caso o valor de um pixel (x, y) fosse menor que o limiar **T**, seria atribuído zero ao pixel e em seguida, os pixels restantes (maiores que zero) receberiam o valor de 255. Dadas as imagens citadas em “Materiais e Métodos”, optou-se por adotar que o que seria objeto o que fosse menor que o limiar **T**, para manter *invert_background=False* como padrão. Mas a título de teste, usou-se *invert_background=True* para a imagem **peppers.pgm** apenas no método global, com a finalidade de confirmar a funcionalidade dessa variável.

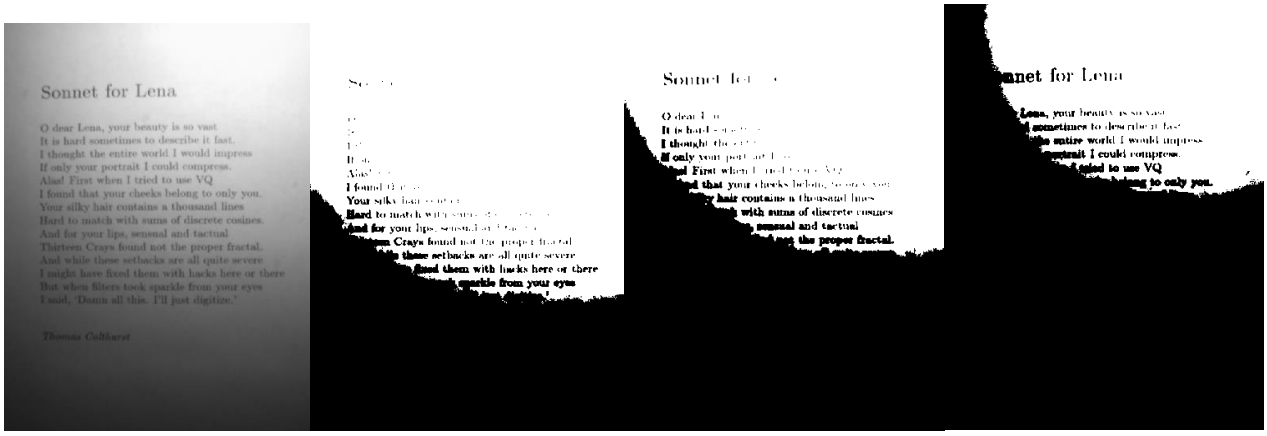
(a) original (b) $T = 100$ (c) $T = 128$ (d) $T = 175$

Figura 6 - Resultados obtidos para imagem *baboon* usando o Método Global.

Figura	Fração dos pixels pretos
b ($T = 100$)	0,24
c ($T = 128$)	0,46
d ($T = 175$)	0,83

Tabela 1 - Fração dos pixels pretos dos resultados para imagem *baboon*.

Em relação aos resultados da Figura 6 para a imagem *baboon*, notou-se que tal método se portou adequado uma vez que no limiar $T = 128$, a imagem resultante trouxe a informação de que havia babuíno reapresentado no arquivo de entrada. Porém, tendendo ao 255, há um grande escurecimento da mesma.



(a) original (b) $T = 100$ (c) $T = 128$ (d) $T = 175$

Figura 7 - Resultados obtidos para imagem *sonnet* usando o Método Global.

Figura	Fração dos pixels pretos
b ($T = 100$)	0,4
c ($T = 128$)	0,52
d ($T = 175$)	0,69

Tabela 2 - Fração dos pixels pretos dos resultados para imagem *sonnet*.

Em contrapartida, analisando os resultados da Figura 7 para a imagem *sonnet*, tal método não produziu bons resultados, pois o efeito de luminosidade teve grande impacto no ajuste do limiar T, que mesmo com valores mais baixos, não capturava o texto, e para valores altos, captou-se ruído.

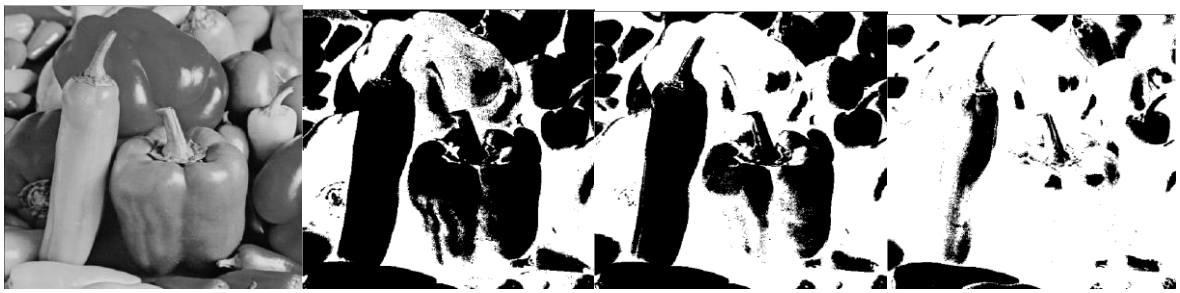
(a) original (b) $T = 100$ (c) $T = 128$ (d) $T = 175$

Figura 8 - Resultados obtidos para imagem *wedge* usando o Método Global.

Novamente, para a imagem *wedge*, percebeu-se que tal método não produziu bons resultados, dado que também o efeito de luminosidade teve grande impacto no ajuste do limiar T, mas teve resultados mais satisfatórios que os resultados da imagem *sonnet*, visto que para $T = 100$, boa parte da informação foi transmitida (sólido de formato “T”).

Figura	Fração dos pixels pretos
b ($T = 100$)	0,26
c ($T = 128$)	0,60
d ($T = 175$)	0,82

Tabela 3 - Fração dos pixels pretos dos resultados para imagem *wedge*.



(a) original (b) $T = 100$ (c) $T = 128$ (d) $T = 175$

Figura 9 - Resultados obtidos para imagem *peppers* usando o Método Global.

Figura	Fração dos pixels pretos
b ($T = 100$)	0,59
c ($T = 128$)	0,47
d ($T = 175$)	0,18

Tabela 4 - Fração dos pixels pretos dos resultados para imagem *peppers*.

Por fim, para a imagem *peppers* - colocada somente nos resultados do Método Global a finalidade de testar a flag de inverter o *background* - tal método produziu resultados semelhantes ao da imagem *wedge*: não tão satisfatórios, porém com $T = 100$, parte considerável da informação foi transmitida.

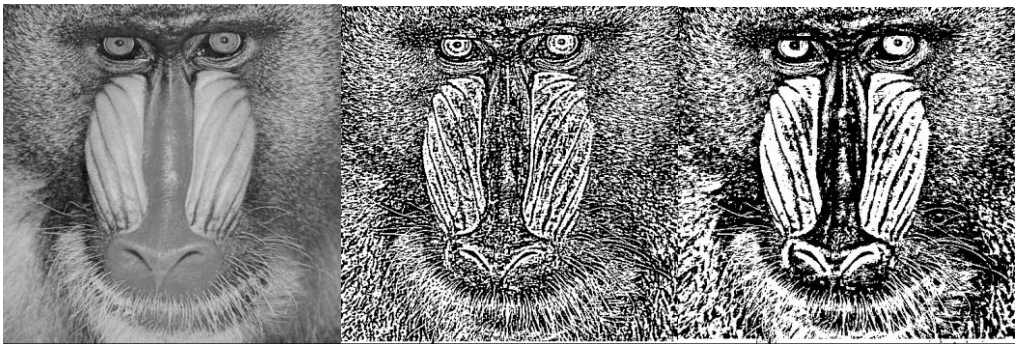
3.2 Método de Bernsen

Neste método, para cada pixel (x, y), considera-se uma vizinhança $(n\ x\ n)$ centrada em (x, y), calculando o limiar como $(Z_{min} + Z_{max})/2$, onde Z_{min} e Z_{max} são os valores de níveis de cinza mínimo e máximo, respectivamente. Para ter um centro adequado, adotou-se números ímpares para n. A Figura 10 exibe como foi desenvolvido esse método:

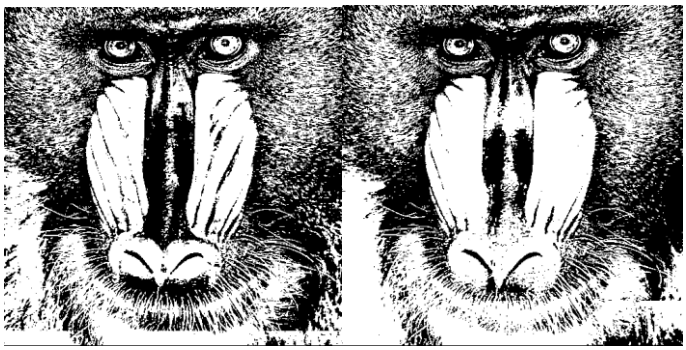
```python
img = cv.imread(f'input/{image_name}.pgm', cv.IMREAD_GRAYSCALE)
a = np.array(img)
a = a.astype(int)

n = neighbours # número de vizinhos
padding_dimens = n // 2
a_padded = np.pad(a, [(padding_dimens, padding_dimens), (padding_dimens, padding_dimens)], mode='reflect')

rows = len(a)
collumns = len(a[0])
center = padding_dimens

for i in range(rows):
    for j in range(collumns):

        # i:i+n, j:j+n
        neighbour_matrix = a_padded[i:i+n, j:j+n]

        aux = neighbour_matrix[center,center]

        neighbour_matrix[center,center] = 0
        max_neighbour = neighbour_matrix.max()
        neighbour_matrix[center,center] = 255
        min_neighbour = neighbour_matrix.min()
        T_limiar = (min_neighbour + max_neighbour)/2

        neighbour_matrix[center,center] = aux
        expression = a[i,j] < T_limiar if invert_background else a[i,j] > T_limiar

        # i, j
        a[i,j] = 255 if expression else 0
```

Figura 10 - Implementação do Método de Bernsen

De acordo com a Figura 10, percebe-se que, inicialmente, foi adicionado uma borda, escolhida como tipo *reflect* (*padding* - numpy.pad) para atender ao formato da matriz de vizinhos $(n\ x\ n)$ sendo o seu centro cada pixel da imagem. Percorreu-se cada pixel, selecionando a matriz de vizinhança do pixel; para achar o valor máximo (numpy.max) sem considerar o centro (que é o pixel), atribuiu-se valor de zero ao centro - se o valor máximo fosse zero, então significa que todos os vizinhos seriam zero, ou seja, é uma manipulação para mascarar o valor do centro. Da mesma forma, para achar o valor mínimo (numpy.min), atribuiu-se valor de 255 ao centro, mascarando-o. Com os valores Z_{min} e Z_{max}, calculou-se o limiar dinâmico T. Vale ressaltar, que precisou-se recuperar o valor original do pixel central uma vez que não feito, alteraria a matriz com a borda adicionada. Dadas as imagens citadas em “Materiais e Métodos”, optou-se por adotar que o que seria objeto o que fosse maior que o limiar T, para manter *invert_background=False* como padrão.



(a) original (b) $n = 7$ (c) $n = 15$



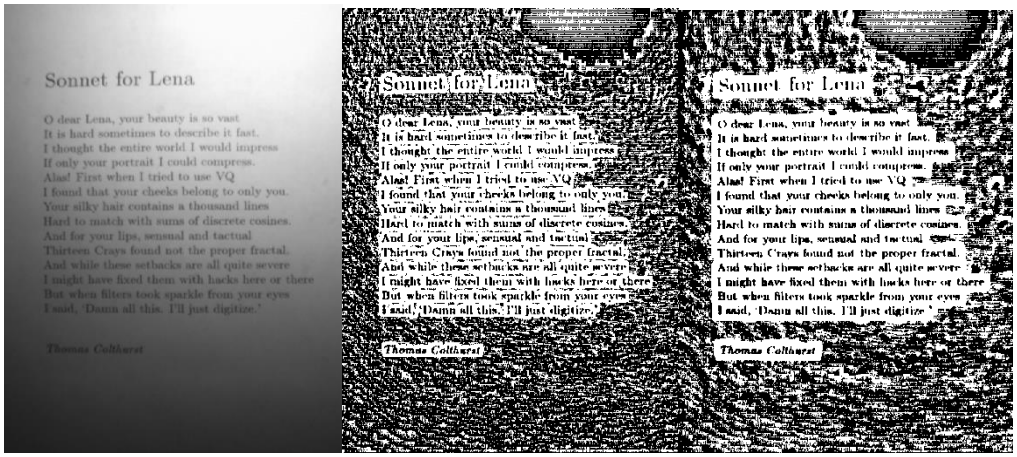
(d) $n = 45$ (e) $n = 135$

Figura 11 – Resultados obtidos para imagem *baboon* usando o Método de Bernsen.

