Práctica 4
Procesamiento digital de imágenes

Programación
ETSIT - UPV
Departamento de Sistemas Informáticos y computación

Curso 2020-2021

Índice

1. Objetivos de aprendizaje

- Aprender a resolver problemas que involucren el manejo de matrices.
- Implementar algoritmos de recorrido total o parcial de una matriz.
- Aprender a integrar las llamadas en un programa gráfico dirigido por menú.
- Entender cómo una imagen puede ser representada mediante una matriz.
- Resolver mediante operaciones matriciales un problema real de procesamiento digital de imágenes.

2. Representación de imágenes mediante matrices

Una imagen digital puede verse como una matriz de **MxN** píxeles, siendo **M** el número de líneas de la imagen y **N** el número de píxeles por línea. En el caso más sencillo de trabajar con imágenes en escala de grises, se puede representar cada píxel de la imagen mediante un número entero entre **0** y **MAX**, donde **0** representa el negro, **MAX** el blanco y cualquier valor intermedio los distintos matices de gris. Habitualmente se emplean 256 niveles de gris (valores entre 0 y 255). Para almacenar una imagen de estas características en la memoria del ordenador bastará con declarar una matriz de enteros de **MxN** y almacenar el nivel de gris que contiene cada píxel de la imagen en un elemento de la matriz.

En el caso de imágenes en color, es habitual almacenar para cada píxel la cantidad de rojo, de verde y de azul que contiene. Este triplete es conocido como **RGB** (red, green, blue). Si empleamos un byte para codificar cada uno de estos tres componentes, estaremos trabajando con imágenes de 24 bits (3 bytes = 24 bits), lo que permite codificar más de 16 millones de colores (exactamente 2^{24} colores).

En esta práctica, para simplificar el problema, se trabajará con imágenes en escala de grises.

3. Estructura del programa propuesto

Se pretende implementar un programa que permita realizar distintos tipos de procesamiento digital de imagen. Se dispone de una clase gráfica que nos presenta una interfaz (Figura 1) donde se dispone de un botón para cargar una imagen y otros controles que permiten seleccionar distintas operaciones a realizar con la imagen cargada.

Figura 1: Entorno gráfico de trabajo

El código se estructura en las siguientes tres clases:

- **ImageProcessingGUI**

  Contiene el código que implementa la Intergaz Gráfica de Usuario o GUI (por sus iniciales en inglés). Esta clase ya está implementada y no debes modificar ni añadir nada en ella. Únicamente debes saber que contiene, entre otras cosas, dos métodos públicos que permiten (1) mostrar una imagen almacenada en una matriz de enteros y (2) obtener la matriz de la imagen actual, con los perfiles que se muestran a continuación:

  - `public void setImage(int [][] img)`: muestra la imagen almacenada en `img`.

- `public int [][] getImage()`: devuelve una matriz con los píxeles de la imagen actualmente visible.

Atención: observa que estos métodos no son estáticos (son métodos de instancia o de objeto). Ello significa que para invocarlos debes usar un objeto de tipo `ImageProcessingGUI`.

- **ImageProcessingActions**

  Contiene métodos que se invocan automáticamente cuando el usuario realiza alguna acción sobre la interfaz gráfica de usuario. Todos estos métodos reciben un objeto `gui` de tipo `ImageProcessingGUI` que puedes usar para invocar los métodos `getImage` y `setImage` descritos anteriormente. Dentro de cada uno de estos métodos deberás:

  1. obtener la imagen actual (`getImage`);
  2. invocar algún método de la clase `ImageUtils` (descrita más abajo) al que le pasaremos la imagen actual y devolverá la imagen modificada según la operación correspondiente;
  3. visualizar (mediante `setImage`) la imagen devuelta por el método anterior.

  De cada uno de estos métodos está escrita únicamente la cabecera, por lo que deberás completarlos escribiendo el cuerpo con las tres operaiones descritas anteriormente. Concretamente, los métodos a completar son los siguientes::

  - `public static void modificarBrilloPressed(ImageProcessingGUI gui, int value)`
    Se invoca automáticamente cuando el usuario hace click sobre el botón `Brillo`. Recibe como parámetro el objeto `gui` y el factor de modificación brillo a aplicar (`value`).
  - `public static void binarizarPressed(ImageProcessingGUI gui, int value)`
    Se invoca automáticamente cuando el usuario hace click sobre el botón `Binarizar`. Recibe como parámetro el objeto `gui` y el umbral de binarización a aplicar (`value`).
  - `public static void entrelazarPressed(ImageProcessingGUI gui)`
    Se invoca automáticamente cuando el usuario hace click sobre el botón `Entrelazar`. Recibe como parámetro el objeto `gui`.
  - `public static void pixelarPressed(ImageProcessingGUI gui, int value)`
    Se invoca automáticamente cuando el usuario hace click sobre el botón `Pixelar`. Recibe como parámetro el objeto `gui` y el tamaño de píxel (`value`).
  - `public static void sobelPressed(ImageProcessingGUI gui)`
    Se invoca automáticamente cuando el usuario hace click sobre el botón `Sobel`. Recibe como parámetro el objeto `gui`.

