COM 3105

Processamento Digital de Imagens

LN06 - 2023

Operações Pontuais | Parte I

Prof. Eng. Marcos Cordeiro d'Ornellas, Ph.D.

Revisado emL 26.03.2023.

1 Operações pontuais

As operações pontuais realizam uma modificação dos valores de pixel sem alterar o tamanho, a geometria ou a estrutura local da imagem. Cada novo valor de pixel $a' = I'(u, v)$ depende exclusivamente do valor anterior $a' = I'(u, v)$ na mesma posição e é, portanto, independente de qualquer outro valor de pixel, em particular de qualquer um dos pixels vizinhos. Os valores de pixel originais são mapeados para os novos valores por uma função $f(a)$,

$$\begin{aligned} a' &\leftarrow f(a) \\ I'(u, v) &\leftarrow f(I(u, v)) \end{aligned} \tag{1}$$

para cada posição da imagem (u, v). Se a função $f()$ for independente das coordenadas da imagem (ou seja, as mesmas em toda a imagem), a operação é chamada de “global” ou “homogênea”. Exemplos típicos de operações pontuais homogêneas incluem, entre outros,

- modificar o brilho ou contraste da imagem;
- aplicar transformações de intensidade arbitrárias;
- limiarização global;
- correção gama;
- transformações de cores.

Em contraste, a função de mapeamento $g()$ para uma operação de ponto não homogênea também levaria em consideração a coordenada da imagem atual (u, v); isto é,

$$\begin{aligned} a' &\leftarrow g(a, u, v) \\ I'(u, v) &\leftarrow g(I(u, v), u, v) \end{aligned} \tag{2}$$

Uma operação não homogênea típica é o ajuste local de contraste ou brilho usado, por exemplo, para compensar a iluminação irregular durante a aquisição da imagem.

1.1 Invertendo a imagem

Inverter uma imagem de intensidade é uma operação de ponto simples que inverte a ordem dos valores de pixel (multiplicando por -1) e adiciona um valor constante para mapear o resultado para a faixa admissível novamente. Assim, para um valor de pixel $a = I(u, v)$ no intervalo $[0, a_{max}]$, a

operação de ponto correspondente é

$$f_{invert}(a) = -a + a_{max} = a_{max} - a \tag{3}$$

A inversão de uma imagem em tons de cinza de 8 bits com $a_{max} = 255$ é fácil de entender. Observe que, neste caso, nenhuma fixação é necessária porque a função sempre mapeia para a faixa original de valores

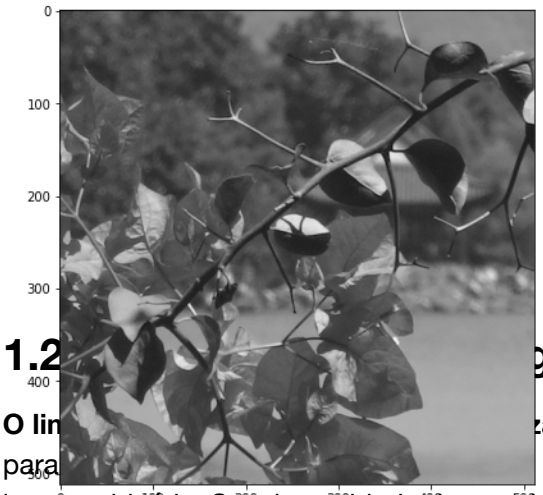
In [1]:

