Proyecto 2 : Mosaico de fotos

Un mosaico fotográfico es una imagen cuya composición está basada en varios fragmentos los cuales son imágenes mas pequeñas, que vamos a llamar miniaturas. El mosaico es creado a partir de una imagen de origen, que vamos a llamar **source**. Las miniaturas se fabricaran a partir de una colección de imágenes, llamada **banco de imágenes**.

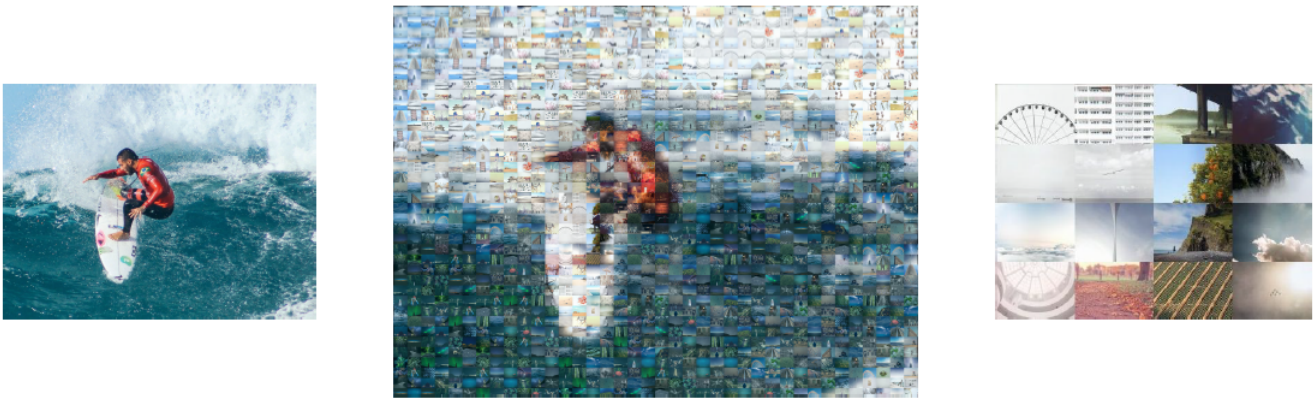

Fig 1 - Mosaico fotográfico de un surfista compuesto de 1600 miniaturas - De izq. a derecha : la imagen source, el mosaico fotográfico y 16 miniaturas (Detalle del pie del surfista)

La construcción de un mosaico fotografico y su calidad son determinados según varios parámetros :

- La estructura de la imagen source utilizada (tamaño, forma, etc..) ;
- El número de imágenes del banco de imágenes ;
- Los algoritmos utilizados para :
  - seleccionar las imagenes correctamente (partie IV) ;
  - redimensionar las imagenes (partie II) ;

En este proyecto los mosaicos se crearán en forma de bloques rectangulares regulares, es decir, el mosaico estará construido con miniaturas rectangulares, todas de la misma dimensión dispuestas lado a lado.
Es recomendable instalar los modulos math, numpy, matplotlib e importarlos en el proyecto. Para importalos se puede utilizar la siguiente instrucción :

```
import math, numpy as np, matplotlib.pyplot as plt, random
```

A lo largo de todo el proyecto :

- El termino “lista” se aplicará a cualquier variable de tipo `list`.
- Los términos “vector” o “tabla” designan variables del tipo `np.ndarray`, indiferentemente de su dimensión.
- El termino “secuencia” representa cualquier serie iterable e indexable independientemente su tipo de Python ; e.g, una tupla, una lista y un vector son todas series iterables.

Los encabezados de las funciones se anotan para precisar los tipos de cada parámetro y el tipo del resultado de la siguiente forma :

```
def unaFuncion(n :int, X :[float], c : str, u) -> np. ndarray :
```

Esto significa que la función `unaFuncion` toma cuatro argumentos n, X, c y u donde n es un entero, X es una lista de flotantes, c una cadena de caracteres y el tipo de u no se precisa. Esta función debe devolver una tabla de numpy. Las funciones se encuentran anexadas en el archivo `proyecto2.py`.
El uso de comentarios será bien apreciado.

I Pixeles e imágenes

1.1 Pixeles

Un pixel es un elemento de color homogéneo utilizado para representar una imagen digital. El color de un pixel se puede representar de diversas formas. Un método bastante común es la descomposición en tres componentes que corresponden a los colores Rojo, Verde y Azul (R, G, B). Cada uno de los tres componentes indica la intensidad del color, donde 0 indica la ausencia de dicho color. Por lo tanto, el vector $(0,0,0)$ indica el color negro.

