IDENTIFICACIÓN DE ANOMALIÁS ENCEFÁLICAS A PARTIR DEL PROCESAMIENTO DE IMÁGENES

JUAN DIEGO OROZCO GOMEZ

UNIVERSIDAD PONTIFICIA BOLIVARIANA

ESCUELA DE INGENIERÍAS Y ARQUITECTURA

PROGRAMA DE INGENIERÍA ELECTRÓNICA

MONTERÍA

2023

IDENTIFICACIÓN DE ANOMALIAS ENCEFALICAS A PARTIR DEL PROCESAMIENTO DE IMAGENES

JUAN DIEGO OROZCO GOMEZ

TRABAJO DE GRADO PARA OPTAR EL TITULO DE INGENIERO ELECTRONICO

ASESORA

ANA MILENA LÓPEZ LÓPEZ

MsC.

UNIVERSIDAD PONTIFICIA BOLIVARIANA

ESCUELA DE INGENIERÍAS Y ARQUITECTURA

PROGRAMA DE INGENIERÍA ELECTRÓNICA

MONTERÍA

2023

CONTENIDO

ÍNDICE:

1. RESUMEN:

Debido a los diferentes avances realizados en los algoritmos de aprendizaje para la identificación de imágenes, y la necesidad de realizar diagnósticos en el área de la medicina, de manera rápida y confiable de diversas condiciones, por medio de imágenes médicas especializadas. En este se presentó una herramienta de apoyo para el diagnóstico de anomalías encefálicas, por lo que se propuso desarrollar dos algoritmos en base a los métodos de procesamiento de datos k-means y máquina de soporte vectorial: el primero, es un algoritmo de agrupamiento de datos con aprendizaje no supervisado; el segundo, es un algoritmo de clasificación de aprendizaje supervisado, para así comparar los resultados de cada uno, dada la misma base de datos de imágenes, de resonancia magnética cerebrales. Sobre la base de lo anterior se investigó y detalló en este documento cómo funcionan, el procesamiento de imágenes en general y los tipos de algoritmos utilizados, primero para aplicarlos en la segmentación de datos y de este modo emplearlos en la tarea de identificación, de esta manera se implementaron varios códigos, dos para el proceso de segmentación, en base a los métodos de k-means y SVM. Para así aplicar estos algoritmos en otros dos códigos, en la tarea de identificación de imágenes, en los cuales se presentan sus resultados en una interfaz gráfica.

Palabras claves: Algoritmo, identificación, aprendizaje, procesamiento

ABSTRACT:

Due to the different developments accomplished in the learning algorithms of image identification, and the need to perform diagnoses in the medical area, in a fast and trustworthy way, in different conditions, through specialized medical imaging, a project was developed, which is shown in the following text. In this in presented a support tool to brain anomalies diagnosis, so is proposed develop two algorithms in base of data processing k-means and support vector machine (SVM): the first is an algorithm of data clustering with unsupervised learning, the second is a classification algorithm with supervised learning, to compare the result of each one, given the same image database, of brain MRI (magnetic resonance image). Based on the above it was investigated a detailed in this paper how Works, the image processing in general and the type of algorithms used, first to apply them in the data segmentation and in this way use them in the identification task, thus it was implemented several codes, two to the segmentation process, based in the methods of k-means and SVM. To apply these algorithms in other two codes, in the image identification task, in which is presented their results in a graphical user interface.

2. INTRODUCCIÓN

Para realizar diagnósticos de manera rápida y certera, se utilizan diferentes métodos y técnicas, para observar los síntomas y el estado de los tejidos del paciente, en los cuales algunas de estas dan como resultados imágenes diagnósticas. Es importante destacar que, para lograr identificar la patología de un órgano de forma no invasiva (Shoeibi, A., Ghassemi, N., Khodatars, M., Moridian, P., Khosravi, A., Zare, A., Gorriz, J. M., Chale-Chale, A. H., Khadem, A., & Acharya, U. 2022), por ejemplo, del cerebro, se utilizan diferentes técnicas que permitan observar la estructura y funciones de este, al mostrar las conexiones de las redes neuronales en este (Sadeghi, D., Shoeibi, A., Ghassemi, N., Moridian, P., Khadem, A., Alizadehsani, R., Teshnehlab, M., Górriz, J. M., & Nahavandi, S. 2022). En este contexto la NHS (National health service), reporto que entre diciembre del 2021 a diciembre de 2022 se realizaron 43,12 millones de este tipo de pruebas en Inglaterra, un aumento del 23,5 % con respecto al reporte del año pasado (National Health Service [NHS England]. 2021, 18 noviembre. 2023, 20 abril). Dado lo anterior es necesario que especialistas analicen las imágenes de forma detallada, para generar un reporte médico de manera que se pueda realizar el tratamiento adecuado para el paciente. Lo anterior es un proceso eficaz, pero debido al alto influjo de pacientes y el tiempo requerido para realizarlo, en los últimos años, se han desarrollado métodos de análisis de datos para coadyuvar al diagnóstico, como lo es el procesamiento de imágenes. De manera similar, se ha utilizado el machine learning para esta tarea, donde puede ser utilizada en la detección de patologías o elementos críticos para su diagnóstico, de enfermedades como la diabetes, tumores cerebrales y la esquizofrenia.

Por lo anterior, se propuso implementar dos técnicas de procesamiento de imágenes, en este caso k-means, el cual es un método de clustering (agrupamiento) en donde se agrupan datos dentro de un número predeterminado de centroides, donde sea menor la distancia de los puntos al centroide, este puede ser utilizado para diferentes aplicaciones, en este caso para segmentación e identificación, pero también es utilizado en predicción de resultados. Por otra parte, se tiene máquina de soporte vectorial, que son algoritmos donde se separan datos con dos diferentes etiquetas, mediante un hiperplano el cual tenga el mayor margen o espacio entre este y los datos posibles, para observar cuán eficientes pueden ser en la identificación de anomalías encefálicas a través de la interfaz gráfica, de modo que pueda ser utilizado por un especialista como insumo para el diagnóstico.

Por lo que se desarrolló un programa capaz de clasificar una imagen cerebral dentro de 3 categorías de anomalías y una categoría de control, la cual es mostrada en una interfaz que presenta los resultados de los dos algoritmos implementados, esto por medio del lenguaje de programación Python, utilizando sus librerías, módulos y funciones para realizar los procesos de lectura, preprocesamiento, segmentación e identificación, como lo serian *OpenCv*, *pandas* y *Sklearn.svm*.

3. MARCO TEÓRICO/ESTADO DEL ARTE:

A continuación se presentaran los conceptos claves utilizados en el desarrollo del proyecto, en los que se encuentran técnicas en base al concepto de machine learning y al procesamiento de imágenes, que se escogieron debido a que permite la automatización en la clasificación de anomalías, imágenes u objetos (Srinivas, C., S, N. P. K., Zakariah, M., Alotaibi, Y. A., Shaukat, K., Partibane, B., & Awal, H. 2022), dado a la capacidad de los algoritmos de interpretar datos y aprender de estos. Es necesario mencionar que los métodos utilizados son de aprendizaje supervisado y no supervisado, de modo que el primero necesita datos de entrenamiento y el segundo no (Tack, C. (2019)).

3.1 SEGMENTACIÓN

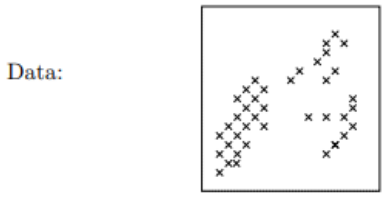
El procesamiento de imágenes es un campo el cual, permite separar y agrupar píxeles, de acuerdo a sus propiedades dentro de la imagen a examinar, de modo que puedan ser interpretados y analizados en el software, lo anterior es utilizado en diferentes aplicaciones, por lo que en este caso en que se requiere identificar anomalías encefálicas es necesario primero, constatar un método de segmentación para así dividir la imágenes, y extraer de las partes que son de interés del resto de esta en partes, además a ciertas etiquetas dado a características como el color, intensidad, textura, entre otros, se agrupan estos píxeles. (Sultana, Sufian, & Dutta, 2020) Por lo anterior se examinan dos métodos, los cuales son ampliamente utilizados en diversos ámbitos.

3.2 K-MEANS

Primero se examinará el método k-means el cual es un algoritmo de agrupamiento ampliamente utilizado en diferentes campos (médicos, análisis estadísticos empresariales) (Mohiuddin , Raihan, & Syed Mohammed, 2020), este tipo de algoritmo pone N puntos de datos, en un espacio I dimensiones, dentro de K centroides, de forma similar a lo que se ve en la figura 1 los cuales deben ser parametrizados por un vector $m^{(k)}$, el cual es llamado mean (media). Por lo anterior si quisiéramos tener un indicador que define la distancia entre puntos, donde se tenga un espacio en el que *x* exista, se debe denotar los puntos por $\{x^n\}$ donde un superíndice n va desde 1 hasta N y si cada vector *x* tiene *I* componentes x_i, (MacKay, 2003.) de modo que se tenga un proceso como las figuras 2 y 3 lo anterior se puede definir como:

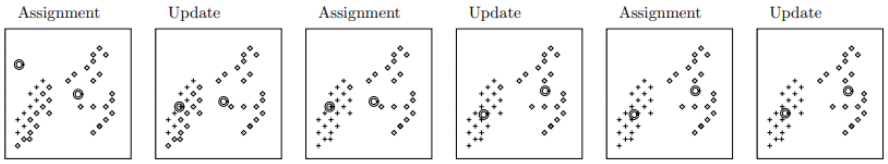
$$d(x,y) = \frac{1}{2}\sum_{i} \quad (x_i - y_i)^2$$

Figura 1
Gráficos de conjunto de datos para realizar k-means



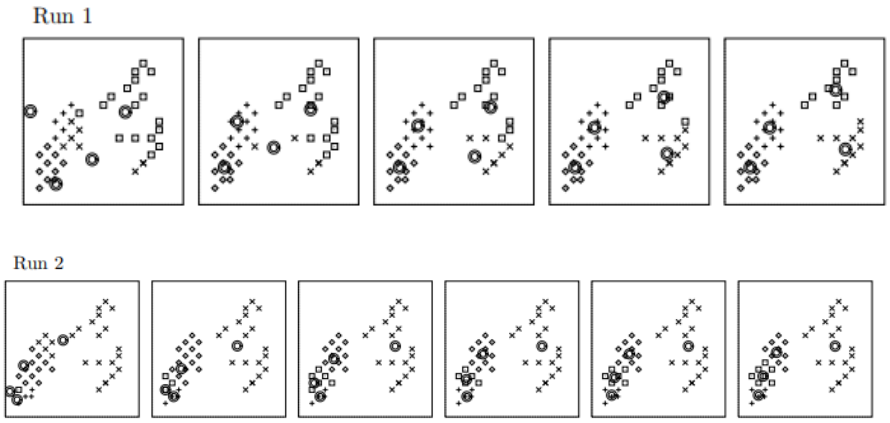
Nota. 40 puntos de datos los cuales se le aplicará el algoritmo de k-means con dos centroides. Adaptado de *Information theory inference, and learning algorithms.* (p.287), por D. J. C. MacKay, 2003.

