Projeto para o Code Marathon do MecanIST 2021

Duarte Valério

Fotos cedidas por João C. Henriques, Paulo Peças, Marco Leite, Manuel Sardinha

1 Objetivo

Desenvolver uma aplicação em Matlab para tratamento de imagens em tons de cinzento, implementando os métodos abaixo descritos.

Usar essa aplicação para melhorar imagens fornecidas.

2 Linguagem de programação

A aplicação deve ser desenvolvida em Matlab, e tem de correr na versão 2019a, a mais antiga atualmente disponibilizada pelos Serviços de Informática do IST.

A aplicação deve implementar os algoritmos de tratamento de imagem descritos na secção seguinte por manipulação das matrizes envolvidas, sem recorrer a funções de toolboxes, e nomeadamente às funções da toolbox *Image Processing*, que o fazem. As funções dessa toolbox podem ser usadas para importar e visualizar imagens.

3 Funcionalidades

O utilizador interage com a aplicação por meio de uma GUI (graphical user interface), que permite selecionar e mostrar uma imagem, e aplicar-lhe, pela ordem que ele quiser, um ou mais dos métodos de tratamento de imagem abaixo descritos, com parâmetros selecionados pelo utilizador.

4 Métodos de tratamento de imagem

Uma imagem em tons de cinzento é uma matriz de números, em que cada elemento corresponde a um pixel. No formato mais habitual, os números são inteiros que se acham entre 0 (preto) e 255 (branco).

Seja $\mathbf{A}$ a matriz da imagem original, e um pixel desta imagem $\mathbf{A}_{l,c}$. Seja $\mathbf{B}$ a matriz da imagem tratada, e o pixel correspondente desta imagem $\mathbf{B}_{l,c}$.

O programa deve implementar:

- Correção da tonalidade com uma transformação linear

$$\mathbf{B}_{l,c} = \alpha\, \mathbf{A}_{l,c} + \beta \tag{1}$$

  onde α e β são parâmetros que o utilizador do programa pode ajustar, e os valores de $\mathbf{B}$ têm de ser inteiros e estar limitados ao intervalo permitido.

- Conversão da imagem para preto e branco

$$\mathbf{B}_{l,c} = \begin{cases} 255, & \text{if } \mathbf{A}_{l,c} \geq \tau \\ 0, & \text{if } \mathbf{A}_{l,c} < \tau \end{cases} \tag{2}$$

  onde τ é um parâmetro que o utilizador do programa pode ajustar.

Nos métodos abaixo, $\mathbf{B}_{l,c}$ é obtido por combinação linear de $\mathbf{A}_{l,c}$ e dos pontos circundantes. Os coeficientes são agrupados numa pequena matriz chamada kernel. Por exemplo, se o kernel for

$$\mathbf{K} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \tag{3}$$

então

$$\mathbf{B}_{l,c} = 5\,\mathbf{A}_{l,c} - \mathbf{A}_{l-1,c} - \mathbf{A}_{l,c-1} - \mathbf{A}_{l,c+1} - \mathbf{A}_{l+1,c} \tag{4}$$

No caso genérico

$$\mathbf{K} = \begin{bmatrix} \mathbf{K}_{l-1,c-1} & \mathbf{K}_{l-1,c} & \mathbf{K}_{l-1,c+1} \\ \mathbf{K}_{l,c-1} & \mathbf{K}_{l,c} & \mathbf{K}_{l,c+1} \\ \mathbf{K}_{l+1,c-1} & \mathbf{K}_{l+1,c} & \mathbf{K}_{l+1,c+1} \end{bmatrix} \tag{5}$$

teremos

$$\begin{aligned} \mathbf{B}_{l,c} = {} & \mathbf{K}_{l-1,c-1}\,\mathbf{A}_{l-1,c-1} + \mathbf{K}_{l-1,c}\,\mathbf{A}_{l-1,c} + \mathbf{K}_{l-1,c+1}\,\mathbf{A}_{l-1,c+1} + \\ & + \mathbf{K}_{l,c-1}\,\mathbf{A}_{l,c-1} + \mathbf{K}_{l,c}\,\mathbf{A}_{l,c} + \mathbf{K}_{l,c+1}\,\mathbf{A}_{l,c+1} + \\ & + \mathbf{K}_{l+1,c-1}\,\mathbf{A}_{l+1,c-1} + \mathbf{K}_{l+1,c}\,\mathbf{A}_{l+1,c} + \mathbf{K}_{l+1,c+1}\,\mathbf{A}_{l+1,c+1} \end{aligned} \tag{6}$$

Repare que todas as quantidades em (6) são escalares. Nos exemplos acima, o kernel é uma matriz 3×3, e o programa só tem obrigatoriamente de utilizar kernels 3×3; mas poderiam usar-se matrizes maiores, por exemplo 5×5. Uma vez mais, os valores de $\mathbf{B}$ têm de ser inteiros no intervalo permitido.

