Práctica 5 "Superpixeles e Histogramas"

Leyva P. José Luis , *Facultad de Ingeniería, UNAM,* Lopez Cruz Luis Enrique, *Facultad de Ingenieria, UNAM*
Najera S. Isaac Israel , *Facultad de Ingeniería, UNAM,* Sandoval M. Joaquin, *Facultad de Ingeniería, UNAM,*

***Resumen*—El presente reporte de practica expone la implementación de un proyecto cuyo propósito es el reconocimiento de características, enfocado a la distinción del del ventrículo izquierdo de un corazón. Este se basa en la teoría de generación de superpixeles y calculo de distancias entre histogramas.**

Como se vera a lo largo del texto se realizaron los cálculos y procesos necesarios para obtener un discriminante que nos permita distinguir entre superpixeles que representa una región de interés.

***Index Terms*—IEEE, IEEEtran, journal, LaTeX, paper, template.**

I. Objetivo

El alumno generará superpixeles dentro de una imagen para lograr la extracción de características.Aprenderá a calcular el histograma local de la Imagen.Entenderá cómo calcular la distancia entre dos histogramas. Evaluará la eficiencia de su algoritmo mediante métricas utilizadas en aprendizaje de máquina.

II. Introducción

II-A. Tipos de superpixeles

LOs algoritmos de segmentación de superpíxeles comúnmente utilizados son SLIC, SEEDS y LSC.

II-A1. SLIC: Este algoritmo genera superpíxeles agrupando píxeles en función de su similitud de color y proximidad en el plano de la imagen. Esto se hace en el espacio [labxy] de cinco dimensiones, donde [lab] es el vector de color de píxeles en el espacio de color CIELAB y xy es la posición del píxel. Necesitamos normalizar las distancias espaciales para usar la distancia euclidiana en este espacio 5D porque la distancia máxima posible entre dos colores en el espacio CIELAB es limitada, mientras que la distancia espacial en el plano xy depende del tamaño de la imagen.

II-A2. SEEDS: El algoritmo utiliza un algoritmo eficiente de escalada de colinas para optimizar la función de energía de los superpíxeles que se basa en histogramas de color y un término límite, que es opcional. La función de energía fomenta que los superpíxeles sean del mismo color, y si se activa el término límite, los superpíxeles tienen límites suaves y son de forma similar. En la práctica, comienza a partir de una cuadrícula regular de superpíxeles y mueve los píxeles o bloques de píxeles en los límites para refinar la solución.

II-A3. LSC: LSC se realiza adoptando una formulación de cortes normalizada de la segmentación de superpíxeles basada en una métrica de similitud que mide la similitud de color y la proximidad del espacio entre los píxeles de la imagen. Sin embargo, en lugar de usar el algoritmo tradicional basado en eigen, aproximamos la métrica de similitud utilizando una función del kernel que conduce a un mapeo explícito de valores de píxeles y coordenadas en un espacio de características de alta dimensión.

II-B. Turbopixeles

TurboPixels es uno de los primeros algoritmos de superpíxeles (es decir, el algoritmo fue, en contraste con Quick Shift y el enfoque de Felzenswalb y Huttenlocher, originalmente destinado a generar superpíxeles). Inspirados en los contornos activos, después de colocar los centros de superpíxeles en una cuadrícula reglar, los superpíxeles crecen en función de un contorno en evolución. El contorno se implementa como conjunto de niveles de la función

$$\psi : \mathbb{R}^2 x[0,\tau) -> \mathbb{R}^2$$

La evolución está formalmente definida por

$$\psi_t = -v \mid\mid \nabla\psi \mid\mid_2$$

Donde $\nabla\psi$ denota el gradiente de ψ y ψ_t es la derivada temporal.Aquí, la velocidad v describe la evolución futura del contorno.En práctica, ψ será la distancia euclidiana con signo y la evolución se lleva a cabo utilizando una discretización de primer orden. El contorno en iteración (T+1) viene dado por

$$\psi^{(T+1)} : \psi^{(T)} - V_I V_B \mid\mid \nabla\psi^{(T)} \mid\mid \Delta t$$

La velocidad v se divide en dos componentes: v_I que depende del contenido de la imagen y v_B lo que garantiza que los superpíxeles no se superpongan.