Figura 2
Gráfica del proceso de k-means



Nota. Se muestra cómo se asignan los datos y se “actualizan” los centroides. Adaptado de *Information theory inference, and learning algorithms.* (p.287), por D. J. C. MacKay, 2003.

Figura 3
Gráfica del proceso de k-means con k=4



Nota. Se muestra cómo en dos iteraciones diferentes un conjunto de 40 datos es asignado de manera diferente por el algoritmo. Adaptado de *Information theory inference, and learning algorithms.* (p.287), por D. J. C. MacKay, 2003.

El algoritmo anterior consiste en dividir y reunir los datos en grupos (clusters) en relación a varios puntos, los cuales pueden corresponder a un criterio, estos serían los centros o también conocidos como centroides, que representan cuán similar es el grupo de datos al analizar cuán cerca o lejos están los datos de los puntos, de manera que los datos encerrados dentro de los grupos sea congruentes entre sí y los que estén dentro de otros sean diferentes unos de los otros (Sinaga & Yang, 2020), de modo que exista el menor solapamiento de información posible. De este modo al realizar la operación del método teniendo en cuenta los parámetros anteriores se colocarán centroides aleatorios en la imagen y se asignan a los datos los más cercanos, luego se calculará el centro del grupo, sumando la distancia elevadas al cuadrado de los datos y se repetirá la función anterior hasta que la función converja (que los centroides, los datos no cambien) o se alcancen el máximo de iteraciones. (Fränti & Sieranoja, 2019) En este método antes de realizar la operación se tiene que elegir el número de centros y grupos, de modo que se puede tener convergencia de los datos inesperada, esto y los datos no balanceados resulta en ser una de sus mayores problemáticas (Sinaga & Yang, 2020) (Fränti & Sieranoja, 2019).

3.3 MÁQUINA DE SOPORTE VECTORIAL

También se tiene al método de máquina de soporte vectorial, un enfoque a la clasificación desarrollado en 1990 en este método se tiene que su principal propósito es maximizar la generalidad de un modelo (Cervantes, Garcia-Lamont,

Rodríguez-Mazahua, & Lopez, 2020) y es ampliamente utilizado en tareas de clasificación (Wang & Chen, 2020), este consiste en separar los datos mediante un "hiperplano", el cual es escogido de cierto número de estas conjeturas aleatorias, el cual separa datos en dos grupos y se tenga el máximo margen entre los grupos, este es llamado hiperplano de separación óptima, lo anterior solo es posible si la separación es lineal, en otras palabras no hay ninguna intersección con los datos, es decir este método está destinado para clasificación binaria, de modo que se utiliza otros procedimientos como hiperplanos no lineales o kernels, lleva los datos a un espacio de dimensión mayor en donde sea posible la separación de los datos(Cervantes, Garcia-Lamont, Rodríguez-Mazahua, & Lopez, 2020).

Este hiperplano está definido como un subespacio afín de dimensión de p-1, dentro de un espacio vectorial, de modo que si se tiene un espacio de 2 dimensiones se tendría un hiperplano de 1 dimensión (una línea). Siendo lo anterior definido por la ecuación.

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 = 0$$

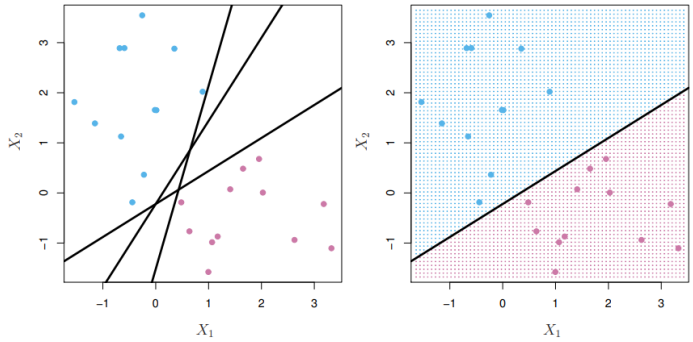
La cual puede ser extendida para espacios de p dimensiones

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p = 0$$

La ecuación anterior nos muestra cuando un hiperplano divide, exactamente en dos espacios iguales el espacio, de modo que, si un punto x satisface la ecuación, entonces este se encontrará en el hiperplano, pero si x satisface $\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p < 0$ o $\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p > 0$, entonces el punto estará a un lado del hiperplano, esto se puede observar en la figura 4.

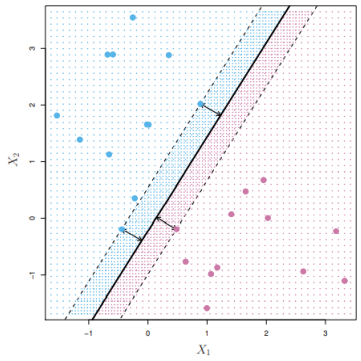
De este modo si tuviéramos una matriz de datos de n observaciones en el espacio vectorial, que representan dos clases (en este caso {1,-1}), es posible construir un hiperplano que separe los datos según las etiquetas dadas, de modo que el hiperplano tenga la propiedad que $\beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \cdots + \beta_p X_{ip} > 0$ si $X_i = 1$ y $\beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \cdots + \beta_p X_{ip} < 0$ si $X_i = -1$,(James, Witten, Hastie, Tibshirani, 2021) y así obtener un hiperplano con el mayor margen posible como se expone en la figura 5 .

Figura 4
Gráficos de hiperplanos separando datos.



*Nota.*El gráfico de la izquierda muestra los diferentes hiperplanos posibles, en la derecha se muestra una de las posibles clasificaciones. Adaptado de *An introduction to statistical learning:With applications in R* (p.370), por G. James, D. Witten, T. Hastie, R. Tibshirani, 2021, Springer

Figura 5
Gráficos de hiperplanos separando datos.



Nota. El gráfico muestra el hiperplano con mayor margen de clasificación, en el cual los datos y el hiperplano tiene la mayor distancia entre los dos. Adaptado de *An introduction to statistical learning:With applications in R* (p.372), por G. James, D. Witten, T. Hastie, R. Tibshirani, 2021, Springer

Sin embargo, se debe tener en cuenta varias debilidades pertinentes de este método que son el gran costo computacional, los problemas con más de dos parámetros, problemas con datos no balanceados. (Cervantes, Garcia-Lamont, Rodríguez-Mazahua, & Lopez, 2020)

3.4 IMÁGENES DIAGNOSTICAS

Por otra parte, los datos a analizar mediante los métodos analizados son provenientes de imágenes médicas cerebrales, de modo que de entre las técnicas más utilizadas para el análisis de patologías en la imagenologia, como los son los rayos x, ultrasonidos, tomografía computarizada y resonancia magnética, se escoge la técnica de resonancia magnética, la cual permite la observación de los tejidos suaves del cerebro, de forma no invasiva (Arabahmadi, M., Farahbakhsh, R., & Rezazadeh, J. 2022). Esta consiste en que un poderoso imán, gradientes de campos magnéticos y ondas de radio generadas por computadoras, escanean y muestran los órganos internos del cuerpo (Srinivas, C., S, N. P. K., Zakariah, M., Alotaibi, Y. A., Shaukat, K., Partibane, B., & Awal, H. 2022), de modo que, los algoritmos sean capaces de identificar las diferencias en los tejidos afectados, en las enfermedades a analizar. Estas patologías a identificar son:

- Neuromielitis óptica
- Encefalopatía posterior reversible
- Isquemia

3.5 ESTADO DEL ARTE

Referencia	Alcance	Título del trabajo	Aporte significativo
Ahmed, M., Seraj, R., & Islam, S. M. S. (2020). The k-means Algorithm: A Comprehensive Survey and Performance Evaluation. Electronics, 9(8), 1295. 10.1007/s00500-018-3618-7	Internacional	K-Means clustering and neural network for object detecting and identifying abnormality of brain tumor	Detectar e identificar la anormalidad debido al tumor cerebral, además de caracterizar el sistema para mejorar la imagen de resonancia magnética para el proceso segmentación.
Lee, L. H., Chen, C. H., Chang, W. C., Lee, P. L., Shyu, K. K., Chen, M. H., Hsu, J. W.,	Internacional	Evaluating the performance of machine learning models for automatic	La clasificación correcta de pacientes, de un grupo de muestra (pacientes con

Bai, Y. M., Su, T. P., & Tu, P. C. (2021a). Evaluating the performance of machine learning models for automatic diagnosis of patients with schizophrenia based on a single site dataset of 440 participants. European Psychiatry, 65(1). https://doi.org/10.1192/j.eurpsy.2021.2248		diagnosis of patients with schizophrenia based on a single site dataset of 440 participants	esquizofrenia y de control), del 85% por medio del método de máquina de soporte vectorial, la conectividad funcional tálamo-cerebelo funcionaria como un potencial marcador biológico para los pacientes con esquizofrenia, el método utilizado funciona mejor con grupos heterogéneos, para predicciones de sujetos singulares.
Melo Riveros, N. A., Cardenas Espitia, B. A., & Aparicio Pico, L. E. (2019). Comparison between K-means and Self-Organizing Maps algorithms used for diagnosis spinal column patients. Informatics in Medicine Unlocked, 16, 100206. https://doi.org/10.1016/j.imu.2019.100206	Nacional	Comparison between K-means and Self-Organizing Maps algorithms used for diagnosis spinal column patients	El método SOM (self-organizing maps), fue el más adecuado para detectar pacientes con anomalías espinales, mientras que el k-means lo es para pacientes normales, además que en este caso el primer método obtuvo los mejores resultados y fue la solución menos compleja.
Rodríguez-Bastidas, O., & Vargas-Rosero, H. F. (2020).	Nacional	Generación de modelos 3D de tumor desde imágenes DICOM,	Utilizando el método de segmentación de

Generation of 3D Tumor Models from DICOM Images for Virtual Planning of its Recession. Revista Facultad de Ingeniería, 29(54), e10173. https://doi.org/10.19053/01211129.v29.n54.2020.10173		para planificación virtual de su recesión	k-means y algoritmos de Triangulación de Delaunay se generaron modelos 3D de las imágenes encefálicas con posibilidad de navegar en este.
Viloria, A., Herazo-Beltran, Y., Cabrera, D., & Pineda, O. B. (2020). Diabetes Diagnostic Prediction Using Vector Support Machines. Procedia Computer Science, 170, 376–381. https://doi.org/10.1016/j.procs.2020.03.065	Regional	Diabetes Diagnostic Prediction Using Vector Support Machines	Presentar la viabilidad de utilizar un método como máquina de soporte vectorial, como herramienta durante el diagnóstico.

