MO443 - Introdução ao Processamento de Imagem Digital - Trabalho 3

Mauro Roberto Costa da Silva - 192800
maurorcsc@gmail.com

I. Introdução

O objetivo deste trabalho é detectar objetos em imagens digitais e obter algumas medidas desses objetos, como centroide, área, perímetro, excentricidade e solidez.

II. Execução do Código

Além deste relatório, foi entregue o código fonte e os demais arquivos citados no texto. O programa foi desenvolvido em *Python* 3.6.9, utilizando as bibliotecas *numpy* 1.19.5, *OpenCV* 3.2.0 e *Matplotlib* 3.3.4, e pode ser executado através do arquivo ***trabalho3.py***.

A. Parâmetros

Para a execução do programa, é necessário a utilização dos seguintes parâmetros:

- -i: imagem de entrada;
- -o: arquivo de saída (opcional);

Após o nome do arquivo *python* (*trabalho3.py*), é necessário inserir o nome do arquivo da imagem de entrada através do parâmetro **-i**. É possível inserir também um parâmetro **-o** que indica o nome do arquivo de saída, por padrão esse nome é "*./output.png*".

Durante a execução do programa, todas as operações são executadas. Isto é, todas as propriedades dos objetos são calculadas e as imagens do contorno, regiões rotuladas, histograma e versão binária da imagem de entrada são geradas. Cada imagem gerada possui o nome formado pelo valor passado no parâmetro *-o* concatenado ao nome da operação aplicada.

Segue um exemplo de como executar o programa:

```
python3 trabalho3.py -i ./imagens/objetos3.png -o objetos3.png
```

Nesse exemplo, o programa recebe como imagem de entrada o arquivo *./imagens/objetos3.png* e calcula todas as propriedades dos objetos. São geradas imagens no formato PNG para o mapa de bordas, os contornos, as regiões rotuladas, histograma e a versão binária da imagem de entrada. Essas imagens são salvas com o nome *objetos3* concatenado ao nome da operação. A entrada de dados foi testada com imagens cedidas pelo professor. Todas as imagens testadas estão no diretório "*./imagens*".

III. Processo da Implementação

Nesta seção, serão descritas as decisões que levaram à implementação final do programa.

A imagem de entrada é lida através do método **imread** da biblioteca *OpenCV* e armazenada em uma matriz do tipo *numpy.array*. Os arquivos de saída são gerados utilizando também a biblioteca *OpenCV*, com o método **imwrite**.

A. Transformação de Cores

Na leitura da imagem, usando **cv2.imread**, é passado o parâmetro *cv2.IMREAD_GRAYSCALE* que indica que a imagem deve ser lida em tons de cinza. Em seguida, a imagem é convertida para preto e branco usando o método **cv2.threshold** e passando como parâmetro o limiar 230, que indica que todo *pixel* com tom abaixo desse valor será convertido para preto e os demais *pixels* para branco. Foi escolhido um limiar alto para que objetos claros também fossem destacados do fundo da imagem.

B. Contornos dos Objetos

Para a obtenção dos contornos, primeiramente é criado um mapa de bordas da imagem. Em uma primeira versão do código, esse mapa foi criado usando o método **cv2.Canny**. Esse método não obteve bons resultados em todas as imagens, resultando em descontinuidade nos cantos de alguns objetos. Por esse motivo, na versão final foi utilizado um operador Laplaciano para a detecção de bordas, implementado usando **cv2.Laplacian** e tamanho de *kernel* 3. Esse operador obteve bons resultados nas imagens testadas, detectando todos os objetos.

A partir do mapa de bordas da imagem, é obtido os contornos dos objetos com o método **cv2.findContours**. Foi usado o modo *cv2.RETR_EXTERNAL* para que fossem devolvidos apenas os contornos externos dos objetos e a implementação utilizada foi a *cv2.CHAIN_APPROX_NONE*, que devolve todos os pontos de bordas dos elementos. Ter todos os pontos de borda é importante para a criação de elipses ao redor dos elementos no cálculo de excentricidade.

Com a lista de contornos, é possível criar uma imagem com os contornos dos objetos. Essa imagem é inicializada com valor $(255, 255, 255)$ (branco) e cada contorno é desenhado da cor vermelha $(255, 0, 0)$ usando o método **cv2.drawContours**.

C. Extração de Propriedades dos Objetos

No código desenvolvido, todas as propriedades dos objetos são calculadas a partir dos contornos extraídos.

1) Área e perímetro: a área de um objetivo é calculada usando o método **cv2.contourArea** e o perímetro é calculado usando o método **cv2.arcLength**.

2) Centroide: para obtenção do centroide dos objetos, foram utilizados momentos. O centroide pode ser obtido pelas fórmulas

$$c_x = \frac{M_{10}}{M_{00}} \qquad \text{e} \qquad c_y = \frac{M_{01}}{M_{00}},$$

em que M são os momentos. Os momento são obtidos usando **cv2.moments** e c_x e c_y são calculados como $cx = int(M['m10']/M['m00'])$ e $cy = int(M['m01']/M['m00'])$.

