Trabalho 03 de Introdução ao Processamento de Imagem Digital

Fabio Fogliarini Brolesi <brolesi@gmail.com> - RA 023718

30 de maio de 2021

Sumário

1 Transformação de cores

1.1 Monocromático

Para a transformação de cores para o monocromático, utilizamos a função `rgb2gray`[1] do módulo `color` da biblioteca scikit-image[2].

A documentação informa que o único argumento a ser passado para o método é a imagem no formato RGB, e o método então "computa a luminância de uma imagem RGB"[1].

```
1 grayscale = rgb2gray(image)
```

[1] tradução livre

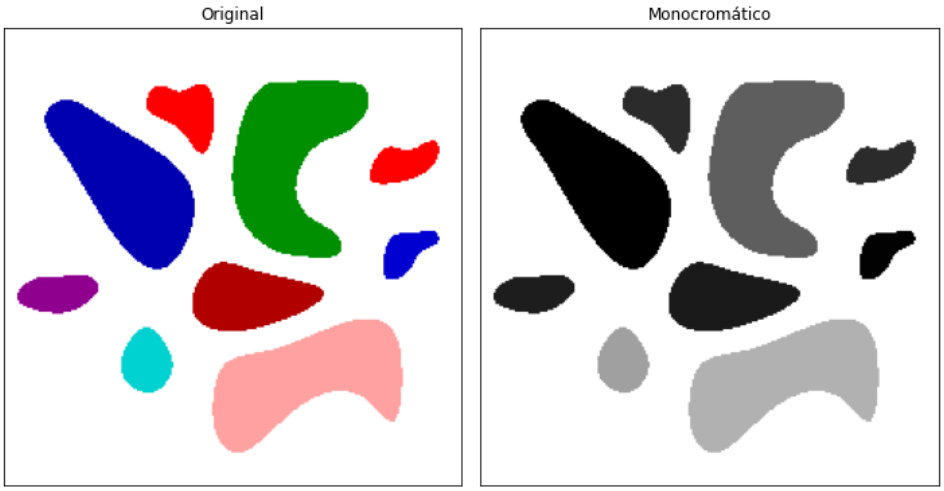

Figura 1: imagem original (esquerda) e a transformada monocromática a partir da original (direita).

O resultado típico é a figura 1.

1.2 Contornos

Para a etapa de contorno, partimos da imagem monocromática já citada, e criamos uma imagem com dimensões iguais a da imagem original (largura e altura).

A partir dos contornos dos objetos identificados, realizamos a impressão de cada um deles na nova imagem, criada a partir das dimensões originais. Para tanto, utilizamos o método `find_contours`[3] do módulo `measure` do pacote `scikit-image`.

```
1 grayscale = rgb2gray(image)
2 contours = measure.find_contours(grayscale, 0.8)
```

O segundo argumento, com o valor de `0.8` é referente ao parâmetro `level` que é um tipo `float`, que a documentação descreve como "Valor ao longo do qual será encontrado contornos na matriz. Por padrão, o valor é computado como `(max(image) + min(image)) / 2`"[2].

Testemos sem o argumento, e verificamos que ele não suportava todos os contornos das imagens dadas, então decidimos por realizar testes variando estes parâmetros, adotando o `0.8` para o caso deste trabalho.

Um exemplo de resultado esperado é o que segue na figura 2.

