Práctica 2: Reducción de imágenes usando familias A y B y la composición max-min

El objetivo de esta práctica es reducir la información que debemos almacenar de una imagen con la que podamos reconstruirla lo mejor posible. Es decir, para almacenar una imagen compuesta por Fil filas y Col columnas necesitaríamos guardar FilxCol píxeles. Si en lugar de almacenar todos los píxeles de la imagen disponemos de un método a partir del cual con unas pocas filas y columnas conseguimos reconstruir la imagen, esto ahorraría mucho espacio de almacenamiento.

Ejemplo: para almacenar una imagen de 600 filas y 1,000 columnas necesitamos almacenar 600,000 píxeles (números). Si en lugar de almacenar las 600 filas y las 1000 columnas almacenamos solamente 16 filas y 16 columnas ahorraríamos casi un 96% de espacio ($\frac{16*600+16*1000}{600*1000} * 100 = 4.2667\%\ del\ tamaño\ original$).

Existen numerosos métodos para realizar el proceso de reducción. En esta práctica vamos a implementar el que está basado en las familias A y B y la reconstrucción de la imagen utilizando la composición max-min a partir de dichas familias.

Vamos a dividir el desarrollo del método en varias etapas.

Etapa 1: Composición max-min y cálculo del error

Esta fase consiste en construir una matriz $\check{R}$ (imagen) de dimensión FilxCol a partir de dos matrices de menores dimensiones (A de dimensión CxFil y B de dimensión CxCol). Para ello deberemos aplicar la composición max-min:

$$\check{R}(x,y) = \bigvee_{i=1}^{C} A_i(x) \wedge B_i(y)$$

Es decir, para calcular el elemento situado en la posición (x, y) de la nueva matriz $\check{R}$ debemos coger la columna x de la matriz A y la columna y de la matriz B, hacer el mínimo elemento a elemento y finalmente, calcular el máximo de los mínimos obtenidos.

Una vez que tenemos la nueva matriz $\check{R}$ podemos calcular el error cometido con respecto a la matriz original R. Para ello deberemos aplicar la siguiente función de coste:

$$Q = \sum_{x=1}^{Fil} \sum_{y=1}^{Col} \left(R(x,y) - \check{R}(x,y) \right)^2$$

Se debe crear una función llamada **calculoError.m** que reciba como parámetros de entrada las matrices A, B y la matriz original R y que devuelva el error cometido al construir la matriz $\check{R}$.

Pruebas

Prueba 1: matrices numéricas

Probar la función con los siguientes parámetros:

- R = [0.5 0.2 0.9 0.83 0.87; 0.3 0.1 0.7 0.26 0.4; 0.5 0.26 0.69 0.26 0.35; 0.3 0.48 0.75 0.49 0.14] → dimensión 4x5
- A = [0.5 0.1 0.3 0.8] → dimensión 1x4
- B = [0.2 0.7 0.6 0.5 0.9] → dimensión 1x5

El error que se debe obtener es: 1.8658

Prueba 2: matrices para representación de imágenes

En MatLab una imagen está representada mediante una matriz de números comprendidos entre 0 y 255. Para obtenerla (matriz R), se debe ejecutar el siguiente comando:

```
>> R = double(imread('nombreImagen'));
```

Debemos normalizar esta matriz para que los números sean valores comprendidos entre 0 y 1. Para ello debemos dividir todos los elementos de R por 255.

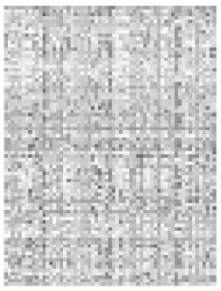
```
>> R = R/255;
```

Esta prueba consiste en leer y normalizar la imagen "circles.png", crear las matrices A y B de forma aleatoria utilizando 16 como valor de C y obtener el error cometido.

- El tamaño de la matriz A debe ser 16 x numeroFilasDeR
- El tamaño de la matriz B debe ser 16 x numeroColumnasDeR.

Etapa 2: Mejora de los valores de las familias A y B

Tal y como podemos apreciar en el resultado de la prueba 2, el error cometido al construir la imagen utilizando las matrices A y B es muy grande. Esto es debido a que dichas matrices están compuestas por valores aleatorios. Si mostráramos la imagen construida obtendríamos una imagen completamente de ruido como la siguiente:



Para tratar de reducir el error lo que debemos hacer es cambiar los valores que componen las matrices A y B. Esto nos lleva a que existen infinitas combinaciones de valores a probar, por lo que necesitamos un método que nos ayude a modificarlos de forma "inteligente", es decir, que la modificación de los valores esté relacionada con el error cometido.