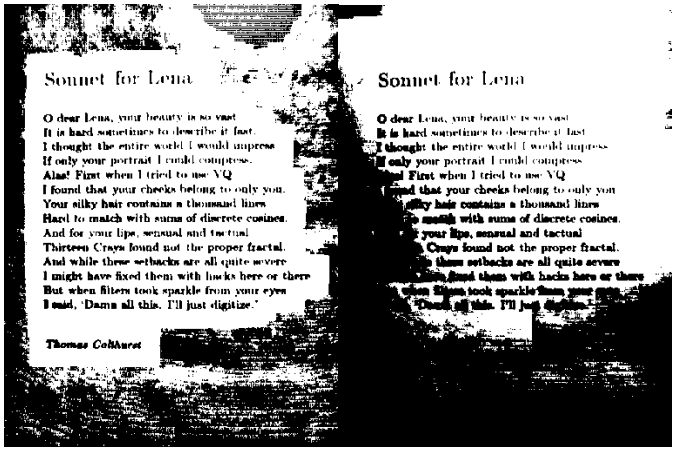
Figura	Fração dos pixels pretos
b ($n = 7$)	0,51
c ($n = 15$)	0,5
d ($n = 45$)	0,45
e ($n = 135$)	0,4

Tabela 5 - Fração dos pixels pretos dos resultados para imagem *baboon*.

Em relação aos resultados para a imagem *baboon*, percebe-se que para vizinhos maiores, obteve-se um resultado melhor, mas é interessante ressaltar que, visualmente, o melhor deles foi com $n = 45$, ou seja, não necessitando aumentar deveras a quantidade de vizinhos para atingir bons resultados.



(a) original (b) $n = 7$ (c) $n = 15$



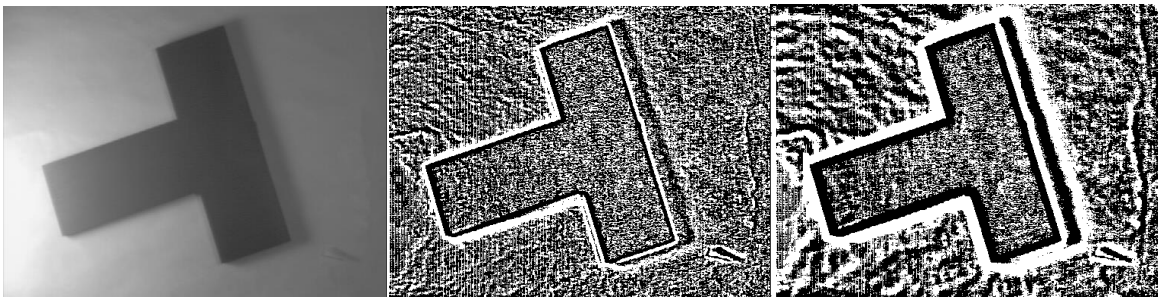
(d) $n = 45$ (e) $n = 135$

Figura 12 - Resultados obtidos para imagem *sonnet* usando o Método de Bernsen.

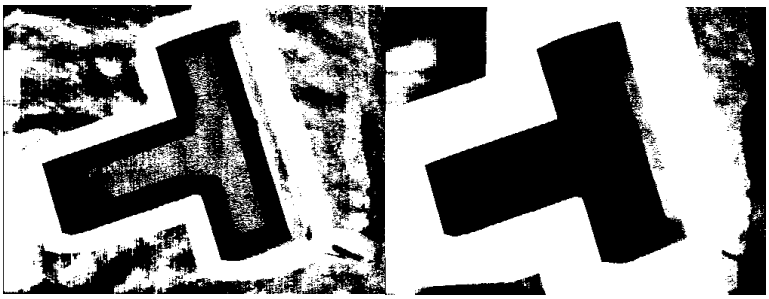
Figura	Fração dos pixels pretos
b ($n = 7$)	0,52
c ($n = 15$)	0,45
d ($n = 45$)	0,37
e ($n = 135$)	0,46

Tabela 6 - Fração dos pixels pretos dos resultados para imagem *sonnet*.

Contudo, analisando os resultados da imagem *sonnet*, notou-se que se obteve resultados não muito satisfatórios, mesmo para vizinhos maiores que lidaram melhor com os ruídos de luminosidade. Embora não tendo um resultado satisfatório, com $n = 45$, toda a informação do texto foi capturada, sem ter ocultá-la.



(a) original (b) $n = 7$ (c) $n = 15$



(d) $n = 45$ (e) $n = 135$

Figura 13 - Resultados obtidos para imagem *wedge* usando o Método de Bernsen.

Figura	Fração dos pixels pretos
b ($n = 7$)	0,57
c ($n = 15$)	0,53
d ($n = 45$)	0,49
e ($n = 135$)	0,43

Tabela 7 - Fração dos pixels pretos dos resultados para imagem *wedge*.

Por fim, para a imagem *wedge*, tal método produziu um resultado razoável para $n = 135$, porém ainda considerando parte do ruído da luminosidade produzindo sombra. Para os demais valores de n, era possível identificar o "T" por sua borda, porém o objeto não era evidenciado.

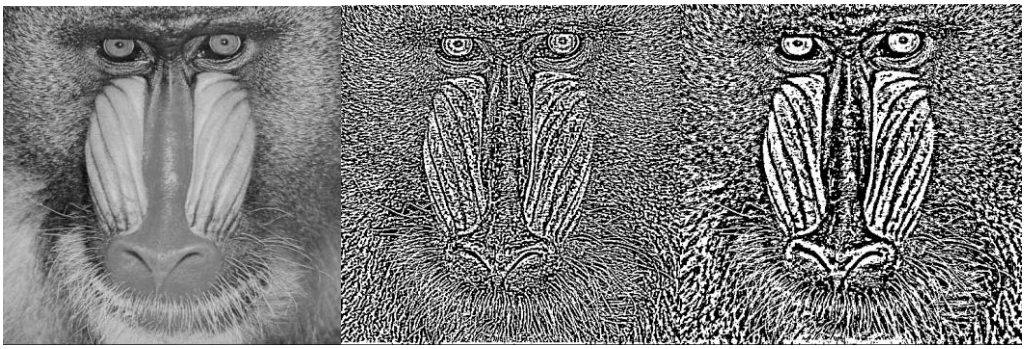
3.3 Método de Niblack

O objetivo principal deste método era, para cada pixel (x, y), considerar-se uma vizinhança $(n\ x\ n)$ centrada em (x, y), calculando o limiar como $\mu(x, y) + k * \sigma(x, y)$, onde μ é a média entre os vizinhos e σ é o desvio padrão entre os mesmos; também, tem-se o parâmetro k, o qual é usado para ajustar a fração da borda do objeto a ser considerada como parte dele. Para ter um centro adequado, adotou-se números ímpares para n. A Figura 14 mostra como realizou-se tal método:

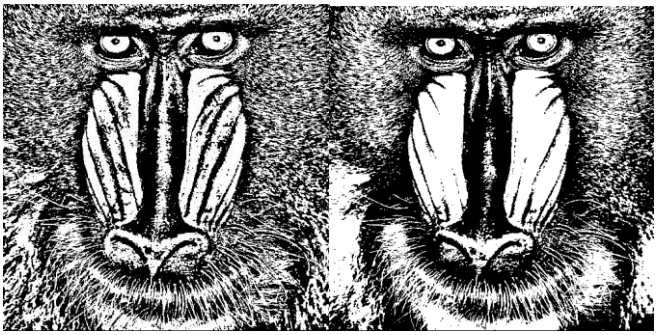
```python
img = cv.imread(f'input/{image_name}.pgm', cv.IMREAD_GRAYSCALE)

a = np.array(img)
a = a.astype(int)

n = neighbours # número de vizinhos
k = k_value

padding_dimens = n // 2
a_padded = np.pad(a, [(padding_dimens, padding_dimens), (padding_dimens, padding_dimens)], mode='reflect')

rows = len(a)
collumns = len(a[0])
center = padding_dimens

mask_center = np.ones(n*n).reshape(n,n)
mask_center[center, center] = 0
mask_center = mask_center.astype(bool)

for i in range(rows):
    for j in range(collumns):

        # i:i+n, j:j+n
        neighbour_matrix = a_padded[i:i+n, j:j+n]

        neighbour_matrix_center_masked = neighbour_matrix[mask_center]
        neighbour_mean = np.mean(neighbour_matrix_center_masked)
        neighbour_standard_deviation = np.std(neighbour_matrix_center_masked)

        T_limiar = neighbour_mean + (k*neighbour_standard_deviation)
        expression = a[i,j] < T_limiar if invert_background else a[i,j] > T_limiar

        # i, j
        a[i,j] = 255 if expression else 0
```

Figura 14 - Implementação do Método de Niblack

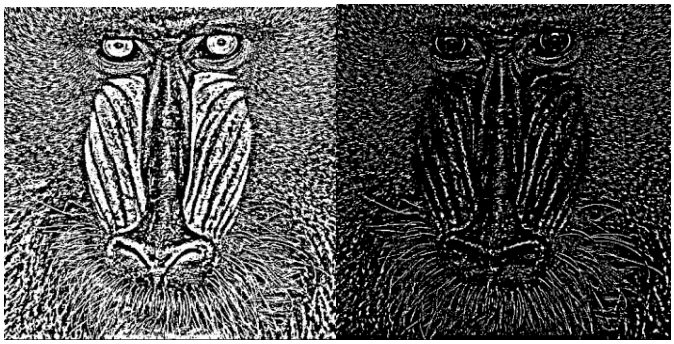
Para reproduzir tal método, foi necessário adicionar uma borda, escolhida como tipo *reflect* (*padding* - numpy.pad) para atender ao formato da matriz de vizinhos $(n\ x\ n)$ sendo o seu centro cada pixel da imagem. Além disso, criou-se uma máscara para não considerar o centro nos cálculos da média e desvio padrão. Em seguida, percorreu-se cada pixel, selecionando a matriz de vizinhança do pixel; para achar média (numpy.mean) e desvio padrão (numpy.std) sem considerar o centro, utilizou-se a máscara criada na matriz de vizinhança. Com tais valores, calculou-se a equação $\mu(x,y) + k * \sigma(x,y)$ para alguns $k's$. Dadas as imagens citadas em "Materiais e Métodos", optou-se por adotar que o que seria objeto o que fosse maior que o limiar T, para manter *invert_background=False* como padrão.



