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

Lo que vamos a hacer en esta práctica es calcular una familia de c conjuntos difusos A y c conjuntos difusos B tales que al componerlos utilizando la expresión anterior obtengamos una aproximación de la imagen original R. Es evidente que si el valor de c no es muy elevado, estaremos almacenando menos información en nuestros conjuntos A y B que en la imagen original.

Voy a hacer una descripción de lo que el método nos dice que tenemos que implementar:

Para empezar, se debe leer y normalizar la imagen original R. A continuación, deberemos crear c vectores A y c vectores B de forma aleatoria. El valor c es el índice de Schein y será mayor o igual a 2. El tamaño de cada uno de los vectores A es igual al número de filas de la imagen original y el tamaño de cada uno de los vectores B es igual al número de columnas de la imagen original. Para ello, consideramos que el problema de descomponer la relación R en conjuntos A y B puede ser visto como un problema de optimización que trate de minimizar la siguiente función de coste :

$$Q = \sum_{(x,y) \in X \times Y} (R(x,y) - \tilde{R}(x,y))^2$$

Para obtener una imagen a partir de las dos familias obtenidas deberemos aplicar la composición max-min:

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

El proceso de encontrar estos valores de A y B va a ser iterativo. Denotaremos por iter el no de iteración que estamos tratando actualmente. Así, la función de coste en cada iteración puede reescribirse como :

$$Q_{(iter)} = \sum_{(x,y) \in X \times Y} \left(R(x,y) - \bigvee_{i=1}^{c} A_i^{(iter)}(x) \wedge B_i^{(iter)}(y) \right)$$

Para actualizar los valores de A y B de la anterior formula, utilizamos:

$$A_i^{(\text{iter}+1)} = A_i^{(\text{iter})} - \alpha \frac{\partial Q_{\text{iter}}}{\partial A_i^{(\text{iter})}(x)}$$

$$B_i^{(\text{iter}+1)} = B_i^{(\text{iter})} - \alpha \frac{\partial Q_{\text{iter}}}{\partial B_i^{(\text{iter})}(y)}$$

Donde el valor recomendado para el parámetro α es 0.005 y las derivadas se corresponden con las siguientes expresiones (siendo $A_i = A_i^{iter}$ y $B_i = B_i^{iter}$):

$$\frac{\partial Q}{\partial A_l(x')} = -2 \sum_{y \in Y} \left(R(x', y) - \bigvee_{i=1}^{c} (A_i(x') \wedge B_i(y)) \right) \varphi \left((A_l(x') \wedge B_l(y), \bigvee_{i=1, i \neq l}^{c} (A_i(x') \wedge B_i(y)) \right) \psi (A_l(x') , B_l(y))$$

$$\frac{\partial Q}{\partial B_l(y')} = -2 \sum_{x \in X} \left(R(x, y') - \bigvee_{i=1}^{c} (A_i(x) \wedge B_i(y')) \right) \varphi \left(A_l(x) \wedge B_l(y'), \bigvee_{i=1, i \neq l}^{c} A_i(x) \wedge B_i(y') \right) \psi (B_l(y'), A_l(x))$$

Las expresiones de las funciones a aplicar son las siguientes:

$$\varphi(a,b) = \begin{cases} 1, & \text{if } a \geq b, \\ 0, & \text{otherwise,} \end{cases}$$

$$\psi(a,b) = \begin{cases} 1, & \text{if } a \leq b, \\ 0, & \text{otherwise,} \end{cases}$$

La última cuestión que hay que tener en cuenta es cuándo debemos terminar nuestra ejecución. Al ser un proceso iterativo, necesitamos una condición de parada. Vamos a considerar que hemos llegado al final de nuestro proceso cuando en dos iteraciones consecutivas, la diferencia entre la función de coste Q en cada una de ellas sea lo suficientemente pequeña, es decir,

$$Q^{(\text{iter}+1)} - Q^{(\text{iter})} < \varepsilon,$$

Para el algoritmo que he construido la tolerancia es un parámetro de entrada, en todos los ejemplos he usado el valor 0.1.