Para este fin vamos a utilizar el algoritmo del gradiente, puesto que es un algoritmo de optimización (minimizar el error en este caso) que nos permite modificar los valores de los parámetros del sistema (los de las matrices A y B en este caso). La forma en que este algoritmo actualiza los valores de las matrices A y B es la siguiente:

$$A_i^{nuevo}(x) = A_i(x) - \alpha \frac{\partial Q}{\partial A_i(x)} = A_i(x) - \alpha \frac{\partial \sum_{x=1}^{Fil} \sum_{y=1}^{Col} \left(R(x,y) - \bigvee_{i=1}^{C} A_i(x) \wedge B_i(y)\right)^2}{\partial A_i(x)}$$

$$B_i^{nuevo}(x) = B_i(x) - \alpha \frac{\partial Q}{\partial B_i(x)} = B_i(x) - \alpha \frac{\partial \sum_{x=1}^{Fil} \sum_{y=1}^{Col} \left(R(x,y) - \bigvee_{i=1}^{C} A_i(x) \wedge B_i(y)\right)^2}{\partial B_i(x)}$$

Tal y como podemos ver, el nuevo valor para cada elemento de las matrices A y B se calcula restando al valor actual de cada elemento la derivada del error con respecto a sí mismo multiplicado por el parámetro α, que determina la velocidad de convergencia del algoritmo.

Vamos a ver cómo sería la actualización para el primer elemento de la matriz A, $A_1(1)$.

NOTAS

- La derivada de una constante es 0 y la derivada de una variable con respecto a sí misma es 1.
- El elemento $A_1(1)$ interviene solamente en el cálculo de todos los elementos de la primera fila de la nueva matriz $\check{R}$. Esto implica:
    - La derivada del error de cualquier elemento de la primera fila de $\check{R}$ tendrá valor si ese elemento es $A_1(1)$. En caso contrario será 0.
    - La derivada del error de cualquier elemento del resto de filas de $\check{R}$ es 0 puesto que $A_1(1)$ no interviene en su cálculo.

Atendiendo a las ecuaciones anteriores la actualización del elemento $A_1(1)$ se haría aplicando:

$$A_1^{nuevo}(1) = A_1(1) - \alpha \frac{\partial Q}{\partial A_1(1)}$$

Ahora vamos a desarrollar la derivada del error con respecto al elemento $A_1(1)$. Como hemos indicado anteriormente solo va a poder tomar valores en los elementos de la primera fila, es decir, tenemos que calcular lo siguiente:

$$\frac{\partial Q}{\partial A_1(1)} = \frac{\partial \left(R(1,1) - \bigvee_{i=1}^{C} A_i(1) \wedge B_i(1)\right)^2}{\partial A_1(1)} + \\ + \frac{\partial \left(R(1,2) - \bigvee_{i=1}^{C} A_i(1) \wedge B_i(2)\right)^2}{\partial A_1(1)} + \\ + \cdots + \\ + \frac{\partial \left(R(1,Col) - \bigvee_{i=1}^{C} A_i(1) \wedge B_i(Col)\right)^2}{\partial A_1(1)}$$

Ahora debemos derivar cada uno de los valores del sumatorio.

Hay que tener en cuenta que si $f(x) = u(x)^2$ su derivada es $f'(x) = 2 * u(x) * u'(x)$.

Debemos darnos cuenta de los siguientes dos hechos:

- El valor de cualquier elemento de R es constante con respecto a $A_1(1)$ y por tanto su derivada es 0.
- La derivada del máximo de los mínimos será 1 si el resultado es $A_1(1)$ y 0 en otro caso.

Por tanto para cada sumando tenemos:

$$\frac{\partial \left(R(1,1) - \bigvee_{i=1}^{C} A_i(1) \wedge B_i(1)\right)^2}{\partial A_1(1)} \\ = -2 * \left(R(1,1) - \bigvee_{i=1}^{C} A_i(1) \wedge B_i(1)\right) * \left(A_1(1) \wedge B_1(1) \geq \bigvee_{i=2}^{C} A_i(1) \wedge B_i(1)\right) \\ * \left(A_1(1) \leq B_1(1)\right)$$

$$\vdots$$

$$\frac{\partial \left(R(1,Col) - \bigvee_{i=1}^{C} A_i(1) \wedge B_i(Col)\right)^2}{\partial A_1(1)} \\ = -2 * \left(R(1,Col) - \bigvee_{i=1}^{C} A_i(1) \wedge B_i(Col)\right) * \left(A_1(1) \wedge B_1(Col) \geq \bigvee_{i=2}^{C} A_i(1) \wedge B_i(Col)\right) \\ * \left(A_1(1) \leq B_1(Col)\right)$$