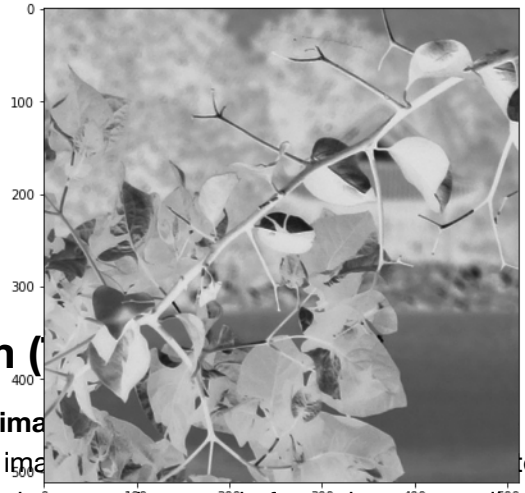
```
import cv2
import numpy as np
import matplotlib.pyplot as plt

plt.figure(figsize=(15,15))

img1 = cv2.imread('image_folder/westlake512.jpg', cv2.IMREAD_GRAYSCALE)

img2 = 255 - img1

# img2 = img2 + 40

plt.subplot(2,2,1)
plt.imshow(img1,cmap='gray')

plt.subplot(2,2,2)
plt.imshow(img2,cmap='gray')

plt.subplot(2,2,3)
plt.grid(True)

plt.hist(img1.ravel(),256,[0,256])
plt.xlim([0,256])

plt.subplot(2,2,4)
plt.grid(True)

plt.hist(img2.ravel(),256,[0,256])
plt.xlim([0,256])
```

executed in 1.94s, finished 19:13:27 2023-03-26

Out[1]:

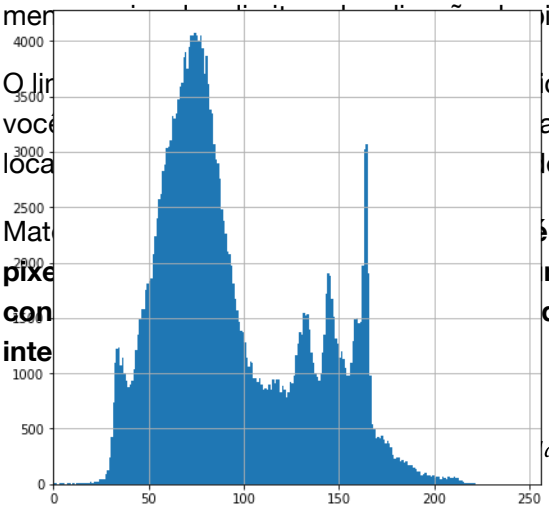
```
(0.0, 256.0)
```

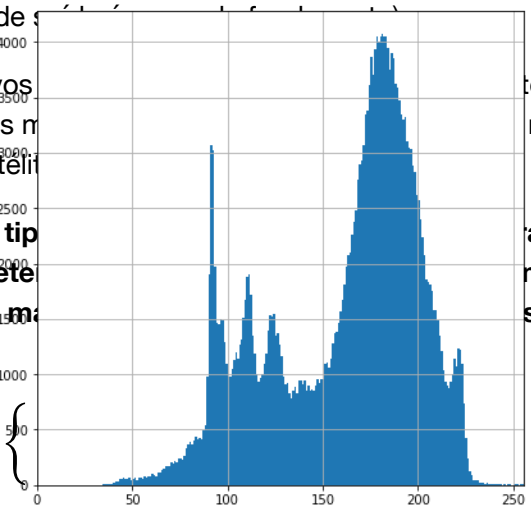




1.2 gem (

O lin **zar a ima** **pixel.** A entrada para uma ima te. A saída é uma imagem binária. Se a intensidade de um pixel na imagem de entrada for maior que um limite, o pixel de saída correspondente é marcado como branco (primeiro plano) e se a intensidade do pixel de entrada for men ixel de s





O lir icativos to. Por exemplo, voc agens m mamografia ou loca le satélit

Mat **é um tip** **a os valores de pixe** **m dete** **mente é con** **d}(a) ma** **s valores de inte**

$$d = \begin{cases} \\ \end{cases} \tag{4}$$

com $0 < a_{th} \leq a_{max}$. Uma aplicação comum é binarizar uma imagem de intensidade com os valores $a_0 = 0$ e $a_1 = 1$.

