Proyecto final: Análisis de imágenes cardíacas

Álvarez Loran JP. * **Carrillo Sánchez R.** **
Martínez Soto M. ***

* *División de Ingeniería Eléctrica, Facultad de Ingeniería, Ciudad de México, México (e-mail:juanpa$_a$l@comunidad.unam.mx)*
** *División de Ingeniería Eléctrica, Facultad de Ingeniería, Ciudad de México, México (e-mail:r.carrillosanchez@comunidad.unam.mx)*
*** *División de Ingeniería Eléctrica, Facultad de Ingeniería, Ciudad de México, México (e-mail:marjo$_s$oto@comunidad.unam.mx)*

Abstract: El objetivo de este proyecto es utilizar técnicas de prepocesamiento, labeling, entrenamiento y clasificación sobre conjuntos de imágenes médicas para análisis cardíaco. Se trata principalmente de identificar la válvula aórtica del corazón humano. Adicionalmente se presenta un avance para la clasificación de otros componentes cardiacos aprovechando su aparición en el estudio realizado (MRI).

Keywords: Segmentación, análisis, médicas, ventrículo, técnicas, MRI, clasificación.

1. OBJETIVOS

Se hará uso de algoritmos para la:

- Visualización de máscaras de segmentación,
- Aplicación de dichas máscaras a las imágenes originales
- Segmentación del ventrículo izquierdo
- Etiquetado de la zona de interés (ventrículo izquierdo)
- Clasificación de otros componentes importantes del corazón que puedan ser visualizados en la MRI

2. INTRODUCCIÓN

Según los CDC (Centros de Control y la Prevención de Enfermedades) el corazón tiene cuatro cavidades: dos aurículas y dos ventrículos. Las arterias y las venas entran y salen del corazón. Las arterias llevan la sangre hacia afuera del corazón y las venas la llevan hacia adentro. El flujo de sangre a través de los vasos y las cavidades del corazón es controlado por válvulas. El ventrículo izquierdo (VI) bombea la sangre oxigenada a través de la válvula aórtica (VAo) hacia la aorta (Ao), la principal arteria que transporta sangre oxigenada al resto del cuerpo.
Todos los componentes del corazón resultan de vital importancia; Particularmente en este caso enfatizamos la identificación de la válvula aórtica. Uno de ellos, según Mayoclinic, es la hipertrofia ventricular izquierda, que es básicamente el agrandamiento y engrosamiento (hipertrofia) de las paredes de la cavidad de bombeo principal del corazón (ventrículo izquierdo). La pared del corazón engrosada pierde elasticidad, lo que lleva a un aumento de la presión para permitir que el corazón llene su cavidad de bombeo para poder enviar la sangre al resto del cuerpo. Al final, el corazón puede dejar de bombear con la fuerza necesaria. Por esta y más razones es importante que un especialista realice un análisis adecuado del estado de los componentes del corazón y evitar complicaciones como coágulos sanguíneos, accidentes cerebrovasculares o insuficiencia cardíaca.
Una forma de hacerlo es a tráves de una Resonancia Mágnetica Cardiaca (MRI), la cual genera imágenes médicas de formato DCM. Es aquí donde este proyecto pretende ayudar, facilitando su identificación para agilizar el trabajo al cardíologo en cuestión.

3. DESARROLLO

A continuación explicaremos paso a paso la implemetanción de cada etapa de programación.

3.1 Carga de librerías

Utilizamos numpy para trabajar de manera más eficiente con cada una de las imágenes, también hicmos uso de scikit-image para la aplicación de filtros y segmentación, así también scikit-learn para el aprendizaje automático, matplotlib para el despliegue de imágenes y cv2 para el manejo de las mismas.

```
from sklearn.model_selection import train_test_split #para
    dividir sets de entrenamiendo y prueba

import numpy as np
import cv2
import matplotlib.pyplot as plt
from os import listdir
import pandas as pd

from skimage.filters import gaussian #filtro para la
    deteccion de bordes
from skimage.filters import sobel #filtro para la
    deteccion de bordes
from skimage.filters import unsharp_mask #filtro para la
    deteccion de bordes
```