II-C. Comparación de histogramas

Hay dos grupos de medidas con las que podemos comparar histogramas, bin to bin y cross-bin.

En la medida bin a bin se utiliza la distancia entre los bins de nuestros histogramas, para esto, es importante tener en consideración que los histogramas son independientes entre ellos. De este modo a traves de la suma de promedios de entre los bins se calcula la similitud de histogramas

La forma en que se calcula la distancia entre bins de los histogramas se hace por medio de la distancia euclideana, la distancia eulcideana al cuadrado, la distancia manhattan y la entropia. Además presenta la correlación entre medidas de distancia de histogramas, de manera que encuentra cuales medidas son similares o no.

El segundo grupo hace enfasis en los valores adyacentes al bin que corresponde a ser tratado, es decir,

III. Desarrollo

Lo primero fue la lectura de las imágenes. Los datos de prueba fueron brindados en formato **.mat**, el cual es específico de Matlab. Al trabajar con python, tuvimos que hacer unos cuantos trucos para poder trabajar con las imágenes. Primero, se leía el archivo usando la función `loadmat()` de la biblioteca `scipy.io`. Una vez leídos los datos, observamos que regresaba un diccionario, donde las imágenes estaban en la llave de `volumenes`.

Sacamos los primeros dos volúmenes para los datos de entrenamiento. Sin embargo, al trabajar con las imágenes directamente en ese formato, los valores iban desde -1024 hasta 1024, con lo cual no funcionaba con el algoritmo `slic()` con el que obteníamos los superpixeles y que esperaba valores de 0 a 255. Para solucionarlo, guardamos la imagen leída en formato **PNG** y luego volvimos a leer los datos.

```
mat = loadmat('01_CHAVE.mat')
imagen1 = (mat['volumenes'][:,:,2])

plt.imshow(imagen1,cmap=plt.cm.gray)
plt.imsave('volumen2.png',imagen1)
```

III-A. Variación de Parámetros

En nuestro primer intento de obtener superpixeles y observar cómo funcionaba el algoritmo, hicimos varias iteraciones variando el *compactness*, siempre con 100, 200 y 300 segmentos. El código para hacer estas pruebas fue el siguiente:

```
image = img_as_float(plt.imread('volumen0.png'))
image = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
print(image.nonzero())
fig = plt.figure("Superpixels -- original")
ax = fig.add_subplot(1,1,1)
ax.imshow(image,cmap=plt.cm.gray)
plt.axis('off')
for numSegments in (100,200,300):
  segments = slic(image, n_segments=numSegments)

  fig = plt.figure("Superpixels -- %d" %(numSegments
    ))
  ax = fig.add_subplot(1,1,1)
  ax.imshow(mark_boundaries(image, segments))
  plt.axis('off')

plt.show()
```

Nuestro primer resultado con *compactness* 0 fue el siguiente, sin ningún super pixel marcado correctamente:

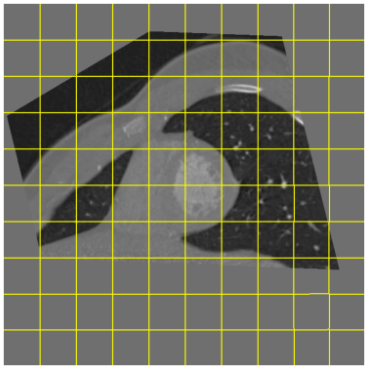

Figura 1: 100 segmentos sin compactness

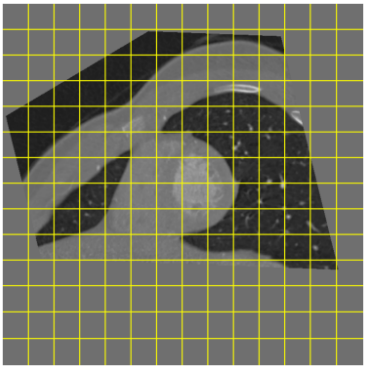

Figura 2: 200 segmentos sin compactness

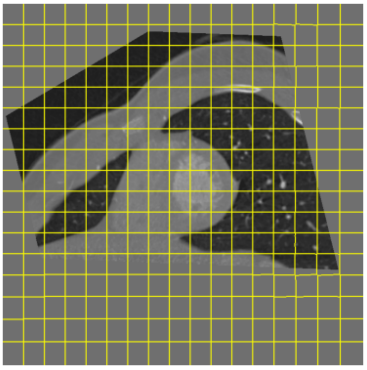

Figura 3: 300 segmentos sin compactness

Leyendo la documentación, recomiendan variar el nivel de compactness en escala logarítmica, con valores como 0.1, 1, 10, y así. Probando con un nivel de 0.01 obtenemos el siguiente resultado:

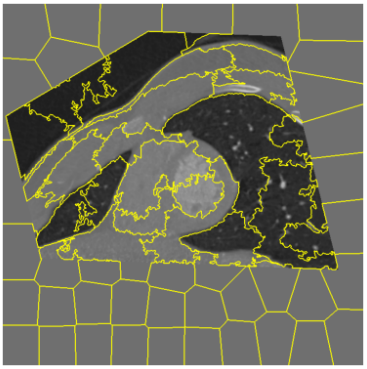

Figura 4: 100 segmentos y compactness 0.01

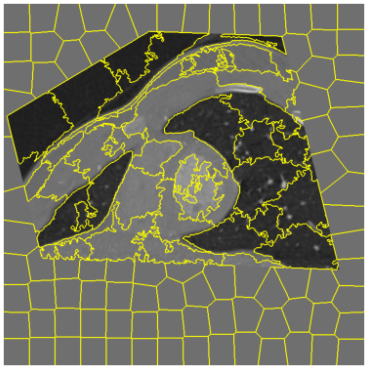

Figura 5: 200 segmentos y compactness 0.01

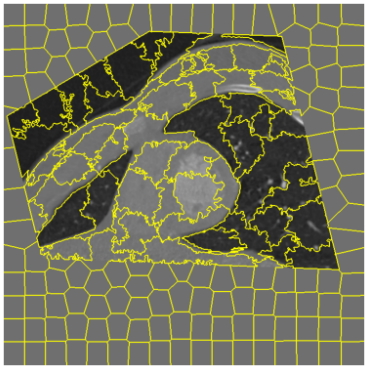

Figura 6: 300 segmentos y compactness 0.01

Podemos observar que el desempeño del algoritmo mejoró bastante. Sin embargo, sentimos que los superpixeles toman formas muy extrañas y angostas, lo que, a nuestra consideración, puede afectar el desempeño de la práctica.

La siguiente prueba que hicimos fue con compactness de 0.1, donde obtuvimos mejores resultados.