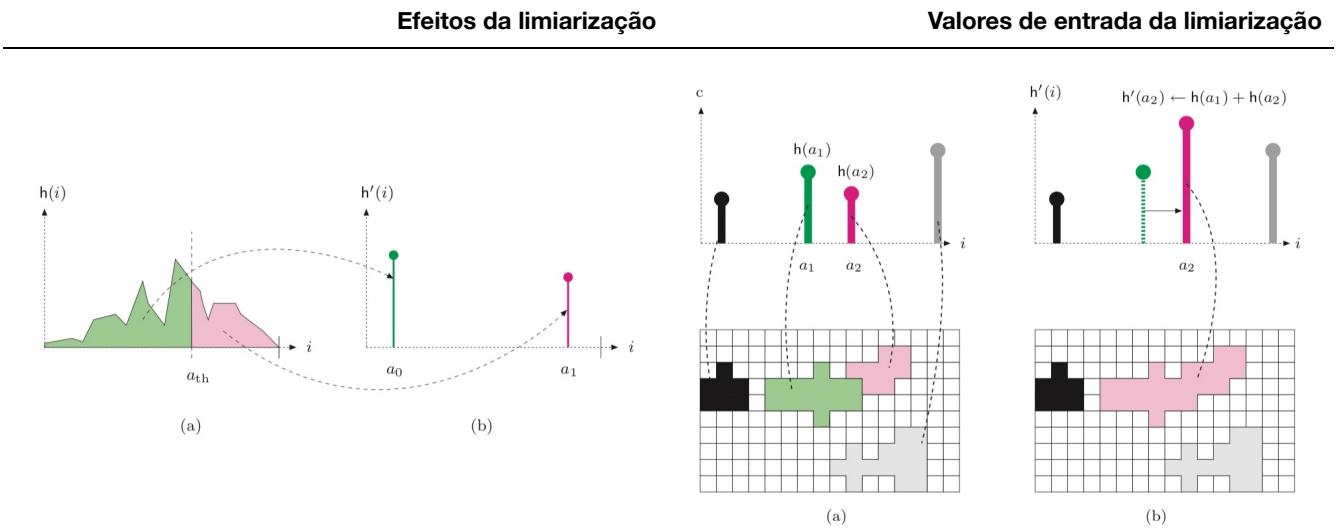
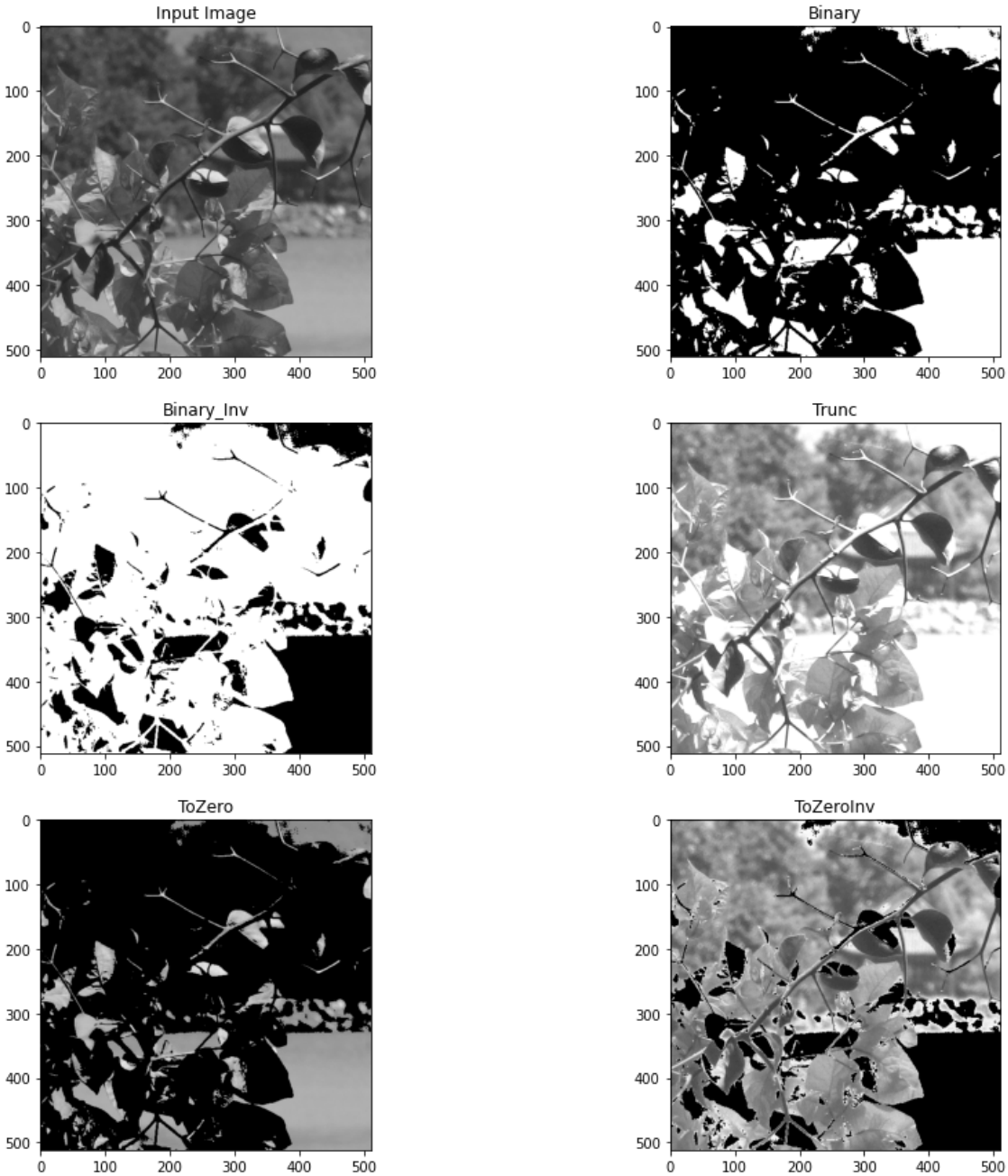