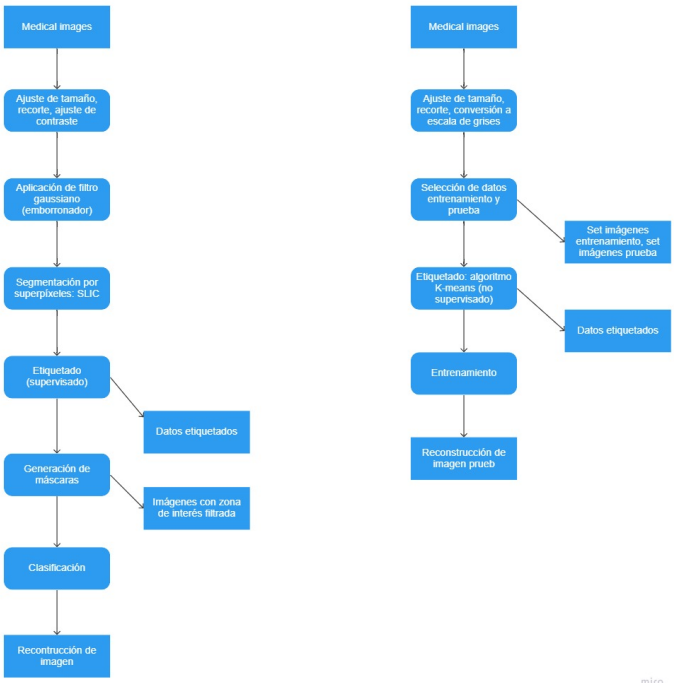


Fig. 1. Diagrama de bloques

```
from skimage.segmentation import slic
from skimage.segmentation import mark_boundaries
from skimage.util import img_as_float
from skimage import io
from skimage.color import rgb2gray

import matplotlib.pyplot as plt
import argparse

from sklearn.metrics import roc_curve, RocCurveDisplay,
    auc, f1_score, confusion_matrix,
    ConfusionMatrixDisplay
```

3.2 Carga de imágenes y su preprocesamiento

Realizamos la carga de imágenes, leyéndolas con la función imread de la biblioteca cv2. Se realiza de forma iterativa por cada elemento que exista en el directorio.
Posteriormente, durante la misma iteración, se realiza un preprocesamiento a estas imágenes. Se les realizan modificaciones iniciales como:

- Recorte a partir del establecimiento de pixeles
- Ajuste de tamaño para que éstas sean de 256x256
- Ajuste su contraste para que los matices de grises sean más fácil de distinguir.
- Creación de máscaras
- Aplicación de filtro de emborronamiento y eliminación de ruido (gaussiano)

Estas imágenes de salida se almacenarán en un arreglo llamado "images".
En la figura 1 podemos ver un ejemplo de salida.

```
images = []

for filename in listdir('data'):
    #img=img_as_float(io.imread(f'data/{filename}'))
    img = cv2.imread(f'data/{filename}')
    img = img[:, 174:550] #recorte de imagen
    img = cv2.resize(img, [256,256])

    img = cv2.convertScaleAbs(img, alpha=1.1, beta=1)
        #alpha contrast - beta = brightness
    img = unsharp_mask(img, radius=1, amount=50)
    img = gaussian(img, sigma=(1.5,1.5), truncate=3.5,
        channel_axis=2)

    images.append(img)

print(images[0].shape)
plt.imshow(images[0], cmap='binary', vmin=0, vmax=255)
```