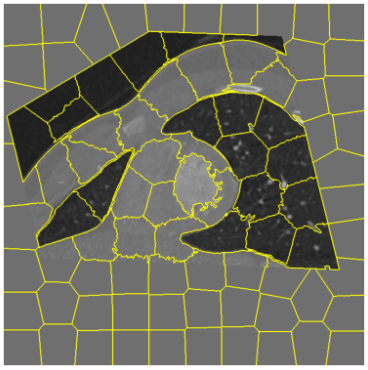

Figura 7: 100 segmentos y compactness 0.1

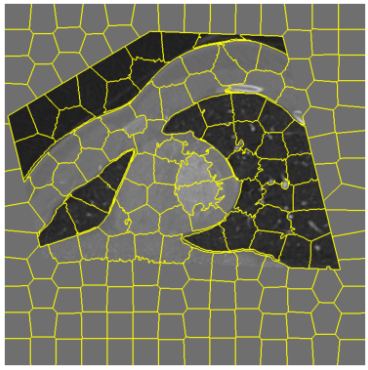

Figura 8: 200 segmentos y compactness 0.1

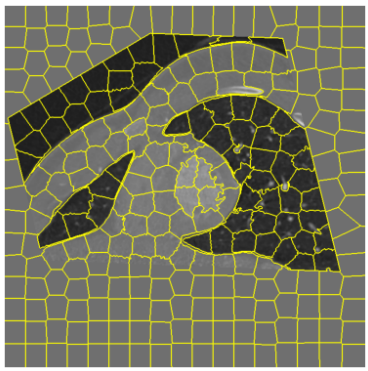

Figura 9: 300 segmentos y compactness 0.1

Por último, solo para ver si mejoraba o no, intentamos usar un compactness de 1:

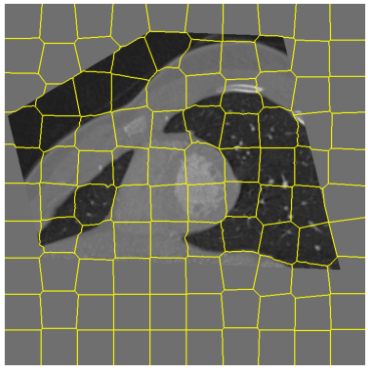

Figura 10: 100 segmentos y compactness 1

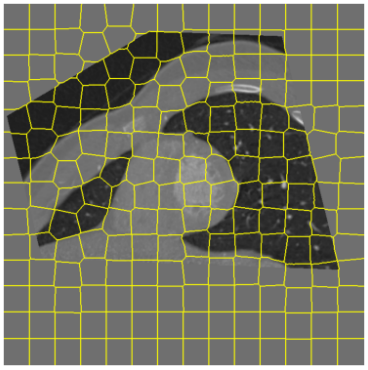

Figura 11: 200 segmentos y compactness 1

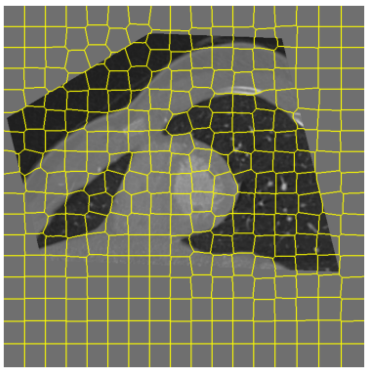

Figura 12: 300 segmentos y compactness 1

Con los resultados anteriores, decidimos trabajar con 200 segmentos y un compactness de 0.1, pues creímos que brindarían los mejores resultados.

III-B. Obtención de histogramas

Ya con un numero de superpixeles y una compacidad adecuada, para poder calcular el histograma representativo de la región, fue necesario que se ubicar manualmente los superpixeles de interés (las que hacen referencia musculo del ventrículo izquierdo), junto a su respectivo identificador.

Una vez con estos datos se obtuvo el histograma único para cada uno de los superpixeles seleccionados y se guardaron para su posterior análisis, como se puede observar en la siguiente figura:

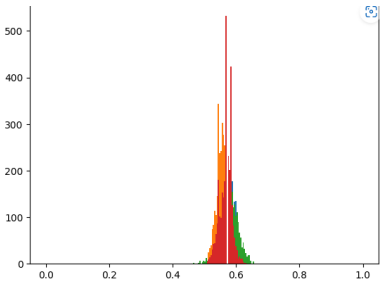

Figura 13

Para nuestra prueba mostraremos el histograma de los superpixeles de nuestra imagen, para esto, determinaremos nuestros superpixeles y con un barrido obtendremos el numero de cuentas de cada bin para nuestro histograma y así crearlo.

```
    for super_pixel_id in range(1,201):
    superPixel = getSuperpixel(vol2,superPixel2,
    super_pixel_id)
    tmp = superPixel[superPixel != 0].ravel()
    n,bins,patches = plt.hist(tmp,256,[0,1])
```