Figura 1: O processo de limiarização da imagem.

In [2]:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

plt.figure(figsize=(15,15))

img = cv2.imread('image_folder/westlake512.jpg', cv2.IMREAD_GRAYSCALE)

ret,thresh1 = cv2.threshold(img,127,255,cv2.THRESH_BINARY)
ret,thresh2 = cv2.threshold(img,127,255,cv2.THRESH_BINARY_INV)
ret,thresh3 = cv2.threshold(img,127,255,cv2.THRESH_TRUNC)
ret,thresh4 = cv2.threshold(img,127,255,cv2.THRESH_TOZERO)
ret,thresh5 = cv2.threshold(img,127,255,cv2.THRESH_TOZERO_INV)

titles = ['Input Image','Binary','Binary_Inv','Trunc','ToZero','ToZeroInv']

images = [img, thresh1, thresh2, thresh3, thresh4, thresh5]

for i in range(6):
    plt.subplot(3,2,i+1),plt.imshow(images[i],'gray')
    plt.title(titles[i])
    #plt.xticks([]),plt.yticks([])
```

executed in 666ms, finished 19:13:28 2023-03-26



1.3 Limiarização adaptativa e dinâmica

A limiarização adaptativa é uma forma de limite que leva em consideração as variações espaciais na iluminação. **Para imagens com iluminação pobre e não uniforme, é necessário um limiar adaptativo para separar os objetos de interesse do fundo. A limiarização adaptativa é usada para separar objetos de imagem de primeiro plano desejáveis do fundo com base na diferença nas intensidades de pixel de cada região.** A desvantagem desse método é que ele é oneroso em termos computacionais e, portanto, não é apropriado para aplicativos de tempo real.

> Um limiar é usado para segmentar uma imagem, definindo todos os pixels cujos valores de intensidade estão acima de um limite para um valor de primeiro plano e todos os pixels restantes para um valor de fundo.

Enquanto o operador de limite convencional usa um limite global para todos os pixels, **o limite adaptativo altera o limite dinamicamente na imagem.** Esta versão mais sofisticada de limiarização pode acomodar mudanças nas condições de iluminação na imagem, por exemplo, aqueles que ocorrem como resultado de um forte gradiente de iluminação ou sombras.

O limiar adaptativo normalmente usa uma imagem em tons de cinza ou colorida como entrada e, na implementação mais simples, gera uma imagem binária que representa a segmentação. Para cada pixel da imagem, um limite deve ser calculado. Se o valor do pixel estiver abaixo do limite, ele é definido como o valor do fundo, caso contrário, ele assume o valor do primeiro plano.

Existem duas abordagens principais para encontrar o limiar: **a abordagem de Chow e Kaneko** e o **limiar local.** A suposição por trás de ambos os métodos é que regiões de imagem menores são mais propensas a ter iluminação aproximadamente uniforme, sendo, portanto, mais adequadas para limiar. **Chow e Kaneko dividem uma imagem em uma matriz de subimagens sobrepostas e, em seguida, encontram o limite ideal para cada subimagem investigando seu histograma.** O limite para cada pixel é encontrado interpolando os resultados das subimagens. A desvantagem desse método é que ele é caro em termos computacionais e, portanto, não é apropriado para aplicativos de tempo real.