4 MATERIALES Y METODOS

Dado la información anterior se procedió a realizar el proyecto que dividió en tres etapas, en las que primero se investigó y se utilizó el concepto de segmentación dados los dos métodos escogidos, en donde se leyeron y reconfiguraron las imágenes iniciales escogidas para ambos casos, mediante las funciones de Python, de modo que en el caso de k-means se graficó el “ruido” de las imágenes a segmentar, se eligió un número de centroides, e iteraciones, se llevó a cabo el proceso de la técnica y recuperación de la imagen, para el método de SVM se aplicó una serie de filtros a los datos obtenidos, se llevó a efecto el proceso que conlleva la técnica y se recuperó la imagen. Luego se crearon los modelos de identificación de imágenes dados los modelos mencionados, en donde se leyeron y reconfiguraron, las imágenes de la base de datos obtenidas, en cada caso se configuraron los parámetros necesarios para poner en ejecución las técnicas

escogidas, con los datos obtenidos, de manera que los algoritmos, identificaron las patologías escogidas con cierta efectividad. Finalmente, se diseña la interfaz para presentar los resultados de los procesos anteriores.

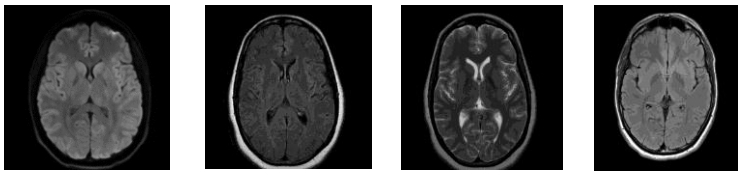
4.1 En el lenguaje de programación Python, se examinó cómo realizar el proceso de segmentación por medio de k-means, en imágenes cerebrales por resonancia magnética, de modo que se realizó el código necesario para la segmentación, en este se realiza la lectura de una de las figuras de muestra, en la que se realiza la lectura de una de las imágenes dadas, la cual es convertida a dos dimensiones, para realizar el procedimiento de k-means, antes de esto se realizaron las gráficas del arreglo 2d y el histograma para cuantizar cuáles serían los mejores parámetros a utilizar, de modo que después de ser realizado el procedimiento con ciertas, variables estas pueden ser modificadas dados los resultados previos.

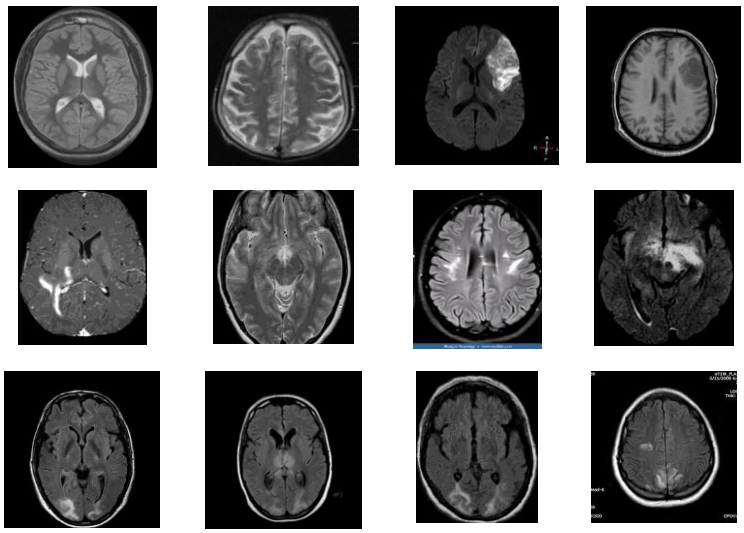
4.1.1 Previamente se escogieron las imágenes a utilizar para este proceso inicial se tiene 16 imágenes 4 por cada categoría, dado que se quiso iniciar con un numero reducidos de datos para las pruebas de los métodos iniciales a evaluar, de modo que se tiene los datos a leer para realizar las diferentes técnicas a utilizar, para lo anterior se lee una de las imágenes, en este caso con la función *imread* del módulo *io* del paquete *skimage*, que es una herramienta, de código abierto para él preprocesamiento de imágenes o con el módulo cv2 de la librería *OpenCV, que* permite realizar tareas de procesamiento de imágenes y visión de computadoras, esta función convierte la imagen en un arreglo de datos, dada la dirección o el nombre del archivo a utilizar.

Comentado [MAOP1]: Esta librería de OpenCV es importante de mencionar.

Figura 6

Las 16 Imágenes iniciales a segmentar con los métodos de k-means y SVM.





Nota. Las imágenes a utilizar estas son del formato png y tienen diferentes resoluciones

4.1.2 Las imágenes escogidas de la figura 6 se identifican con su nombre S_Br, un número y una letra dependiendo a la anomalía que se presenten, N por neuromielitis óptica (NMO), P por el síndrome de encefalopatía posterior reversible (PRES), I por isquemia o ninguna por las imágenes sin anomalías

Figura 7
Código y variables resultantes de la lectura de imágenes, dadas las funciones utilizadas.

```
7
8   from skimage import io
9   import cv2
10
11  image = io.imread('C:/Users/GLORIA GOMEZ/.spyder-py3/S_Br1.png',as_gray=True)/255.0
12  img = cv2.imread('S_Br4.png')
13  img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
14
```

Nam	Type	Size	Value
image	Array of float64	(2118, 2118)	[[0. 0. 0. ... 0. 0. 0.] [0. 0. 0. ... 0. 0. 0.]
img	Array of uint8	(630, 630, 3)	[[[5 5 5] [5 5 5]

Nota. Se utiliza dos imágenes diferentes para cada método de lectura y se observa que las variables son de diferentes tipos de arreglos, float para *io* y int para *cv2*.

4.1.3 En este caso como se observa en la figura 7, se agregó el “argumento” gray, en *imread* para tener la imagen monocromática y se normalizo los datos dividiendo éste por 255, de modo que los datos dentro de este sean menores a 1 y en *cvtColor* se utiliza dentro de esta *COLOR_BGR2RGB* para cambiar el espacio de color, es decir se cambia la configuración de colores a otra, de esta manera se tienen arreglos que corresponden a la resolución de la imagen, pero con diferentes tipos y uno teniendo un parámetro adicional para el color.

4.1.4 Seguidamente se configuro los datos de modo que se pueda utilizar para realizar las gráficas y el proceso de k-means por lo que se cambia el espacio de color de los datos por medio de *cvtColor* función de cv2, después se separan los valores rgb de manera que se puedan convertir en “arreglos 2d” por medio de *split* y *flatten* respectivamente, lo anterior también puede ser realizado mediante la función *reshape*, la cual cambia los datos de la imagen, de un arreglos de tres parámetros a dos.

Figura 8
Código de variables reconfiguradas para el proceso de k-means.

```
import cv2
from skimage import io


image = io.imread('C:/Users/GLORIA GOMEZ/.spyder-py3/S_Br1.png',as_gray=True)/255.0
h, w = image.shape

numcolors = 3

image_2d = image.reshape(h*w,1)

img = cv2.imread('S_Br4.png')
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
b, g, r = cv2.split(img)
b = b.flatten()
g = g.flatten()
r = r.flatten()

vec2 = img.reshape((-1, 1))
```

Nota. Las variable *img* cambia a tres, las cuales tiene los datos de la imagen original, dividido en colores y se vectoriza en *vec2*, y la variable *image* cambia a tener los datos en un solo parámetro del arreglo.

4.1.5 En esta aplicación de k-means para graficar los datos en 2d y el histograma se dividió *img* en tres, dados sus valores BGR mediante la función *split* y se cambia estos arreglos a una dimensión mediante la función *flatten*, además se cambia el arreglo *img* a uno de una dimensión mediante *reshape*, el cual se guardó en *vec2*, como se ve en la figura 8.

4.1.6 Al tener los datos necesarios se puede graficar de diversas formas, como se expone en la figura anterior (figura 8), las que se utilizó fue creando un gráfico 3d con *Axes3d* del módulo *mpl_toolkits.mplot3d*, de la librería *mpl_toolkitK*, que permite graficar multivectores, de modo que se pueda graficar elementos 3D para visualizar los datos en 2d y la función *.hist* para graficar el histograma.

4.1.7 De este modo se realizó el proceso de k-means, mediante el módulo de cv2 que tiene una función kmeans que realizar el proceso, para esto se tiene que tener las muestras, que serían los datos obtenidos por los procesos anteriores, el número de centroides, los intentos a realizar y el criterio, en donde se para las iteraciones si se tiene la precisión deseada o si el número máximo de iteraciones es alcanzado. Por lo que los resultados, en este caso la imagen segmentada es presentada.

Figura 9
Código para la realización del proceso de k-means.

```
43
44  vec = img.reshape((-1, 3))
45  vec = np.float32(vec)
46  print(vec.shape)
47  criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 100, 0.2)
48
49  K = 4
50  attempts = 10
51  ret,label,center = cv2.kmeans(vec, K, None, criteria, attempts, cv2.KMEANS_RANDOM_CENTERS)
52  label = label.flatten()
53
54  center = np.uint8(center)
55  res = center[label.flatten()]
56  result_image = res.reshape((img.shape))
```

4.1.8 Para la segmentación en k-means se vectoriza los datos de la imagen por *reshape*, se crea la variable criteria que por medio de *TEM_CRITERIA*, *TEM_CRITERIA_MAX_ITER*, y otros parámetros para el proceso, se escoge el número de centroides K y de intentos (attempts), para realizar el proceso de k-means, por medio de la función de *cv2*, kmeans, dado los datos anteriores y centroides aleatorios, y así obtener, las etiquetas y los centros resultantes, como se ve en la figura 9, ya presentada, de modo que se obtiene la imagen segmentada.