Q 1. Se supone que cada uno de los tres componentes RGB de un pixel se representa por un numero entero positivo o nulo, codificado en 8bits. ¿Cuántos colores diferentes se pueden representar con un solo pixel?

Q 2. A continuación se representa cada pixel por un vector (tabla de numpy de una sola dimensión) de tres elementos que son enteros de tipo np.uint8 (entero no negativo codificado en 8bits). A partir de ahora, consideraremos este tipo de variable como `pixel`. De una instrucción que permita crear un vector correspondiente a un pixel blanco.

Q 3. En Python, como en muchos lenguajes de programación, las operaciones de suma, resta, multiplicación, división entera, modulo y potencia (los operadores +, -, *, //, %, **, respectivamente) se aplican a variables del mismo tipo y devuelven resultados del mismo tipo. Esto puede traer casos de *desbordamiento* y, por lo tanto, errores de calculo ya que los *desbordamientos* ocurren de manera "silenciosa", es decir, no producen errores mientras el programa se ejecuta.
El operador de división (/) entre dos enteros siempre produce un resultado del tipo float, incluso si dicha división es exacta ($12/2 \rightarrow 6.0$). Lo mismo ocurrirá con cada funcion que llame a este operador de manera implícita como `np.mean`.
Considerando `a = np.uint8(280)` y `b = np.uint8(240)`. ¿Qué valen a, b, a+b, a-b, a//b y a/b?

Q 4. Para representar una imagen en escala de grises (donde 0 es negro y 1 es blanco), se puede remplazar cada pixel por un solo entero, cuyo valor corresponde a la mejor aproximación entera del promedio de los tres componentes RGB de cada pixel. Escribir la función

```
def gris (p :pixel) -> np.uint8 :
```

La cual calcula el nivel de gris correspondiente a cada pixel p.

1.2 Imágenes

Una imagen en escala de grises de talla $w \times h$ (w pixeles de ancho, h pixeles de alto) está asociada a una tabla de bytes (tipo `np.uint8`) de dos dimensiones de h líneas y de w columnas. Cada elemento de dicha tabla representa el nivel de gris de cada pixel. Por lo tanto, la tabla de dos dimensiones de la imagen `img1` se define como :

```
img1=np.array([[ 85,   0, 127, 170, 85, 150],
   [119, 102, 102, 123, 81, 170],
   [255, 170,  90, 112, 63,  97],
   [171, 212, 225, 186, 162, 171]], np.uint8}
```

La cual define una imagen de dimensión 6×4 representada en la figura 2.

Nota : A partir de ahora, cuando nos referirimos al tipo `imagen`, nos referiremos a tablas de bytes de dos dimensiones. (Es decir, trabajaremos solo en escala de grises)

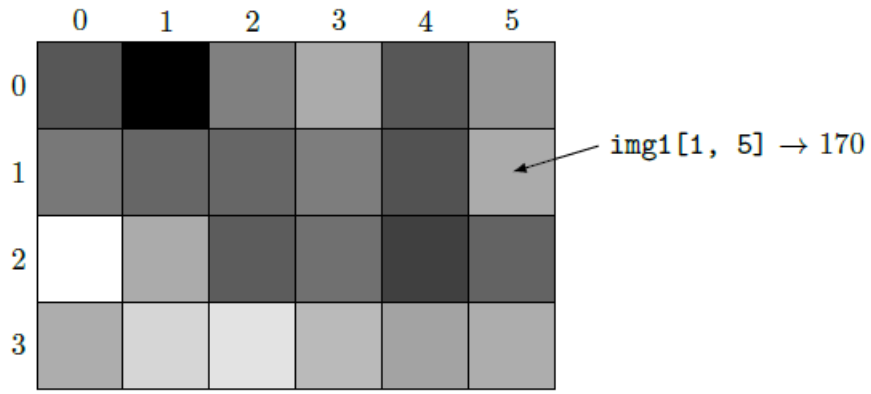

Fig 2 - Visualización de la imagen img1 -

Q 5. Implementar la función

```
def conversion(a :np.ndarray) -> image :
```

la cual genera una imagen en niveles de gris correspondiente a la conversión de la imagen en color `a`.