Como o **limiar global**, o **limiar adaptativo é usado para separar objetos de imagem de primeiro plano desejáveis do fundo com base na diferença nas intensidades de pixel de cada região.** O limiar global usa um limite fixo para todos os pixels na imagem e, portanto, funciona apenas se o histograma de intensidade da imagem de entrada contiver picos nitidamente separados correspondendo ao(s) assunto(s) e fundo(s) desejados. Portanto, não pode lidar com imagens que contenham, por exemplo, um forte gradiente de iluminação.

Na seção anterior, **usamos um valor global como valor limite**. Mas pode não ser bom em todas as condições em que a imagem tem diferentes condições de iluminação em diferentes áreas. Nesse caso, optamos pelo limiar adaptativo. Neste, o algoritmo calcula o limite para pequenas regiões da imagem. Portanto, obtemos limites diferentes para regiões diferentes da mesma imagem e isso nos dá melhores resultados para imagens com iluminação variável.

Há três parâmetros de entrada e apenas um argumento de saída.

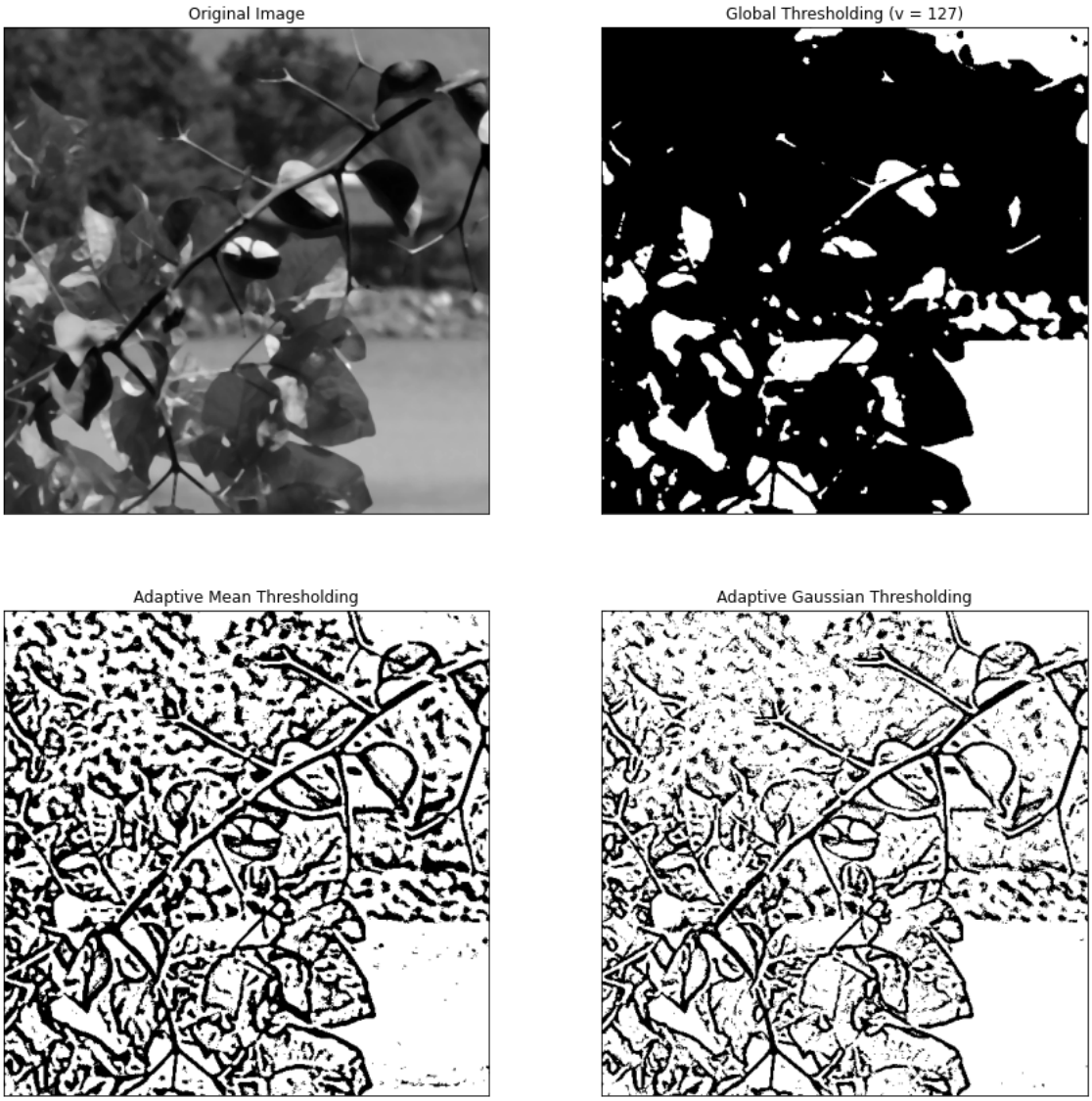
- Método adaptativo - decide como o valor de limiar é calculado.
  - cv2.ADAPTIVE_THRESH_MEAN_C: o valor limite é a média da área da vizinhança.
  - cv2.ADAPTIVE_THRESH_GAUSSIAN_C: o valor limite é a soma ponderada dos valores da vizinhança onde os pesos são uma janela gaussiana.
- Tamanho do Bloco - Decide o tamanho da área do bairro.
- C - É apenas uma constante que é subtraída da média ou média ponderada calculada.

O código abaixo compara o limiar global e o limiar adaptativo para uma imagem com iluminação variável:

In [3]:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

plt.figure(figsize=(15,15))

img = cv2.imread('image_folder/westlake512.jpg', cv2.IMREAD_GRAYSCALE)
img = cv2.medianBlur(img,5)

ret,th1 = cv2.threshold(img,127,255,cv2.THRESH_BINARY)

th2 = cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_MEAN_C,\
            cv2.THRESH_BINARY,11,2)
th3 = cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,\
            cv2.THRESH_BINARY,11,2)

titles = ['Original Image', 'Global Thresholding (v = 127)',
            'Adaptive Mean Thresholding', 'Adaptive Gaussian Thresholding']

images = [img, th1, th2, th3]

for i in range(4):
    plt.subplot(2,2,i+1),plt.imshow(images[i],'gray')
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])
plt.show()
```