[2]tradução livre

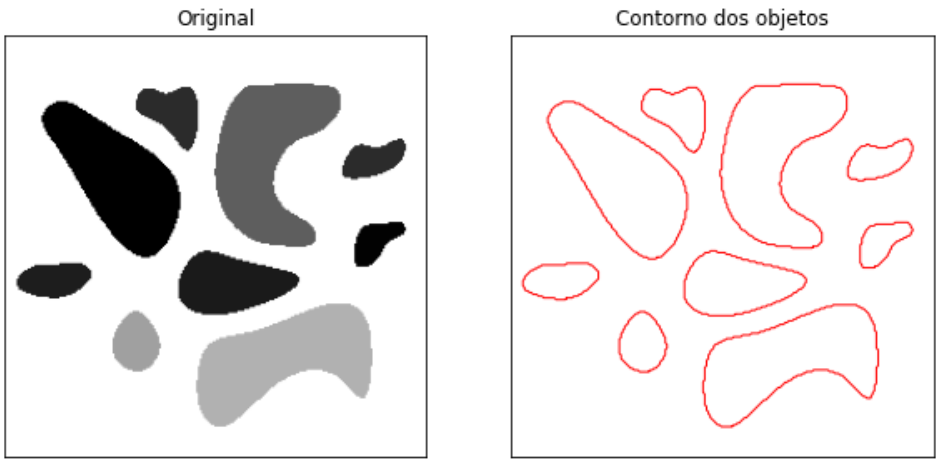

Figura 2: imagem monocromática a partir da imagem original (esquerda) e a transformada com contornos marcados a partir da original (direita).

2 Extração de propriedades

Para todos os itens abaixo, utilizamos inicialmente o método `label` do módulo `measure` do pacote `scikit-image`. Este método recebe como argumentos: `grayscale, background=1, connectivity=grayscale.ndim`.

O primeiro, é a imagem transformada pelo método `rgb2gray` já citado na subseção Monocromático.

O parâmetro `background` é um parâmetro do tipo inteiro, opcional, e que "Considera todos os pixels com este valor como pixels de fundo e rotula todos eles como 0. Por padrão, valor 0 são considerados pixels de fundo."[3]. No nosso caso, utilizamos apenas imagens com fundo branco e por isso após alguns testes o valor `1` mostrou-se adequado.

O último parâmetro, `connectivity` também é opcional e do tipo inteiro. Ele é a quantidade de saltos ortogonais para considerar, no nosso caso, um pixel como vizinho. Para este caso, explicitamos que o valor a ser considerado é a propriedade `ndim` da imagem em escala de cinza `grayscale`.

Em seguida, utilizamos o método `regionprops`[4] do módulo `measure` do mesmo pacote `scikit-image`.

O código do método `regionprops` está em [5]. No repositório do Github, basicamente há algumas definições no método e a invocação de uma classe `RegionProperties`[6]. Nela encontramos várias propriedades, algumas delas com cache. As subseções abaixo trazem luz ao que cada etapa faz.

O código do trecho que invoca os métodos é o que segue:

[3] tradução livre

```
grayscale = rgb2gray(image)
label_img = measure.label(grayscale, background=1, connectivity
    =grayscale.ndim)
props = measure.regionprops(label_img)
```

2.1 Centróide

No caso do centróide, o local onde fisicamente o centro de massa se concentra, no módulo `measure` ele define como a média das coordenadas da região:

```
def centroid(self):
    return tuple(self.coords.mean(axis=0))
```

O método `measure.regionprops` nos dá dois conjuntos de valores, o `centroid` e o `local_centroid`, o primeiro referente ao centróide relativamente à imagem e o segundo referente ao centróide do objeto considerando o retângulo que envolve este objeto (também chamado *bounding box*).

Para exibir no resultado final, consideramos o valor vindo da propriedade `centroid`.

2.2 Perímetro

O perímetro está como propriedade `perimeter` e é calculado a partir do perímetro com vizinhança 4. Um método de uma bilbioteca `utils`[7] é carregado e invoca o cálculo com este tipo de vizinhança. A referência para este método é a [8] conforme a própria biblioteca do scikit-image coloca.

2.3 Área

Para a área do objeto simplesmente utilizamos o atributo `area` do resultado do método `measure.regionprops`.

2.4 Excentricidade

Para a definição de excentricidade nos apoiamos na figura 3 conforme [10] (adaptado) e temos a equação 1:

$$excentricidade = \frac{A}{B} \tag{1}$$

O método `measure.regionprops` retorna, entre outras, duas propriedades: `major_axis_length` e `minor_axis_length` que são eixos maiores e menores respectivamente, porém, da elipse que tem os mesmos momentos centrais de segunda ordem normalizados do objeto, e não propriamente A