(a) original (b) $n = 7$ (c) $n = 15$



(d) $n = 45$ (e) $n = 135$



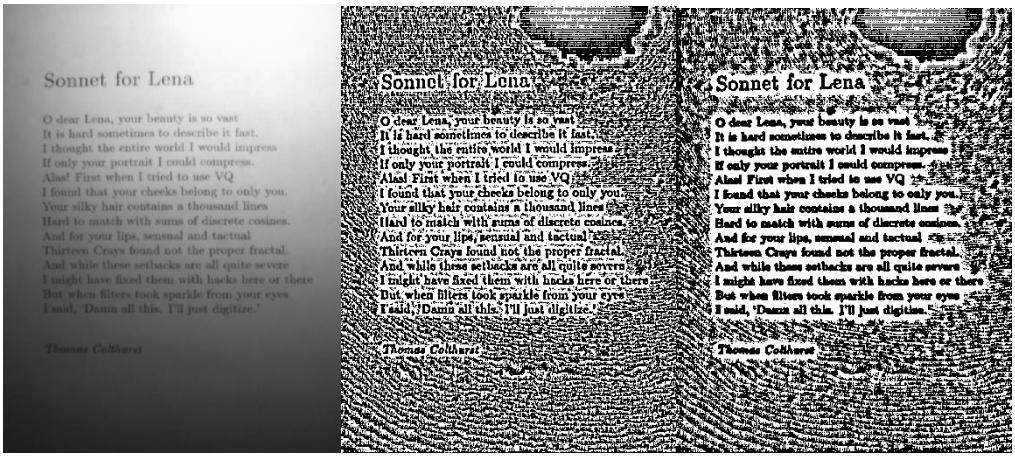
(f) $n = 15, k = 0.1$ (g) $n = 15, k = 1$

Figura 15 - Resultados obtidos para imagem *baboon* usando o Método de Niblack.

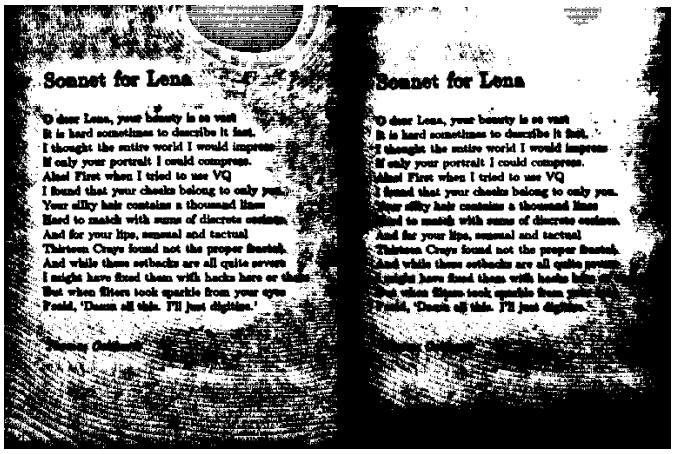
Figura	Fração dos pixels pretos
b ($n = 7$)	0,54
c ($n = 15$)	0,53
d ($n = 45$)	0,52
e ($n = 135$)	0,52
f ($n = 15, k = 0.1$)	0,53
g ($n = 15, k = 1$)	0,86

Tabela 8 - Fração dos pixels pretos dos resultados para imagem *baboon*.

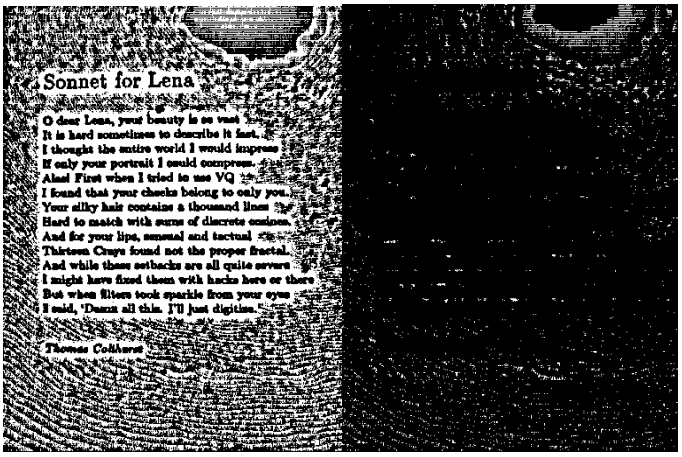
E, para a imagem *baboon*, percebe-se que os vizinhos maiores novamente retornaram um resultado melhor, e também que colocando $n = 45$, foi suficiente para transmitir a informação contida na imagem de entrada, aparecendo os traços característicos do animal. Vale comentar que este método permite a alteração do parâmetro k, o qual foi adotado como 0,1: nos testes realizados com a ordem decimal, teve-se melhores resultados como ilustra a Figura 15 (g) (ele causa grande impacto na informação a ser transmitida pela imagem - somente nesse caso alterou-se $k = 1$).



(a) original (b) $n = 7$ (c) $n = 15$



(d) $n = 45$ (e) $n = 135$



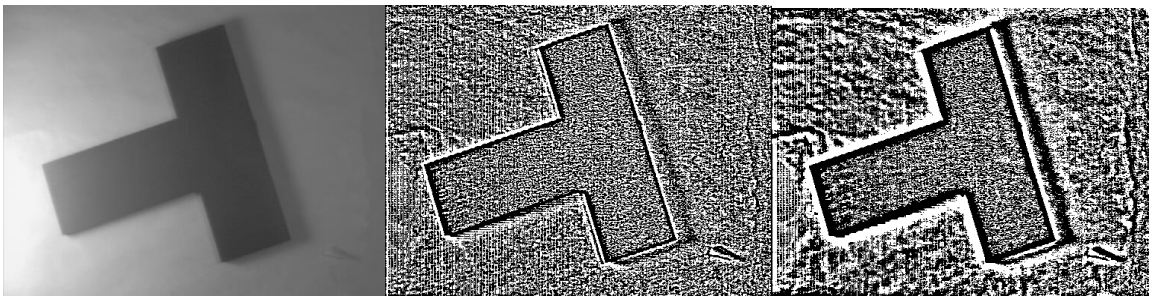
(f) $n = 15, k = 0.1$ (g) $n = 15, k = 1$

Figura 16 - Resultados obtidos para imagem *sonnet* usando o Método de Niblack.

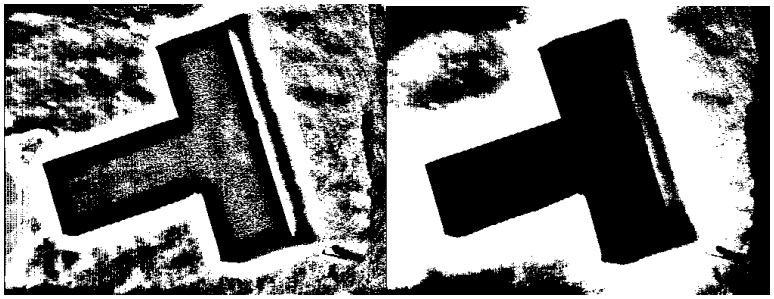
Figura	Fração dos pixels pretos
b ($n = 7$)	0,48
c ($n = 15$)	0,48
d ($n = 45$)	0,49
e ($n = 135$)	0,52
f ($n = 15, k = 0.1$)	0,48
g ($n = 15, k = 1$)	0,97

Tabela 9 - Fração dos pixels pretos dos resultados para imagem *sonnet*.

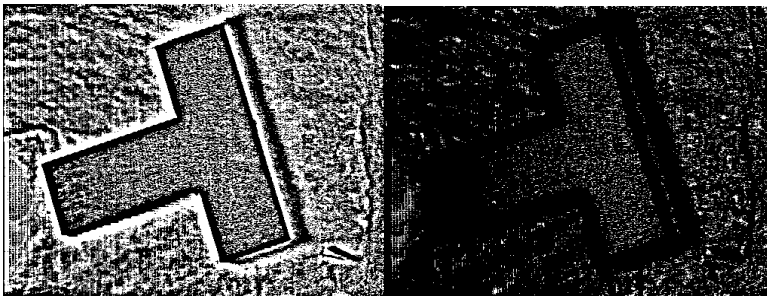
Entretanto, analisando os resultados da imagem *sonnet*, notou-se que obteve-se resultados não muito desejados, mesmo para vizinhos maiores que tentaram lidar melhor com os ruídos, mas ainda não de forma suficiente. Embora não tendo um resultado satisfatório, com $n = 45$, toda a informação do texto foi capturada. Também, vale ressaltar que adotou-se k como 0,1 para imagem *sonnet*, pois novamente nos testes realizados com a ordem decimal, teve-se melhores resultados como ilustra a Figura 16 (g) (somente nesse caso alterou-se $k = 1$).



(a) original (b) $n = 7$ (c) $n = 15$



(d) $n = 45$ (e) $n = 135$



(f) $n = 15, k = 0.1$ (g) $n = 15, k = 1$

Figura 17 - Resultados obtidos para imagem *wedge* usando o Método de Niblack.

Figura	Fração dos pixels pretos
b ($n = 7$)	0,54
c ($n = 15$)	0,54
d ($n = 45$)	0,55
e ($n = 135$)	0,5
f ($n = 15, k = 0.1$)	0,54
g ($n = 15, k = 1$)	0,93

Tabela 10 - Fração dos pixels pretos dos resultados para imagem *wedge*.

Além disso, para a imagem *wedge*, notou-se que tal método produziu resultados mais adequados para vizinhos maiores e, nesse caso, com $n = 135$, mas ainda considerando parte do ruído. Por fim, destaca-se que se adotou k como 0,1 para tal imagem, pois novamente nos testes realizados com a ordem decimal, teve-se melhores resultados como ilustra a Figura 17 (g) (somente nesse caso alterou-se $k = 1$).

3.4 Método de Sauvola e Pietaksinen

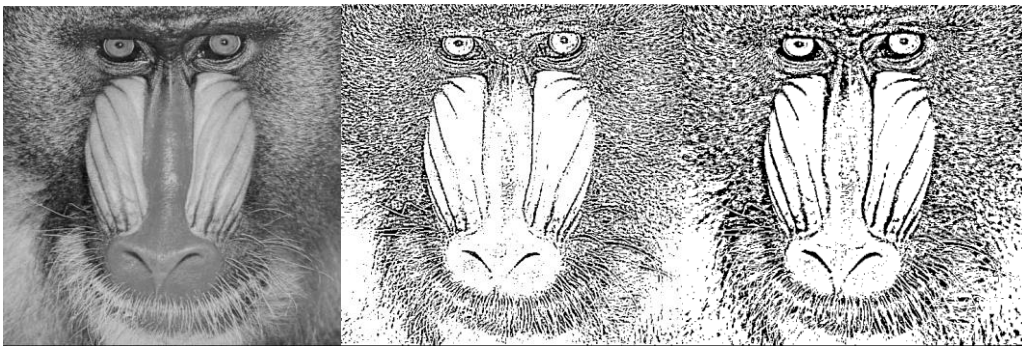
Neste método, era necessário considerar-se uma vizinhança $(n\ x\ n)$ centrada em (x, y) para cada pixel (x, y), calculando o limiar como $\mu(x,y)\left[1 + k * \left(\frac{\sigma(x,y)}{R} - 1\right)\right]$, onde μ é a média entre os vizinhos e σ é o desvio padrão entre os mesmos; também, tem-se o parâmetro k que geralmente possui uma grandeza baixa (valor baixo) e R, que pode ser colocado com valores da ordem de dezena, centena. Para ter um centro adequado, adotou-se números ímpares para n. A seguir, é exibido pela Figura 18, a solução proposta para este método:

```python
img = cv.imread(f'input/{image_name}.pgm', cv.IMREAD_GRAYSCALE)
a = np.array(img)
a = a.astype(int)

n = neighbours # número de vizinhos
k = k_value
R = R_value

padding_dimens = n // 2
a_padded = np.pad(a, [(padding_dimens, padding_dimens), (padding_dimens, padding_dimens)], mode='reflect')

rows = len(a)
collumns = len(a[0])
center = padding_dimens

mask_center = np.ones(n*n).reshape(n,n)
mask_center[center, center] = 0
mask_center = mask_center.astype(bool)

for i in range(rows):
    for j in range(collumns):

        # i:i+n, j:j+n
        neighbour_matrix = a_padded[i:i+n, j:j+n]

        neighbour_matrix_center_masked = neighbour_matrix[mask_center]
        neighbour_mean = np.mean(neighbour_matrix_center_masked)
        neighbour_standard_deviation = np.std(neighbour_matrix_center_masked)

        T_limiar = neighbour_mean * (1 + ( k * ( (neighbour_standard_deviation/R) - 1) ) )

        expression = a[i,j] < T_limiar if invert_background else a[i,j] > T_limiar

        # i, j
        a[i,j] = 255 if expression else 0
```