II Redimensión de imágenes

En esta parte nos interesamos en diferentes algoritmos de redimensión de una imagen A, de tamaño $W \times H$ (W pixeles de ancho por H pixeles de alto) en una imagen a, de tamaño $w \times h$ (w pixeles de ancho por h pixeles de alto).

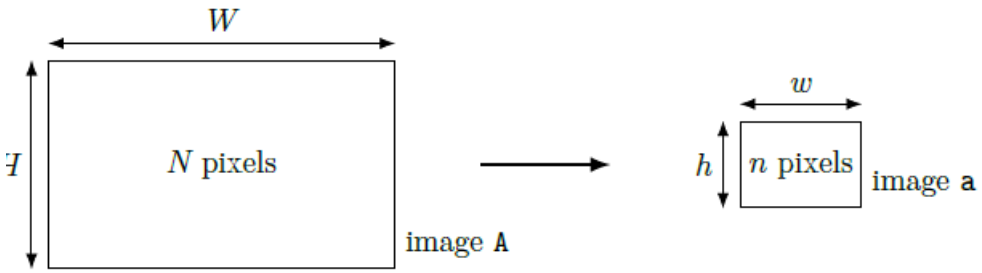

Fig 3-Redimensión de una imagen-

Sin embargo, Redimensionar una imagen implica aumentar o reducir la cantidad de pixeles en total. La nueva imagen redimensionada debe permanecer lo más fiel posible a la imagen original y, para lograr este objetivo, existen algoritmos que permiten determinan qué valor de pixeles asignar en la nueva imagen redimensionada.

Nota : para simplificar este problema lo más posible, vamos a considerar que todas las redimensiones cumplen las siguientes condiciones :

- En el caso de reduccion, W/w y H/h son enteros
- Para el caso de agrandar una imagen, w/W y h/H son enteros.

2.1 Algoritmo de Interpolación por el vecino más cercano

Esta interpolación está definida por la fórmula $a(i,j) = A\left(\left\lfloor \frac{iH}{h} \right\rfloor, \left\lfloor \frac{jW}{w} \right\rfloor\right)$ donde $\lfloor x \rfloor$ es la parte entera de x.

Q 6. Desarrollar la función

```
def vecinoProximo(A :imagen, w  :int, h  :int) -> imagen :
```

que devuelve una nueva imagen correspondiente a la redimensión de la imagen A, de talla w × h utilizando la Interpolación por el vecino más cercano.

2.2 Algoritmo de reducción por medias locales

Suponemos que las dimensiones de la imagen `a` dividen aquellas de la imagen `A`. Es decir : H/h y W/w son enteros. Para mejorar la calidad de la reducción, se propone la función `mediaLocal`.

```
def mediaLocal(A:np.ndarray, w:int, h:int):
    a=np.empty((w,h),np.uint8)
    H,W=A.shape
    ph,pw=H//h, W//w
    for I in range(0,H,ph):
        for J in range(0,W,pw):
            a[I//ph,J//pw]=round(np.mean(A[I:I+ph, J:J+pw]))
    return a
```

Q 7. Explique en algunas líneas como funciona este código.

2.3 Optimizacion de la reducción por medias locales

Para acelerar el calculo de la media local, precalculamos para cada imagen su tabla de suma. La tabla de suma de una imagen A , que se define como una tabla de H lineas y W columnas, es la tabla S de $H+1$ lineas y $W+1$ columnas, definida como :

$$\forall l \in [\![0, H]\!], \quad \forall c \in [\![0, W]\!], \quad S(l,c) = \sum_{\substack{0 \leq i < l \\ 0 \leq j < c}} A(i,j),$$

Nótese que $S(0,0) = 0$

Q 8. Escribir la función

```
def tablaSuma(A :image)-> np. ndarray :
```

que calcule la tabla de suma de una imagen A.

Q 9. Suponemos, como siempre, que las dimensiones de la imagen A dividen aquellas de la imagen a : H/h y W/w son enteros. Se propone la funcion `reduccionSuma1`, que toma como parametro la imagen A y su tabla de suma S, asi que las dimensiones de la imagen que se desea obtener.

```
101  def reduccionSumas1(A:np.ndarray, S:np.ndarray, w:int, h:int) :
102      a = np.empty((h, w), np.uint8)
103      H, W = A.shape
104      ph, pw = H // h, W // w
105      nbp = ph * pw
106      for I in range(0, H, ph):
107          for J in range(0, W, pw):
108              X = (S[I+ph, J+pw] - S[I+ph, J]) - (S[I, J+pw] - S[I, J])
109              a[I // ph, J // pw] = round(X / nbp)
110      return a
```

Explicar en algunas líneas el principio de `reduccionSuma1`.