executed in 479ms, finished 19:13:28 2023-03-26



Um problema com o limiar simples é que você **precisa especificar manualmente o valor do limite.** Podemos verificar manualmente o quão bom é um limite tentando valores diferentes, mas é tedioso e pode falhar no mundo real.

Portanto, precisamos encontrar uma maneira de determinar automaticamente o limite. A **técnica do Otsu** em homenagem a seu criador Nobuyuki Otsu é uma alternativa para obter o limiar automaticamente.

1.4 O Método de Otsu

Os algoritmos de limite global automático geralmente têm as seguintes etapas:

1. Processe a imagem de entrada
2. Obtenha o histograma da imagem (distribuição de pixels)
3. Calcule o valor limite T
4. Substitua os pixels da imagem em branco nessas regiões, onde a saturação é maior que T, e em preto nos casos opostos.

Normalmente, algoritmos diferem na etapa 3.

Vamos entender a ideia por trás da abordagem de Otsu. **O método processa o histograma da imagem, segmentando os objetos por meio da minimização da variância em cada uma das classes. Normalmente, essa técnica produz os resultados apropriados para imagens bimodais. O**

histograma dessa imagem contém dois picos claramente expressos, que representam diferentes faixas de valores de intensidade.

A ideia central é **separar o histograma da imagem em dois grupos com um limite definido como**

Toda a equação de cálculo pode ser descrita como: $\sigma_w^2(t) = w_1(t)\sigma_1^2(t) + w_2(t)\sigma_2^2(t)$, onde $w_1(t), w_2(t)$ são as probabilidades das duas classes divididas por um limite t, cujo valor está dentro do intervalo de 0 a 255, inclusive.