Para la distancia y comparación entre histogramas calculamos la distancia chi

Estos resultados se guardaran en un documento llamado salidas.txt

```
mean_distance = 0

  for hist in histogramas:
  mean_distance += chi_distance(n,hist)
  mean_distance = mean_distance / len(histogramas)

  f.write(str(mean_distance)+'\n')

  if mean_distance < 2000:
  id_interes.append(super_pixel_id)
```

Por ultimo para realzar el resultado final mostramos nuestra zona de interés que fue bien reconocida haciendo satisfactorio el resultado

```
fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.imshow(mark_boundaries(vol2,superPixel2), cmap=
    plt.cm.gray)
plt.axis('off')

detected_areas = np.zeros(vol2.shape[:2])
for id in id_interes:
  detected_areas += getSuperpixel(vol2, superPixel2,
    id)

fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.imshow(detected_areas, cmap=plt.cm.gray)
plt.axis('off')
```

IV. Resultados

En los histogramas obtenidos de la prueba se puede observar la distancia que existe entre los superpixeles obtenidos.

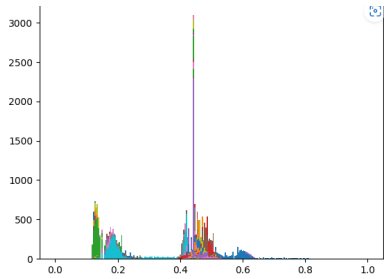

Figura 14

Posteriormente usando un umbral de 2000 de distancia promedio se separo correctamente el area, obteniendo la foto con los superpixeles generados y mas abajo el área separada exitosamente. El resultado que se obtuvo en la prueba resulto ser correcto, ya que se pudo obtener el área gracias a que los superpixels delimitaran bien las zonas.