- **ImageUtils**

  En esta clase debes implementar los métodos estáticos que realizan cada uno de los procesamientos de imagen que se piden realizar en esta práctica. Tal y como se ha explicado, estos métodos deberán ser invocados desde los métodos correspondientes de

la clase `ImageProcessingActions`. Los métodos deberán recibir la imagen a procesar (además de cualquier otro parámetro que pueda ser necesario) y devolver la imagen procesada.

4. Actividades propuestas

Se propone implementar una serie de métodos que permitan realizar los procesamientos de imagen mostrados en la Figura 4.

(a) Imagen original

(b) Modificar brillo

(c) Binarizar

(d) Pixelar

(e) Entrelazar

(f) Filtro de Sobel

Actividad #1: modificar (aumentar o disminuir) el brillo

Escribie un método `brillo` en la clase `ImageUtils` que reciba una matriz de enteros con la imagen a procesar y un factor de brillo, y devuelva una nueva matriz con el brillo modificado; para ello se debe sumar a cada pixel de la imagen el factor de brillo, que podrá ser positivo o negativo. Factores positivos provocarán una imagen más clara y factores negativos una más oscura. Deberá controlarse que, en ningún caso, un píxel de la imagen resultante tenga un valor superior a 255 ni inferior a 0. Se ha de devolver una matriz con la imagen resultante.

A continuación, completa el método `modificarBrilloPressed` de la clase `ImageProcessingActions` para que obtenga la imagen actual, modifique su brillo mediante una llamada al método an-

terior y visualice la imagen resultante.

Actividad #2: binarizar

Escribe un método `binarizar` en la clase `ImageUtils` que reciba una matriz de enteros con la imagen a procesar y un valor de umbral entre 0 y 255, y devuelva la matriz binarizada. El método debe convertir a blanco (255) todos los píxeles que igualen o superen el umbral y a negro (0) todos los que queden por debajo del mismo. Se ha de devolver una matriz con la imagen resultante.

A continuación, completa el método `binarizarPressed` de la clase `ImageProcessingActions` para que obtenga la imagen actual, la binarice mediante una llamada al método anterior y visualice la imagen resultante.

Actividad #3: pixelar

El efecto de pixelado se consigue dividiendo la imagen en cuadrados de un ancho determinado y asignando a todos los píxeles pertenecientes a un mismo cuadrado el valor medio de dichos píxeles.

Para resolver esta operación se propone implementar en primer lugar los siguientes métodos **private** en la clase `ImageUtils`:

- `private static int mediaSubMatriz (int [][] m, int row, int col, int w)`

  Método que dada una matriz `m`, un índice de fila `row`, un índice de columna `col` y un ancho (tamaño de píxel) `w`, devuelva la media del los píxeles pertenecientes a la submatriz de `m` comprendida entre `row, col` y `row+w, col+w`. Deberá tenerse en cuenta que tanto `row+w` como `col+w` pueden exceder los límites de la matriz `m`. Por lo tanto, para evitar un error de tipo `ArrayIndexOutOfBoundsException` deberá calcularse la media únicamente sobre la porción de submatriz que cae dentro de la matriz `m`.

- `private static void ponerValoresSubmatriz (int [][] m, int row, int col, int w, int value)`

  Método que dada una matriz `m`, un índice de fila `row`, un índice de columna `col`, un ancho (tamaño de píxel) `w` y un valor `value`, asigna a todos los píxeles de la matriz `m` comprendidos entre `row, col` y `row+w, col+w` el valor `value`. Deberá tenerse en cuenta que tanto `row+w` como `col+w` pueden exceder los límites de la matriz `m`. Por lo tanto, para evitar un error de tipo `ArrayIndexOutOfBoundsException`, deberá asignarse este nuevo valor únicamente a aquellos elementos que realmente pertenecen a la matriz `m`.

Una vez implementados los métodos descritos anteriormente, se deberá implementar un método **public** en la clase `ImageUtils` con perfil:

`public static int [][] pixelar (int [][] m, int w)`

que dada una matriz `m` y un tamaño de píxel `w`, devuelva una nueva matriz resultante de pixelar `m`. Para ello deberá recorrerse la matrz `m` con saltos de `w` en `w`, tanto en filas como en columnas, y para cada una de estas posiciones `i,j` calcular (mediante el método `mediaSubMatriz`) la media de los valores comprendidos entre `i,j` e `i+w, j+w`, y almacenar seguidamente en la nueva matriz (mediante el método `ponerValoresSubMatriz`) dicha media en esas mismas las posiciones.