Há duas opções para encontrar o limite. O primeiro é minimizar a variância dentro da classe definida acima $\sigma_w^2(t)$, o segundo é maximizar a variância entre as classes usando a expressão abaixo: $\sigma_b^2(t) = w_1(t)w_2(t)[\mu_1(t) - \mu_2(t)]^2$, onde μ_i é uma média da classe i.

A probabilidade P é calculada para cada valor de pixel em dois clusters separados C_1, C_2 usando as funções de probabilidade de cluster expressas como:

$w_1(t) = \sum_{i=1}^{t} P(i),$

$w_2(t) = \sum_{i=t+1}^{I} P(i)$

Ressalta-se que a imagem pode ser apresentada como função intensidade $f(x, y)$, cujos valores estão em nível de cinza. A quantidade de pixels com um nível de cinza especificado i é denotado por i. O número total de pixels na imagem é n.

Assim, a probabilidade de ocorrência de nível de cinza i é:

$P(i) = \frac{n_i}{n}.$

Os valores de intensidade de pixel para C_1 estão em $[1, t]$ e para C_2 estão em $[t + 1, I]$, onde I é o valor máximo de pixel (255).

A próxima fase é obter as médias para C_1, C_2, que são denotadas por $\mu_1(t)$, $\mu_2(t)$ apropriadamente:

$\mu_1(t) = \sum_{i=1}^{t} \frac{iP(i)}{w_1(t)},$

$\mu_2(t) = \sum_{i=t+1}^{I} \frac{iP(i)}{w_2(t)}$

Agora, vamos lembrar a equação acima da variância ponderada dentro das classes. Encontraremos o resto de seus componentes (σ_1^2, σ_2^2) misturando todos os ingredientes obtidos acima:

$sigma_1^2(t) = \sum_{i=1}^{t} [i - \mu_1(t)]^2 \frac{P(i)}{w_1(t)},$

$\sigma_2^2(t) = \sum_{i=t+1}^{I} [i - \mu_2(t)]^2 \frac{P(i)}{w_2(t)}.$

Deve-se notar que, se o limite fosse escolhido incorretamente, a variância de alguma classe seria grande. Para obter a variância total, simplesmente precisamos resumir as variâncias dentro da classe e entre as classes: $\sigma_T^2 = \sigma_w^2(t) + \sigma_b^2(t)$, onde $\sigma_b^2(t) = w_1(t)w_2(t)[\mu_1(t) - \mu_2(t)]^2$. A variância total da imagem (σ_T^2) não depende do limite.

Assim, **o pipeline do algoritmo geral para a opção de maximização de variância entre classes** pode ser representado da seguinte maneira:

1. calcular o histograma e as probabilidades do nível de intensidade
2. inicializar $w_i(0), \mu_i(0)$
3. iterar sobre os limites possíveis: $t = 0, \dots, max\_intensity$

   A. atualizar os valores de w_i, μ_i, onde w_i é uma probabilidade e μ_i é uma média da classe i
   B. calcula o valor de variância entre as classes $\sigma_b^2(t)$
4. o limite final é o valor máximo $\sigma_b^2(t)$

Apesar do método de Otsu ter sido criado em 1979, ele ainda constitui a base de algumas soluções complexas

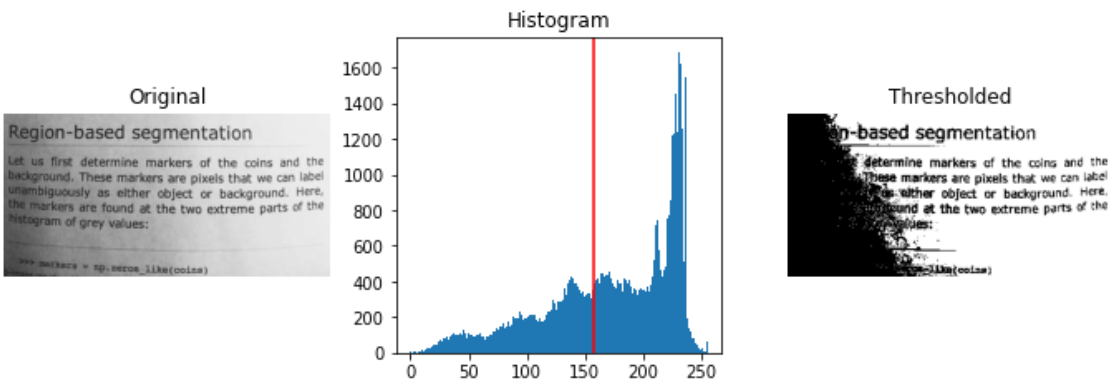
Ilustramos como aplicar o método de Otsu. Este método calcula um limite "ideal" (marcado por uma linha vermelha no histograma abaixo) maximizando a variação entre duas classes de pixels, que são separadas pelo limite. De forma equivalente, esse limite minimiza a variância intraclasse.