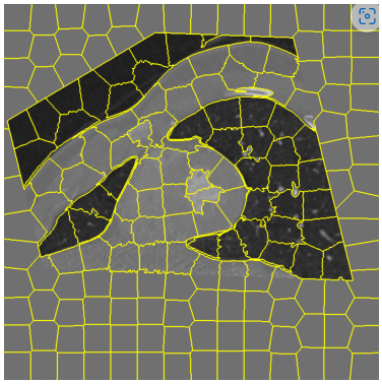

Figura 15

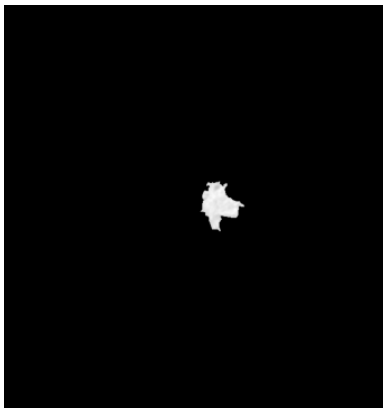

Figura 16

V. CÓDIGO

```python
from scipy.io import loadmat
from skimage.segmentation import slic,
    mark_boundaries
from skimage.util import img_as_float
import matplotlib.pyplot as plt
import cv2
import numpy as np

def getSuperpixel(img: np.ndarray,lbl:np.ndarray,id:
     int):
  shp = img.shape
  res = np.zeros(shp)
  for x in range(img.shape[0]):
    for y in range(img.shape[1]):
      res[x,y] = img[x,y] if lbl[x,y] == id else 0.0
  return res

def chi_distance(counts1, counts2):
  l = len(counts1)
  distance = 0

  for i in range(l):
    if (counts1[i] + counts2[i]):
      distance += ((counts1[i]-counts2[i])**2) / (
    counts1[i] + counts2[i])
  return distance

# Datos de entrenamiento
vol0 = plt.imread('volumen0.png')
vol0 = cv2.cvtColor(vol0, cv2.COLOR_RGB2GRAY)

vol1 = plt.imread('volumen1.png')
vol1 = cv2.cvtColor(vol1, cv2.COLOR_RGB2GRAY)

# Datos de prueba
vol2 = plt.imread('volumen2.png')
vol2 = cv2.cvtColor(vol2, cv2.COLOR_RGB2GRAY)

# Superpixeles
superPixel0 = slic(img_as_float(vol0),200,0.1)
superPixel1 = slic(img_as_float(vol1),200,0.1)
superPixel2 = slic(img_as_float(vol2),200,0.1)

# Graficaci n de superpixeles
fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.imshow(mark_boundaries(vol1, superPixel1))
plt.axis('off')

# Regiones obtenidas de manera manual
regiones_interes = [getSuperpixel(vol0,superPixel0
    ,81),getSuperpixel(vol0,superPixel0,92),
    getSuperpixel(vol1,superPixel1,83),getSuperpixel
    (vol1,superPixel1,90)]

histogramas = []
for region in regiones_interes:
  temp = region[region != 0].ravel()
  n,bins,patches = plt.hist(temp,256,[0,1])
  histogramas.append(n)

# Prueba
id_interes = []
f = open('salidas.txt','w')

for super_pixel_id in range(1,201):
  superPixel = getSuperpixel(vol2,superPixel2,
    super_pixel_id)
  tmp = superPixel[superPixel != 0].ravel()
  n,bins,patches = plt.hist(tmp,256,[0,1])

  mean_distance = 0

  for hist in histogramas:
    mean_distance += chi_distance(n,hist)
  mean_distance = mean_distance / len(histogramas)

  f.write(str(mean_distance)+'\n')

  if mean_distance < 2000:
    id_interes.append(super_pixel_id)

f.close()

# Validaci n Gr fica
fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.imshow(mark_boundaries(vol2,superPixel2), cmap=
    plt.cm.gray)
plt.axis('off')

detected_areas = np.zeros(vol2.shape[:2])
for id in id_interes:
  detected_areas += getSuperpixel(vol2, superPixel2,
    id)

fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.imshow(detected_areas, cmap=plt.cm.gray)
plt.axis('off')
```

VI. Conclusion

La práctica nos enseñó a aplicar superpixeles, el cual es un algoritmo para el análisis de imágenes que facilita la obtención de información de patrones al agrupar pixeles con características y ubicaciones similares. Aprendimos a implementarlo en python y lo aplicamos en un caso de uso real, lo cual fue muy interesante, pues podemos ver cómo lo que vemos en clase se usa en el mundo real.

El desarrollo de la práctica fue algo complicado, al trabajar con un formato de imagen diferente y andar buscando los superpixeles correctos, calcular la distancia chi entre histogramas y otras cosas que no habíamos hecho antes. A pesar de esto, el resultado de la práctica fue exitoso, pues nuestra prueba pudo reconocer correctamente el área de interés. Estamos satisfechos con el desarrollo y los resultados de la práctica.

Referencias

[] Stutz, D. (2015, 28 febrero). ≪TurboPixels: Fast superpixels using geometric flows≫, Levinshtein, Stere, Kutulakos, Fleet, Dickinson, and Siddiqi •. David Stutz. https://davidstutz.de/turbopixels-fast-superpixels-using-geometric-flows-levinshtein-stere-kutulakos-fleet-dickinson-siddiqi/

[] Superpixel segmentation using Linear Spectral Clustering. (2015, 1 junio). IEEE Conference Publication — IEEE Xplore. https://ieeexplore.ieee.org/document/7298741

[] Adrian Rosebrock. (2014, 28 julio). Segmentation: A Slic Superpixel Tutorial using Python. Recuperado 15 de noviembre de 2022, de https://pyimagesearch.com/2014/07/28/a-slic-superpixel-tutorial-using-python/