Figura 18 - Implementação do Método de Sauvola e Pietaksinen

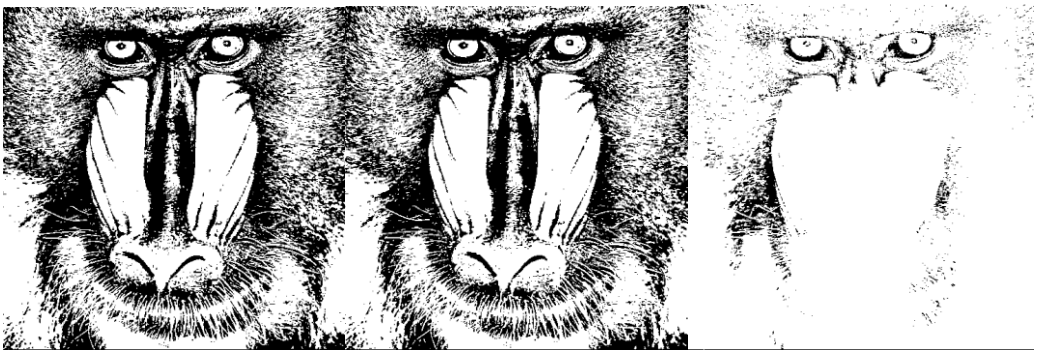
Primeiramente, adicionou-se uma borda, escolhida como tipo *reflect* (*padding* - numpy.pad) para atender ao formato da matriz de vizinhos $(n\ x\ n)$

sendo o seu centro cada pixel da imagem. Além disso, criou-se uma máscara para não considerar o centro nos cálculos da média e desvio padrão. Em seguida, percorreu-se cada pixel, selecionando a matriz de vizinhança do pixel; para achar média (numpy.mean) e desvio padrão (numpy.std) sem considerar o centro, utilizou-se a máscara criada na matriz de vizinhança. Com tais valores, calculou-se a equação $\mu(x,y)\left[1 + k * \left(\frac{\sigma(x,y)}{R} - 1\right)\right]$ para alguns $k's$ e $R's$. Dadas as imagens citadas em “Materiais e Métodos”, optou-se por adotar que o que seria objeto o que fosse maior que o limiar T, para manter *invert_background=False* como padrão.



(a) original (b) $n = 7$ (c) $n = 15$

(d) $n = 45$ (e) $n = 135$



(f) $k = 0.1, R = 90$ (g) $k = 0.1, R = 180$ (h) $k = 1, R = 90$

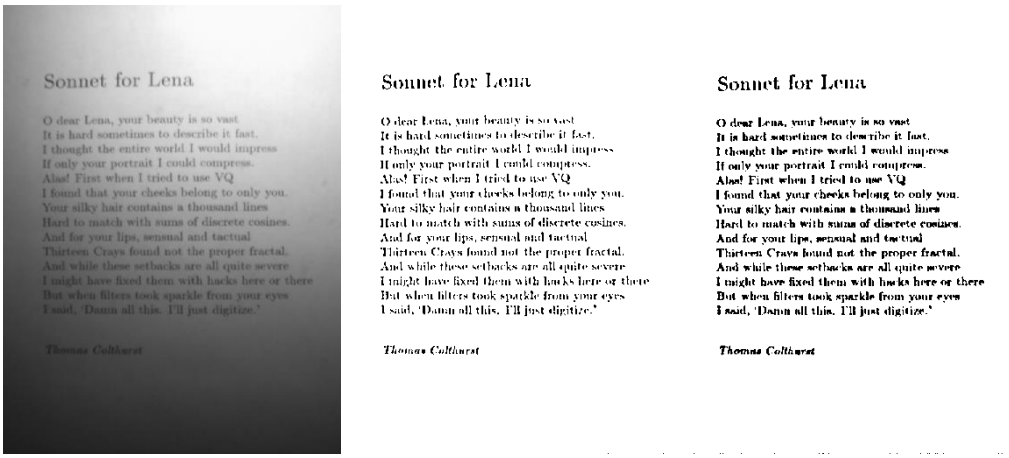
Figura 19 - Resultados obtidos para imagem *baboon* usando o Método de Sauvola e Pietaksinen.

Figura	Fração dos pixels pretos
b ($n = 7$)	0,27
c ($n = 15$)	0,3
d ($n = 45$)	0,34
e ($n = 135$)	0,4
f ($n = 135, k = 0.1, R = 90$)	0,4
g ($n = 135, k = 0.1, R = 180$)	0,37
h ($n = 135, k = 1, R = 90$)	0,05

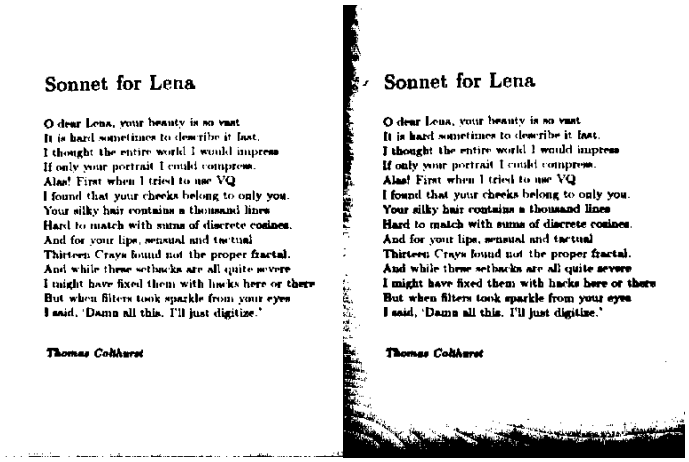
Tabela 11 - Fração dos pixels pretos dos resultados para imagem *baboon*.

Em relação aos resultados obtidos para a imagem *baboon*, percebe-se que os vizinhos maiores novamente retornaram um resultado melhor, e também que colocando $n = 45$; mas é pertinente comentar que já para um $n = 15$ tem-se uma versão melhor definida comparada aos 3 métodos apresentados anteriormente.

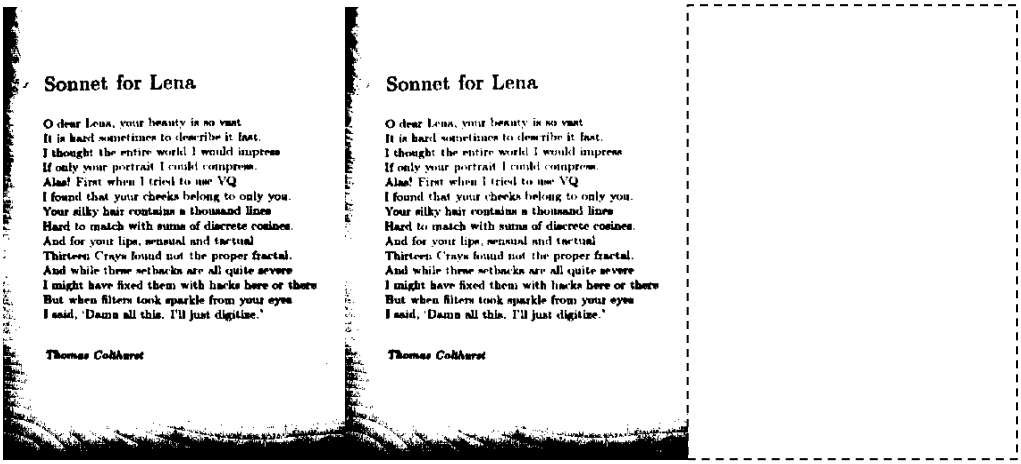
Vale comentar que este método permite a alteração do parâmetro k e R, os quais foram adotados como 0,1 e 90, respectivamente. Nos testes realizados, para k a ordem decimal se manteve como a mais adequada, como ilustra a Figura 19 (h); ao passo que, R não exibiu grandes diferenças mesmo dobrando seu valor, como as imagens Figuras 19 (f) e (g) mostram.



(a) original (b) $n = 7$ (c) $n = 15$



(d) $n = 45$ (e) $n = 135$



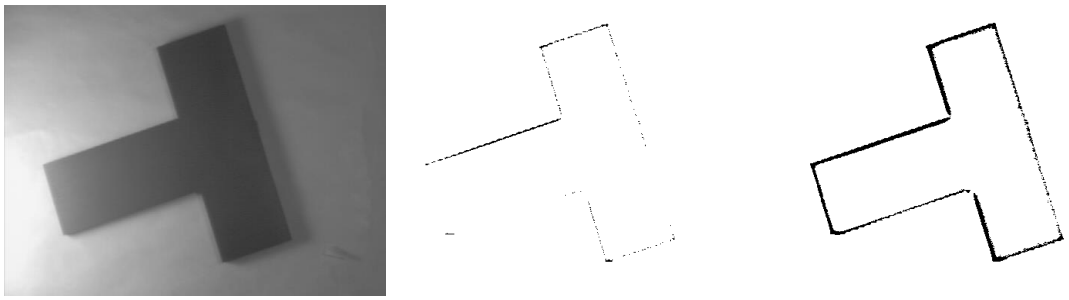
(f) $k = 0.1, R = 90$ (g) $k = 0.1, R = 180$ (h) $k = 1, R = 90$

Figura 20 - Resultados obtidos para imagem *sonnet* usando o Método de Sauvola e Pietaksinen.

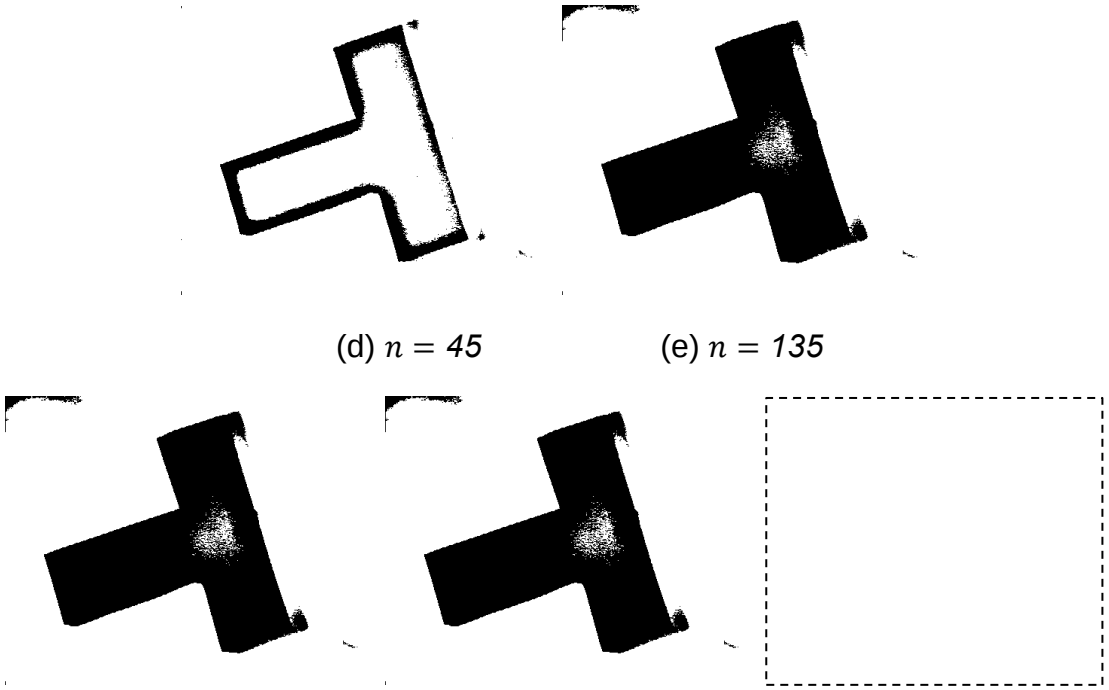
Figura	Fração dos pixels pretos
b ($n = 7$)	0,04
c ($n = 15$)	0,06
d ($n = 45$)	0,07
e ($n = 135$)	0,15
f ($n = 135, k = 0.1, R = 90$)	0,15
g ($n = 135, k = 0.1, R = 180$)	0,12
h ($n = 135, k = 1, R = 90$)	0,0

Tabela 12 - Fração dos pixels pretos dos resultados para imagem *sonnet*.