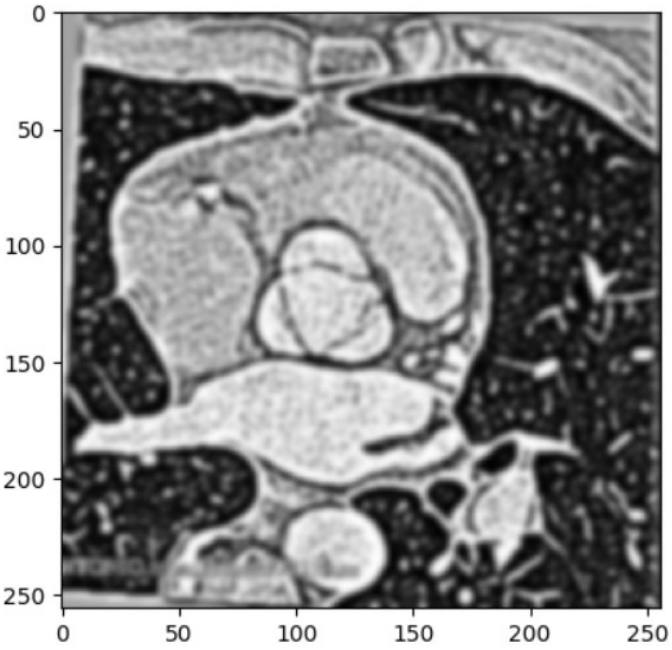


Fig. 2. Una de las imágenes preprocesada del set

3.3 Segmentación por superpíxeles

Creamos la función gen_superpixels que únicamente hace uso de slic proveniente de la biblioteca scikit learn para la segmentación de píxeles. Veremos que el resultado es bastante convincente, ya que identifica de manera adecuada a los ventrículos. Esta función recibe varios parámetros, que de acuerdo a nuestras necesidades podemos omitir o utilizar. El primero es la imagen a la cual se le aplicará el algoritmo el número de superpixeles que buscamos generar y sigma que indica el nivel de suavidad con la que se aplicará el kernel. Existen otros parámetros que en este momento se ignoraron.

```
def gen_superpixels(image, numSegments = 110, sigma=3.5):
image_superpixels = slic(image, n_segments = numSegments,
    sigma = sigma)
return image_superpixels
```

Segmentos almacena cada uno de los superpixeles obtenidos a través de la función slic, únicamente de la imagen prueba que hemos tomado la cual es images[0].

Después desplegamos los superpixeles uniformizados utilizando la imagen original y el conjunto de superpíxeles que los delimitarán, esto gracias a la función mark_boundaries de scikit-image.

```
numSegments = 110
segments = slic(images[0], n_segments = numSegments, sigma
    = 3.5)

# show the output of SLIC
fig = plt.figure("Superpixels -- %d segments" %
    (numSegments))
ax = fig.add_subplot(1, 1, 1)

superpixels_image = mark_boundaries(images[0], segments)
ax.imshow(superpixels_image)
plt.axis("off")
# show the plots
plt.show()
```

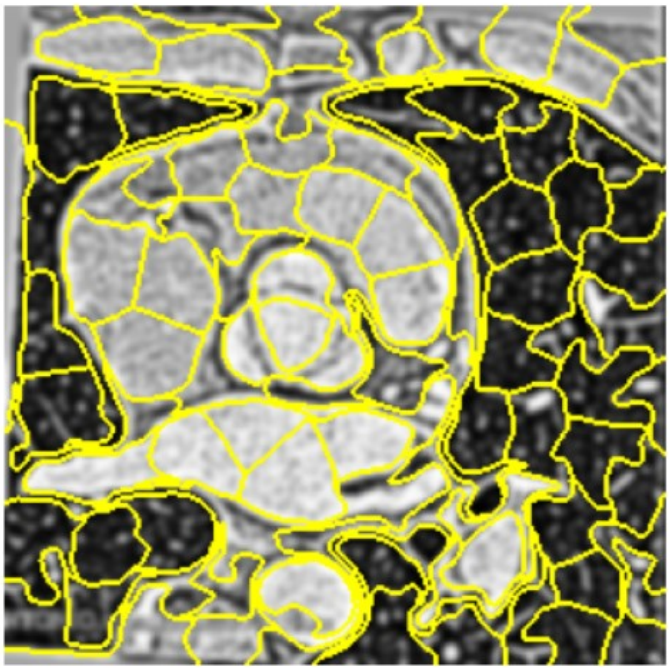


Fig. 3. Superpíxeles de la imagen preprocesada anterior

Aquí aplicamos la función que genera los superpixeles a nuestro set de datos preprocesados anteriormente, esta vez de forma general y se almacenan en un nuevo arreglo llamado "superpixels_images".

```
superpixels_images = []

for img in images :
    sp = gen_superpixels(img)
    superpixels_images.append(sp)
```

```
superpixels_image = mark_boundaries(images[0], segments)
ax.imshow(superpixels_image)
plt.axis("off")
# show the plots
plt.show()
```