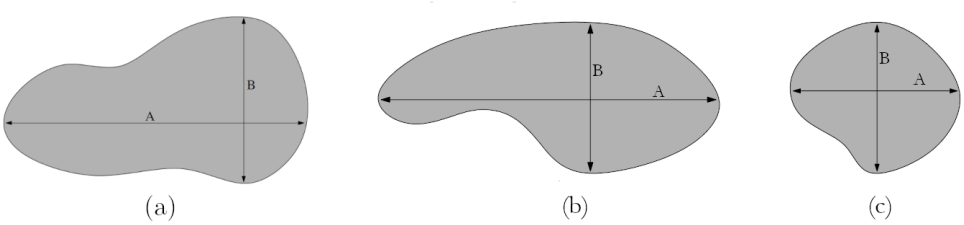

Figura 3: Objetos e suas respectivas dimensões para cálculo de excentricidade [10] (adaptado) - (a) ilustração de eixos maior e menor de um objeto; (b) objeto com excentricidade alta; (c) objeto com excentricidade baixa.

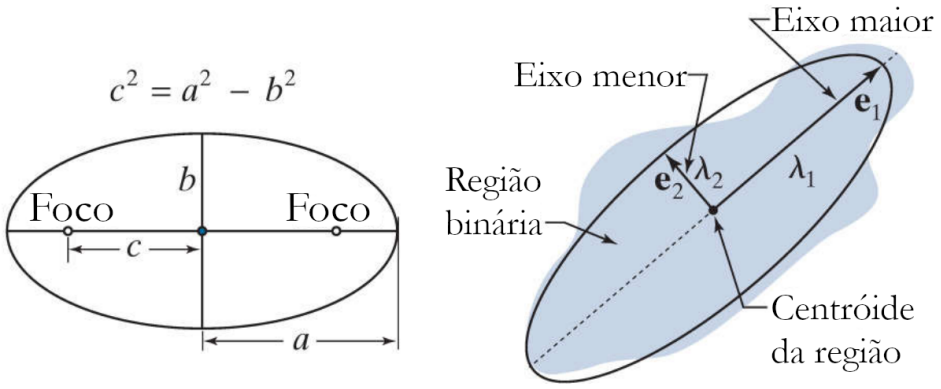

Figura 4: Uma outra forma de calcular a excentricidade conforme [9]. À esquerda, uma elipse na forma padrão. À direita, a aproximação um objeto numa orientação arbitrária conforme [9] (adaptado). e_1, λ_1 e e_2, λ_2 são os autovetores e seus respectivos autovalores da matriz de covariância das coordenadas da região.

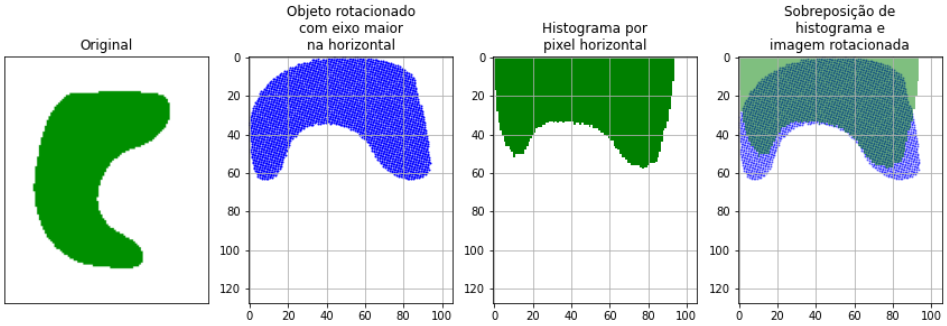

Figura 5: A imagem original, imagem com eixo maior na horizontal, histograma da imagem rotacionada com eixo maior na horizontal, sobreposição de histograma e imagem rotacionada.

como o maior eixo do objeto, e B como maior eixo ortogonal ao eixo anterior, conforme ilustrado na figura 4.

Para o cálculo, usaremos a abordagem de [10].

Conforme [11], o eixo maior são os pontos do maior segmento de reta que pode ser definido dentro do objeto. Eles são computados a partir da avaliação de todos os pixels de borda e encontrando o par com a maior distância, digamos $P_1 = x_1, y_1$ e $P_2 = x_2, y_2$.

Então a dimensão do eixo maior do objeto será dada por:

$$\textit{tamanho do eixo maior} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{2}$$

Para a nossa abordagem, decidimos recuperar o objeto, e rotaciona-lo para que o maior eixo fique na horizontal (com ângulo 0). Para tanto, calculamos a matriz de rotação para o ângulo correto, e realizamos a operação para o cálculo da inversa desta matriz.