Esta serie de ecuaciones la podemos reescribir de esta forma

$$\frac{\partial Q}{\partial A_1(1)} = -2 * \sum_{y=1}^{Col} \left(R(1,y) - \bigvee_{i=1}^{C} A_i(1) \wedge B_i(y) \right) * \left(A_1(1) \wedge B_1(y) \geq \bigvee_{i=2}^{C} A_i(1) \wedge B_i(y) \right) * \left(A_1(1) \leq B_1(y) \right)$$

Por tanto la actualización del elemento $A_1(1)$ queda como

$$A_1^{nuevo}(1) = A_1(1) + 2 * \alpha * \sum_{y=1}^{Col} \left(R(1,y) - \bigvee_{i=1}^{C} A_i(1) \wedge B_i(y) \right) * \left(A_1(1) \wedge B_1(y) \geq \bigvee_{i=2}^{C} A_i(1) \wedge B_i(y) \right) * \left(A_1(1) \leq B_1(y) \right)$$

En general, la actualización del elemento situado en la fila r y en la columna t de la matriz A sería

$$A_r^{nuevo}(t) = A_r(t) + 2 * \alpha * \sum_{y=1}^{Col} \left(R(t,y) - \bigvee_{i=1}^{C} A_i(t) \wedge B_i(y) \right) * \left(A_r(t) \wedge B_r(y) \geq \bigvee_{\substack{i=1 \\ i \neq r}}^{C} A_i(t) \wedge B_i(y) \right) * \left(A_r(t) \leq B_r(y) \right)$$

Escribe una función llamada **actualizaElementoDeA.m** que reciba como parámetros de entrada las matrices A, B y R, el parámetro α y los índices r y t que definen el elemento a actualizar y devuelva el valor actualizado.

Para actualizar los elementos de la matriz B seguimos el mismo razonamiento pero teniendo en cuenta que en lugar de intervenir en el cálculo los elementos de las filas de $\check{R}$ interviene en el cálculo de los elementos de las columnas.

Por tanto, la actualización del elemento situado en la fila r y en la columna t de la matriz B sería

$$B_r^{nuevo}(t) = B_r(t) + 2 * \alpha * \sum_{x=1}^{Fil} \left(R(x,t) - \bigvee_{i=1}^{C} A_i(x) \wedge B_i(t) \right) * \left(A_r(x) \wedge B_r(t) \geq \bigvee_{\substack{i=1 \\ i \neq r}}^{C} A_i(x) \wedge B_i(t) \right) * \left(B_r(t) \leq A_r(x) \right)$$

Escribe una función llamada **actualizaElementoDeB.m** que reciba como parámetros de entrada las matrices A, B y R, el parámetro α y los índices r y t que definen el elemento a actualizar y devuelva el valor actualizado.

Etapa 3: Minimización del error en la reconstrucción

En este punto **sabemos**:

- Construir la matriz $\check{R}$ a partir de las matrices A y B.
- Calcular el error cometido en la reconstrucción, Q.
- Actualizar cada elemento de las matrices A y B en base al error cometido.

Lo último que nos queda por hacer es desarrollar el algoritmo iterativo en el que en cada iteración se actualicen todos los valores de las matrices A y B de forma que se minimice el error. Este proceso iterativo se repetirá hasta que se cumpla una de las dos siguientes condiciones:

- La mejora del error de una iteración a otra esté por debajo de un umbral

$$Q^{iter} - Q^{iter-1} < umbral$$

- Se alcancen el número máximo de iteraciones permitidas

$$iter > iterMax$$

Desarrolla una función llamada **reduccionImagen.m** que reciba como parámetros de entrada el nombre de la imagen a reducir, el valor del parámetro α, el umbral de error, el número máximo de iteraciones y el número de filas y columnas que deseamos guardar (parámetro C) y nos devuelva las matrices A y B obtenidas.
Esta función debe hacer lo siguiente:

- Lectura y normalización de la imagen a tratar.
- Creación de las matrices aleatorias A y B.
- Construcción de la matriz $\check{R}$ inicial y calcular su error.
- Proceso iterativo hasta que se cumpla una condición de parada
  - Actualización de todos los valores de la matriz A
  - Actualización de todos los valores de la matriz B
  - Construcción de la matriz $\check{R}$ y obtener su error

Prueba: Reducción de la imagen circles.png

Probar utilizando la siguiente configuración

- C = 16
- α = 0.005
- umbral = 0
- maxIter = 200

Probar el algoritmo con una foto de vuestras caras, ¿Qué tal resultado se obtiene?