Analizar:

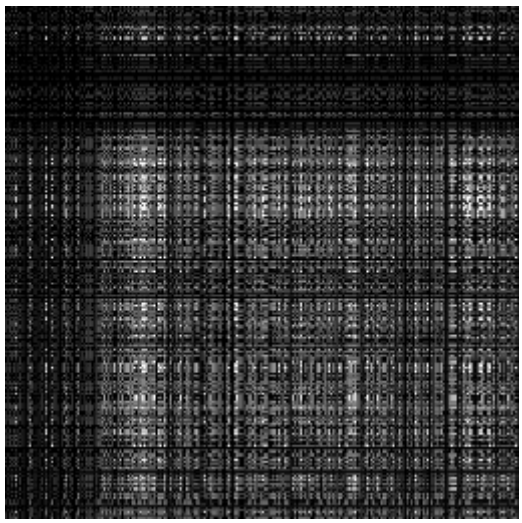
Tiempo de ejecución del algoritmo

En mi opinión, el tiempo de ejecución con un 'c' bajo, es más que aceptable y se obtiene una imagen realmente aceptable en calidad. Al pasar 'c' de un valor 10, el algoritmo empieza a tardar demasiado tiempo en correlación a la mejora que se obtiene respecto de un valor de 'c' menor. También se puede controlar la duración del algoritmo aumentando el valor de 'alpha', pero también se obtienen resultados peores (no mucho peores). Estudiando un poco 'alpha', se observa que el algoritmo hace bastantes menos iteraciones con un resultado aceptable hasta valores de 'alpha' entre 0.01 y 0.02.

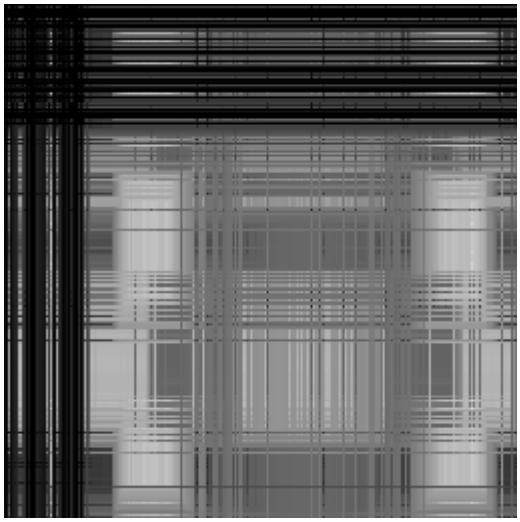
Cómo afecta el parámetro c a la calidad del resultado

Al aumentar el valor de 'c', conseguimos una mayor calidad en la imagen que se nos devuelve. Esto ocurre porque almacenamos más vectores con información sobre la imagen. También hay que tener en cuenta que al aumentar este parámetro, el tiempo de ejecución del algoritmo es bastante mayor.

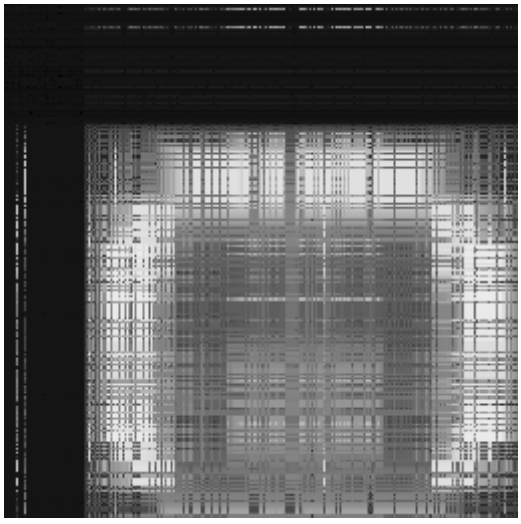
Ejemplos con diferentes imágenes



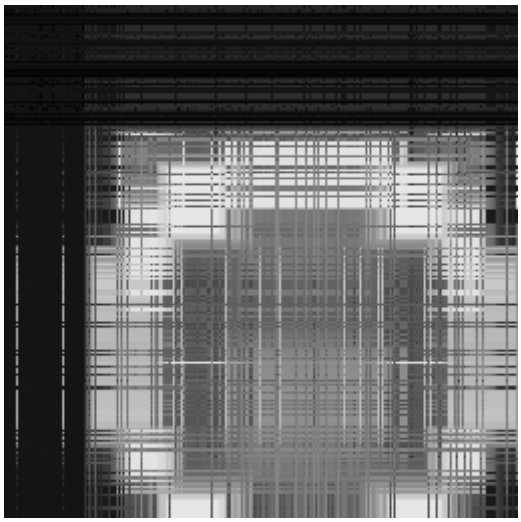
→ c=1 iteraciones=1 3seg.