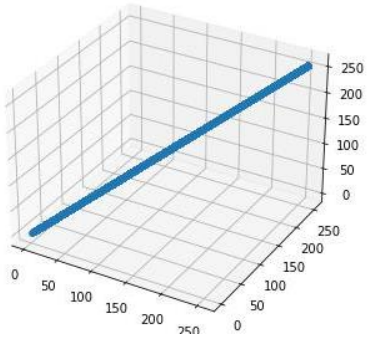
Figura 10

Código de segmentación de imágenes por k-means en Python

```
import numpy as np
from matplotlib import pyplot as plt
import cv2
from skimage import io
from mpl_toolkits.mplot3d import Axes3D
image = io.imread('C:/Users/GLORIA GOMEZ/.spyder-py3/S_Br1.png', as_gray=True)/255.0
h, w = image.shape
numcolors = 3
image_2d = image.reshape(h*w, 1)
img = cv2.imread('S_Br4.png')
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
r, g, b = cv2.split(img)
r = r.flatten()
g = g.flatten()
b = b.flatten()
vec2 = img.reshape((-1, 1))
fig = plt.figure()
ax = Axes3D(fig)
ax.scatter(r, g, b)
ax2 = fig.add_subplot(1, 2, 2)
ax2.hist(vec2, 256)
plt.show()
vec = img.reshape((-1, 3))
vec = np.float32(vec)
print(vec.shape)
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 100, 0.2)
K = 4
attempts = 10
ret,label,center = cv2.kmeans(vec, K, None, criteria, attempts, cv2.KMEANS_RANDOM_CENTERS)
label = label.flatten()
center = np.uint8(center)
res = center[label.flatten()]
result_image = res.reshape((img.shape))
res = center[label.flatten()]
result_image = res.reshape((img.shape))
figure_size = 10
plt.figure(figsize=(figure_size, figure_size))
plt.subplot(1, 2, 1), plt.imshow(img)
plt.title('Original Image'), plt.xticks([]), plt.yticks([])
plt.subplot(1, 2, 2), plt.imshow(result_image)
plt.title('Segmented Image when K = %i' % K), plt.xticks([]), plt.yticks([])
```

Figura 11
Gráfico de los datos de la imagen de entrada inicial, S_Br4 en dos dimensiones para el análisis, donde el eje x sería los valores en rojo de la imagen, el eje y los valores azules de la imagen y el eje z los valores verdes de la imagen.

z (Valores verdes)

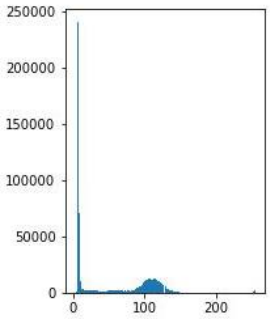


y (Valores Azules)

x (Valores rojos)

Figura 12
Gráfico del histograma de la imagen de entrada inicial, S_Br4 para la visualización de ruido. Donde el eje x se tienen la escala de colores de 0 a 255 y en eje y se corresponden los valores en cada uno de estas escalas, de modo que se observa que el negro o 0 es el color dominante en la imagen.

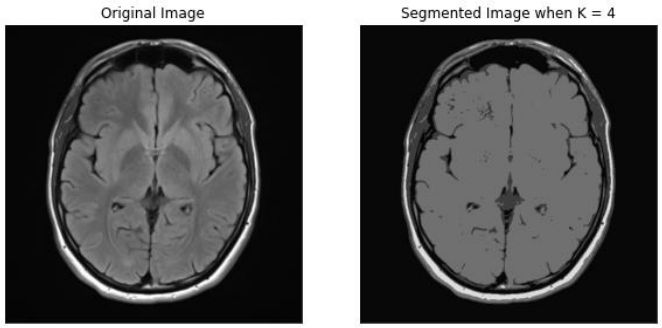
Comentado [MAOP2]: ¿Qué significan los ejes? Coloque las etiquetas en los ejes de las gráficas. Explique en el texto también.



Valores de la imagen

Escala de colores

Figura 13
Entrada e imagen segmentada S_Br4, donde en la izquierda se tiene la original y en la derecha está después del proceso con 4 centroides.



4.1.9 Se realizó una gráfica a través del mismo código en Python, se grafican el histograma de la imagen segmentada, para observar el ruido y una gráfica en donde, al analizarla se muestran los clusters óptimos para la tarea, como se observa en las figuras 11 y 12.

4.1.10 Después de tener el número de centroides e intentos necesarios, como se observa en las figuras 10 y 13 se efectuó la segmentación, utilizando una

función dentro de las librerías de Python, y se reconstruye la imagen segmentada.

4.2 Por otra parte, se llevó a cabo el proceso de segmentación para las imágenes expuestas previamente mediante el método de SVM, de manera similar al proceso anterior, en el que se lee la imagen, se vectoriza los datos obtenidos de la imagen, después de esto se pasan estos por una serie de filtros, se crea el modelo SVM, se guarda los datos en un archivo y se realiza el proceso, para así obtener la imagen segmentada.

Figura 14
Código de lectura de imagen y creación de tablas de datos para SVM, dada las funciones utilizadas.

```
img = cv2.imread('C:/Users/GLORIA GOMEZ/.spyder-py3/S_BrI.png')
img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

df = pd.DataFrame()

img2 = img.reshape(-1)
df['Original Image'] = img2
```

4.2.1 Para esto se utiliza la función *imread* de *cv2*, de modo que al tener el directorio de la imagen se obtiene los datos de esta, se convierte los valores de estos a blanco y negro con *cvtColor*, se crea una tabla de datos con la función *Dataframe* del módulo pandas, para la manipulación y análisis de datos, la cual se utilizó para implementar diferentes filtros a la imagen, como se observa en la imagen anterior (figura 14).

4.2.2 Para aplicar los diferentes filtros, para así detectar más fácilmente los bordes primero se utilizó un for anidado donde se implementa el filtro gabor, para los otros se utiliza las funciones del módulo *skimage.filter*, y se tomó los datos resultantes, a la tabla de datos.

Figura 15
Código para aplicar filtro gabor a la imagen de entrada S_BrI, después de la lectura

```
for theta in range(2):
    theta = theta/4*np.pi
    for sigma in (1, 3):
        for lamda in np.arange(0, np.pi/4):
            for gamma in (0.05, 0.5):

                gabor_label = 'Gabor' + str(num)
                ksize = 5
                kernel = cv2.getGaborKernel(
                    (ksize, ksize), sigma, theta, lamda, 0, ktype=cv2.CV_32F)
                kernels.append(kernel)
                fimg = cv2.filter2D(img2, cv2.CV_8UC3, kernel)
                filtered_img = fimg.reshape(-1)
                df[gabor_label] = filtered_img

                num += 1
```

4.2.3 En el for se utiliza la función de *cv2* de *getGaborKernel* para implementar el filtro dado, de modo que el resultado se guardó en las tablas de datos, de forma “vectorizada” como se observa en la figura 15.

Figura 16
Código para aplicar varios filtros a la imagen de entrada

```
edges = cv2.Canny(img, 100, 200)
edges1 = edges.reshape(-1)
df['Canny Edge'] = edges1

edge_roberts = roberts(img)
edge_roberts1 = edge_roberts.reshape(-1)
df['Roberts'] = edge_roberts1

edge_sobel = sobel(img)
edge_sobel1 = edge_sobel.reshape(-1)
df['Sobel'] = edge_sobel1

edge_scharr = scharr(img)
edge_scharr1 = edge_scharr.reshape(-1)
df['Scharr'] = edge_scharr1

edge_prewitt = prewitt(img)
edge_prewitt1 = edge_prewitt.reshape(-1)
df['Prewitt'] = edge_prewitt1

gaussian_img = nd.gaussian_filter(img, sigma=3)
gaussian_img1 = gaussian_img.reshape(-1)
df['Gaussian s3'] = gaussian_img1

gaussian_img2 = nd.gaussian_filter(img, sigma=7)
gaussian_img3 = gaussian_img2.reshape(-1)
df['Gaussian s7'] = gaussian_img1

median_img = nd.median_filter(img, size=3)
median_img1 = median_img.reshape(-1)
df['Median s3'] = median_img1

labeled_img = cv2.imread('S_BrI.png')
labeled_img = cv2.cvtColor(labeled_img, cv2.COLOR_BGR2GRAY)
labeled_img1 = labeled_img.reshape(-1)
df['Label'] = labeled_img1
```

Nota. Los filtros aplicados son roberts, sobel, scharr, prewitt, gaussiano y de la media.

4.2.4 Como se muestra en la figura 16 se creó el modelo de SVM y se cargó los datos de la imagen filtrada, para así realizar el proceso y obtener la imagen segmentada, como se expone en las imágenes anteriores.

Figura 17
Código de creación y proceso del modelo SVM

```
Y = df['Label'].values

X = df.drop(labels=['Label'], axis=1)

X_train, X_test, Y_train, Y_test = train_test_split(
    X, Y, test_size=0.4, random_state=20)

model = LinearSVC(max_iter=250)

model.fit(X_train, Y_train)
```

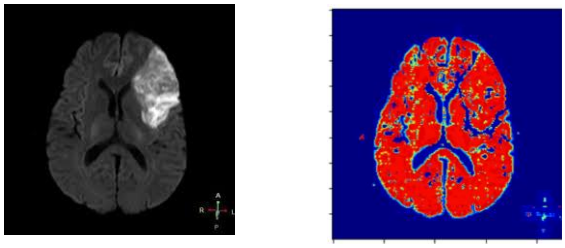
4.2.5 De este modo, como se presenta en la figura 17, se creó el modelo para realizar el proceso de necesario para obtener la imagen segmentada, mediante la función de *LinearSVC* del módulo *Sklearn.svm*, la cual hace parte de la biblioteca *scikit-learn*, que se utiliza para proyectos de aprendizaje automático. Lo anterior se logró dado diferentes intentos o iteraciones dados en el código, después de esto se ingresó los datos anteriores, preprocesados al modelo y se tiene los datos necesarios para obtener la imagen, esto dado la función *train_test_split*, para obtener datos que pudieran ser utilizados por la función, y la función *fit* por la cual se obtiene los datos segmentados.

Figura 18
Código de carga de datos y visualización de la segmentación

```
filename = 'brain_1'
pickle.dump(model, open(filename, 'wb'))

load_model = pickle.load(open(filename, 'rb'))

result = load_model.predict(X)

segmented = result.reshape((img.shape))

plt.imshow(segmented, cmap='jet')
```

4.2.6 Como se muestra en la figura 18 y la figura 19, mediante la librería pickle, la cual se utiliza para serializar y deserializar datos, de modo que se cargó los datos en un archivo, en este caso “brain_1”, se abrió este y mediante la función predict se obtuvo los datos de la imagen segmentada y se construyó la imagen mediante reshape, de modo que se pueda graficar.

Figura 19
Imagen de entrada S_BrI y la salida segmentada por el método de SVM.



4.3 Para la identificación de imágenes, de forma similar para el proceso de segmentación se leen las imágenes a utilizar de forma que se puedan realizar el método elegido y así entrenar el algoritmo, como se observa en las figuras 20, 21 y 22 se realiza una serie de pasos para ello, se declaran la variable de la dirección donde se encuentran las carpetas con los archivos y arreglos vacíos, los cuales tendrán los datos y las etiquetas de las categorías cuando se realiza el for anidado, este abre la carpeta con las subcarpetas con las imágenes, mediante *os.path.join* función del módulo os (el cual permite utilizar elementos del sistema operativo a Python) de modo que en un paso abre una de estas subcarpetas, después se toma el directorio de la imagen, se leen los datos de cada imagen de la subcarpeta mediante *os.path.join*, para tener el directorio de la imagen, y por la función de *cv2.imread* y se crea uno o dos arreglos, dependiendo del método, con los datos y las categorías, con la función *append*.