Em seguida, já com o eixo maior na horizontal, temos a garantia de que o valor para o eixo menor é no formato $x = y$, com y variando entre 0 e *tamanho do eixo maior*.

Desta forma, decidimos por criar um hisograma da imagem, com a quantidade de bins sendo igual ao tamanho do eixo maior adicionado de 1 unidade. Com isso, é como se projetássemos a imagem para que ela ficasse como que colapsada a partir de um eixo. Com isso, recuperamos o maior valore de y para termos a maior dimensão do eixo ortogonal ao maior eixo da imagem.

Figura 5 ilustra as etapas da abordagem utilizada.

Os valores então, da razão de `major_axis_length` e `minor_axis_length` do resultado do método `measure.regionprops` possuem valores diferentes em relação ao cálculo feito considerando a abordagem acima.

Assim, conforme a abordagem acima, o código que calcula a excentricidade é:

```
def calculate_eccentricity(poly):
    theta = get_angle(poly.coords)
    inverse_matrix = np.linalg.inv(theta)

    rotated_object = np.dot(poly.coords, inverse_matrix)

    y_min = min(rotated_object[:,0])
    x_min = min(rotated_object[:,1])
    x_max = max(rotated_object[:,1])
    if x_min < 0:
        rotated_object[:,1] += -x_min
    if y_min < 0:
        rotated_object[:,0] += -y_min

    y, x = np.histogram(rotated_object[:, 1], bins=int(np.round
    (x_max-x_min + 1)))

    delta_x = (x_max - x_min)
    delta_y = max(y)

    ratio = max(delta_x, delta_y) / min(delta_x, delta_y)

    return ratio
```

2.5 Solidez

A solidez é calculada como:

$$solidez = \frac{área\ do\ objeto\ imagem}{fecho\ convexo\ do\ objeto} \tag{3}$$

Ela indica o quando to fecho convexo é utilizado para compor o objeto. Quanto maior a solidez, maior área dentro do fecho ele ocupa.

Para o caso do trabalho proposto, utilizando as propriedades do objeto calculado, temos então o seguinte:

```
def calculate_solidity(poly):
    return poly.area / poly.convex_area
```

3 Histograma

Para o histograma e a segregação de tamanhos de imagens, executamos as seguintes etapas:

1. Transformamos a imagem original em escala de cinza;
2. Recuperamos os rótulos dela a partir do método `measure.label`;

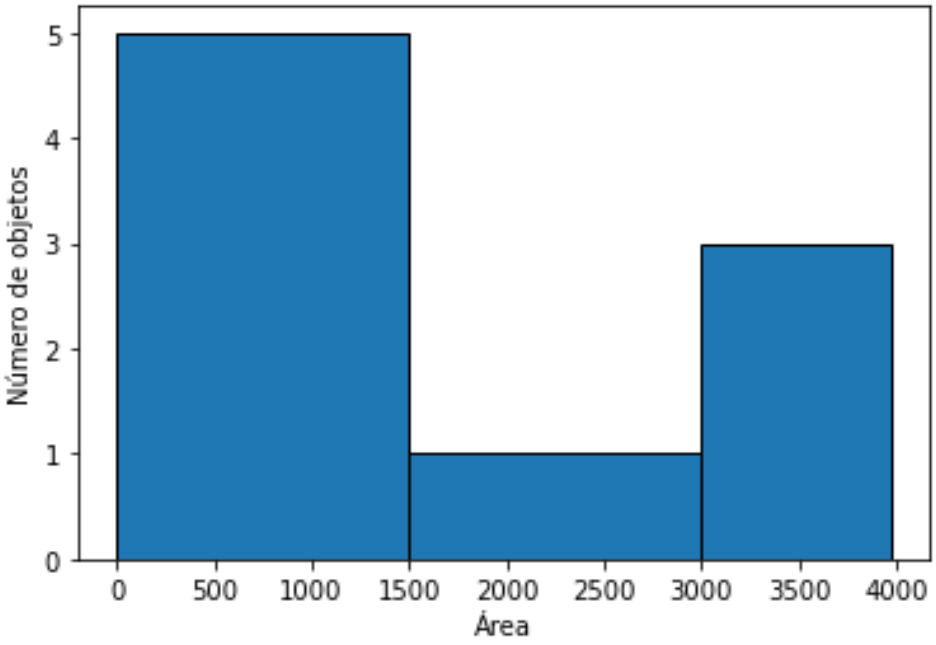

Figura 6: Histograma do resultado de objetos na figura classificados como pequenos, médios e grandes.