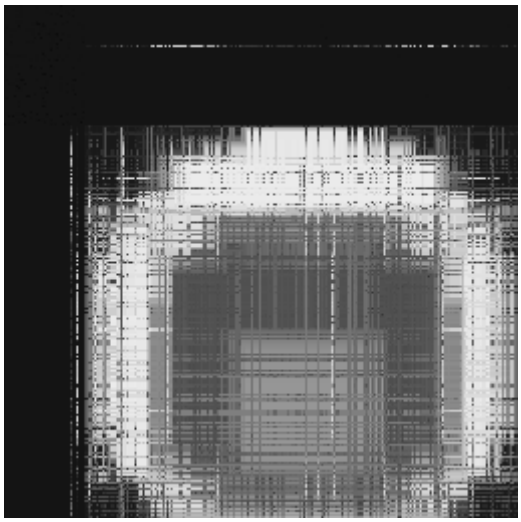
→ c=2 iteraciones=9 59seg.



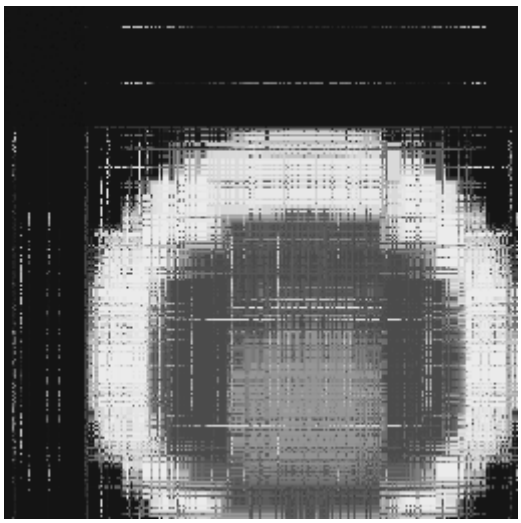
→ c=4 iteraciones=45 453seg.



→ c=5 iteraciones=35 586seg.



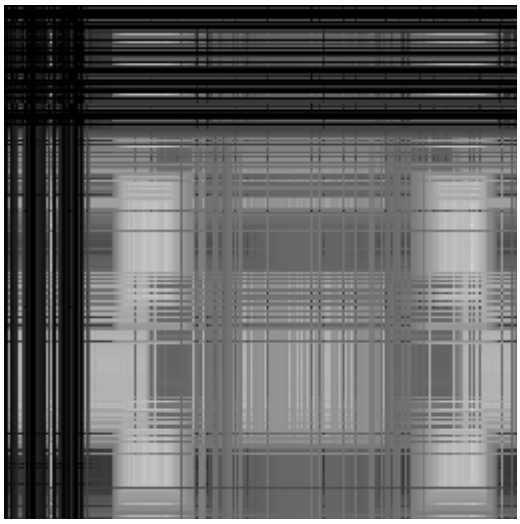
→ c=10 iteraciones=90 3858seg.



→ c=20 iteraciones=167 15259seg.

¿Qué ocurre si cambiamos la expresión de la derivada de Q por la siguiente expresión?

El algoritmo apenas cambia su forma de funcionar, pero el tiempo en el que lo hace se reduce considerablemente, consiguiendo resultados casi idénticos (las iteraciones también son las mismas). Excepto con una 'c' baja:



Con 'c' = 2; el de la izquierda es el de la nueva expresión. A partir de aquí los resultados son casi idénticos (en lo que a imagen se refiere). Voy a hacer una comparación de tiempos:

'c' / tiempo	Grad1	Grad2
2	59 segundos	79 segundos
4	453 segundos	176 segundos
5	586 segundos	
10	3858 segundos	1144 segundos
20	15310 segundos	4259 segundos

Podemos observar que el primer método casi cuadruplica el tiempo de ejecución del segundo. Y que sólo obtenemos un resultado anómalo en la situación de 'c' = 2.

FIN PRÁCTICA 2