Figura 20
Código de lecturas de imágenes en SVM

```
import numpy as np
import cv2
import os

dir = 'C:\\Users\\GLORIA GOMEZ\\.spyder-py3\\BrIm'

Catg = ['isquemia', 'NMO', 'Normal', 'PRES']

data = []

for category in Catg:
    path = os.path.join(dir, category)
    label = Catg.index(category)
    for img in os.listdir(path):
        imgpath = os.path.join(path, img)
        Br_img = cv2.imread(imgpath, 0)
        try:
            Br_img = cv2.resize(Br_img, (50, 50))
            image = np.array(Br_img).flatten()
            data.append([image, label])
        except Exception as e:
            pass
```

Nota. Con *resize* se cambia la resolución de la imagen, de modo que los datos resultantes sean más manejables.

Figura 21
Código de lecturas de imágenes en k-means

```
np.random.seed(42)
dir = 'C:\\Users\\GLORIA GOMEZ\\.spyder-py3\\BrIm'
Catg = ['isquemia', 'NMO', 'Normal', 'PRES']
data = []
data1 = []
labela = []
data_label = []
d = 0
SZ = 32

for category in Catg:
    path = os.path.join(dir, category)
    label = Catg.index(category)

    for img in os.listdir(path):
        imgpath = os.path.join(path, img)
        Br_img = cv2.imread(imgpath, 0)
        try:
            Br_img = cv2.resize(Br_img, (SZ, SZ))
            image1 = np.array(Br_img).flatten()
            Br_img = Br_img.astype('float32')
            if "I" in img[:6]:
                # a += 1
                labela.append("isquemia")
            elif "N" in img[:6]:
                # b += 1
                labela.append("NMO")
            elif "P" in img[:6]:
                # c += 1
                labela.append("PRES")
            else:
                if d==4: continue
                d += 1
                labela.append("Normal")
            data1.append(Br_img)
            # data.append([image1, label])

        except Exception as e:
            pass
data1 = np.array(data1)
```

Figura 22
Código de categorización de imágenes en k-means

```
for i in labela:
    if i == "isquemia":
        data_label.append(0)
    elif i == "NMO":
        data_label.append(1)
    elif i == "Normal":
        data_label.append(2)
    else:
        data_label.append(3)
data_label = np.array(data_label)
```

4.4 Se examinó la identificación de imágenes por medio de k-means, por lo que se configura los datos adquiridos para el proceso.

Figura 23
Código de proceso de k-means y resultados previos

```
data1 = data1/255.0



#k-means accept data with less than 3 dimensions
reshaped_data = data1.reshape(len(data1), -1)
reshaped_data.shape


kmeans = KMeans(n_clusters=4, random_state=0)
clusters = kmeans.fit_predict(reshaped_data)
kmeans.cluster_centers_.shape
# 2,1024
kmeans.cluster_centers_ = kmeans.cluster_centers_*255
plt.figure(figsize = (10,9))
bottom = 0.35
for i in range(2):
    plt.subplots_adjust(bottom)
    plt.subplot(4,4,i+1)
    plt.imshow(kmeans.cluster_centers_[i].astype(int).reshape(32,32,-1))
```

4.4.1 Para realizar el proceso se normalizo los valores del arreglo *data1*, dividiéndolo por 255 y dado que en este caso, para realizar el proceso de k-means se necesita que los datos sean de dos dimensiones, de modo que por medio de *reshape*, se cambia de 3 a dos dimensiones, el arreglo con la información, dado lo anterior se realiza el proceso de k-means, con cierto número de centroides y se muestra el resultado de este proceso dada las coordenadas de los centroides obtenidas por *cluster_center*, de modo que se pueda observar en una imagen graficada por *imshow*, esto se expone en la figura anterior (figura 23).

Figura 24
Código de proceso de visualización de resultados mediante gráfico.

```
x_data = [i for i in range(1024)]
plt.scatter(x_data, kmeans.cluster_centers_[0], color='red', alpha=0.2, s=70)
plt.scatter(x_data, kmeans.cluster_centers_[1], color='blue', alpha=0.2, s=50)
plt.scatter(x_data, kmeans.cluster_centers_[2], color='green', alpha=0.2, s=30)
plt.scatter(x_data, kmeans.cluster_centers_[3], color='yellow', alpha=0.2, s=10)
```

4.4.2 La anterior figura muestra que después de ser realizado el proceso anterior, primero se observó la separación de datos realizados por el algoritmo, dadas las 4 categorías mediante *scatter*, de manera que se observa gráficamente la efectividad de este.

Figura 25
Código de extracción de categorías y de predicción o identificación de categorización.

```
reference_labels = {}


def get_reference_dict(clusters, data_label):
    reference_label = {}
    # For loop to run through each label of cluster label
    for i in range(len(np.unique(clusters))):
        index = np.where(clusters == i, 1, 0)
        num = np.bincount(data_label[index == 1]).argmax()
        reference_label[i] = num
    return reference_label


#Mapping predictions to original labels
def get_labels(clusters, refernce_labels):
    temp_labels = np.random.rand(len(clusters))
    for i in range(len(clusters)):
        temp_labels[i] = reference_labels[clusters[i]]
    return temp_labels


reference_labels = get_reference_dict(clusters, data_label)
predicted_labels = get_labels(clusters, reference_labels)

print(accuracy_score(predicted_labels, data_label))
```

4.4.3 Además del procedimiento anterior se extrae las “etiquetas” de referencias y las identificadas mediante funciones creadas, *get_reference_dict* y *get_labels*, respectivamente, la primera toma los datos diferentes del arreglo de centroides, con la función de *numpy unique*, y extrae los datos buscados con la *np.where*, y la segunda funciona tomando datos aleatorios en arreglos temporales con *np.ramdon.rand* dado el tamaño de los centroides, lo que se observa en la figura 25. De esta manera se obtiene la precisión y la identificación realizadas.

Figura 26
Código de exposición de resultados del proceso de identificación por k-means.

```
ran_numb = random.randint(0, 15)

img_identify = predicted_labels[ran_numb]

img_in = data_label[ran_numb]


Catg = ['isquemia', 'NMO', 'Normal', 'PRES']

print(Catg[int(img_identify)])
print(Catg[int(img_in)])

Nk_brain = data2[ran_numb]
plt.imshow(Nk_brain, cmap='gray')
```

4.4.4 En la figura 26 se muestra que mediante la función *random.randint* un número aleatorio, de modo que se escoja una imagen y categorización aleatoria respectiva, para ser mostrada con *print* y *imshow* en la consola

4.5 Se examinó el proceso de obtención de las imágenes por máquina de soporte vectorial, por lo que se escrutará como se configuran los datos en la imagen, dado que esto es lo que dicta cuál es la la eficacia del algoritmo.

4.5.1 De modo que al tener los datos de las imágenes mediante la librería *pickle* se escriben los datos obtenidos en un archivo en este caso data1.pickle esto mediante las funciones *open* y *pickle.dump*. de esta manera el archivo puede ser leído a través de open *pickle.load*

4.5.2 Se examinó el proceso de obtención de las imágenes por máquina de soporte vectorial, en Python, por lo que se escogió cómo se debían configurar los datos en la imagen, de modo que se separaron los datos de forma efectiva, de modo que se encuentren la mejor separación de estos.

4.5.3 Después de lo anterior al realizar pruebas previas al algoritmo se “mezclan” los datos del archivo, después se obtuvo las etiquetas y características de los datos, mediante un for que las junta en un arreglo mediante *append*, y se separan las variables en conjuntos de prueba y entrenamiento, por la función de *train_test_split* del módulo *sklearn.model_selection*, de los cuales se utilizara los de entrenamiento dentro del modelo de SVM, después de ser creado, para así ser guardado en un archivo con la función *dump*.

Figura 27
Mezcla de datos, entrenamiento y guardado del algoritmo.

```
random.shuffle(data)
features = []
labels = []

for feature, label in data:
    features.append(feature)
    labels.append(label)

xtrain, xtest, ytrain, ytest = train_test_split(features, labels, test_size=0.5)

model = SVC(C=1, kernel='poly', gamma='auto')
model.fit(xtrain, ytrain)

pick = open('model.sav', 'wb')
pickle.dump(model, pick)
pick.close()
```

4.5.4 Mediante el código escrito en Python, que se muestra en la figura 27 se realizó la identificación de la imagen, dado los datos a analizar. De modo que se “mezclan” y extraen los datos del archivo mediante las funciones *shuffle* y *append*, y el *form* en donde se extrajeron los datos de las imágenes y la categorización hecha de cada una de estas, para ser divide en muestras mediante *train_test_split*, se creó el modelo con la función *SVC*, y se realiza la identificación con la función *fit*, para así guardarlas en un archivo mediante *dump*.

Figura 28
Código de presentación de resultados de identificación por SVM

```
pick = open('model.sav', 'rb')
model = pickle.load(pick)
pick.close()

prediction = model.predict(xtest)
accuracy = model.score(xtest, ytest)

Catg = ['isquemia', 'NMO', 'Normal', 'PRES']

print('Accuracy', accuracy)

print('Prediction is :', Catg[prediction[0]])

Nbrain = xtest[0].reshape(50, 50)
plt.imshow(Nbrain, cmap='gray')
plt.show()
```

4.5.5 Como se muestra en la figura 28, se evaluó los resultados de este estadio, se lee el archivo creado con *load*, de modo que se "predice" las categorías mediante *predict* del módulo de *sklearn* y se muestra la precisión del algoritmo, comparando las predicciones, con los datos de entrenamiento con la función score del módulo de *sklearn* y se presentó la imagen identificada, después de se cambió su resolución por *reshape*, mediante *imshow*.

4.6 Se muestran los datos de los métodos de segmentación anteriores en forma de porcentajes de precisión con respecto a la probabilidad de una predicción correcta

4.6.1 Se realizó un diseño de interfaz, la cual muestra la información recolectada de manera más intuitiva posible, y que es simple de comprender por el usuario.

Figura 29
Código de creación de la ventana de la interfaz y configuración de ciertos parámetros.

```
root = Tk()
root.title("GUI software de identificación de imagenes en base *")
root.iconbitmap('ico_Br.ico')
root.geometry("800x600")

root['background'] = '#19232d'
```

4.6.2 Para esto se utiliza librería *tkinter*, de modo que mediante sus "widgets" se creó una interfaz que cumple con las necesidades planteadas. Para esto primero se crea la raíz de la interfaz con el "widget" *Tk*, de esta manera se puede implementar las otras aplicaciones, pero primero se le da un título, un icono, el tamaño inicial, y el color del fondo de la ventana mediante los

comandos, *title*, *iconbitmap*, *geometry* y *background*, respectivamente como se ve en la figura 29.

