Relatório - Atividade 1

Universidade Estadual de Campinas
Instituto de Computação
Introdução ao Processamento Digital de Imagens

Março de 2018

Felippe Trigueiro Angelo
210479

1 Descrição geral da atividade 1

A atividade 1 da disciplina Introdução ao Processamento Digital de Imagens teve como objetivo a realização de alguns processamentos básicos em imagens digitais, como por exemplo, (i)detecção de bordas de objetos contidos na imagem; (ii)conversão de uma imagem cromática em uma imagem monocromática; (iii)cálculo de alguns parâmetros dos objetos tais como, perímetro, área e centróide e (iv)classificação dos objetos de acordo com o valor de sua área.

2 Executando o Programa

Para executar o programa, entre no terminal do linux, navegue até a pasta onde os códigos estão localizados e digite o seguinte comando:

```
sudo python3 atividade1.py [nome-da-imagem]
Ex: sudo python3 atividade1.py objetos2.png
```

2.1 Dados de Entrada

Para a correta execução do programa é necessário que alguns cuidados sejam tomados com os dados de entrada:

- As imagens de entrada devem estar na mesma pasta em que está contido o código-fonte;
- As imagens de entrada devem ser coloridas.
- As imagens devem possuir o fundo branco e os objetos de alguma outra cor.

2.2 Dados de Saída

Após os processamentos descritos no primeiro ítem deste texto, o programa gera como saída um conjunto de quatro imagens, que correspondem à imagem original, uma imagem contendo a imagem de entrada juntamente com a imagem em tons de cinza, uma imagem contendo as bordas dos objetos da imagem original e uma imagem contendo os rótulos dos objetos. As imagens de saída estão localizadas na pasta /Output_Images, localizada no diretório do código-fonte, e estão no formato .png.

Algumas informações adicionais não são exibidas na forma de imagens, mas são exibidas no terminal. São elas:

- Número de objetos;
- Características dos objetos, tais como, área, perímetro e centróide (não exibido no terminal);
- Quantidade de objetos pequenos, médios e grandes.

3 Descrevendo o Código

3.1 Algoritmos e Estruturas de Dados utilizadas

3.1.1 Conversão para Imagem Monocromática

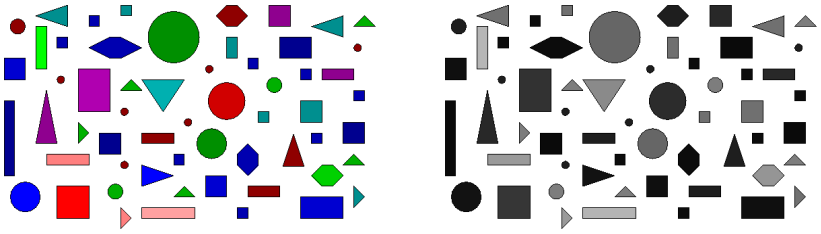
Com a finalidade de se obter as bordas dos objetos contidos na imagem de entrada, converteu-se a mesma para tons de cinza, facilitando assim o processamento em questão. Tal facilitamento advém do fato de que imagens coloridas possuem no mínimo três camadas, ao contrário das imagens monocromáticas, que possuem apenas uma. Portanto, como a cor não é o objeto de interesse desse processamento, apenas a luminância possui a necessidade de ser preservada.

Para a realização dessa operação, foi utilizada a função *rgb2gray()*, presente no submódulo *color* da biblioteca *SKIMAGE*.

```
1 img_gray = color.rgb2gray(img)
```

A função *rgb2gray()* recebe uma imagem colorida no formato RGB ou RGBA e a converte para tons de cinza por meio da equação 1, onde RGB são as matrizes contendo os canais vermelho, verde e azul. A função retorna uma matriz MxN contendo os valores de luminância normalizados entre 0 e 1. Um exemplo que mostra o efeito dessa função pode ser visto na figura 1.

$$Y = 0.2125R + 0.7154G + 0.0721B \tag{1}$$



(a) Imagem Colorida (b) Imagem Monocromática

Figure 1: Uso da função rgb2gray().