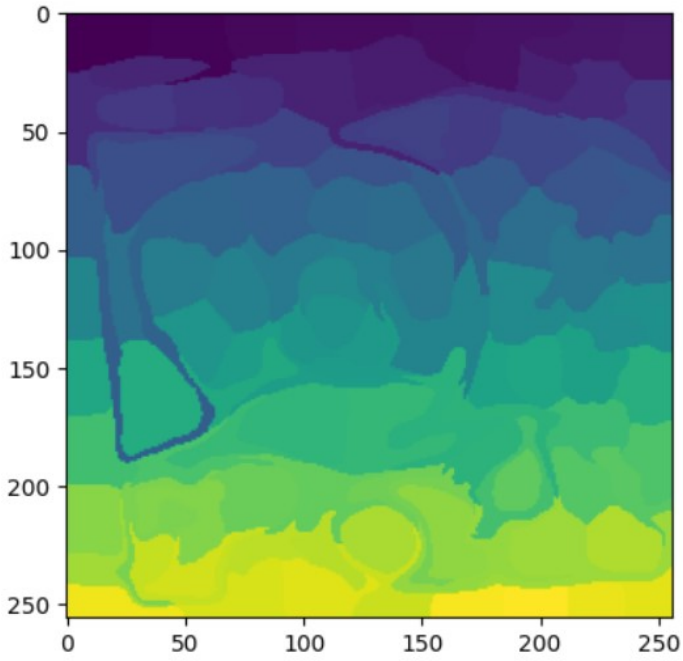


Fig. 4. Boundaries de la imagen de la figura 1, con un mapa de colores por defecto

3.4 Etiquetado

Para el etiquetado de nuestras áreas de interés, es decir la selección de los superpíxeles que representan las áreas dle corazón, se programó una función que permite manualmente seleccionarlas con el mouse.

```
def get_label(x,y, labels):
    return labels[y,x]
```

Esta función maneja eventos de selección sobre el superxpíxel que deseamos.

```
coords=[]
def click_event(event, x, y, flags, params):

    # checking for left mouse clicks
    if event == cv2.EVENT_LBUTTONDOWN:

        # displaying the coordinates
        # on the Shell
        print(x, ' ', y)

        # displaying the coordinates
        # on the image window
        font = cv2.FONT_HERSHEY_SIMPLEX
        cv2.putText(superpixels_image, str(x) + ',' +
                    str(y), (x, y), font,
                    0.3, (0, 255, 0), 1)
        cv2.imshow('image', superpixels_image)

    # checking for right mouse clicks
    if event == cv2.EVENT_RBUTTONDOWN:
```

```
        # displaying the coordinates
        # on the Shell
        print(x, ' ', y)

        # displaying the coordinates
        # on the image window
        font = cv2.FONT_HERSHEY_SIMPLEX
        cv2.putText(superpixels_image,
             str(get_label(x,y,segments)),
                   (x, y), font, 0.3,
                   (255, 0, 255), 1)
        cv2.imshow('image', superpixels_image)
        coords.append([x,y])
```

Muestra los superpixeles de la imagen, Se manda llamar a la función que se ha definido arriba, y sale con al presionar alguna tecla una vez terminado el proceso de selección.

```
cv2.imshow('image', superpixels_image)

# setting mouse handler for the image
# and calling the click_event() function

cv2.setMouseCallback('image', click_event)

# wait for a key to be pressed to exit
cv2.waitKey(0)

# close the window
cv2.destroyAllWindows()
```

Una vez teniendo las etiquetas de las imágenes, se crearon máscaras a partir de ellas, y con bitwise_or una operación or se lograron trasponer la imagen original con la máscara creada. De esta forma, se ignora todo lo demás que no ha sido sleccionado durante la etapa de etiquetado previo.

```
mask = np.zeros(images[0].shape[:2], dtype = "uint8")

for x,y in coords:
    mask[segments == get_label(x,y, segments)] = 255

masked_image = cv2.bitwise_or(images[0], images[0], mask =
     mask)
cv2.imshow("Applied",masked_image)
cv2.waitKey(0)
```