Figura 30
Código de creación de cuatro marcos de la ventana y organización.

```
upFrame = Frame(root, width=200, height=600)
upFrame.grid(row=0, column=0, padx=70, pady=2)

upFrame1 = Frame(root, width=200, height=600, bg='#19232d')
upFrame1.grid(row=0, column=1, padx=70, pady=2)


rightFrame = Frame(root, width=200, height=600)
rightFrame.grid(row=1, column=1, padx=70, pady=2)

leftFrame = Frame(root, width=200, height=600)
leftFrame.grid(row=1, column=0, padx=50, pady=10)
```

Nota. En el código *upFrame* sería el marco superior izquierdo, *upFrame1* es el marco superior derecho, *rigthFrame* sería el marco inferior izquierdo, *leftFrame* es el marco inferior derecho.

4.6.3 Luego para la organización como se ve en la figura 30 y visualización con el widget Frame, se crearon 4 marcos y con el comando grid se organizaron estos.

Figura 31
Código de lectura y reconfiguración de imágenes en la interfaz.

```
img_1 = Image.open("UPB_Logo.jpg")

rsz_img1 = img_1.resize((220, 110), Image.ANTIALIAS)
N_img1 = ImageTk.PhotoImage(rsz_img1)
imgtk = ImageTk.PhotoImage(image=Image.fromarray(Nbrain))
```

4.6.4 Como se expone en la figura 31, se realiza la lectura y cambio de tamaño de las imágenes que son mostradas en la interfaz, en este caso la del logo de UPB y la imagen a realizar la identificación con las funciones *open* y *PhotoImage*, para la lectura y *resize*, para el cambio de tamaño.

Figura 32
Código de creación entradas para nombre y edad del paciente.

```
e = Entry(rightFrame, width=30)
e_1 = Entry(rightFrame, width=30)

e.grid(row=0, column=1)

e_1.grid(row=1, column=1)


def getName():
    name = e.get()
    label_a = Label(rightFrame, text=name)
    label_a.grid(row=3, column=1, padx=20)


def getAge():
    name = e_1.get()
    label_e = Label(rightFrame, text=name)
    label_e.grid(row=4, column=1, padx=20)
```

4.6.5 Como se ve en la figura 32, se crean las entradas para que el usuario pueda ingresar el nombre y la edad del paciente con el “widget”, *Entry*, y se crean las funciones que permiten al pulsar un botón tener estas entradas en la interfaz con la función *get* y *Label*.

Figura 33
Código de creación del texto e imágenes en la interfaz.

```
label_logo = Label(upFrame, image=N_img1).grid(row=0, column=0)
label_img = Label(leftFrame, image=imgtk).grid(row=0, column=0)
label_a = Label(rightFrame, text="Nombre: ").grid(row=3, column=0, padx=20)
label_e = Label(rightFrame, text="Edad: ").grid(row=4, column=0, padx=20)
label_1 = Label(
    rightFrame, text="Anomalia: "
    + str(Catg[ytest[0]]))
label_1.grid(row=5, column=0, padx=20)
label_2 = Label(rightFrame,
                text="Identificada: "
                + str(Catg[prediction[0]]))
label_2.grid(row=6, column=0, padx=20)
label_3 = Label(rightFrame, text="Porcentaje de confiabilidad: "
                + str(accuracy_percent) + '%')
label_3.grid(row=7, column=0, padx=20)
label_5 = Label(upFrame1,
                text="UNIVERSIDAD PONTIFICIA BOLIVARIANA",
                bg='#19232d', fg='#fff').grid(row=0, column=0, padx=20)
label_6 = Label(upFrame1,
                text="Programa de ingenieria electronica",
                bg='#19232d', fg='#fff').grid(row=1, column=0, padx=20)
label_7 = Label(upFrame1,
                text="Desarrollado por: Juan D.",
                bg='#19232d', fg='#fff').grid(row=3, column=0, padx=20)
```

4.6.6 Después se muestran las imágenes y el texto necesario para presentar los resultados, como Anomalía y porcentaje de confiabilidad mediante la función *Label*, lo cual es presentado en la imagen anterior (figura 33).

Figura 34
Código para la identificación de una imagen de entrada dada por el usuario en base al algoritmo SVM.

```
data_img = []


def openf():
    global label_img
    global img_l
    global img_r
    global label_2
    global label_3
    root.filename = filedialog.askopenfilename(
        initialdir='C:\\Users\\GLORIA GOMEZ\\.spyder-py3\\BrIm',
        title="Selecciona una imagen", filetypes=(("png files", "*.png"),
                                                  ("all files", "*.*")))
    img_l = Image.open(root.filename)
    Br_imgtemp = cv2.imread(root.filename, 0)
    Br_imgtemp = cv2.resize(Br_imgtemp, (50, 50))
    imagetemp = np.array(Br_imgtemp).flatten()
    data_img.append(imagetemp)
    prediction_t = model.predict(data_img)
    rzs_img = img_l.resize((250, 250), Image.ANTIALIAS)
    img_r = ImageTk.PhotoImage(rzs_img)
    label_img = Label(leftFrame, image=img_r).grid(row=0, column=0)
    label_1.grid_forget()
    label_2.grid_forget()
    label_2 = Label(
        rightFrame, text="Identificada: "
        + str(Catg[int(prediction_t)])).grid(row=6, column=0, padx=20)
```

Figura 35
Código para la identificación de una imagen de entrada dada por el usuario en base al algoritmo k-means.

```
def openf():
    global label_img
    global img_l
    global img_r
    global label_1
    global label_2
    root.filename = filedialog.askopenfilename(
        initialdir='C:\\Users\\GLORIA GOMEZ\\.spyder-py3\\BrIm',
        title="Selecciona una imagen", filetypes=(("png files", "*.png"),
                                                  ("all files", "*.*")))
    img_l = Image.open(root.filename)
    img_temp = cv2.imread(root.filename, 0)
    img_temp = cv2.resize(img_temp, (SZ, SZ))
    # img_tempo1 = np.array(img_temp).flatten()
    img_temp = img_temp.astype('float32')
    img_temp1 = img_temp/255
    reshaped_img = img_temp1.reshape(-1, 1024)
    reshaped_img.shape
    cluster_img = kmeans.predict(reshaped_img)
    pred_img = get_labels(cluster_img, reference_labels)
    rzs_img = img_l.resize((250, 250), Image.ANTIALIAS)
    img_r = ImageTk.PhotoImage(rzs_img)
    label_img = Label(leftFrame, image=img_r).grid(row=0, column=0)
    label_1.grid_forget()
    label_2.grid_forget()
    label_2 = Label(rightFrame,
                    text="Identificada: "
                    + str(Catg[int(pred_img)])).grid(row=6, column=0, padx=20)
```

4.6.7 Se crea un comando, la cual al presionar un botón, se lee una imagen, la cual puede ser exterior de la base de datos de entrenamiento, con las funciones open y imread, se realiza el preprocesamiento necesario, para utilizar la función predict, de esta forma se muestran los resultados en la interfaz con la función Label, como se ve en la figura 34 y 35.

Figura 36
Código para la creación de botones e interfaz en funcionamiento.

```
button_name = Button(rightFrame, text="nombre",
                     command=getName).grid(row=0, column=0)

button_age = Button(rightFrame, text="edad",
                    command=getAge).grid(row=1, column=0)

button_open = Button(leftFrame, text="Open", bg="#6F7997", fg="white",
                     command=openf).grid(row=1, column=0)

root.mainloop()
```

4.6.8 Finalmente se crean los botones y con *mainloop*, se mantiene en funcionamiento la ventana como se expone en la figura 36.

5 RESULTADOS Y DISCUSIÓN.

Dado las tareas y métodos realizados en la sección anterior se tiene como resultado el código del método de k-means, dada la metodología planteada en la sección 3.4 el cual, al ser entrenado con la base de datos, realiza la identificación de una anomalía dada una imagen, que se muestra en el anexo 7.1, también se obtuvo un segundo código de identificación desarrollado dada la metodología planteada en la sección 3.5 para el método SVM, que se muestra en el anexo 7.2.

Dado lo realizado en la metodología en la sección 3.6 se tiene como resultado la interfaz que muestra todos los resultados pertinentes que se muestran en las siguientes figuras 37 y 38.

Figura 37
Interfaz para la presentación de resultados de identificación por SVM

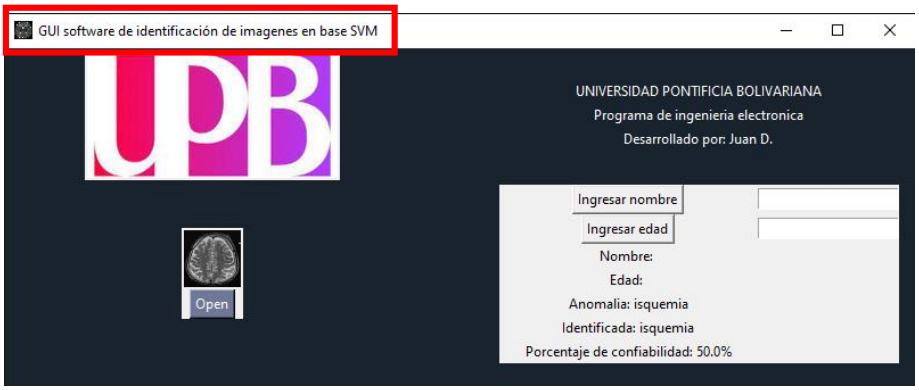
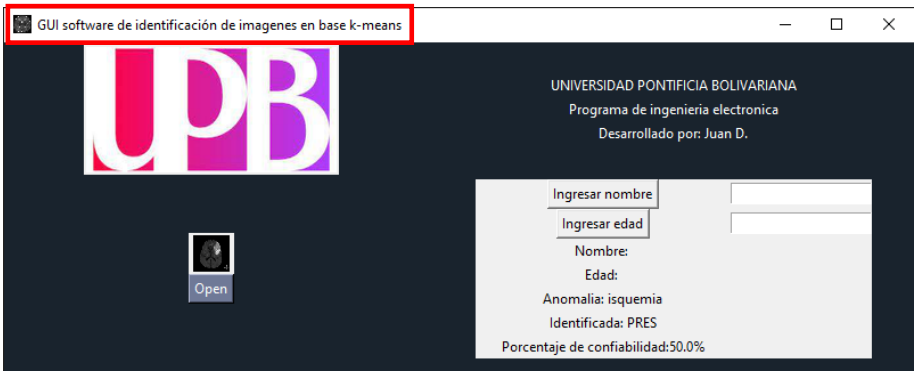


Figura 38

Interfaz para la presentación de resultados de identificación por k-means



Dado lo anterior se realizan pruebas para observar la eficiencia y confiabilidad de los algoritmos en base a los métodos anteriores, que se muestra en la tabla 1.

Tabla 1
Comparación de porcentaje de éxito entre los dos algoritmos desarrollados.

