Introdução ao Processamento Digital de Imagem (MC920) - Trabalho 1

Heitor Boschirolli - 169477

2018
March

1 Executando o código

O tar HeitorBoschirolli_169477.tar contém o código main.py - responsável pelo processamento das imagens -, duas imagens para teste e este PDF.

Para executar o arquivo main.py, basta digitar "python main.py" em um terminal linux com python 2 e pressionar a tecla enter. Assim que o código for executado, ele solicitará ao usuário que digite o nome da imagem a ser processada na entrada padrão do terminal; o nome da imagem deve incluir sua extensão e, caso a imagem não esteja no mesmo diretório que o script, é preciso passar o caminho da imagem a partir do diretório do script como nome.

Em seguida algumas propriedades da imagem serão exibidas na saída padrão do terminal e as imagens <nome da imagem>_grayscale.png, <nome da imagem>_edges.png, <nome da imagem>_labeled.png e <nome da imagem>_histogram.png que contém a imagem original convertida para níveis de cinza, o contorno dos objetos da imagem original, os objetos da imagem original númerados e um histograma com o número de objetos para cada tamanho (pequeno, médio ou grande), respectivamente aparecerão no mesmo diretório em que a imagem original estava. <nome da imagem> representa o nome da imagem inserida pelo usuário no início da execução.

2 Solução

2.1 Transformação de cores

Para converter o conteúdo da imagem original para níveis de cinza, foi usada a função *rgb2gray* da biblioteca scikit-image. Como o nome sugere, a função espera que a imagem de entrada esteja na representação RGB.

Para converter os três canais da imagem RGB para apenas um canal na representação em níveis de cinza, a função *rgb2gray* usa a equação $Y = 0.2125R + 0.7154G + 0.0721B$ onde R, G, e B são as intensidades de cada pixel na escala RGB e Y é a intensidade do pixel em níveis de cinza [1].

2.2 Contornos dos objetos

Para encontrar o contorno dos objetos no fundo branco, foi usada a função *roberts* da biblioteca scikit-image. Como o nome sugere, a função usa o operador cruzado de Roberts para encontrar a magnitude das bordas das imagens e retorna o mapa de borda cruzada de Robert [2].

2.3 Extração de propriedades dos objetos

Todas as propriedades dos objetos foram extraídas usando a função *regionprops* da biblioteca scikit-image. Essa função exige que os objetos da imagem já estejam rotulados, para isso foi usada a função *label* da mesma biblioteca (os rótulos também serviram para numerar os objetos posteriormente). A imagem foi, antes de ser passada para a função label, binarizada (para garantir que a função teria o comportamento esperado); para a binarização foi convertido todo pixel com intensidade diferente de 1 (branco) para 0 (preto) e os pixel de intensidade 1 permaneceram inalterados.

A medida de área é feita contanto o número de pixels que compõe o objeto [2], mesmo assim, na fase de testes, houve alguma variação de áreas de objetos com mesmo tamanho, sugerindo que a medida tem uma incerteza (não descrita na documentação até onde vi). Os perímetros são calculados usando caminho 4 entre os pixels [2].

Para fazer o histograma foi usada a função *hist* da biblioteca matplotlib com as áreas já medidas dos objetos como parâmetro. O número de barras

foi restringido para 3 de forma que a primeira barra contaria todos objetos de área 0 a 1500 (pequenos), o segundo de 1500 a 3000 (médios) e a terceira de 3000 até o tamanho do maior objeto (grandes).

3 Exemplo de entrada e saída

Para o caso em que a imagem de entrada é objetos1.png (Figura 1), abaixo encontram-se as entradas e saídas em um terminal linux.

```
python main.py
Enter the image name: objetos1.png
region: 0  perimeter: 190 area: 2352
region: 1  perimeter: 190 area: 2352
region: 2  perimeter: 62  area: 272
region: 3  perimeter: 62  area: 272
region: 4  perimeter: 188 area: 2304
region: 5  perimeter: 62  area: 272
region: 6  perimeter: 64  area: 289
region: 7  perimeter: 190 area: 2352
region: 8  perimeter: 64  area: 289
region: 9  perimeter: 64  area: 289
region: 10 perimeter: 190 area: 2352
region: 11 perimeter: 190 area: 2352
region: 12 perimeter: 64  area: 289
region: 13 perimeter: 62  area: 272
region: 14 perimeter: 188 area: 2304
region: 15 perimeter: 190 area: 2352
region: 16 perimeter: 62  area: 272
```

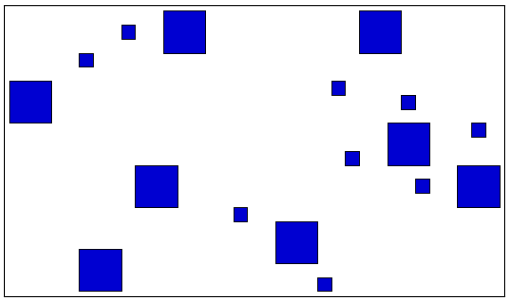

Figure 1: objetos1.png

E no diretório em que a imagem objetos1.png estava, as imagens objetos1_grayscale.png (Figura 2), objetos1_edges.png (Figura 3), objetos1_labeled.png (Figura 4) e objetos1_histogram.png (Figura 5) aparecerão.

