Homework 1: Filtragem de imagens

Fis. Computacional 2022-23 (P4)

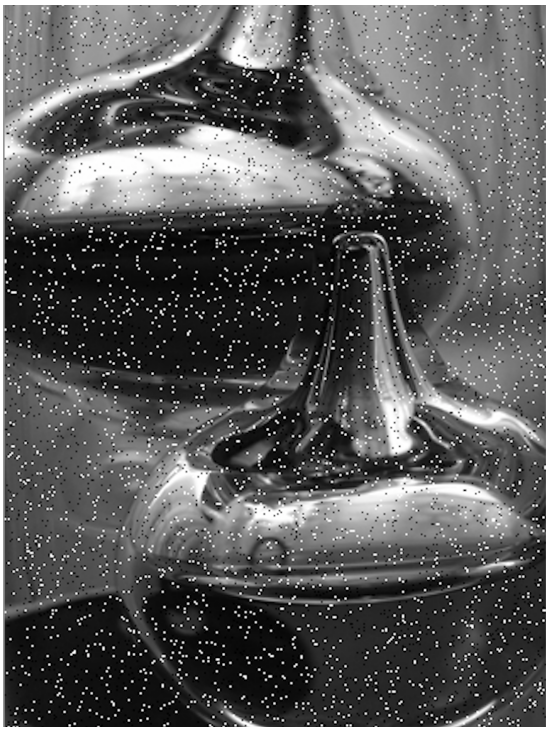
Fernando Barão

3 de maio de 2023

Filtragem de imagens

Actualmente, 45% dos portugueses têm uma conta Instagram ativa. De facto, cada vez que carregamos uma foto nas nossas redes sociais, a nossa maior preocupação é escolher um conjunto de filtros que melhore a qualidade das imagens. Portanto, o objetivo do exercício proposto é criar vários algoritmos que permitam melhorar a qualidade das nossas fotografias.

Para este efeito, consideraremos uma imagem .pgm em preto e branco. O protocolo de codificação `.pgm` atribui a cada pixel da imagem um número natural entre 0 e N. O número 0 corresponde à cor preta e por outro lado o número N corresponde à cor branca. O ficheiro glassware_noisy.ascii.pgm é um exemplo de imagem codificada através do protocolo PGM.



Mostra-se de seguida o excerto das primeiras 14 linhas do ficheiro `.pgm`.

```
P2
320 428
255
118 120 119 130 125 0 148 156 154 141 130 130 138 142 145 140 141
 ↪  142 140 139
131 132 143 156 145 142 134 113 100 99 100 106 109 112 255 116 116
 ↪  122 130 139
147 152 156 160 159 163 161 161 163 165 165 162 157 155 0 155 154
 ↪  154 155 157
157 159 159 159 162 164 164 166 168 167 166 165 163 160 154 150 147
 ↪  144 141
139 0 140 140 142 142 146 149 151 152 152 153 152 150 145 142 136
 ↪  131 127 126
124 122 121 255 119 119 118 120 121 121 122 126 126 124 124 131 255
 ↪  137 140
140 140 143 150 152 139 132 133 127 127 132 133 137 139 133 115 91
 ↪  73 67 74 82
90 106 122 133 143 151 164 179 198 207 208 205 209 194 188 186 178
 ↪  165 153 148
152 0 160 162 161 156 0 134 133 133 124 108 87 67 47 29 22 20 21 32
 ↪  0 95 123
```

```
255 153 157 158 158 155 152 147 146 144 134 130 125 127 129 133 142
 ↪  153 159
159 161 159 160 160 161 160 163 163 163 164 166 166 166 165 166 166
 ↪  166 166
```

A primeira linha do ficheiro contém a sigla *P2*, que identifica o protocolo de codificação do ficheiro. Para além disso, a segunda linha define a dimensão da imagem: no exemplo proposto, os números *320* e *428* determinam o número de pixels da fotografia ($320 \times 428 = 136960$). De facto, uma fotografia é uma matriz cujas entradas correspondem à codificação da cor dos pixéis. Portanto, os dois números armazenados na segunda linha do ficheiro PGM representam o tamanho da matriz associada à imagem.

Por outro lado, a terceira linha do ficheiro tem o papel de identificar a gradação máxima da cor N que corresponde à cor branca. Na imagem anexada, o valor correspondente ao branco será 255.