In [4]:

```
import matplotlib.pyplot as plt
from skimage import data
from skimage.filters import threshold_otsu

image = data.page()
thresh = threshold_otsu(image)
print(thresh)
binary = image > thresh


fig, axes = plt.subplots(ncols=3, figsize=(12, 3.5))
ax = axes.ravel()
ax[0] = plt.subplot(1, 3, 1)
ax[1] = plt.subplot(1, 3, 2)
ax[2] = plt.subplot(1, 3, 3, sharex=ax[0], sharey=ax[0])

ax[0].imshow(image, cmap=plt.cm.gray)
ax[0].set_title('Original')
ax[0].axis('off')

ax[1].hist(image.ravel(), bins=256)
ax[1].set_title('Histogram')
ax[1].axvline(thresh, color='r')

ax[2].imshow(binary, cmap=plt.cm.gray)
ax[2].set_title('Thresholded')
ax[2].axis('off')

plt.show()
```

executed in 1.69s, finished 19:13:30 2023-03-26

```
157
```



1.5 O Método de Otsu modificado (Multi-Otsu)

O método multi-Otsu é um algoritmo de limiar que é usado para separar os pixels de uma imagem de entrada em várias classes diferentes, cada uma obtida de acordo com a intensidade dos níveis de cinza na imagem.

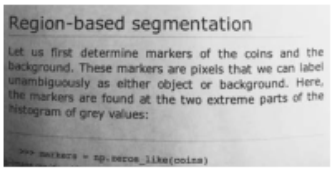
Multi-Otsu calcula vários limites, determinados pelo número de classes desejadas. O número padrão de classes é 3: para obter três classes, o algoritmo retorna dois valores de limite. Eles são representados por uma linha vermelha no histograma a seguir:

In [5]:

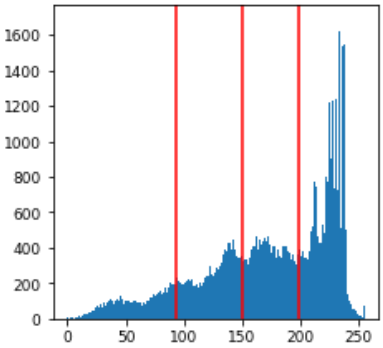
```
import matplotlib
import matplotlib.pyplot as plt
import numpy as np

from skimage import data
from skimage.filters import threshold_multiotsu

# Setting the font size for all plots.
matplotlib.rcParams['font.size'] = 9

# The input image.
image = data.page()

# Applying multi-Otsu threshold for the default value, generating
# three classes.
# thresholds = threshold_multiotsu(image,3)
# https://scikit-image.org/docs/stable/api/skimage.filters.html#skimage.filt

thresholds = threshold_multiotsu(image,4)

# Using the threshold values, we generate the three regions.
regions = np.digitize(image, bins=thresholds)

fig, ax = plt.subplots(nrows=1, ncols=3, figsize=(12, 3.5))

# Plotting the original image.
ax[0].imshow(image, cmap='gray')
ax[0].set_title('Original')
ax[0].axis('off')

# Plotting the histogram and the two thresholds obtained from
# multi-Otsu.
ax[1].hist(image.ravel(), bins=255)
ax[1].set_title('Histogram')
for thresh in thresholds:
    ax[1].axvline(thresh, color='r')

# Plotting the Multi Otsu result.
ax[2].imshow(regions, cmap='gray')
ax[2].set_title('Multi-Otsu result')
ax[2].axis('off')

plt.subplots_adjust()

plt.show()
```

executed in 649ms, finished 19:13:30 2023-03-26

Original



Histogram



Multi-Otsu result