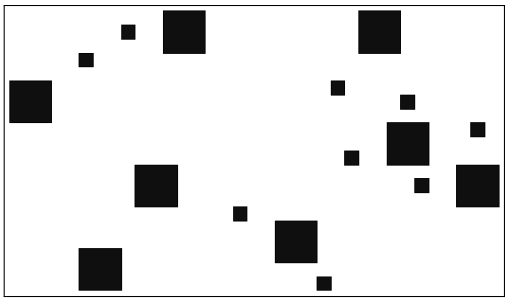

Figure 2: Imagem da Figura 1 convertida para níveis de cinza

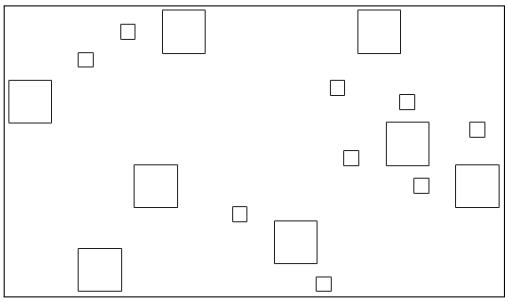

Figure 3: Bordas dos objetos da imagem da Figura 1

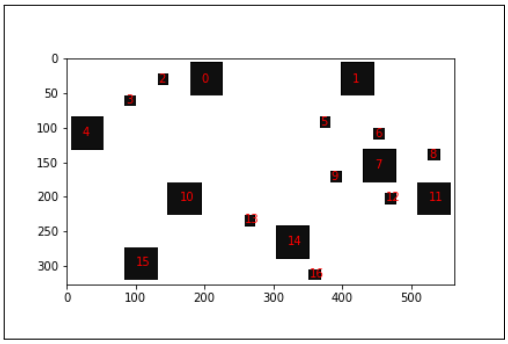

Figure 4: Objetos da imagem da figura 1 convertidos para escala de cinza e rotulados

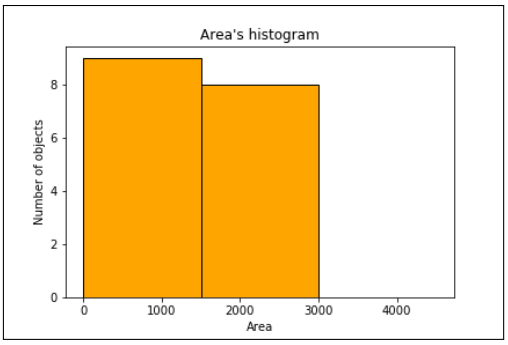

Figure 5: Histograma mostrando quantos objetos na imagem da Figura 1 são pequenos (primeira barra), médios (segunda barra) e grandes (terceira barra)

3.1 Limitações

A função responsável por rotular os objetos é bastante sensível a ruído. Cada píxel de cor diferente do fundo é considerado como pertencente a um objeto. Na Figura 7 pode ser visto o resultado da numeração dos objetos da Figura 6 que possui ruído sal e pimenta introduzido artificialmente.

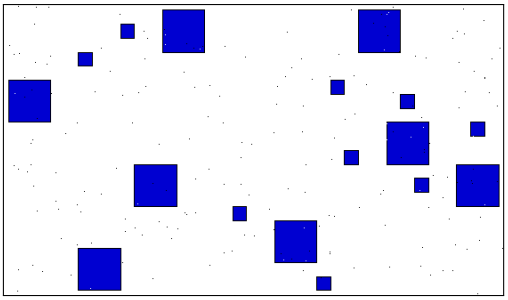

Figure 6: Imagem da Figura 1 com ruído sal e pimenta introduzido artificialmente

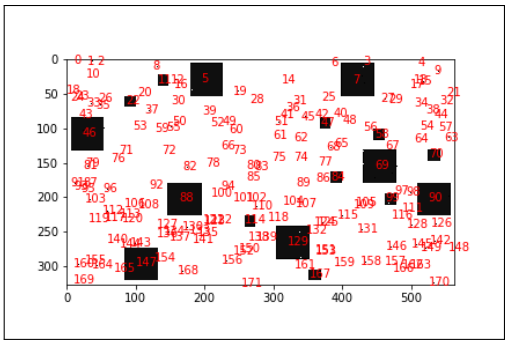

Figure 7: Resultado da rotulação feita na imagem da Figura 6

Como pode ser visto, cada ponto escuro do ruído que está fora dos quadrados azuis foi considerado como um objeto novo, resultando em informações incorretas sobre a imagem. Na figura 8 é possível ver o histograma resultante dessa númeração:

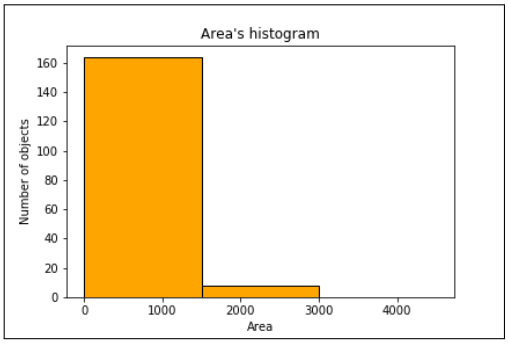

Figure 8: Histograma mostrando a contagem de objetos pequenos, médios e grandes na imagem da figura 6

Idealmente o ruído não afetaria o processo de contagem de objetos e o histograma resultante seria igual ao da Figura 5. A diferença entre os histogramas das Figuras 5 e 8 ilustra o estrago causado pelo ruído.

Foi considerado estabelecer um tamanho mínimo para os objetos de forma que a contagem de pixels de ruído não ocorresse, mas como o estabelecimento de quão grande um conjunto de pixels precisa ser para ser considerado um objeto é dependente da aplicação, preferiu-se deixar que o código funcionasse adequadamente apenas para imagens sem ruído.

References

[1] Module: color — skimage v0.14dev docs - scikit-image, http://scikit-image.org/docs/dev/api/skimage.color.html

[2] Module: filters — skimage v0.14dev docs - scikit-image, http://scikit-image.org/docs/dev/api/skimage.filters.html