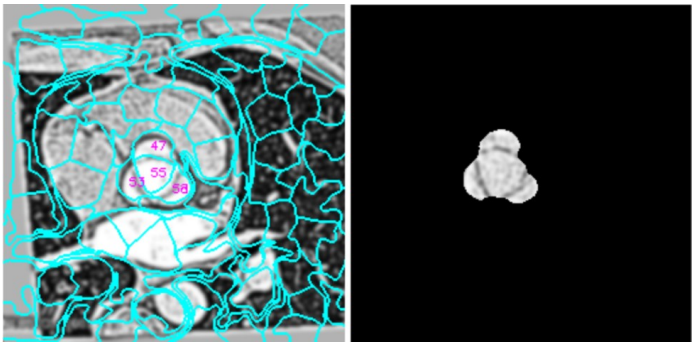


Fig. 5. Primera imagen ya etiquetada de la lista y su máscara aplicada

Obteniendo los las mascaras de interes en la fase de entrenamiento se procede a obtener su histograma caracteristico

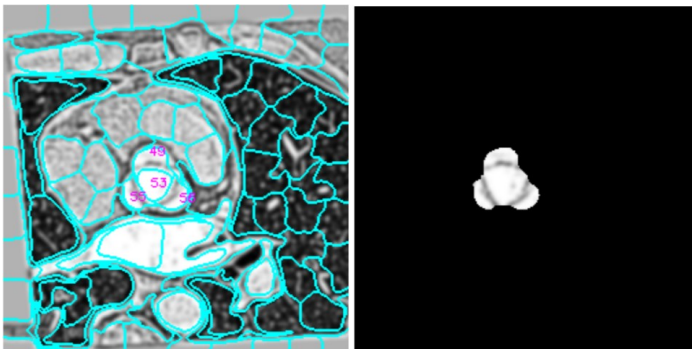


Fig. 6. Segunda imagen ya etiquetada de la lista y su máscara aplicada

de cada una de las mascaras con el fin de poder obtenter las un histograma representativo del set de entrenamiento.

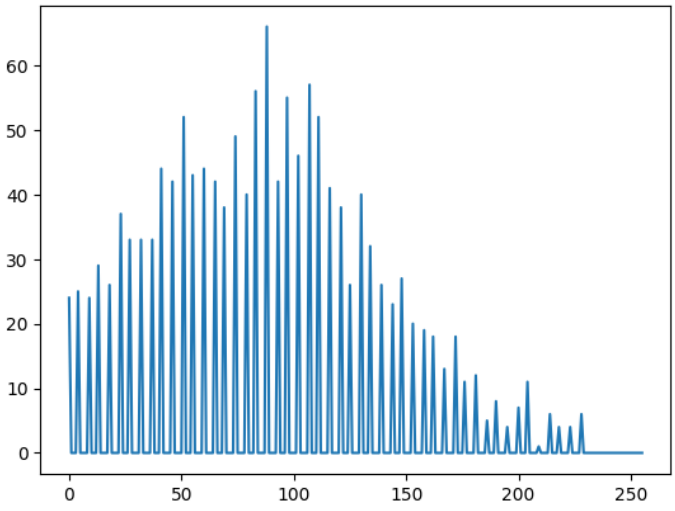


Fig. 7. Histograma de la primera mascara

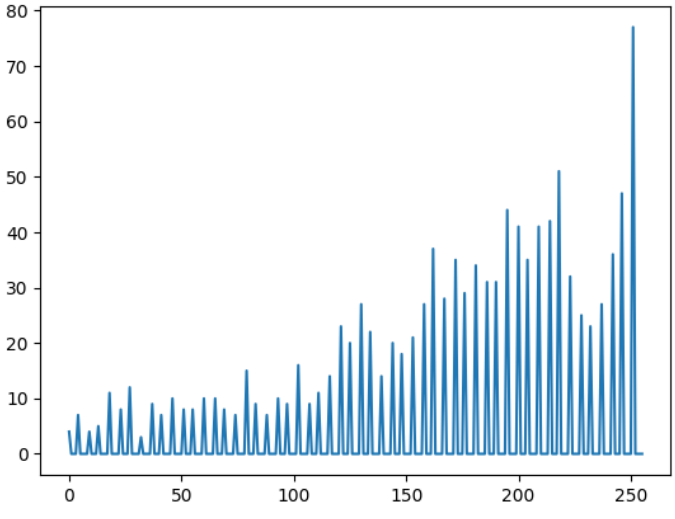


Fig. 8. Histograma de la segunda mascara

La fase de prueba consiste en la replica de los pasos de entrenamiento, obtener los superpixeles e iterarlos para obtener los histogramas de cada unos de los superpixeles

