Coordenadas Laplacianas 2D

Por: Edinson Orlando Dorado Dorado
Código: 1941966

1. Matriz W

i y j de I (intensidades, np array de la imagen, width*height) es diferente a i y j de W (width*height, width*height)

Imagen (width,height)

1.1 Constante Beta

```
beta=0.1
```

1.2 Hallar la raíz cuadrada del valor absoluto de la diferencia al cuadrado de las intensidades.

Escalar—> $||I_i - I_j||_\infty^2$

1.3 Hallar sigma

$$\sigma = \max_{(P_i,P_j)\in E} ||I_i - I_j||_\infty$$

Illustrative image pixels

P_1	P_2	P_3
P_4	P_5	P_6
P_7	P_8	P_9

$$w_{ij} = \exp\left(-\frac{\beta||I_i - I_j||_\infty^2}{\sigma}\right)$$

(width*height, width*height)

1. **Crear la matriz pesos W:** Esta matriz la cual tiene una dimensión de (width*height,width*height), donde width es el ancho de la imagen en px y height es el alto de la imagen en px, es donde se guardan los ponderados asignados a las artistas que conectan cualquier voxel (píxel (i,j) en done i < height y j < width) con otro voxel (pixel de coordenadas (i,j) en done i < height y j < width). Para calcular la matriz, a su vez se puede seguir estos pasos:

2. Matriz Laplaciana $\mathbf{L = D - W}$

2.1 Crear el vector D

2.1 **Crear el vector D:** Este vector representa la suma de los pesos de todas las conexiones para cada nodo, y es crucial para la formulación de la matriz Laplaciana. También se puede entender como la valencia de los pesos de cada voxel (voxel siendo un píxel de coordenadas (i,j) en done i < height y j < width).

assigned to edge (P_i, P_j) of the graph. The set $N(i) = \{j : (P_i, P_j) \in E\}$ represents the indices of the pixels P_j that share an edge with pixel P_i and $d_i = \sum_{j \in N(i)} w_{ij}$ is the weighted valency of P_i.

Figure 23: Vector D

```
import numpy as np
D = np.sum(W, axis=1)
```

La matriz W, debido a que en cada fila i, tiene 0's en las columnas en donde no tiene conexión y un valor distinto de 0 cuando si tiene conexión y por tanto un peso, se suman todos los pesos de esa fila para obtener el resultado de la fila i en D. Este vector tiene una dimensión de (width*height), siendo unidimensional.

```
>>> np.sum([[0, 1], [0, 5]], axis=1)
array([1, 5])
```

2.2 Calcular la Matriz Laplaciana

Para calcular la matriz Laplaciana, haciendo la diferencia de D y W, hay que tener en cuenta que W tiene una dimensión de (width*height,width*height) o sea una arreglo de dos dimensiones, mientras que D tiene una dimensión por ahora de (width*height) o sea unidimensional, por lo que antes de realizar la resta, se debe transformar a D en un array de dos dimensiones (de igual número de filas y columnas que W), y, según el documento base, los valores actuales en D, deben de quedar en la diagonal, mientras que los demás valores que no están en la diagonal, deben ser 0's. Así pues lo que se hace es crear una matriz esparsa de D, lo cual es mucho mas eficiente que convertir D a una matriz de dimensiones (width*height,width*height) en donde muchas de esas celdas serán 0's.

```
import scipy.sparse as sp
L = sp.diags(D) - W
```

3. Crear el sistema

```
A = I_s + L2
solve = factorized(A)
x = solve(b)
```

3.1 Crear L2

```
import numpy as np
L2 = L.dot(L)
```

3.2 Entender la matriz de índices de los Píxeles

3.2 **Entender la matriz de índices de los Píxeles:** Esta matriz de ejemplo tiene 9 píxeles, con nombres P1 hasta P9. Los números del 1 al 9 son importantes para transformar posiciones (i,j) en esos números del 1 al 9 (útiles por ejemplo para transformar valores de semillas a valores en la matriz Is). Cabe resaltar que i puede tomar valores desde 0 hasta height-1 y j puede tomar valores desde 0 hasta width-1 en esta matriz de nueve Píxeles.