Além disso, analisando os resultados da imagem *sonnet*, obteve-se resultados satisfatórios desta vez, pois o ruído foi tratado adequadamente já para vizinhos pequenos, ou seja, não necessitando de grandes processamentos para atingir um resultado plausível (para $n = 7$, desconsiderou-se bem o ruído). Vale comentar que este método permite a alteração do parâmetro k e R, os quais foram adotados como 0,1 e 90, respectivamente. Nos testes realizados para essa imagem, k de ordem decimal se manteve como a mais adequado, como ilustra a Figura 20 (h) onde o resultado produzido foi inteiramente 255; paralelamente, R não exibiu grandes diferenças mesmo dobrando seu valor, como as imagens Figuras 20 (f) e (g) mostram.



(a) original (b) $n = 7$ (c) $n = 15$



(d) $n = 45$ (e) $n = 135$

(f) $k = 0.1, R = 90$ (g) $k = 0.1, R = 180$ (g) $k = 1, R = 90$

Figura 21 - Resultados obtidos para imagem *wedge* usando o Método de Sauvola e Pietaksinen.

Figura	Fração dos pixels pretos
b ($n = 7$)	0,003
c ($n = 15$)	0,02
d ($n = 45$)	0,082
e ($n = 135$)	0,25
f ($n = 135, k = 0.1, R = 90$)	0,25
g ($n = 135, k = 0.1, R = 180$)	0,23
h ($n = 135, k = 1, R = 90$)	0,0

Tabela 13 - Fração dos pixels pretos dos resultados para imagem *wedge*.

Por fim, para a imagem *wedge*, notou-se que tal método produziu resultados mais adequados para vizinhos maiores e, nesse caso, com $n = 135$,

mas ainda considerando parte do ruído, similar ao Niblack. Ademais, vale ressaltar que adotou-se os parâmetros k e R como 0,1 e 90, respectivamente. Nos testes realizados para essa imagem, k de ordem decimal se manteve como a mais adequado, como ilustra a Figura 21 (h) - resultado inteiramente 255; paralelamente, R não exibiu grandes diferenças mesmo dobrando seu valor, como é exibido pelas Figuras 21 (f) e (g).

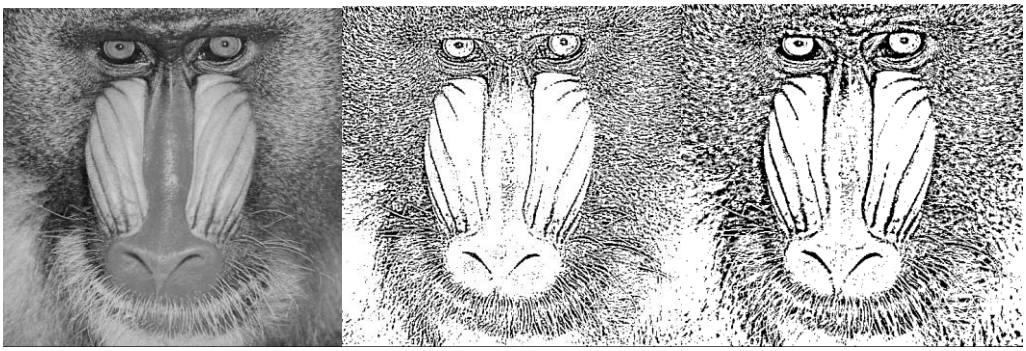
3.5 Método de Phansalskar, More e Sabale

O objetivo deste método era considerar uma vizinhança $(n\ x\ n)$ centrada em (x, y), calculando o limiar como $\mu(x,y)\left[1 + p * e^{(-q*\mu(x,y))} + k * \left(\frac{\sigma(x,y)}{R} - 1\right)\right]$, onde μ é a média entre os vizinhos e σ é o desvio padrão entre os mesmos; também, tem-se o parâmetro k que geralmente possui uma grandeza baixa (valor baixo) e R, que pode ser colocado com valores da ordem de dezena, centena. Além disso, o parâmetro p não possui grande dimensão bem como q (unidade, dezena) e como foi sugerido pelos autores $p = 2$ e $q = 10$, nota-se que geralmente $q > p$. Para ter um centro adequado, adotou-se números ímpares para n. Na Figura 22, mostra-se como reproduziu-se tal método:

```python
img = cv.imread(f'input/{image_name}.pgm', cv.IMREAD_GRAYSCALE)
a = np.array(img)
a = a.astype(int)

n = neighbours # número de vizinhos
k = k_value
R = R_value
p = p_value
q = q_value

padding_dimens = n // 2
a_padded = np.pad(a, [(padding_dimens, padding_dimens), (padding_dimens, padding_dimens)], mode='reflect')

rows = len(a)
collumns = len(a[0])
center = padding_dimens

mask_center = np.ones(n*n).reshape(n,n)
mask_center[center, center] = 0
mask_center = mask_center.astype(bool)

for i in range(rows):
    for j in range(collumns):

        # i:i+n, j:j+n
        neighbour_matrix = a_padded[i:i+n, j:j+n]

        neighbour_matrix_center_masked = neighbour_matrix[mask_center]
        neighbour_mean = np.mean(neighbour_matrix_center_masked)
        neighbour_standard_deviation = np.std(neighbour_matrix_center_masked)

        T_limiar = neighbour_mean * (1 + (p * (math.exp((-q)*neighbour_mean))) + ( k * ( (neighbour_standard_deviation/R) - 1) ) )

        expression = a[i,j] < T_limiar if invert_background else a[i,j] > T_limiar

        # i, j
        a[i,j] = 255 if expression else 0
```

Figura 22 - Implementação do Método de Phansalskar, More e Sabale.

Inicialmente, adicionou-se uma borda, escolhida como tipo *reflect* (*padding* - numpy.pad) para atender ao formato da matriz de vizinhos $(n\ x\ n)$ sendo o seu centro cada pixel da imagem. Além disso, criou-se uma máscara para não considerar o centro nos cálculos da média e desvio padrão. Em seguida, percorreu-se cada pixel, selecionando a matriz de vizinhança do pixel; para achar média (numpy.mean) e desvio padrão (numpy.std) sem considerar o centro, utilizou-se a máscara criada na matriz de vizinhança. Com tais valores, calculou-se a equação $\mu(x,y)\left[1 + p * e^{(-q*\mu(x,y))} + k * \left(\frac{\sigma(x,y)}{R} - 1\right)\right]$ para alguns $k's$ e $R's$. Também, alguns valores para p e q foram utilizados, porém como não apresentaram muitas alterações adotou-se nos resultados o que foi sugerido pelos autores do método ($p = 2, q = 10$). Dadas as imagens citadas em “Materiais e Métodos”, optou-se por adotar que o que seria objeto o que fosse maior que o limiar T, para manter *invert_background=False* como padrão.



(a) original (b) $n = 7$ (c) $n = 15$

(d) $n = 45$ (e) $n = 135$

(f) $k = 0.1, R = 90$ (g) $k = 0.1, R = 180$ (h) $k = 0.7, R = 90$ (i) $k = 0.7, R = 180$

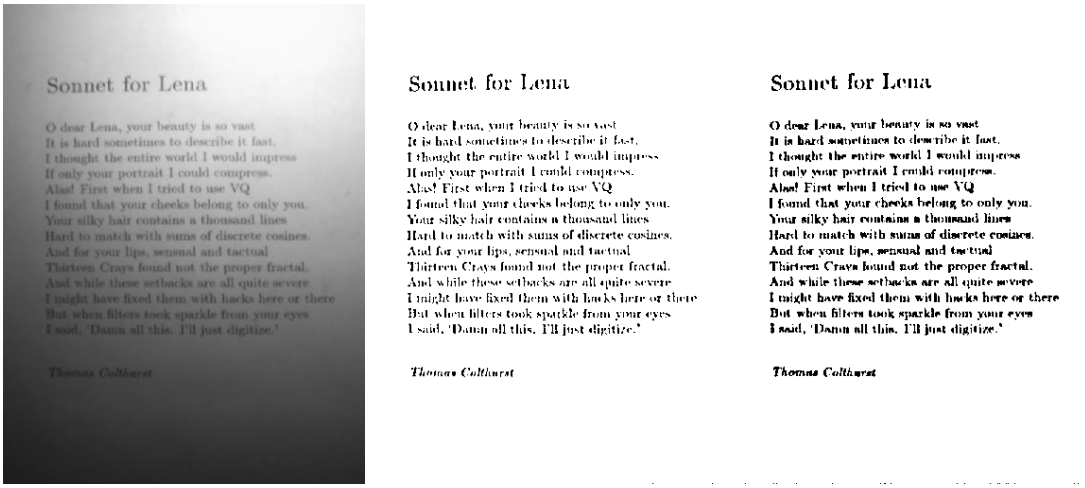
Figura 23 - Resultados obtidos para imagem *baboon* usando o Método de Phansalskar, More e Sabale.

Em relação aos resultados obtidos para a imagem *baboon*, percebe-se que os vizinhos maiores novamente retornaram um resultado melhor, e também que colocando $n = 45$; mas é pertinente comentar que já para um $n = 15$ tem-se uma versão razoável, similar ao em Sauvola e Pietaksinen.

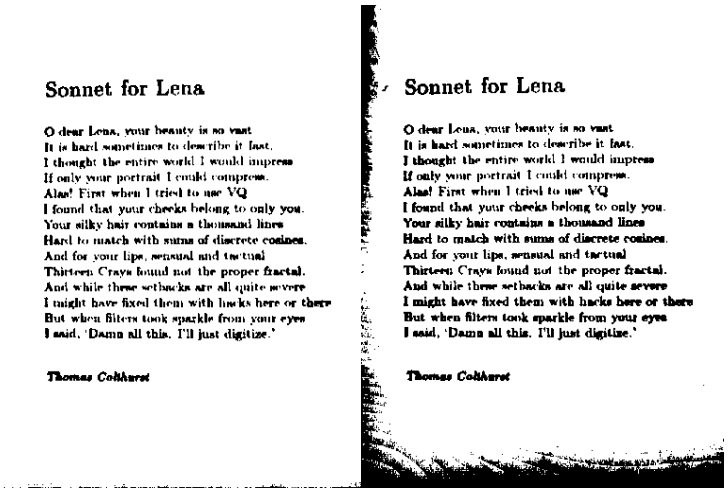
E, destaca-se que este método permite a alteração do parâmetro k e R, os mesmos foram adotados como 0,1 e 90, respectivamente. Também é possível alterar p e q, os quais foram mantidos em 2 e 10 (sugestão dos autores). Porém, foi feito um teste simples na imagem *wedge* e será discutido brevemente sobre eles. Nos testes realizados, a ordem decimal para k se manteve como a mais adequada, como ilustra a Figura 23 (h) e (i), onde já se perdeu um pouco da imagem com $k = 0.7$; ao passo que, R não exibiu grandes diferenças mesmo dobrando seu valor, como as imagens Figuras 23 (f) e (g) mostram. É interessante pontuar que mesmo R não tendo grandes impactos para imagens com mais pixels zero, quando observa-se as Figuras 23 (h) e (i), percebe-se um ligeiro clareamento.