3) Solidez: a solidez de um objeto pode ser definida como a razão entre sua área e a do seu fecho convexo:

$$solidez = \frac{area}{area\ fecho\ convexo}.$$

O fecho convexo do objeto é obtido usando o método **cv2.convexHull** e a sua área obtida usando **cv2.contourArea**.

4) Excentricidade: inicialmente, foi utilizada excentricidade de elipse para calcular a excentricidade dos objetos das imagens. Sejam A e B, respectivamente, os eixos maior e menor de uma elipse e sejam $a = A/2$ e $b = B/2$, respectivamente, os semieixos maior e menor da mesma elipse, a excentricidade dessa elipse corresponde a:

$$excentricidade = \frac{\sqrt{a^2 - b^2}}{a} = \sqrt{1 - \frac{b^2}{a^2}} = \sqrt{1 - \left(\frac{b}{a}\right)^2} = \sqrt{1 - \left(\frac{2b}{2a}\right)^2} = \sqrt{1 - \left(\frac{B}{A}\right)^2}. \quad (1)$$

Quanto mais próximo de 0 é a excentricidade, mais parecida com um círculo é a elipse.

Como os objetos das imagens não necessariamente são elipses, foi usado o método **cv2.fitEllipse** para encontrar a menor elipse que contenha todo o objeto. A partir dessa elipse, são obtidos o menor e maior eixos e é calculada a excentricidade aproximada do objeto, de acordo com a equação (1). Esse método resultou em valores próximos ao esperado.

Na versão final, foi utilizado o cálculo de excentricidade a partir de momentos. Usando momentos, a excentricidade de um objeto pode ser calculada como:

$$\sqrt{1 - \frac{\lambda_2}{\lambda_1}}, \quad (2)$$

em que λ_1 e λ_2 são os autovalores da matriz de covariância da imagem e correspondem ao maior e menor eixos da imagem, respectivamente. Eles podem ser calculados como:

$$\lambda_1 = \frac{\mu'_{20} + \mu'_{02} + \sqrt{4{\mu'_{11}}^2 + (\mu'_{20} - \mu'_{02})^2}}{2}$$

e

$$\lambda_2 = \frac{\mu'_{20} + \mu'_{02} - \sqrt{4{\mu'_{11}}^2 + (\mu'_{20} - \mu'_{02})^2}}{2}.$$

Os valores de μ'_{20}, μ'_{02} e μ'_{11} são obtidos através dos momentos, da seguinte forma:

$$\mu'_{20} = \frac{M_{20}}{M_{00}} - c_x^2 \qquad \mu'_{02} = \frac{M_{02}}{M_{00}} - c_y^2 \qquad \mu'_{11} = \frac{M_{11}}{M_{00}} - c_y c_x,$$

em que c_x e c_y são os pontos do centroide e M são os momentos.

Para cada contorno, são obtidos os momentos com o método **cv2.moments** e calculada a excentricidade seguindo a equação (2). Os resultados obtidos com essa segunda implementação foram melhores que os obtidos com excentricidade de elipse, e, por esse motivo, essa foi a implementação escolhida na versão final.

5) Imagem com objetos rotulados: é criada uma imagem com os objetos rotulados. Essa imagem é inicializada com a cor branca e, em seguida, cada objeto é desenhado na imagem a partir do seu contorno, usando o método **cv2.fillPoly**. Os rótulos são adicionados usando o método **cv2.putText** e posicionados de acordo com o centroide de cada objeto. São subtraídos 5 *pixels* do eixo x do centroide para cada caractere do texto e adicionados 5 *pixels* no eixo y do centroide no posicionamento dos textos. Isso é feito para tentar compensar o tamanho da fonte, já que a posição passada para o método **cv2.fillPoly** corresponde ao início do texto, não ao centro.

D. Histograma de Área dos Objetos

O vetor de área dos objetos calculados na Seção III-C é utlizado para classificar os objetos em pequenos, médios e grandes, seguindo os critérios:

- objeto pequeno: Área $<$ 1500 *pixels*
- objeto médio: 1500 $\leq$ Área $<$ 3000 *pixels*
- objeto grande: Área $>$ 3000 *pixels*

O histograma é obtido usando a função **hist** da biblioteca *matplotlib*, passando $bins = [p, 1500, 3000, q]$, em que $p = \min(\min(areas), 0)$, $q = \max(\max(areas), 4000)$ e $areas$ é o vetor das áreas dos objetos,. A saída dessa função também é usada para imprimir a quantidade de objetos de cada tamanho e o histograma é exportado para imagem usando o método **savefig**.

IV. Resultados e Discussão

Para a apresentação dos resultados, será utilizada a imagem *objetos3.png* (Figura 1), disponibilizada pelo professor. Em seguida, serão apresentados os resultados e o que foi observado.