Illustrative image pixels

P_1	P_2	P_3
P_4	P_5	P_6
P_7	P_8	P_9

3. Crear el sistema

```
A = I_s + L2
solve = factorized(A)
x = solve(b)
```

3.3 xB y xF

```
seeds = [
# Fondo
(5, 5), (5, 55), (30, 60), (0, 15), (10,0),(20,0),
(30,0),(20,40),(60,50),(60,0),(70,0),(60,80), (2,20), (40,0),
# Perro izquierdo
(15, 20), (40, 20), (60, 20), (65, 20), (10,20),
(65,25),(55,20),(40,25),(35,20),(70,30),(25,20),(15,20),(45,19)
]
labels = [
'B', 'B', 'B', 'B','B', 'B', 'B', 'B','B', 'B', 'B', 'B','B', 'B', 'B', 'B','B','B'
'F', 'F', 'F', 'F','F', 'F', 'F', 'F','F','F','F','F','F'
]
xB = 1
xF = 0
```

El arreglo de seeds es un arreglo de tuplas de 2 elementos, cada tupla es una posición en la imagen. En seeds, en este caso, primero se especifican algunas anotaciones del fondo (Background) y algunas anotaciones de la parte que se quiere obtener (Foreground). Después se encuentra el arreglo de labels, que es donde se escriben las anotaciones de cada tupla en seeds. En este arreglo labels, se asigna 'B' si la tupla en esa misma posición en seeds es Background, o 'F', si la tupla en esa misma posición en seeds es Foreground. Por último se asigna un valor tanto a xB como a xF, en donde xB debe de ser mayor a xF.

3.4 Crear la matriz Is

Crear la matriz Is: Para hallar la matriz Is, se debe crear una matriz de dimensiones (width*height,width*height) en donde cada semilla b ya sea de Background o Foreground, se debe transformar a una coordenada en la matriz de indicies. Una vez obtenido el número del Píxel, por ejemplo un número i, se debe poner en la posición (i,i) en la matriz Is, un 1. Recordar que se debe hacer esto para todas las semillas, tanto de Background como de Foreground. Las demás posiciones deben de ser 0.

where $\mathbf{I_S}$ is a diagonal matrix such that $I_S(i,i) = 1, i \in S = B \cup F$, and zero, otherwise, b is the vector where

```
I_s = sp.lil_matrix((num_pixels, num_pixels))
for (i, j), label in zip(seeds, labels):
    idx = indices[i, j]
    I_s[idx, idx] = 1
```

3. Crear el sistema

```
A = I_s + L2
solve = factorized(A)
x = solve(b)
```

3.5 Entender y crear el arreglo b

```
import numpy as np
b = np.zeros(num_pixels)
for (i, j), label in zip(seeds, labels):
    idx = indices[i, j]
    b[idx] = xB if label == 'B' else xF
```

Entender y crear el arreglo b: El arreglo b, es un arreglo unidimensional de longitud width*height en donde según el número del Píxel en la matriz de índices que representa tupla en seeds, irá el valor de xB si ese Píxel a sido escogido como Background o xF si ese Píxel a sido escogido como Foreground, o 0 si no está relacionado con ninguna semilla.

```
A = I_s + L2
solve = factorized(A)
x = solve(b)
```

4. Resolver Sistema usando Factorización de Cholesky

```
import scipy.sparse as sp
# Usando factorización Cholesky para resolver el sistema
A = I_s + L2
A = sp.csr_matrix(A)
solve = factorized(A)  # Factorización Cholesky
#asignar a x la solución del sistema
x = solve(b)
```

Resolver el sistema usando Factorización de Cholesky Finalimente con todo lo obtenido, se puede formar un sistema el cual puede ser resuelto por medio de la factorización de Cholesky. Esto da un resultado x, el cual es un arreglo, es único y tiene una longitud de width*height. En x está contenida la solución al sistema.

```
A = I_s + L2
solve = factorized(A)
x = solve(b)
```

5. Asignar etiquetas al resultado de resolver el sistema

$$y_i = \begin{cases} x_B, & \text{if } x_i \geq \frac{x_B + x_F}{2} \\ x_F, & \text{otherwise} \end{cases}$$

```
segmented_image = x.reshape((height, width))
```

```
import numpy as np
def apply_labels(segmented_values,image, xB, xF):
    # Calcular el umbral basado en xB y xF
    threshold = (xB + xF) / 2

    # Asignar etiquetas basado en el umbral
    labels = np.where(segmented_values >= threshold, xB, image)

    return labels
```

Por el parámetro **segmented_values** es donde entra el **x** hallado final de la forma: **x.reshape((height,width))**

Asignar etiquetas al resultado de resolver el sistema Ya con el arreglo x, se procede a clasificar cada elemento de x en Background o Foreground. Para obtener una mejor representación visual, se hizo pequeño cambio en la implementación a como se describe en el documento guía (en vez de colocar xF, se colocó el valor de la intensidad en la imagen original, con el fin de ver en la imagen original que fue lo que se segmentó).