Figura	Fração dos pixels pretos
b ($n = 7$)	0,27
c ($n = 15$)	0,3
d ($n = 45$)	0,34
e ($n = 135$)	0,4

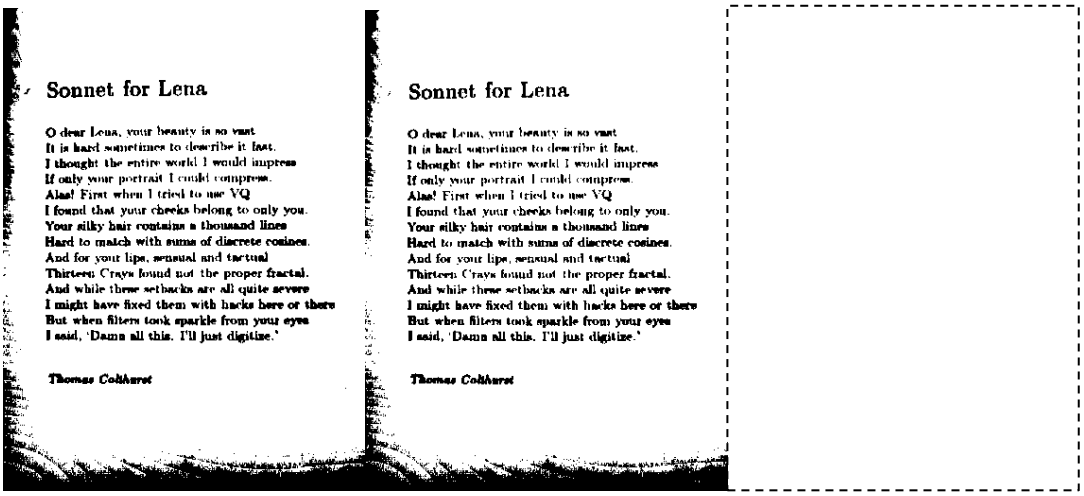
Tabela 14 - Fração dos pixels pretos dos resultados para imagem *baboon*.



(a) original (b) $n = 7$ (c) $n = 15$



(c) $n = 45$ (d) $n = 135$



(e) $k = 0.1, R = 90$ (f) $k = 0.1, R = 180$ (g) $k = 0.7, R = 90$

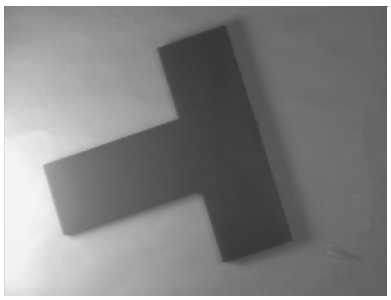
Figura 24 - Resultados obtidos para imagem *sonnet* usando o Método de Phansalskar, More e Sabale.

Além disso, analisando os resultados da imagem *sonnet*, obteve-se resultados satisfatórios, uma vez que o ruído foi tratado adequadamente já para vizinhos pequenos e, por consequência, não necessitando de grandes processamentos (alto número de vizinhos) para atingir um resultado plausível (para $n = 7$, desconsiderou-se bem o ruído). Também, ressalta-se que este método permite a alteração do parâmetro k e R, os mesmos foram adotados

como 0,1 e 90, respectivamente. Também é possível alterar p e q, os quais foram mantidos em 2 e 10 (sugestão dos autores). Nos testes realizados para essa imagem, k de ordem decimal se manteve como a mais adequado, como ilustra a Figura 24 (g) onde o resultado produzido foi inteiramente 255 com um valor k próximo a 1; paralelamente, R não exibiu grandes diferenças mesmo dobrando seu valor, como as imagens Figuras 24 (f) e (g) mostram.

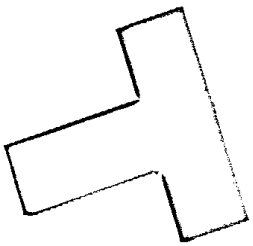
Figura	Fração dos pixels pretos
b ($n = 7$)	0,04
c ($n = 15$)	0,06
d ($n = 45$)	0,07
e ($n = 135$)	0,15

Tabela 15 - Fração dos pixels pretos dos resultados para imagem *sonnet*.

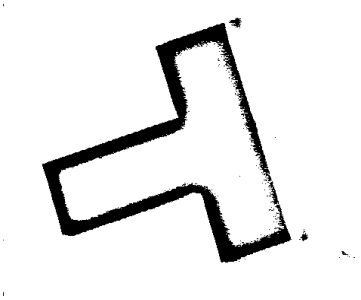


(a) original

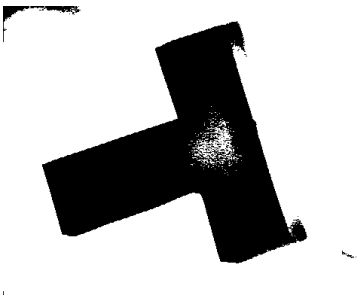
(b) $n = 7$



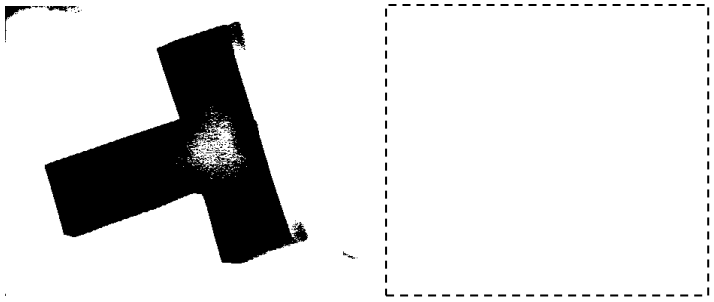
(c) $n = 15$



(c) $n = 45$



(d) $n = 135$



(e) $n = 135, k = 0.1$ (f) $n = 135, k = 0.7$

(g) $p = 1, q = 5$ (h) $p = 2, q = 10$

Figura 25 - Resultados obtidos para imagem *wedge* usando o Método de Phansalskar, More e Sabale.

Figura	Fração dos pixels pretos
b ($n = 7$)	0,003
c ($n = 15$)	0,02
d ($n = 45$)	0,082
e ($n = 135, k = 0.1$)	0,25
f ($n = 135, k = 0.7$)	0
g ($p = 1, q = 5$)	0,24
h ($n = 135, R = 90, k = 0.1, p = 2, q = 10$)	0,24

Tabela 16 - Fração dos pixels pretos dos resultados para imagem *wedge*.

Por fim, para a imagem *wedge*, notou-se que tal método produziu resultados mais adequados para vizinhos maiores e, nesse caso, com $n = 135$, mas ainda considerando parte do ruído. Além disso, vale ressaltar que adotou-se os parâmetros k e R como 0,1 e 90, respectivamente. Nos testes realizados para essa imagem, k de ordem decimal se manteve como a mais adequado, como ilustra a Figura 25 (f) - resultado inteiramente 255 com valor de k próximo a 1; paralelamente, alterou-se os parâmetros de p e q, dobrando-os: inicialmente, não foram notadas grandes alterações - Figuras 25 (g) e (h) -, provavelmente causando mais impacto para valores bem maiores.

3.6 Método do Contraste

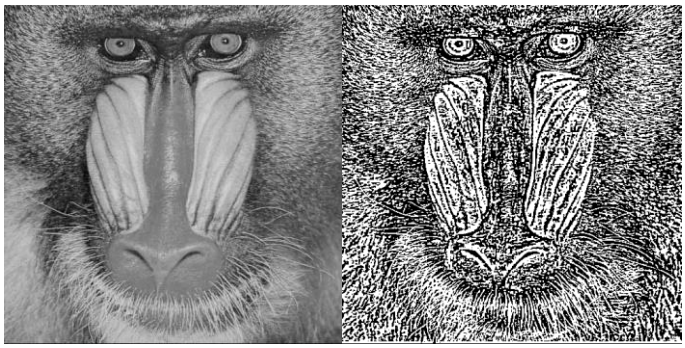
Neste método, para cada pixel (x, y), considera-se uma vizinhança $(n\ x\ n)$ centrada em (x, y), calculando o limiar conforme o pixel esteja mais próximo do Z_{min} ou Z_{max}, onde Z_{min} e Z_{max} são os valores de níveis de cinza mínimo e máximo, respectivamente. Para ter um centro adequado, adotou-se números ímpares para n. A Figura 26 ilustra o desenvolvimento do método:

```python
img = cv.imread(f'input/{image_name}.pgm', cv.IMREAD_GRAYSCALE)
a = np.array(img)

a = a.astype(int)
n = neighbours # número de vizinhos
padding_dimens = n // 2

a_padded = np.pad(a, [(padding_dimens, padding_dimens), (padding_dimens, padding_dimens)], mode='reflect')

rows = len(a)
collumns = len(a[0])
center = padding_dimens

mask_center = np.ones(n*n).reshape(n,n)
mask_center[center, center] = 0
mask_center = mask_center.astype(bool)

for i in range(rows):
    for j in range(collumns):

        # i:i+n, j:j+n
        neighbor_matrix = a_padded[i:i+n, j:j+n]

        aux = neighbor_matrix[center,center]

        neighbor_matrix[center,center] = 0
        max_neighbor = neighbor_matrix.max()
        neighbor_matrix[center,center] = 255
        min_neighbor = neighbor_matrix.min()

        neighbor_matrix[center,center] = aux

        distanceToMax = abs(max_neighbor - aux)
        distanceToMin = abs(aux - min_neighbor)

        expression = distanceToMax < distanceToMin if invert_background else distanceToMax > distanceToMin

        # i, j
        a[i,j] = 0 if expression else 255
```

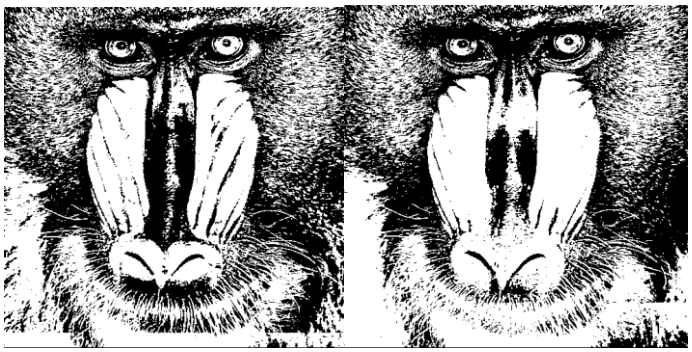
Figura 26 - Implementação do Contraste

Como é apresentado pela Figura 26, inseriu-se uma borda, escolhida como tipo *reflect* (*padding* - numpy.pad) para atender ao formato da matriz de vizinhos $(n\ x\ n)$ sendo o seu centro cada pixel da imagem. Percorreu-se cada pixel, selecionando a matriz de vizinhança do pixel; para achar o valor máximo (numpy.max) sem considerar o centro (que é o pixel), atribuiu-se valor de zero ao centro (como no método de Bernsen). Da mesma forma, para achar o valor mínimo (numpy.min), atribuiu-se valor de 255 ao centro, mascarando-o. Com os valores Z_{min} e Z_{max}, definiu-se que se o pixel estivesse mais próximo do mínimo, ele receberia o valor de zero, caso contrário (mais próximo do máximo) receberia 255 - isso considerando que *invert_background=False*. Vale ressaltar, que

precisou-se recuperar o valor original do pixel central uma vez que não feito, alteraria a matriz com a borda adicionada.