Anomalía	Porcentaje de éxito de k-means (%)	Porcentaje de éxito de SVM (%)
Sin anomalías	25	50
Isquemia	50	75
NMO	25	50
PRES	75	75

Siendo el porcentaje promedio de éxito de k-means con una base de datos limitada de 43.75 y de SVM de 62.5, de modo que si comparamos los resultados reportados por *Diabetes Diagnostic Prediction Using Vector Support Machines* (2020), de efectividad del 99.2% con pacientes colombianos y del 65.6% con pacientes de distintos transfondos etnicos en la identificación de diabetes, y *Comparison between K-means and Self-Organizing Maps algorithms used for diagnosis spinal column patients* (2019), en donde se obtuvo una precisión del 82.6%, para la identificación de problemas lumbares mediante datos.

6 CONCLUSIONES Y RECOMENDACIONES:

La segmentación de imágenes si bien es una herramienta útil en la visión artificial, no es utilizado ampliamente para la identificación de imágenes, debido a que existen

otras herramientas, métodos durante el preprocesamiento de datos, que reduce su utilidad para el proceso planteado. Específicamente dentro de este proyecto, al tener las imágenes a identificar, convertirlas en arreglos es una tarea trivial, por lo que dentro de este se optó por reducir los datos de la imagen, reduciendo su resolución, y cambiar los parámetros del tamaño de los datos como pasos del preprocesamiento.

Tabla 2
Comparación de porcentaje de éxito entre los resultados del proyecto y de los resultados dados por el estado de arte.

Método	**Proyecto**	**Estado del arte**
k-means	43,75%	82,6%
SVM	62,5%	65,6%

El porcentaje mostrado en la tabla del método SVM del proyecto frente al estado del arte, se toma de *Diabetes Diagnostic Prediction Using Vector Support Machines* (2020) y el porcentaje expuesto en la tabla del metodo k-means, del proyecto frente al estado del arte se toma de
Comparison between K-means and Self-Organizing Maps algorithms used for diagnosis spinal column patients (2019).

El método de k-means, sin utilizar otras herramientas, dentro del proceso de entrenamiento del algoritmo, no es el más eficiente para el proceso de identificación de imágenes, dado que este es un algoritmo de tipo no supervisado, de modo que no se puede utilizar los resultados para mejorar el proceso, aunque los resultados muestran que si se tienen dos categorías este puede tener cierta proficiencia, pero cuando se tiene varias categorías, esta decrece a un nivel poco confiable.

El método de máquina de soporte vectorial tiene resultados más confiables en cuanto a la precisión de las categorías identificadas, aun así, esto se podría mejorar, mediante el cambio de ciertos parámetros en las etapas de procesado y durante la creación del modelo, como serian el número de iteraciones en k-means , la magnitud de c y gamma en SVM y el tamaño de la resolución, lo cual debe ser considerado si se requiere expandir o tener una herramienta que pueda ser utilizada en la prestación de servicios de diagnóstico médico.

Se sugiere que se continúe con los conceptos, herramientas y métodos similares utilizados en el documento, de modo que se desarrollen aplicaciones que ayuden al diagnóstico dentro de la herramienta, como enlazar las imágenes a analizar con los datos pertinente del paciente dentro de una base de datos, como lo sería el nombre, la edad y otros síntomas que se observen en este, de modo que se tenga estos para futuros análisis o tratamientos, esto dentro del software de la infraestructura de las EPS e IPS que la utilicen. También se insta a utilizar otros modelos para la identificación o probar la eficiencia de otros métodos como los que utilizan redes neuronales como keras y redes convolucionales.

7 BIBLIOGRAFÍA

Cervantes, J., Garcia-Lamont, F., Rodríguez-Mazahua, L., & Lopez, A. (2020). A comprehensive survey on support vector machine classification: Applications, challenges and trends,. Neurocomputing, 189-215.

Fränti, P., & Sieranoja, S. (2019). How much can k-means be improved by using better initialization and repeats? About the journal, 95-122.

Mohiuddin , A., Raihan, S., & Syed Mohammed, S. I. (2020). The k-means Algorithm: A Comprehensive Survey and Performance Evaluation. Electronics, 1295.

Sinaga, P. K., & Yang, M.-S. (2020). Unsupervised K-Means Clustering Algorithm. IEEE, 80716 - 80727.
Sultana, F., Sufian, A., & Dutta, P. (2020). Evolution of Image Segmentation using Deep Convolutional Neural Network: A Survey. Knowledge-Based Systems, 950-7051.

Wang, M., & Chen, H. (2020). Chaotic multi-swarm whale optimizer boosted support vector machine for medical diagnosis. Applied Soft Computing.

MacKay, D. (2003). Information theory, inference, and learning algorithms. https://www.inference.org.uk/itprnn/book.pdf

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2021). An Introduction to Statistical Learning: With Applications in R (2nd 2021 ed.). Springer.

Sadeghi, D., Shoeibi, A., Ghassemi, N., Moridian, P., Khadem, A., Alizadehsani, R., Teshnehlab, M., Górriz, J. M., & Nahavandi, S. (2022). An overview of artificial intelligence techniques for diagnosis of Schizophrenia based on magnetic resonance imaging modalities: Methods, challenges, and future works. Computers in Biology and Medicine, 146, 105554. https://doi.org/10.1016/j.compbiomed.2022.105554

Shoeibi, A., Ghassemi, N., Khodatars, M., Moridian, P., Khosravi, A., Zare, A., Gorriz, J. M., Chale-Chale, A. H., Khadem, A., & Acharya, U. (2022b). Automatic diagnosis of schizophrenia and attention deficit hyperactivity disorder in rs-fMRI modality using convolutional autoencoder model and interval type-2 fuzzy regression. Cognitive Neurodynamics. https://doi.org/10.1007/s11571-022-09897-w

National Health Service [NHS England]. (2021, 18 noviembre). Diagnostic Imaging Dataset 2021-22 Data. NHS England. Recuperado 28 de abril de 2023, de https://www.england.nhs.uk/statistics/statistical-work-areas/diagnostic-imaging-dataset/diagnostic-imaging-dataset-2021-22-data/

National Health Service [NHS England]. (2023, 20 abril). Diagnostic Imaging Dataset 2022-23 Data. NHS England. Recuperado 28 de abril de 2023, de https://www.england.nhs.uk/statistics/statistical-work-areas/diagnostic-imaging-dataset/diagnostic-imaging-dataset-2022-23-data/

Tack, C. (2019). Artificial intelligence and machine learning | applications in musculoskeletal physiotherapy. Musculoskeletal science and practice, 39, 164-169. https://doi.org/10.1016/j.msksp.2018.11.012

Srinivas, C., S, N. P. K., Zakariah, M., Alotaibi, Y. A., Shaukat, K., Partibane, B., & Awal, H. (2022). Deep Transfer Learning Approaches in Performance Analysis of Brain Tumor Classification Using MRI Images. Journal of Healthcare Engineering, 2022, 1-17. https://doi.org/10.1155/2022/3264367

Arabahmadi, M., Farahbakhsh, R., & Rezazadeh, J. (2022). Deep Learning for Smart Healthcare—A Survey on Brain Tumor Detection from Medical Imaging. Sensors, 22(5), 1960. https://doi.org/10.3390/s22051960

Srinivas, C., S, N. P. K., Zakariah, M., Alotaibi, Y. A., Shaukat, K., Partibane, B., & Awal, H. (2022b). Deep Transfer Learning Approaches in Performance Analysis of Brain Tumor Classification Using MRI Images. Journal of Healthcare Engineering, 2022, 1-17. https://doi.org/10.1155/2022/3264367