Q 10. Mostar que la función `reduccionSuma2` cuyo código se provee a continuación da el mismo resultado que `reduccionSuma1`.

```
def reduccionSumas2(A:np.ndarray, S:np.ndarray, w:int, h:int):
    H, W = A.shape
    ph, pw = H // h, W // w
    sred = S[0:H+1:ph, 0:W+1:pw]
    dc = sred[:, 1:] - sred[:,:-1]
    dl = dc[1:, :] - dc[:-1, :]
    d = dl/ (ph * pw)
    return np.uint8(d.round())
```

III Preparando el banco de imágenes

El siguiente paso es preparar la banca de imagenes que serán utilizadas como miniaturas en el mosaico, para esto es necesario cargar las imagenes, transformarlas de RGB a escala de grises y redimenzionar las imagenes al tamano deseado.

Las imagenes son proporcionada en el documento `imagenes.zip`, se pueden extraer en la carpeta de su preferencia, la función `cargar_imagenes` incluida en el codigo .py toma como parametro la ruta de la carpeta donde se extraerán las imágenes, luego las tranforma en una imagenRGB y las almacena en una lista que será devuelta.

Q 11. Implemente la función

```
def listaRGBtoGris(A :[np.ndarray])-> [image] :
```

que tome como entrada una lista de imagenes en RGB y devuelva una lista con las mismas imagenes en escala de gris.

Q 12. Desarrolle la función

```
def listaRedim(A :[image],h,w)-> [image] :
```

que tome como entrada una lista de imagenes en escala de gris, la altura h y el ancho w a las que se desean redimensionar las imagenes. y que devuelva una lista con las mismas imagenes redimensionadas, utilizando los métodos utilizados en la sección 2.

Q 13. Cada imagen de la banca de imagenes tiene como dimensiones H :500 y W :375 pixeles. Las imagenes source miden cada una H :2500 y W :1875. A qué tamano se deben redimensionar las miniaturas para poder hacer un fotomosaico con 50 miniaturas de cada lado que mida 3750x5000 pixeles? Redimensione las miniaturas de la banca de imágenes a dicha altura.

IV Posicionamiento de miniaturas

Consideramos el caso donde el moisaco estará constituido de p miniaturas de alto por p de ancho. El número total de miniaturas es de $r = p^2$.

Q 14. Implemente la función

```
def initMosaico (source :image, w :int, h : int, p : int) -> image :
```

que toma como parámetro la imagen source, las dimensiones w y h de cada miniatura y el numero p de miniaturas por lado. Esta función reenvia una version redimensionada de la fuente, de la misma talla que el mosaico fotografico final.

Q 15. A partir de ahora, llamaremos **bloque** a cada porción de la imagen **source** que será reemplazada por una miniatura. Para determinar que miniatura de la banca de imagenes es la que mejor conviene para cada bloque, definimos la distancia L_1 entre dos imagenes a y b del mismo tamano $w \times h$ como :

$$L_1(a,b) = \sum_{\substack{0 \leqslant i < h \\ 0 \leqslant j < w}} |a(i,j) - b(i,j)|.$$

Desarrolle la función

```
def L1(a :image, b : image) -> int :
```

que calcula las distancias L_1 entre dos imagenes de la misma talla.
Q 16. Escribir la función

```
def escogerMiniatura(bloque :image, miniaturas : [image]) -> int :
```

que toma como parametro una imagen que corresponde a un bloque y una lista de miniaturas. La función devuelve el indice i tal que `L1(bloque, miniaturas[i])` es el valor minimo (o cualquiera si hay varios que convienen). Esta función no debe modificar la lista de miniaturas.

Q 17. Implementar, con la ayuda de las funciones anteriores, la siguiente funcion :

```
def construirMosaico(source :image, miniaturas : [image], p :int) -> image :
```

la cual construye un mosaico fotográfico de la imagen source con p miniaturas por cada lado.

Q 18. Intente crear un mosaico con cualquiera de las imagenes source en el proyecto con p=5 y, las miniaturas reducidas segun la pregunta 13. Anexe su resultado.

Q19. **Bonus :** Para garantizar una variedad en las miniaturas. Como se puede modificar el algoritmo para evitar que se repitan las mismas miniaturas en bloques vecinos? Proponga una solución modificando las funciones necesarias y pruebelo con una imagen source.