O programa deve implementar os seguintes efeitos. Em todos eles, M é um escalar tal que a soma de todos os elementos do kernel $\mathbf{K}$ é 1 (ou 0, se 1 for impossível).

- Gaussian blur com o kernel

$$\mathbf{K}_b = \frac{1}{M} \begin{bmatrix} e^{-\frac{2}{2\sigma^2}} & e^{-\frac{1}{2\sigma^2}} & e^{-\frac{2}{2\sigma^2}} \\ e^{-\frac{1}{2\sigma^2}} & e^{-\frac{0}{2\sigma^2}} & e^{-\frac{1}{2\sigma^2}} \\ e^{-\frac{2}{2\sigma^2}} & e^{-\frac{1}{2\sigma^2}} & e^{-\frac{2}{2\sigma^2}} \end{bmatrix} \tag{7}$$

  onde σ é um parâmetro real que o utilizador do programa pode ajustar.

- Edge detection (deteção de arestas) com o kernel

$$\mathbf{K}_e = \frac{1}{M} \begin{bmatrix} -\frac{\alpha^2-\alpha+2}{2} & -\frac{\alpha^2-\alpha+2}{2} & -\frac{\alpha^2-\alpha+2}{2} \\ -\frac{\alpha^2-\alpha+2}{2} & 8\alpha & -\frac{\alpha^2-\alpha+2}{2} \\ -\frac{\alpha^2-\alpha+2}{2} & -\frac{\alpha^2-\alpha+2}{2} & -\frac{\alpha^2-\alpha+2}{2} \end{bmatrix} \tag{8}$$

  onde α é um parâmetro real que o utilizador do programa pode ajustar, e cujo valor por omissão pode ser 1.

- Sharpening com o kernel

$$\mathbf{K}_s = \frac{1}{M} \left(\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} + \mathbf{K}_e \right) \tag{9}$$

  onde $\mathbf{K}_e$ é o kernel definido em (8).

Pode convir consultar o artigo

`https://en.wikipedia.org/wiki/Kernel_(image_processing)`

que também discute várias possibilidades de como proceder na borda das imagens. Relativamente a esse assunto, deve usar-se um método que preserve o tamanho, isto é, as dimensões de $\mathbf{A}$ e $\mathbf{B}$ devem ser as mesmas.

5 Imagens fornecidas

Pretende-se detetar as arestas destas imagens, usando o programa desenvolvido, com alguma sucessão de métodos de tratamento e com parâmetros escolhidos para cada caso.

6 Grau de dificuldade adicional

Os kernels (7)–(9) são todos 3×3. O programa **pode** utilizar kernels com dimensões superiores (5×5, 7×7, 9×9, ...), sendo a dimensão do kernel escolhida pelo utilizador, mas **esta generalização é opcional**. Os kernels obtêm-se do seguinte modo:

- As entradas do kernel (7) são dadas por

$$e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{10}$$

  onde x e y são as distâncias, na horizontal e na vertical, ao pixel do centro.

- A matriz do kernel (8) é o simétrico da soma de oito matrizes:

$$\begin{bmatrix} 0 & 0 & 0 \\ 1 & -\alpha & \frac{(-\alpha)(-\alpha+1)}{2} \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ \frac{(-\alpha)(-\alpha+1)}{2} & -\alpha & 1 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 1 & 0 \\ 0 & -\alpha & 0 \\ 0 & \frac{(-\alpha)(-\alpha+1)}{2} & 0 \end{bmatrix} + \begin{bmatrix} 0 & \frac{(-\alpha)(-\alpha+1)}{2} & 0 \\ 0 & -\alpha & 0 \\ 0 & 1 & 0 \end{bmatrix} + \dots$$
$$+ \begin{bmatrix} 1 & 0 & 0 \\ 0 & -\alpha & 0 \\ 0 & 0 & \frac{(-\alpha)(-\alpha+1)}{2} \end{bmatrix} + \begin{bmatrix} \frac{(-\alpha)(-\alpha+1)}{2} & 0 & 0 \\ 0 & -\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 1 \\ 0 & -\alpha & 0 \\ \frac{(-\alpha)(-\alpha+1)}{2} & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & \frac{(-\alpha)(-\alpha+1)}{2} \\ 0 & -\alpha & 0 \\ 1 & 0 & 0 \end{bmatrix} \tag{11}$$

  Os elementos das matrizes são tirados da sequência

$$1,\ -\alpha, \frac{(-\alpha)(-\alpha+1)}{2!}, \frac{(-\alpha)(-\alpha+1)(-\alpha+2)}{3!}, \dots \tag{12}$$

  cujo termo geral é

$$\frac{\Gamma(-\alpha+k)}{k!\,\Gamma(-\alpha)} \tag{13}$$

  onde a função $\Gamma(x)$ está implementada em Matlab como `gamma`. Na generalização para kernels maiores, usam-se mais termos desta sucessão.

- Após a generalização de (8), a generalização de (9) é trivial. A matriz da esquerda tem todas as entradas iguais a 0, exceto a do pixel central, que é 1.