8 ANEXOS

8.1

Código de identificación de anomalías en base k-means

```
import os
import random
import numpy as np
import matplotlib.pyplot as plt
import cv2
from sklearn.cluster import KMeans
from sklearn.metrics import accuracy_score
from tkinter import Tk, Label, Button, Frame, filedialog
from PIL import ImageTk, Image


root = Tk()
root.title("GUI software de identificación de imagenes en base k-means")
root.iconbitmap('ico_Br.ico')
root.geometry("800x600")
root['background'] = '#19232d'

img_1 = Image.open("UPB_logo.jpg")

np.random.seed(42)

dir = 'C:\\Users\\GLORIA GOMEZ\\.spyder-py3\\BrIm'
img_temp = cv2.imread('C:/Users/GLORIA GOMEZ/.spyder-py3/S_BrI.png', 0)

Catg = ['isquemia', 'NMO', 'Normal', 'PRES']
data = []
data1 = []
labela = []
data_label = []

# a = 0
# b = 0
# c = 0
d = 0
SZ = 32
```

```
# Data reading

for category in Catg:
    path = os.path.join(dir, category)
    label = Catg.index(category)

    for img in os.listdir(path):
        imgpath = os.path.join(path, img)
        Br_img = cv2.imread(imgpath, 0)
        try:
            Br_img = cv2.resize(Br_img, (SZ, SZ))
            image1 = np.array(Br_img).flatten()
            Br_img = Br_img.astype('float32')
            if "I" in img[:6]:
                # a += 1
                labela.append("isquemia")
            elif "N" in img[:6]:
                # b += 1
                labela.append("NMO")
            elif "P" in img[:6]:
                # c += 1
                labela.append("PRES")
            else:
                if d==4: continue
                d += 1
                labela.append("Normal")
            data1.append(Br_img)
            # data.append([image1, label])

        except Exception as e:
            pass

data1 = np.array(data1)

for i in labela:
    if i == "isquemia":
        data_label.append(0)
    elif i == "NMO":
        data_label.append(1)
    elif i == "Normal":
        data_label.append(2)
    else:
        data_label.append(3)
data_label = np.array(data_label)

data2 = data1
```

```
data1 = data1/255.0

# k-means accept data with less than 3 dimensions
reshaped_data = data1.reshape(len(data1), -1)
reshaped_data.shape


kmeans = KMeans(n_clusters=4, random_state=0)
clusters = kmeans.fit_predict(reshaped_data)
kmeans.cluster_centers_.shape
# 2,1024
kmeans.cluster_centers_ = kmeans.cluster_centers_*255
plt.figure(figsize = (10, 9))
bottom = 0.35

reference_labels = {}


def get_reference_dict(clusters, data_label):
    reference_label = {}
    # For loop to run through each label of cluster label
    for i in range(len(np.unique(clusters))):
        index = np.where(clusters == i, 1, 0)
        num = np.bincount(data_label[index == 1]).argmax()
        reference_label[i] = num
    return reference_label


# Mapping predictions to original labels
def get_labels(clusters, refernce_labels):
    temp_labels = np.random.rand(len(clusters))
    for i in range(len(clusters)):
        temp_labels[i] = reference_labels[clusters[i]]
    return temp_labels


reference_labels = get_reference_dict(clusters, data_label)
predicted_labels = get_labels(clusters, reference_labels)

print(accuracy_score(predicted_labels, data_label))


ran_numb = random.randint(0, 15)

img_identify = predicted_labels[ran_numb]
```

```
img_in = data_label[ran_numb]


Catg = ['isquemia', 'NMO', 'Normal', 'PRES']

print(Catg[int(img_identify)])
print(Catg[int(img_in)])

Nk_brain = data2[ran_numb]
plt.imshow(Nk_brain, cmap='gray')

upFrame = Frame(root, width=200, height=600)
upFrame.grid(row=0, column=0, padx=70, pady=2)

upFrame1 = Frame(root, width=200, height=600, bg='#19232d')
upFrame1.grid(row=0, column=1, padx=70, pady=2)

rightFrame = Frame(root, width=200, height=600)
rightFrame.grid(row=1, column=1, padx=70, pady=2)

leftFrame = Frame(root, width=200, height=600)
leftFrame.grid(row=1, column=0, padx=50, pady=10)

rsz_img1 = img_1.resize((220, 110), Image.ANTIALIAS)
N_img1 = ImageTk.PhotoImage(rsz_img1)
# N_img = ImageTk.PhotoImage(Nbrain)
imgtk = ImageTk.PhotoImage(image=Image.fromarray(Nk_brain))

e = Entry(rightFrame, width=20)
e_1 = Entry(rightFrame, width=20)

e.grid(row=0, column=1)

e_1.grid(row=1, column=1)


def getName():
    name = e.get()
    label_a = Label(rightFrame, text=name)
    label_a.grid(row=3, column=1, padx=20)


def getAge():
    name = e_1.get()
    label_e = Label(rightFrame, text=name)
```

```
    label_e.grid(row=4, column=1, padx=20)


accuracy_percent = accuracy_score(predicted_labels, data_label)*100


label_logo = Label(upFrame, image=N_img1).grid(row=0, column=0)
label_img = Label(leftFrame, image=imgtk).grid(row=0, column=0)
label_a = Label(rightFrame, text="Nombre: ").grid(row=3, column=0, padx=20)
label_e = Label(rightFrame, text="Edad: ").grid(row=4, column=0, padx=20)
label_1 = Label(
    rightFrame, text="Anomalia: "
    + str(Catg[int(img_in)]))
label_1.grid(row=5, column=0, padx=20)
label_2 = Label(rightFrame,
                text="Identificada: "
                + str(Catg[int(img_identify)]))
label_2.grid(row=6, column=0, padx=20)
label_3 = Label(rightFrame, text="Porcentaje de confiabilidad:"
                + str(accuracy_percent) + "%").grid(
                    row=7, column=0, padx=20)
label_4 = Label(upFrame1,
                text="UNIVERSIDAD PONTIFICIA BOLIVARIANA",
                bg='#19232d', fg='#fff').grid(row=0, column=0, padx=20)
label_5 = Label(upFrame1,
                text="Programa de ingenieria electronica",
                bg='#19232d', fg='#fff').grid(row=1, column=0, padx=20)
label_7 = Label(upFrame1,
                text="Desarrollado por: Juan D.",
                bg='#19232d', fg='#fff').grid(row=3, column=0, padx=20)



def openf():
    global label_img
    global img_l
    global img_r
    global label_1
    global label_2
    root.filename = filedialog.askopenfilename(
        initialdir='C:\\Users\\GLORIA GOMEZ\\.spyder-py3\\BrIm',
        title="Selecciona una imagen", filetypes=(("png files", "*.png"),
                                                  ("all files", "*.*")))
    img_l = Image.open(root.filename)
    img_temp = cv2.imread(root.filename, 0)
    img_temp = cv2.resize(img_temp, (SZ, SZ))
```

```
    # img_tempo1 = np.array(img_temp).flatten()
    img_temp = img_temp.astype('float32')
    img_temp1 = img_temp/255
    reshaped_img = img_temp1.reshape(-1, 1024)
    reshaped_img.shape
    cluster_img = kmeans.predict(reshaped_img)
    pred_img = get_labels(cluster_img, reference_labels)
    rzs_img = img_l.resize((250, 250), Image.ANTIALIAS)
    img_r = ImageTk.PhotoImage(rzs_img)
    label_img = Label(leftFrame, image=img_r).grid(row=0, column=0)
    label_1.grid_forget()
    label_2.grid_forget()
    label_2 = Label(rightFrame,
                    text="Identificada: "
                    + str(Catg[int(pred_img)])).grid(row=6, column=0, padx=20)


button_name = Button(rightFrame, text="Ingresar nombre",
                     command=getName).grid(row=0, column=0)

button_age = Button(rightFrame, text="Ingresar edad",
                    command=getAge).grid(row=1, column=0)

button_open = Button(leftFrame, text="Open", bg="#6F7997", fg="white",
                     command=openf).grid(row=1, column=0)

root.mainloop()
```

8.2 Codigo SVM

```
import os
import numpy as np
import cv2
import matplotlib.pyplot as plt
import pickle
import random
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from tkinter import Tk, Label, Button, Frame, filedialog
from PIL import ImageTk, Image
import numpy as np
```

```
root = Tk()
root.title("GUI software de identificación de imagenes en base SVM")
root.iconbitmap('ico_Br.ico')
root.geometry("800x600")

root['background'] = '#19232d'

img_1 = Image.open("UPB_logo.jpg")


# dir = 'C:\\Users\\GLORIA GOMEZ\\.spyder-py3\\BrIm'
# Catg = ['isquemia', 'NMO', 'Normal', 'PRES']
# data = []

# for category in Catg:
#     path = os.path.join(dir, category)
#     label = Catg.index(category)
#     for img in os.listdir(path):
#         imgpath = os.path.join(path, img)
#         Br_img = cv2.imread(imgpath, 0)
#         try:
#             Br_img = cv2.resize(Br_img, (50, 50))
#             image = np.array(Br_img).flatten()
#             data.append([image, label])
#         except Exception as e:
#             pass

# print(len(data))

# pick_in = open('data1.pickle', 'wb')
# pickle.dump(data, pick_in)
# pick_in.close()

pick_in = open('data1.pickle', 'rb')
data = pickle.load(pick_in)
pick_in.close()


random.shuffle(data)
features = []
```

```
labels = []

for feature, label in data:
    features.append(feature)
    labels.append(label)

Catg = ['isquemia', 'NMO', 'Normal', 'PRES']


xtrain, xtest, ytrain, ytest = train_test_split(
    features, labels, test_size=0.5)

# model = SVC(C=1, kernel='poly', gamma='auto')
# model.fit(xtrain, ytrain)

pick = open('model.sav', 'rb')
model = pickle.load(pick)
pick.close()

prediction = model.predict(xtest)
accuracy = model.score(xtest, ytest)
Catg = ['isquemia', 'NMO', 'Normal', 'PRES']

Nbrain = xtest[0].reshape(50, 50)

upFrame = Frame(root, width=200, height=600)
upFrame.grid(row=0, column=0, padx=70, pady=2)

upFrame1 = Frame(root, width=200, height=600, bg='#19232d')
upFrame1.grid(row=0, column=1, padx=70, pady=2)


rightFrame = Frame(root, width=200, height=600)
rightFrame.grid(row=1, column=1, padx=70, pady=2)

leftFrame = Frame(root, width=200, height=600)
leftFrame.grid(row=1, column=0, padx=50, pady=10)

img_1 = Image.open("UPB_logo.jpg")
```

```
rsz_img1 = img_1.resize((220, 110), Image.ANTIALIAS)
N_img1 = ImageTk.PhotoImage(rsz_img1)
imgtk = ImageTk.PhotoImage(image=Image.fromarray(Nbrain))
# N_img = ImageTk.PhotoImage(Nbrain)

e = Entry(rightFrame, width=20)
e_1 = Entry(rightFrame, width=20)

e.grid(row=0, column=1)

e_1.grid(row=1, column=1)


def getName():
    name = e.get()
    label_a = Label(rightFrame, text=name)
    label_a.grid(row=3, column=1, padx=20)


def getAge():
    name = e_1.get()
    label_e = Label(rightFrame, text=name)
    label_e.grid(row=4, column=1, padx=20)


accuracy_percent = accuracy*100

label_logo = Label(upFrame, image=N_img1).grid(row=0, column=0)
label_img = Label(leftFrame, image=imgtk).grid(row=0, column=0)
label_a = Label(rightFrame, text="Nombre: ").grid(row=3, column=0,
padx=20)
label_e = Label(rightFrame, text="Edad: ").grid(row=4, column=0, padx=20)
label_1 = Label(
    rightFrame, text="Anomalia: "
    + str(Catg[ytest[0]]))
label_1.grid(row=5, column=0, padx=20)
label_2 = Label(rightFrame,
                text="Identificada: "
                + str(Catg[prediction[0]]))
label_2.grid(row=6, column=0, padx=20)
```

```
label_3 = Label(rightFrame, text="Porcentaje de confiabilidad: "
            + str(accuracy_percent) + '%')
label_3.grid(row=7, column=0, padx=20)
label_5 = Label(upFrame1,
            text="UNIVERSIDAD PONTIFICIA BOLIVARIANA",
            bg='#19232d', fg='#fff').grid(row=0, column=0, padx=20)
label_6 = Label(upFrame1,
            text="Programa de ingenieria electronica",
            bg='#19232d', fg='#fff').grid(row=1, column=0, padx=20)
label_7 = Label(upFrame1,
            text="Desarrollado por: Juan D.",
            bg='#19232d', fg='#fff').grid(row=3, column=0, padx=20)


data_img = []


def openf():
    global label_img
    global img_l
    global img_r
    global label_2
    global label_3
    root.filename = filedialog.askopenfilename(
        initialdir='C:\\Users\\GLORIA GOMEZ\\.spyder-py3\\BrIm',
        title="Selecciona una imagen", filetypes=(("png files", "*.png"),
                                          ("all files", "*.*")))
    img_l = Image.open(root.filename)
    Br_imgtemp = cv2.imread(root.filename, 0)
    Br_imgtemp = cv2.resize(Br_imgtemp, (50, 50))
    imagetemp = np.array(Br_imgtemp).flatten()
    data_img.append(imagetemp)
    prediction_t = model.predict(data_img)
    rzs_img = img_l.resize((250, 250), Image.ANTIALIAS)
    img_r = ImageTk.PhotoImage(rzs_img)
    label_img = Label(leftFrame, image=img_r).grid(row=0, column=0)
    label_1.grid_forget()
    label_2.grid_forget()
    label_2 = Label(
        rightFrame, text="Identificada: "
```

```
        + str(Catg[int(prediction_t)])).grid(row=6, column=0, padx=20)


button_name = Button(rightFrame, text="Ingresar nombre",
                     command=getName).grid(row=0, column=0)

button_age = Button(rightFrame, text="Ingresar edad",
                    command=getAge).grid(row=1, column=0)

button_open = Button(leftFrame, text="Open", bg="#6F7997", fg="white",
                     command=openf).grid(row=1, column=0)


root.mainloop()
```