Nas linhas seguintes, os valores armazenados são as codificações que indentificam univocamente a cor de cada pixel da imagem, segundo a convenção mencionada anteriormente (0 = cor preta, N = cor branca).

Sugestão: descarreguem o ficheiro da imagem `.pgm` para o vosso computador e podem verificar o seu conteúdo fazendo:

```
cat <ficheiro_com_a_imagem>.pgm   [enter]
```

Verifiquem que a informação guardada no ficheiro está de acordo com a explicação anteriormente fornecida.

Para a resolução deste exercício, criem um programa em C++ `rImagem.C` (podem usar a extensão .C ou outra que seja aceite como por exemplo .cpp, .cxx, .cc).

Tarefas e questões:

1. **Leitura da imagem `.pgm`**

   Introduzam no vosso programa o código C++ necessário à leitura do ficheiro da imagem e do armazenamento do seu conteúdo num objecto do tipo vector de vectores, `vector<vector<int>>` , que corresponde a uma matriz. Mostra-se de seguida um exemplo da estrutura do código C++ a implementar para esta tarefa.

   ```
   // 1. read image configuration (3 first lines)
   //... abrir o ficheiro da imagem
   ifstream FI("filename");
   string line;
   //... ler a primeira linha
   std::getline(FI, line));
   ```

```
std::getline(FI, line);
//... ler a 2a linha e recuperar o numero de linhas e colunas
std::istringstream iss(line);
int nrows, ncols;
iss >> ncols >> nrows;
//... ler a 3a linha e recuperar o codigo maximo de cor
std::getline(FI, line);
iss.str(line);
int WhiteValue;
iss >> WhiteValue;

// 2. creation of matrix M [nrows, ncols]
vector<vector<int>> M(nrows, vector<int>(ncols));

// 3. read image to matrix
(...)
```

De forma a simplificar o código C++, podem autonomizar a leitura do ficheiro numa função que seja chamada a partir do programa principal. O protótipo da função (declaração da função que contém o seu nome assim como os argumentos de entrada e saída) poderia ser como se indica de seguida, em duas formas diferentes de implementação:

```
vector<vector<int>> ReadImage(string filename); // passing
↪  output by copy
void ReadImage(string filename, vector<vector<int>>& M); //
↪  passing output by reference
```

2. **Criação de um histograma de frequências absolutas das cores da imagem**

   Para tal comecem por criar um `vector` de contagens com a dimensão do número de cores $N+1$, da cor preta à cor branca passando pelos tons de cinzento.

   ```
   vector<int> ColourFreq(N+1);
   ```

   E de seguida, armazenem no `vector` a frequência absoluta das cores existentes na imagem. Algoritmicamente, varram todos os pixéis da imagem e incrementem de 1, o contador existente na posição do `vector` correspondente a cada código de cor. De forma simbólica, calculem o número k_i de pixéis caracterizados pela cor de código i, $\forall i \in 0, ..., N$

   Imprimam no ecrã o `vector` de contagens obtido. Quais as cores mais frequentes?

   Calcule ainda o vector das frequências relativa em percentagem e imprima também no ecrã.

3. **Variância e média das cores da imagem**

Para a análise da imagem podem recorrer as estimadores estatísticos, média $\mu \equiv < x >$ e variância $Var(x) = \sigma^2(x)$, onde x é o código de cor. O primeiro, informa-nos da cor média existente na imagem: quanto maior for esta cor mais clara será a imagem. O segundo, quantifica a variação de cor na imagem, ou seja, a sua não uniformidade do ponto de vista da cor. O cálculo da cor média (μ) para os $N + 1$ cores, pode ser calculado fazendo o varrimento de todos os pixéis e utilizando a cor de cada pixel $x_{i,j}$,

$$\mu_x = \frac{1}{N_{pixels}} \sum_{i=1}^{nrows} \sum_{j=1}^{ncols} x_{i,j}$$

O cálculo da variância da cor pode ser realizado da seguinte forma,

$$Var(x) \equiv \sigma_x^2 = \frac{1}{N_{pixels} - 1} \left[\sum_{i=1}^{nrows} \sum_{j=1}^{ncols} (x_{i,j} - \mu_x)^2 \right]$$

Imprimam no ecrã os valores da média e do desvio padrão.