Figure 1: Imagem de objetos utilizada

A Figura 2 apresenta a imagem de entrada após a alteração de cores para preto e branco, a Figura 3 apresenta o mapa de bordas da imagem obtido pelo operador Laplaciano e a Figura 4 apresenta os contornos dos objetos detectados a partir do mapa de bordas, como descrito na Seção III-B.

Figure 2: Imagem preto e branca

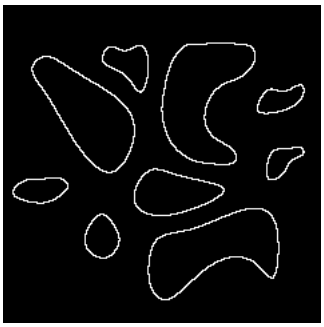

Figure 3: Mapa de Bordas (Laplaciano)

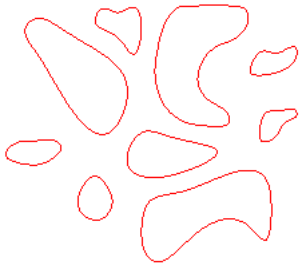

Figure 4: Contorno dos objetos

A Saída 1 apresenta a saída do código com a área, perímetro, excentricidade e solidez de cada região da imagem. Os valores de perímetro, excentricidade e solidez ficaram bem próximos aos apresentados na descrição do trabalho e aos obtidos através do método **regioprops** da biblioteca *Scikit-Image*. A área dos objetos possui uma maior diferença em relação aos valores apresentados no enunciado, porém essa diferença não ultrapassou 10% e não afetou o cálculo das demais propriedades.

Saída 1: Saída padrão do código com as propriedades calculadas para cada região

```
número de regiões: 9

região 0: área: 634 perímetro: 96.325901 excentricidade:
     0.629583 solidez: 0.977623
região 1: área: 3801 perímetro: 305.421354 excentricidade:
     0.913779 solidez: 0.768888
região 2: área: 596 perímetro: 103.012193 excentricidade:
     0.897001 solidez: 0.968293
região 3: área: 1606 perímetro: 174.124890 excentricidade:
     0.871703 solidez: 0.969231
região 4: área: 398 perímetro: 88.769552 excentricidade:
     0.860278 solidez: 0.905575
região 5: área: 496 perímetro: 99.254833 excentricidade:
     0.895691 solidez: 0.900181
região 6: área: 3478 perímetro: 259.462984 excentricidade:
     0.899819 solidez: 0.976963
região 7: área: 738 perímetro: 119.982755 excentricidade:
     0.746918 solidez: 0.890295
região 8: área: 3831 perímetro: 313.764500 excentricidade:
     0.819785 solidez: 0.739718
```

Na Figura 5, as regiões foram desenhadas a partir dos seus contornos, preenchidas com cores aleatórias e rotuladas com a sua numeração correspondente. A numeração foi posicionada de acordo com o centroide de cada objeto, como descrito na Seção III-C5

Figure 5: Objetos rotulados

A Figura 6 mostra o histograma das regiões divididas por área, como descrito na Seção III-D e a Saída 2 apresenta a saída do código com a quantidade de regiões de cada tamanho. O histograma e a quantidade de objetos de cada tamanho são semelhantes ao apresentado na descrição do trabalho.

Saída 2: Saída padrão do código com a quantidade de objetos de cada tamanho

```
número de regiões pequenas: 5
número de regiões médias: 1
número de regiões grandes: 3
```

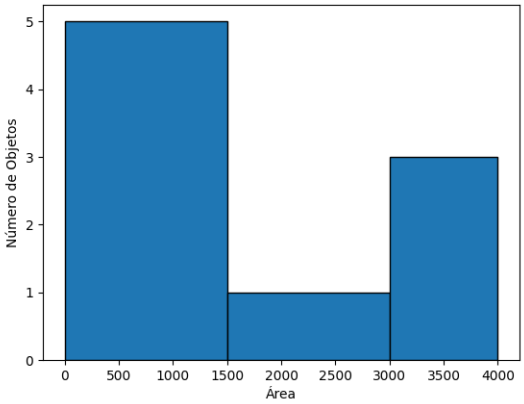

Figure 6: Histograma das áreas das regiões

V. Conclusão

Pudemos aplicar técnicas de detecção de regiões em imagens e calcular algumas de suas propriedades. Os resultados foram satisfatórios, já que nas imagens testadas todas as regiões foram detectadas e os valores das propriedades calculadas tiverem pequenas diferenças em relação aos descritos no enunciado do trabalho e aos obtidos pelo método **regionprops** da biblioteca *Scikit-Image*.

Esses resultados ainda poderiam ser melhorados utilizando operadores morfológicos para a criação do mapa de bordas, podendo resultar em contornos mais precisos.