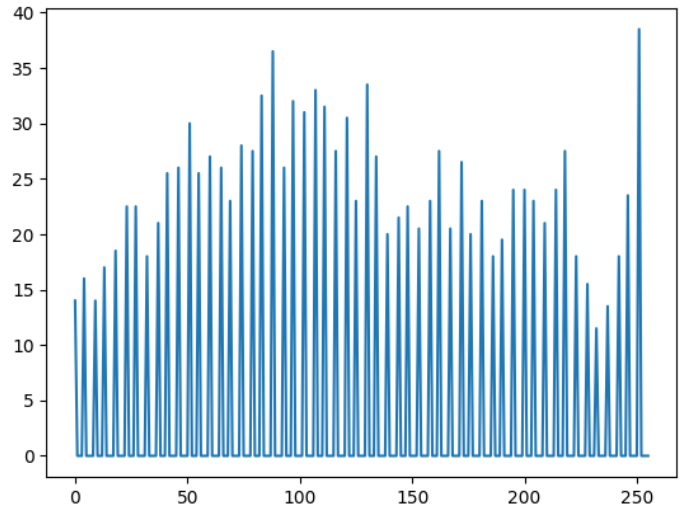


Fig. 9. Histograma promedio

que a partir de estos obtendemos las distancias de chi cuadrada, correlacion y la distancia de Bhattacharyya.

Los resultados obtenidos son varios; principalente notamos que la todas las distancias es posible obtener la cantidad de super pixeles que son representativos a la zona. La distancia de Bhattacharyya es la que nos permitio ajustar de mejor manera los superpixeles optimos para llevar a cabo la segmentacion, lamentablemente no encontramos la mejor forma de preprocesar las imagenes con las que trabajamos, de modo que mucho de la informacion se ha filtrado de modo que la segmentacion del area del ventriculo no es la mejor.

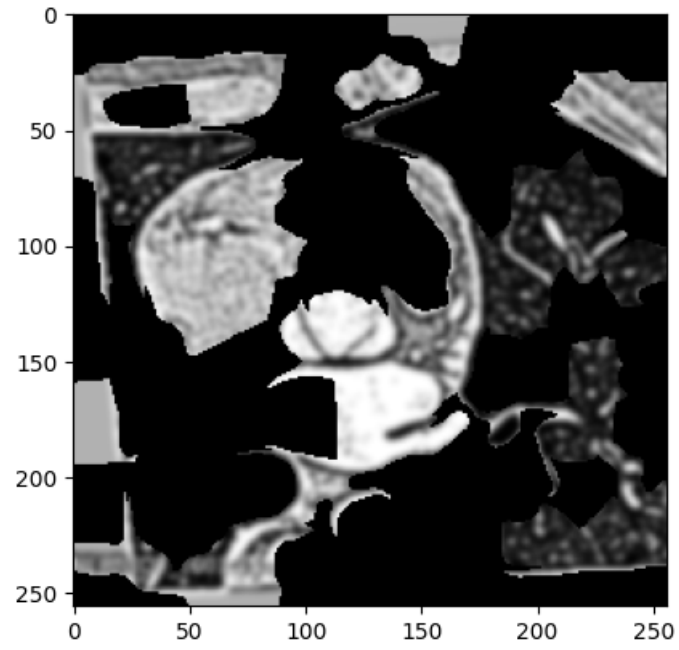


Fig. 10. Resultado

Como se planteó en los objetivos se aprovecharon las imágenes del estudio para identificar otras partes del corazón, las cuales son independientes del objetivo inical de segmentación de la válvula aórtica, pero de cualquier