(a) original (b) $n = 7$ (c) $n = 15$



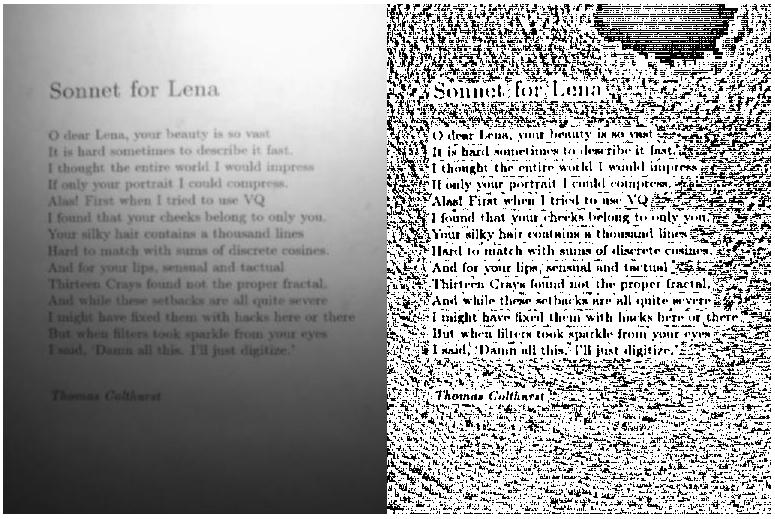
(d) $n = 45$ (e) $n = 135$

Figura 27 - Resultados obtidos para imagem *baboon* usando o Método do Contraste.

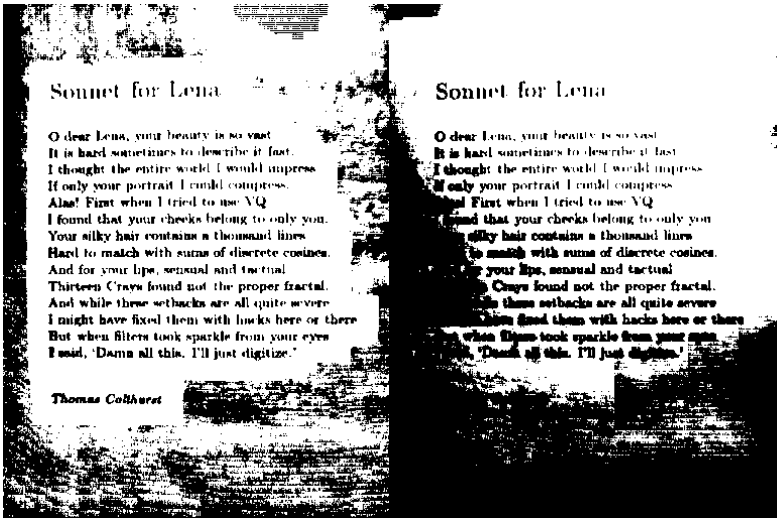
Figura	Fração dos pixels pretos
b ($n = 7$)	0,5
c ($n = 15$)	0,5
d ($n = 45$)	0,45
e ($n = 135$)	0,39

Tabela 17 - Fração dos pixels pretos dos resultados para imagem *baboon*.

Sobre resultados para a imagem *baboon*, percebe-se que para vizinhos maiores, obteve-se um resultado melhor, mas é interessante ressaltar que um resultado adequado foi alcançado com $n = 45$.



(a) original (b) $n = 7$ (c) $n = 15$



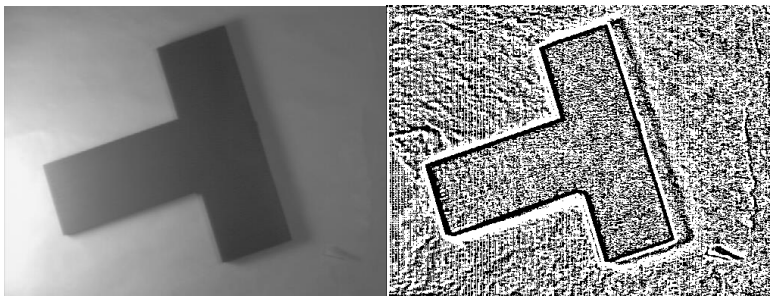
(c) $n = 45$ (d) $n = 135$

Figura 28 - Resultados obtidos para imagem *sonnet* usando o Método do Contraste.

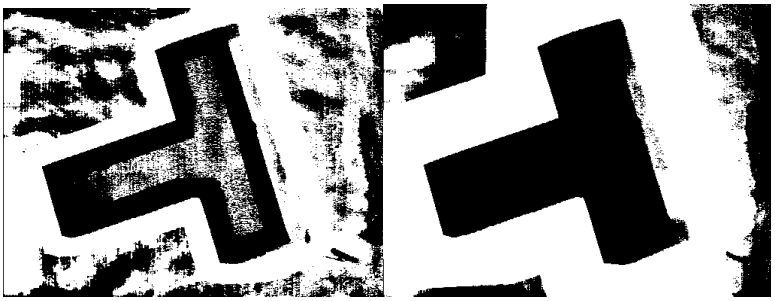
Figura	Fração dos pixels pretos
b ($n = 7$)	0,26
c ($n = 15$)	0,26
d ($n = 45$)	0,27
e ($n = 135$)	0,43

Tabela 18 - Fração dos pixels pretos dos resultados para imagem *sonnet*.

Entretanto, para a imagem *sonnet*, obteve-se resultados não muito satisfatórios, mesmo para vizinhos maiores que lidaram melhor com os ruídos. Embora não tendo um resultado satisfatório, com $n = 45$, toda a informação do texto foi capturada, porém acompanhada de ruídos nos arredores.



(a) original (b) $n = 7$



(c) $n = 45$ (d) $n = 135$

Figura 29 - Resultados obtidos para imagem *wedge* usando o Método do Contraste.

Figura	Fração dos pixels pretos
b ($n = 7$)	0,39
c ($n = 15$)	0,4
d ($n = 45$)	0,43
e ($n = 135$)	0,42

Tabela 19 - Fração dos pixels pretos dos resultados para imagem *wedge*.

Por fim, para a imagem *wedge*, tal método produziu um resultado razoável para $n = 135$, porém ainda considerando parte do ruído da luminosidade produzindo sombra. Para os demais valores de n, era possível identificar o "T" por sua borda, porém o objeto não era evidenciado.

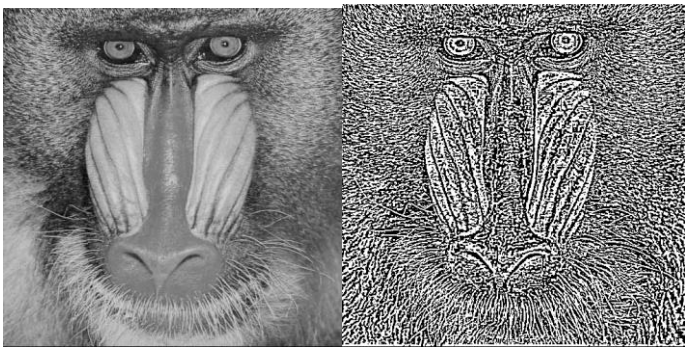
3.7 Método da Média

O objetivo principal deste método era, para cada pixel (x, y), considerar-se uma vizinhança $(n\ x\ n)$ centrada em (x, y), calculando o limiar como sendo a média entre os vizinhos. Para ter um centro adequado, adotou-se números ímpares para n. A Figura 30 mostra como realizou-se tal método:

```python
img = cv.imread(f'input/{image_name}.pgm', cv.IMREAD_GRAYSCALE)
a = np.array(img)
a = a.astype(int)

n = neighbours # número de vizinhos

padding_dimens = n // 2
a_padded = np.pad(a, [(padding_dimens, padding_dimens), (padding_dimens, padding_dimens)], mode='reflect')

rows = len(a)
collumns = len(a[0])
center = padding_dimens

mask_center = np.ones(n*n).reshape(n,n)
mask_center[center, center] = 0
mask_center = mask_center.astype(bool)

for i in range(rows):
    for j in range(collumns):

        # i:i+n, j:j+n
        neighbour_matrix = a_padded[i:i+n, j:j+n]

        neighbour_matrix_center_masked = neighbour_matrix[mask_center]
        neighbour_mean = np.mean(neighbour_matrix_center_masked)

        T_limiar = neighbour_mean
        expression = a[i,j] < T_limiar if invert_background else a[i,j] > T_limiar

        # i, j
        a[i,j] = 255 if expression else 0
```

Figura 30 - Implementação do Método da Média

Para reproduzir tal método, foi necessário adicionar uma borda, escolhida como tipo *reflect* (*padding* - numpy.pad) para atender ao formato da matriz de vizinhos $(n \; x \; n)$ sendo o seu centro cada pixel da imagem. Além disso, criou-se uma máscara para não considerar o centro nos cálculos da média e desvio padrão. Em seguida, percorreu-se cada pixel, selecionando a matriz de vizinhança do pixel; para achar média (numpy.mean) sem considerar o centro, utilizou-se a máscara criada na matriz de vizinhança. Com tais valores, o limiar foi definido como a média calculada. Dadas as imagens citadas em “Materiais e Métodos”, optou-se por adotar que o que seria objeto o que fosse maior que o limiar T, para manter *invert_background=False* como padrão.



(a) original (b) $n = 7$

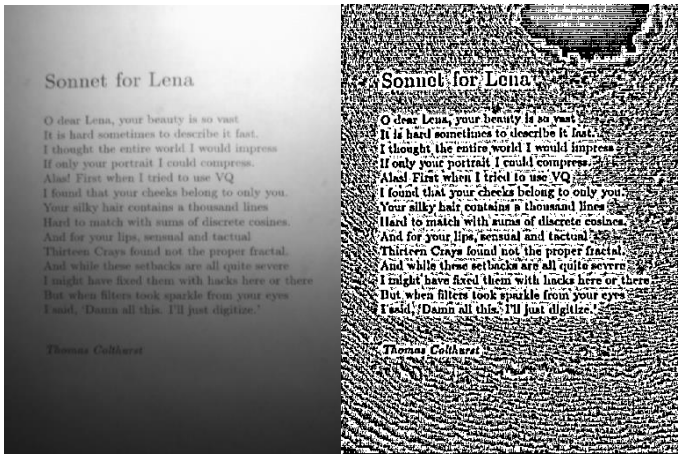
(c) $n = 45$ (d) $n = 135$

Figura 31 - Resultados obtidos para imagem *baboon* usando o Método da Média.

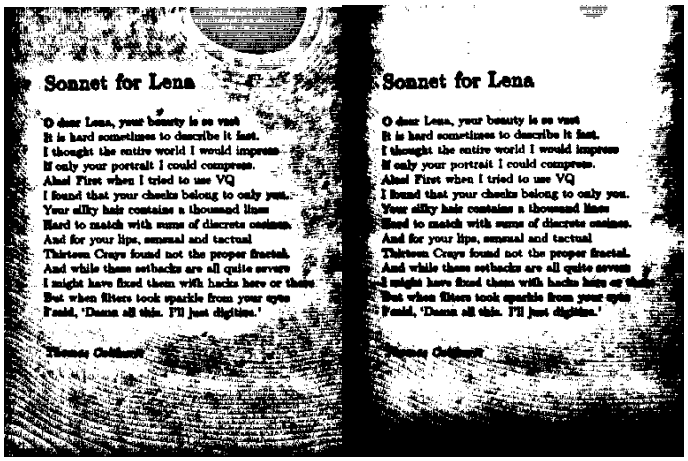
Figura	Fração dos pixels pretos
b ($n = 7$)	0,5
c ($n = 15$)	0,49
d ($n = 45$)	0,48
e ($n = 135$)	0,48

Tabela 20 - Fração dos pixels pretos dos resultados para imagem *baboon*.

Em relação aos resultados para a imagem *baboon*, percebe-se que para vizinhos maiores, obteve-se um resultado melhor, mas adotando $n = 45$ já é obtido um resultado satisfatório.



(a) original (b) $n = 7$



(d) $n = 45$ (e) $n = 135$

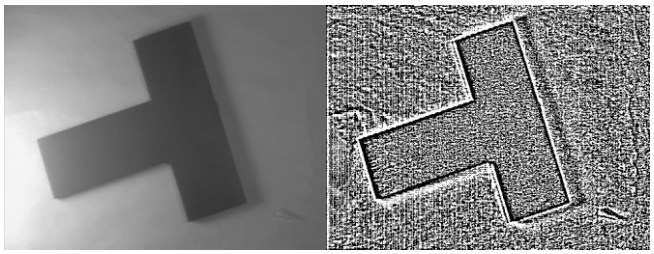
Figura 32 - Resultados obtidos para imagem *sonnet* usando o Método da Média.