3.2 Detecção da Borda dos Objetos

Para se encontrar a borda dos objetos contidos na imagem de entrada utilizou-se a função *find_contours()*, presente no submódulo *measure* da biblioteca *SKIMAGE*.

```
edges = measure.find_contours(img_gray, 0.8)
```

A função *find_contours()* recebe uma imagem em tons de cinza e busca pelas bordas da imagem que contém um valor de luminosidade abaixo do limiar passado como segundo argumento, neste caso 0.8. Para alcançar tal objetivo, *find_contours()* utiliza um algoritmo chamado de *Marching Square Algorithm* (ver [1]). A função passa como retorno uma lista de arrays *ndarray* com tamanho (n, 2), onde n é o número de bordas encontradas.

É importante citar que o valor de 0.8 foi usado pois percebeu-se que ele corresponde a um limite de luminosidade que pode ser obtido quando converte-se uma imagem colorida para monocromática, assim como foi convencionado que o fundo da imagem de entrada seria branco, os valores abaixo de 0.8 corresponderão aos objetos. A figura 2 mostra o resultado da aplicação de *find_contours()* na imagem da figura 1.

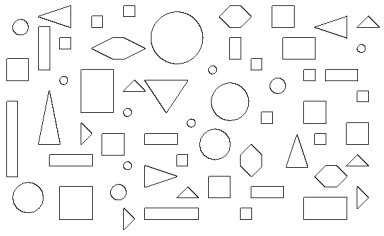


Figure 2: Contorno dos Objetos.

3.3 Cálculo das Propriedades dos Objetos

Para o cálculo das propriedades área, perímetro e centróide foram utilizadas as funções *label()* e *regionprops()*, presentes no submódulo *measure* da biblioteca *SKIMAGE*.

```
bin_im = img_gray < 0.8
label = measure.label(bin_im, connectivity=2)
props = measure.regionprops(label)
```

A função *regionprops()* recebe um array rotulado, ou seja, um array que contém o número de conexões que cada pixel realiza com os pixels de um mesmo objeto na imagem. Tal array é obtido por meio da função *label* que recebe como entrada uma imagem binária que é obtida por meio da comparação lógica presente na linha 1. Assim todos os pixels que possuam valor abaixo de 0.8, ou seja, pixels dos objetos, são mapeados como preto.

Assim, o concatenado conjunto das funções acima gera como retorno uma lista de objetos contendo várias propriedades dos objetos contidos na imagem, dentre eles, área, perímetro (calculado com vizinhança 4) e centróide.

Calculadas as propriedades de área, perímetro e centróde, a área e perímetro dos objetos da figura 1 foram mostrados no terminal, bem como o rótulo das regiões. Abaixo são exibidos algumas linhas obtidas como resultado.

Número de Regiões: 58

Região 0 Perímetro: 138 Área: 816
Região 1 Perímetro: 62 Área: 272
Região 2 Perímetro: 118 Área: 1024
Região 3 Perímetro: 126 Área: 1056
Região 4 Perímetro: 251 Área: 4616
Região 5 Perímetro: 62 Área: 272

3.4 Rótulo dos Objetos da Imagem

Como parte da atividade 1, foi pedido para rotular cada região da imagem de entrada, assim foi criada uma imagem contendo o rótulo de cada objeto contido na imagem. Para tal foi utilizo o trecho de código abaixo.

```
1 centr = props[n].centroid
2 plt.text(centr[1] - 8, centr[0] + 5, str(n), figure=p2)
```

A função *text()* presente no submódulo *pyplot* biblioteca *MatPlotLib*, imprime na tela um texto, nas coordenadas especificadas. Neste caso, o texto em questão corresponde ao rótulo de cada objeto, que por sua vez é imprimido aproximadamente no centróide de cada objeto, calculado como um atributo do objeto retornado por *regionprops()*. A figura 3 mostra o resultado obtido por essa operação.