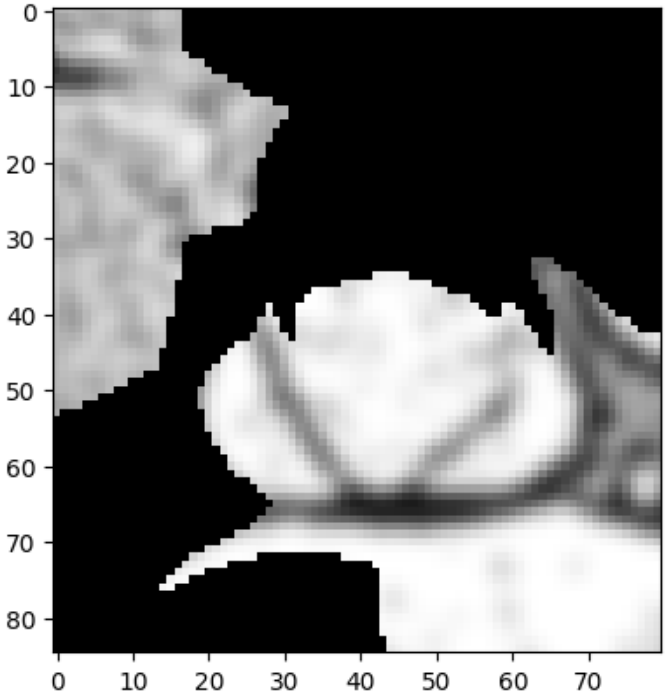


Fig. 11. Resultado

forma resulta interesante e importante. A continuación se presenta más detalle:

3.5 Clasificación mediante Kmeans y SVM

Adicionalmente, se realizó un código alterno el cual tiene como objetivo clasificar las partes del corazón mediante un método no supervisado por lo que se utilizó el algoritmo KMeans para clusterizar y etiquetar los datos de la imagen de la cual no conocemos sus componentes. Una vez obtenidas las etiquetas, esta información se pasa a una máquina de soporte vectorial para crear el modelo de aprendizaje.

Como primer paso preprocesamos las imagenes pasándolas a escala de grises, estandarizando su tamaño a 100x100 pixeles y finalmente reestructurandolas como un arreglo de 1 dimensión para ser procesadas por los algoritmos.

```
dir_list= os.listdir('./img')
images=list()
for i in dir_list:
    images.append(cv2.imread('./img/'+i))

for i in images:
    images[images.index(i)] = cv2.cvtColor(i,
        cv2.COLOR_BGR2GRAY)

for i in images:
    images[images.index(i)] = cv2.resize(i,(100,100))

for i in images:
    images[images.index(i)] = i.reshape((i.shape[0] *
        i.shape[1], 1))
```

Una vez que tenemos en un arreglo todas las imagenes estandarizadas procedemos a separar los datos para el entrenamiento y la prueba.

```
images_train, images_test= train_test_split(images,
    test_size=0.2)
```

Con esta información procedemos a aplicar el algoritmo KMeans a cada imagen y obtener sus etiquetas, almacenando cada uno de estos datos en su correspondiente arreglo.

```
#Set the number of clusters
num_clusters = 5
labeled_images=list()
labels_=list()
cluster_centers_=list()
km = KMeans(n_clusters=num_clusters)

# Run k-means clustering
for i in images_train:
    km.fit(i)
    cluster_centers = km.cluster_centers_.squeeze()
    labels = km.labels_

    # Convert the labels back into a 2D image
    labels = labels.reshape(i.shape[0], i.shape[1])

    # Create an image for each cluster
    segmented_image = np.zeros_like(i)
    for i in range(num_clusters):
        segmented_image[labels == i] =
            cluster_centers[i]

    labeled_images.append(segmented_image)
    labels_.append(labels)
```

Ahora que tenemos datos etiquetados nos resta declarar el modelo y entrenarlo con cada set de datos obtenido.

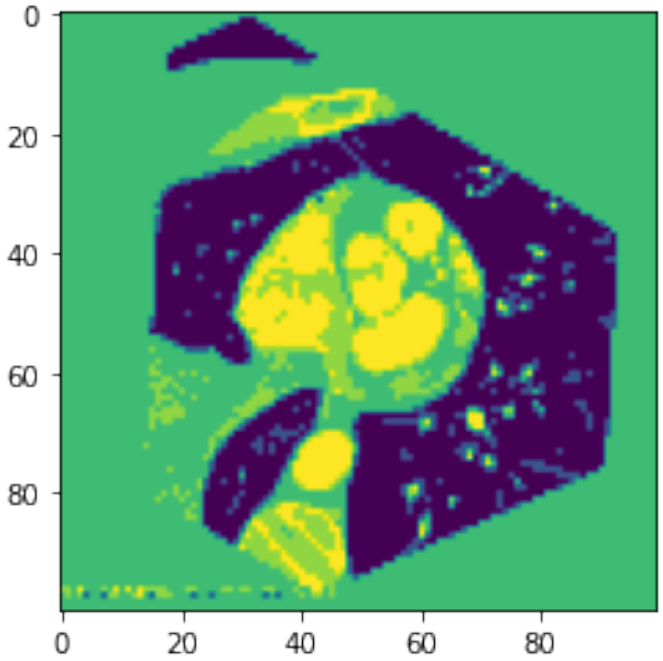
```
model = SVC()
for i in range(len(labeled_images)):
    model.fit(labeled_images[i], labels_[i])
```