4. **Criação da imagem invertida**

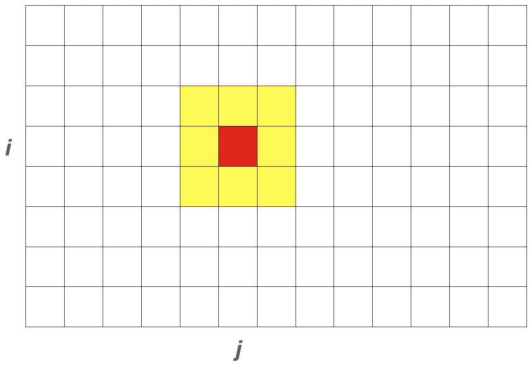
   Para este efeito, considerem que inverter uma imagem significa trocar a cor de cada pixel com a cor complementar ($0 \rightarrow N$, $1 \rightarrow N-1$, $2 \rightarrow N-2$, ...). Portanto, designando $x(i,j)$ como a cor do pixel (i,j) antes de inverter a imagem, valerá a relação $x'(i,j) = N - x(i,j)$, sendo $x'(i,j)$ a cor do pixel (i,j) após a inversão.

   Escreva num novo ficheiro, `glassware_noisy_inverted.ascii.pgm` , a imagem invertida no formato `.pgm` acima descrito.

   Somem a imagem inicial com a imagem inversa. Qual é o resultado esperado?

5. **Eliminação do ruído da imagem**

   Olhando atentamente para a imagem, verificarão que esta possui perturbações de cor (ruído) relacionados o seu envelhecimento. Nesta tarefa, vão procurar melhorar a qualidade da imagem. Para este efeito, considerem a cor de um dado pixel $x(i,j)$ da fotografia. Vamos considerar do ponto vista algorítmico, que *idealmente* a cor de um pixel está relacionado com a sua vizinhança próxima. Nesse sentido, calculem a média μ_8 dos 8 pixéis à volta do pixel $x(i,j)$ e troquem o valor de $x(i,j)$ por μ_8.

   Repitam esta operação para todos os pixéis. Do ponto de vista matemático, isto corresponde a substituir a cor de cada pixel $x(i,j)$ por:

$$x'(i,j) = \frac{1}{8}\left\{\left[\sum_{n=i-1}^{i+1}\sum_{m=j-1}^{j+1} x(n,m)\right] - x(i,j)\right\}$$

Guardem a nova imagem num ficheiro `glassware_reduced_noise.ascii.pgm` .

6. **Variância da cor após eliminação do ruído**

   Repitam o cálculo da variância da variável da cor da imagem e comparem o resultado com o valor obtido anteriormente na alínea 3.

7. **Filtragem da imagem: box filtering**

   O algoritmo explicado na alínea 5 consegue eliminar os pontos isolados brancos e pretos da imagem. Contudo, deteriora a qualidade dos píxels que ficam à volta dos pontos isolados. De facto, o algoritmo de redução do ruído anteriormente descrito não tem em conta a cor original do pixel (i,j) que está a corrigir, podendo por isso introduzir uma deterioração da cor dos pixels na proximidade desses pontos.

   Para melhorar o algoritmo, vamos introduzir a matriz de filtragem W e corrigirmos a cor dos pixels da imagem de acordo com a relação,

$$x'(i,j) = \sum_{\Delta i=-1}^{1}\sum_{\Delta j=-1}^{1} x(i+\Delta i, j+\Delta j)\, W(1+\Delta i, 1+\Delta j)$$

   A definição da matriz W permite atribuir um peso diferente a cada pixel que intervém no processo de correção. Por exemplo, no caso do algoritmo da alínea 5, a matriz W possui o tamanho 3×3, e o valor dos seus elementos são $W(1,1) = 0$ e todos os restantes elementos possuem o valor $1/8$.

   Defina agora um novo algoritmo de filtragem da imagem em que atribua o valor $1/9$ (pesos constantes) a todas as entradas da matriz W.

   Repitam o calculo da variância da cor e comparem com os resultados obtidos anteriormente.

   Guardem a nova imagem com o nome `glassware_box_blur.ascii.pgm` .