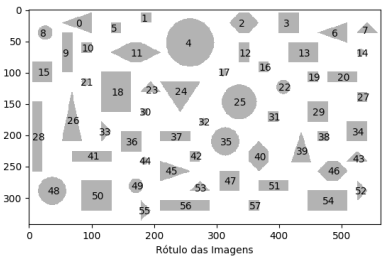


Figure 3: Rótudo dos Objetos.

3.5 Histograma e Classificação dos Objetos

Por fim, foi realizada uma classificação dos objetos detectados, utilizando como métrica, a área de cada um deles. A regra de classificação foi a seguinte:

- Objeto Pequeno: Possui área menor que 1500 pixels;
- Objeto Médio: Possui área maior que 1500 e menor que 3000 pixels;

- Objeto Grande: Possui área maior que 3000.

Assim, dada a figura 1 como entrada, foram obtidos os seguintes resultados:

- Número de regiões pequenas: 47
- Número de regiões médias: 9
- Número de regiões grandes: 2

Os resultados da classificação dos objetos foram resumidos em um histograma de barras. Para a figura 1 foi obtido o histograma da figura 4.

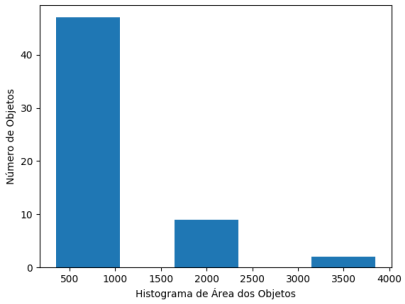


Figure 4: Histograma da Classificação por meio da Área.

3.6 Estruturas de dados utilizadas

No código, foram utilizadas as estruturas de dados array, tupla e lista no armazenamento das informações desejadas. No caso das imagens foram utilizados os arrays providos pela biblioteca *numpy*, gerando arrays da classe *ndarray*. Assim para imagens coloridas, elas são representadas como arrays do tipo MxNx3, onde cada matriz MxN representa um canal do RGB. Já para imagens monocromáticas, são utilizados arrays bidimensionais.

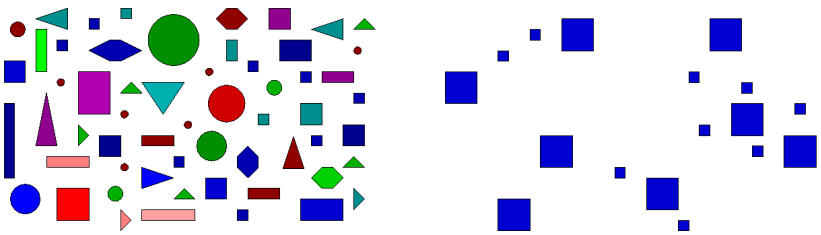
No armazenamento das informações das bordas é utilizada uma lista de *ndarrays*, onde cada array possui dimensões (n,2).

A tupla é utilizada no armazenamento das dimensões das imagens, por meio da extração do atributo *.shape*.

4 Testes adotados e e Limitações do programa

Foram feitos testes com as imagens contidas no endereço da página da disciplina. As imagens podem ser vistas na figura 5. Portanto, foram testadas imagens retangulares no formato .png.

Convém citar novamente aqui, que foi adotada a convenção de que o fundo da imagem é branco. Assim, caso o programa receba como entrada, imagens fora desses padrões, podem ocorrer erros.



(a) Formas Aleatórias (b) Quadrados Azuis

Figure 5: Imagens utilizadas no teste.

5 Referências Bibliográficas

1. Lorensen, William and Harvey E. Cline. **Marching Cubes: A High Resolution 3D Surface Construction Algorithm.** Computer Graphics (SIGGRAPH 87 Proceedings) 21(4) July 1987, p. 163-170).

2. **API Reference for skimage 0.14dev.** Disponível em: http://scikit-image.org/docs/dev/api/api.html

3. **The Matplotlib API Pyplot.** Disponível em: https://matplotlib.org/api/pyplot_summary.html