Una vez que tenemos el modelo entrenado pasamos a realizar las predicciones con el arreglo de prueba que generamos anteriormente. Con las etiquetas predichas se reconstruye la imagen y mostramos la clasificación.

```
for i in images_test:

    predictions= model.predict(i)
    predictedImage = np.zeros_like(i)

for i in range(num_clusters):
    predictedImage[predictions == i] =
        cluster_centers[i]

predictedImage = predictedImage.reshape(100,100)*255.0
plt.imshow(predictedImage)
plt.show()
```

El resultado obtenido en la prueba es el siguiente:

Fig. 12. Clasifiación mediante SVM y KMeans



4. CONCLUSIONES

La calsificación de estudios médicos presenta una complejidad elevada ya que requiere de conocimiento previo en la identificación de los componentes anatómicos así como la complejidad de la programación.

El problema se abordó desde dos perspectivas distintas: la clasificación supervisada y la no supervisada, en ambas se obtuvieron buenos resultados pero dependiendo de la aplicación un método resulta más eficiente que otro.

Si se desea clasificar elementos particulares resulta más eficiente la clasificación supervisada mediante superpixeles ya que nos permite seleccionar de manera dinámica el elemento que deseamos y entrenar al modelo. Por otra parte si se desea clasificar varios elementos, será una buena elección el uso de la clasificación no supervisada, sin embargo, será importante ajustar el uso de clusters para que permita diferenciar los componentes anatómicos.

REFERENCES

[1] The Radiology Assistant: Cardiac Anatomy. (2009, 13 febrero). https://radiologyassistant.nl/cardiovascular/anatomy/cardiac-anatomy

[2] How the Heart Works — Congenital Heart Defects — NCBDDD — CDC. (2018, 26 septiembre). Centers for Disease Control and Prevention. https://www.cdc.gov/ncbddd/spanish/heartdefects/howtheheartworks.html

[3] Hipertrofia ventricular izquierda - Síntomas y causas - Mayo Clinic. (2021, 29 julio). https://www.mayoclinic.org/es-es/diseases-conditions/left-ventricular-hypertrophy/symptoms-causes/syc-20374314

[4] Implementación de algoritmo de superpixeles para la segmentación de imágenes a color (2017), Alan

Hernández Girón, A. González Navarro, Y. Adrián Morales Blas, A. Unidad Profesional Interdisciplinaria en Ingeniería y Tecnologías Avanzadas-IPN. Ciudad de México

[5] scikit-learn: machine learning in Python — scikit-learn 1.2.0 documentation. (s. f.). https://scikit-learn.org/stable/

[6] scikit-image: Image processing in Python — scikit-image. (s. f.). https://scikit-image.org/

[7] Sci-Hub — Classification of Ventricular Septal Defects for the Eleventh Iteration of the International Classification of Diseases Striving for Consensus: A report from the International Society for Nomenclature of Paediatric and Congenital Heart Disease. The Annals of Thoracic Surgery — 10.1016/j.athoracsur.2018.06.020. (s. f.). https://sci-hub.se/10.1016/j.athoracsur.2018.06.020

[8] Sci-Hub — Aorta segmentation with a 3D level set approach and quantification of aortic calcifications in non-contrast chest CT. 2012 Annual International Conference of the IEEE Engineering in Medicine and Biology Society — 10.1109/EMBC.2012.6346433. (s. f.). https://sci-hub.se/10.1109/EMBC.2012.6346433

[9] Module: filters — skimage v0.19.2 docs. (s. f.). https://scikit-image.org/docs/stable/api/skimage.filters.html