Finalmente, completa el método `pixelarPressed` de la clase `ImageProcessingActions` para que obtenga la imagen actual, la pixele mediante una llamada al método anterior y visualice la imagen resultante.

Actividad #4: entrelazar

Escribe un método `entrelazar` en la clase `ImageUtils` que reciba una matriz de enteros con la imagen y devuelva una nueva matriz resultante de intercambiar las filas pares e impares de la original. Esto es, se deberán intercambiar las filas 0 y 1, 2 y 3, 4 y 5, etc.

A continuación, completa el método `entrelazarPressed` de la clase `ImageProcessingActions` para que obtenga la imagen actual, la entrelace mediante una llamada al método anterior y visualice la imagen resultante.

Actividad #5: filtro con efecto dibujo (filtro de Sobel)

La mayoría de los filtros que se aplican sobre imágenes digitales se consiguen mediante las denominadas **operaciones de convolución**. Esta operación consiste en tratar cada píxel de la matriz original mediante otra matriz de menor tamaño denominada *kernel*. En la Figura 2 se muestra un ejemplo de cómo aplicar un *kernel* a un píxel dado de una matriz `m1` para obtener el píxel correspondiente en la matriz resultante `m2`. Esta operación se debe repetir para cada uno de los píxeles en la imagen original, y obtener así los píxeles correspondientes de la imagen resultante. Para los píxeles que se encuentran en el borde de la imagen se debe definir alguna otra operación específica, ya que no es posible superponer el *kernel* en esos casos.

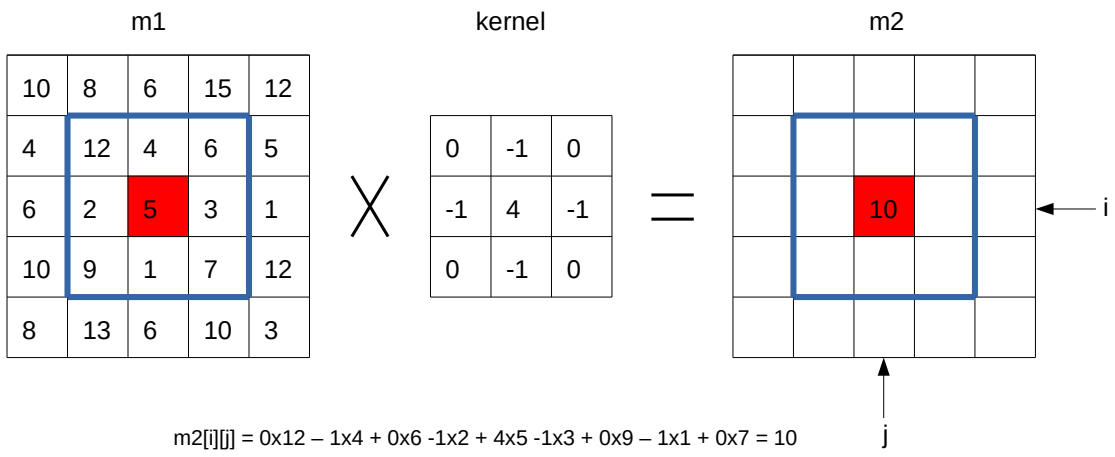

Figura 2: Ejemplo de convolución

Para calcular el nuevo valor del píxel marcado en rojo se suporpone el *kernel* sobre ese píxel y se realiza la operación: $0 \times 47 + 1 \times 22 + 0 \times 25 + 1 \times 52 + 4 \times 51 + 1 \times 50 + 0 \times 47 + 0 \times 47 + 0 \times 47$

En el caso particular del filtro de *Sobel* se utilizan dos kernels:

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$$

De este modo, dada una matriz m que contiene la imagen original, cada nuevo píxel $p2_{i,j}$ de la matriz resultante se calcula a partir de los píxeles $p_{i,j}$ de la matriz original como:

$gx = -p_{i-1,j-1} + p_{i-1,j+1} - 2p_{i,j-1} + 2p_{i,j+1} - p_{i+1,j-1} + p_{i+1,j+1}$
$gy = -p_{i-1,j-1} - 2p_{i-1,j} - 1p_{i-1,j+1} + 1p_{i+1,j-1} + 2p_{i+1,j} + p_{i+1,j+1}$
$p2_{i,j} = \sqrt{gx^2 + gy^2}$

Los píxeles del borde de la imagen resultante (primera y última fila y primera y última columna) se dejarán con su valor por defecto (cero), ya que en este caso particular no es posible realizar esta operación.

Escribe un método `sobel` en la clase `ImageUtils` que reciba una matriz de enteros con la imagen y devuelva una nueva matriz resultante de aplicar el filtro de Sobel tal y como se ha descrito anteriormente.

A continuación, completa el método `sobelPressed` de la clase `ImageProcessingActions` para que obtenga la imagen actual, le aplique un filtro de Sobel mediante una llamada al método anterior y visualice la imagen resultante.