Figura	Fração dos pixels pretos
b ($n = 7$)	0,44
c ($n = 15$)	0,43
d ($n = 45$)	0,4
e ($n = 135$)	0,41

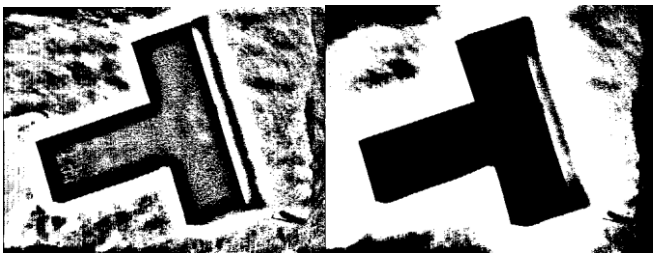
Tabela 21 - Fração dos pixels pretos dos resultados para imagem *sonnet*.

Novamente, para a imagem *sonnet*, obteve-se resultados não muito satisfatórios, mesmo para vizinhos maiores que lidaram melhor com os ruídos.

Embora não tendo um resultado adequado, com $n = 45$, toda a informação do texto foi capturada, porém acompanhada de ruídos nos arredores.



(a) original (b) $n = 7$



(d) $n = 45$ (e) $n = 135$

Figura 33 - Resultados obtidos para imagem *wedge* usando o Método da Média.

Figura	Fração dos pixels pretos
b ($n = 7$)	0,49
c ($n = 15$)	0,48
d ($n = 45$)	0,48
e ($n = 135$)	0,44

Tabela 22 - Fração dos pixels pretos dos resultados para imagem *wedge*.

E, para a imagem *wedge*, tal método produziu um resultado razoável para $n = 135$, porém ainda considerando parte do ruído da luminosidade. Para os demais valores de n, era possível identificar o “T” por sua borda porém o objeto não era evidenciado.

3.8 Método da Mediana

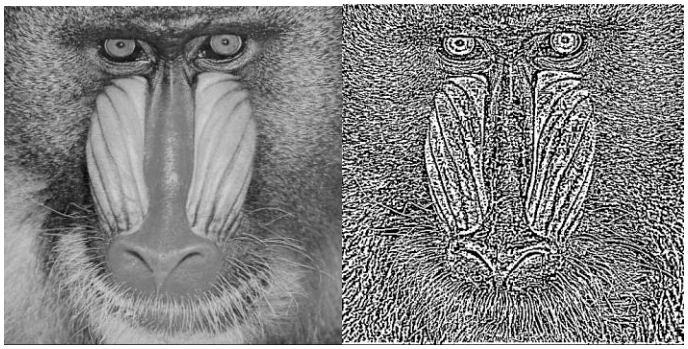
Por fim, esse método calcula o limiar como sendo a mediana entre os vizinhos, ou seja, para cada pixel (x, y), considera-se uma vizinhança $(n\ x\ n)$ centrada em (x, y). Para ter um centro adequado, adotou-se números ímpares para n. A Figura 34 mostra como realizou-se tal método:

```python
img = cv.imread(f'input/{image_name}.pgm', cv.IMREAD_GRAYSCALE)
a = np.array(img)
a = a.astype(int)

n = neighbours # número de vizinhos
padding_dimens = n // 2
a_padded = np.pad(a, [(padding_dimens, padding_dimens), (padding_dimens, padding_dimens)], mode='reflect')

rows = len(a)
collumns = len(a[0])
center = padding_dimens

mask_center = np.ones(n*n).reshape(n,n)
mask_center[center, center] = 0
mask_center = mask_center.astype(bool)

for i in range(rows):
    for j in range(collumns):

        # i:i+n, j:j+n
        neighbour_matrix = a_padded[i:i+n, j:j+n]

        neighbour_matrix_center_masked = neighbour_matrix[mask_center]
        neighbour_median = np.median(neighbour_matrix_center_masked)

        T_limiar = neighbour_median
        expression = a[i,j] < T_limiar if invert_background else a[i,j] > T_limiar

        # i, j
        a[i,j] = 255 if expression else 0
```

Figura 34 - Implementação do Método da Mediana

Para reproduzir tal método, foi necessário adicionar uma borda, escolhida como tipo *reflect* (*padding* - numpy.pad) para atender ao formato da matriz de vizinhos $(n\ x\ n)$ sendo o seu centro cada pixel da imagem. Além disso, criou-se uma máscara para não considerar o centro nos cálculos da média e desvio padrão. Em seguida, percorreu-se cada pixel, selecionando a matriz de vizinhança do pixel; para achar mediana (numpy.median) sem considerar o centro, utilizou-se a máscara criada na matriz de vizinhança. Com tais valores, o limiar foi definido como a mediana calculada. Dadas as imagens citadas em

“Materiais e Métodos”, optou-se por adotar que o que seria objeto o que fosse maior que o limiar T, para manter *invert_background=False* como padrão.



(a) original (b) $n = 7$

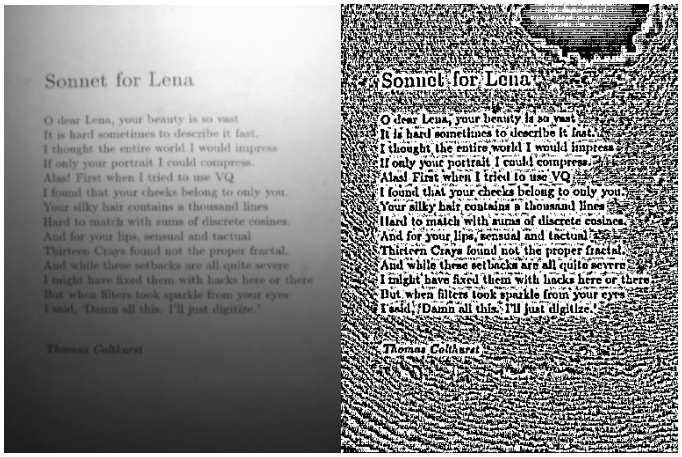
(d) $n = 45$ (e) $n = 135$

Figura 35 - Resultados obtidos para imagem *baboon* usando o Método da Mediana.

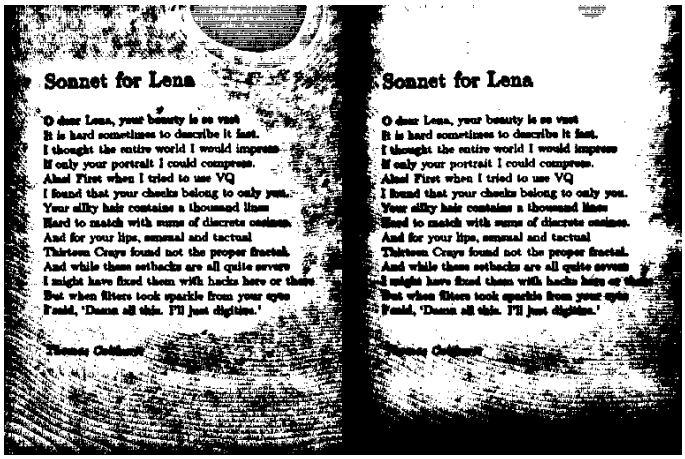
Figura	Fração dos pixels pretos
b ($n = 7$)	0,51
c ($n = 15$)	0,51
d ($n = 45$)	0,5
e ($n = 135$)	0,5

Tabela 23 - Fração dos pixels pretos dos resultados para imagem *baboon*.

Em relação aos resultados para a imagem *baboon*, percebe-se que para vizinhos maiores, obteve-se um resultado melhor, mas é interessante ressaltar que, visualmente, o melhor deles foi com $n = 45$, ou seja, não necessitando aumentar deveras a quantidade de vizinhos para atingir bons resultados.



(a) original (b) $n = 7$



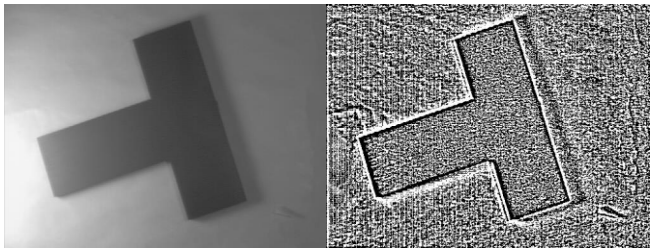
(d) $n = 45$ (e) $n = 135$

Figura 36 - Resultados obtidos para imagem *sonnet* usando o Método da Mediana.

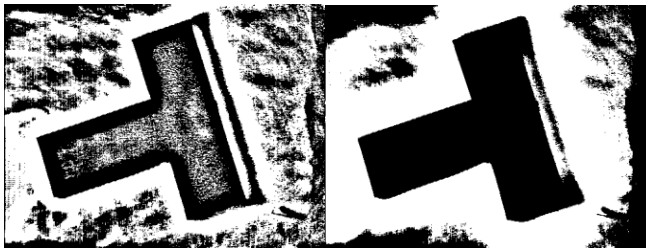
Ainda, para a imagem *sonnet*, obteve-se resultados não muito satisfatórios, mesmo para vizinhos grandes como $n = 135$. Embora não tendo um resultado adequado, com $n = 45$, toda a informação do texto foi capturada, porém acompanhada de ruídos nos arredores, mas mais atenuados do que com $n = 135$.

Figura	Fração dos pixels pretos
b ($n = 7$)	0,78
c ($n = 15$)	0,72
d ($n = 45$)	0,62
e ($n = 135$)	0,45

Tabela 24 - Fração dos pixels pretos dos resultados para imagem *sonnet*.



(a) original (b) $n = 7$



(d) $n = 45$ (e) $n = 135$

Figura 37 - Resultados obtidos para imagem *wedge* usando o Método da Mediana.

Figura	Fração dos pixels pretos
b ($n = 7$)	0,69
c ($n = 15$)	0,68
d ($n = 45$)	0,62
e ($n = 135$)	0,5

Tabela 25 - Fração dos pixels pretos dos resultados para imagem *wedge*.

Por fim, para a imagem *wedge*, tal método produziu um resultado razoável para $n = 135$, porém ainda considerando parte do ruído. Para os demais valores de n, era possível identificar o “T” por sua borda, porém o objeto não era evidenciado.

4. Conclusão

Dessa forma, com a implementação dos métodos propostos, foi possível realizar tratamentos para obter melhores, ou resultados mais adequados que a versão original, a qual muitas vezes está sob a influência de um ruído por exemplo (como a sombra, excesso ou falta de luminosidade).

Em linhas gerais, a imagem *baboon* teve um resultado adequado em todos métodos, de acordo com os parâmetros apresentados nos resultados, porém a princípio era uma imagem sem ruídos trazidos na entrada.

Por outro lado, a imagem *sonnet* tinha informações como caracteres e pouca luminosidade, e mesmo alguns métodos não efetuando um tratamento eficaz, os de Sauvola/Pietaksinen e Phansalskar/More/Sabale exibiram bons resultados com um número de vizinhos baixo. Mesmo exigindo um processamento maior por ter que calcular a média, desvio padrão e operação exponencial (no último método citado), com número de vizinhos baixos não se porta como tão custoso.

Por fim, no caso da imagem *wedge*, havia o problema de sombra, de onde o foco de iluminação estava e por conta disso, em alguns métodos como o Global, teve grande influência no resultado, borrando parte da imagem com preto. Porém, em métodos que exigiam menos computacionalmente comparado aos posteriores como o de Bernsen, atingiu-se um resultado razoável, acompanhado de parte do ruído, mas sendo possível interpretar como um sólido no formato “T”. Também nesse caso, os métodos de Sauvola/Pietaksinen e Phansalskar/More/Sabale conseguiram atenuar boa parte dos ruídos, porém ao custo de necessitar de uma vizinhança de 135x135, ainda tendo uma falta de pixels pretos na parte superior do “T”.