3. Recuperamos cada uma das propriedades de cada objeto rotulado com o método `measure.regionprops`;
4. Colocamos cada área de objeto num array nomeado por `all_histogram_areas`;
5. Computamos o maior tamanho de área
6. Criamos cada um dos bins do histograma
7. Realizamos o plot do histograma
8. Com o resultado, temos cada uma das frequências para objetos pequenos, médios e grandes.

A figura 6 ilustra o resultado do cálculo do histograma dos objetos identificados na figura 1.

A figura

4 Código

De forma simplificada, para o resultado de centróides, perímetros, áreas, excentricidade, solidez, executamos este trecho, uma vez a variável `props` já populada:

```
for r, p in enumerate(props, start=0):
  print(f"região {r}: centroide: ({p.centroid[1]}, {p.centroid
    [0]}), área: {p.area} perímetro: {p.perimeter}
    excentricidade: {eccentricity} solidez: {solidity}")
```

Os valores de `eccentricity` e `solidity` vem dos resultados já citados nos itens Excentricidade e Solidez respectivamente.

5 Limitações

Todos os testes foram feitos com imagens de fundo branco.

Todos os testes foram feitos com imagens coloridas com apenas 3 canais (RGB), sem a 4a camada (transparência).

Não testamos parametrizar o argumento `level` da função `find_contours`, deixando ela com o valor `0.8`, nem como tentar uma automatização a partir de validação de histograma da imagem (no caso de imagens em tons próximos tentar algum cálculo baseado em histogramas).

Referências

[1] Module: color https://scikit-image.org/docs/dev/api/skimage.color.html#skimage.color.rgb2gray Acessado em: 2021-05-19

[2] Scikit-image https://scikit-image.org/ Acessado em: 2021-05-19

[3] Module: measure https://scikit-image.org/docs/dev/api/skimage.measure.html#skimage.measure.find_contours Acessado em: 2021-05-17

[4] Module: measure https://scikit-image.org/docs/dev/api/skimage.measure.html#skimage.measure.regionprops Acessado em: 2021-05-19

[5] scikit-imageskimagemeasure_regionprops.py https://github.com/scikit-image/scikit-image/blob/main/skimage/measure/_regionprops.py#L877-L1170 Acessado em: 2021-05-25

[6] scikit-imageskimagemeasure_regionprops.py https://github.com/scikit-image/scikit-image/blob/00875e67733a950f8f2477ac77a38712f14e62f3/skimage/measure/_regionprops.py#L203 Acessado em: 2021-05-25

[7] scikit-imageskimagemeasure_regionprops_utils.py https://github.com/scikit-image/scikit-image/blob/

00875e67733a950f8f2477ac77a38712f14e62f3/skimage/measure/_regionprops_utils.py#L186 Acessado em: 2021-05-25

[8] K. Benkrid, D. Crookes. Design and FPGA Implementation of a Perimeter Estimator. The Queen's University of Belfast. http://www.cs.qub.ac.uk/~d.crookes/webpubs/papers/perimeter.doc Acessado em: 2021-05-25

[9] GONZALEZ, R. C.; WOODS, R. E. Digital Image Processing. 4th. ed. USA: Pearson Education Ltd., 2018. ISBN 0-13-335672-8

[10] PEDRINI, H. C.; *Introdução ao Processamento Digital de Imagem MC920 / MO443*, 2021. https://www.ic.unicamp.br/~helio/disciplinas/MO443/aula_representacao.pdf

[11] Michael A. Wirth, Ph.D., Shape Analysis and Measurement http://www.cyto.purdue.edu/cdroms/micro2/content/education/wirth10.pdf Acessado em: 2021-05-28