Universidad Nacional Autónoma de México

Facultad de Ciencias

Descubrimiento de Estructuras en imágenes de TC con diagnóstico de COVID-19 utilizando MinHashing, K-means y SIFT

T E S I S

que para obtener el título de:

física

P R E S E N T A :

Mayra Cid Oros

TUTOR

Dr. Ivan Vladimir Meza Ruiz

Ciudad Universitaria, CDMX, 2022

Agradecimientos

Agradezco al Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas por el préstamo de su equipo.

Quiero agradecer a mi asesor de tesis el Dr. Ivan Vladimir Meza Ruiz por su apoyo y conocimiento brindado para llevar a cabo este proyecto y sobre todo por su infinita paciencia al esperar que este trabajo pudiera llegar a su fin.

A mis sinodales, gracias por la oportunidad y por el tiempo que me han dedicado para leer este proyecto.

A mis amigos quiero agradecer todo el apoyo, cariño y animo que me han brindado.

A mi familia, mis seres más queridos y lo más importante que tengo en la vida, por su amor, consejos, paciencia y apoyo incondicional en todo momento, sinceramente Gracias.

Índice general

Índice de figuras

1 Introducción

La pandemia ocasionada por el virus SARS-CoV-2 ha generado un fuerte impacto mundial en diferentes niveles que van desde lo económico, lo social y lo ambiental. Sin embargo, también ha dejado una gran cantidad de datos cuya información es importante para entender y resolver los problemas ocasionados. Estos datos resultan ser tan grandes que en ocasiones no hay suficientes expertos por lo que se necesita cooperación de todas las áreas posibles.

Una alternativa que da el área de visión por computadora, rama de la inteligencia artificial, es la que permite desarrollar teoría y tecnología necesaria para emular la percepción humana, mediante lo que llaman aprendizaje supervisado y no supervisado. Donde el aprendizaje supervisado es aquel que nos dice que para determinado dato x se necesita que el algoritmo aprenda determinado valor y, es decir, se le proporciona orientación. Mientras que el no supervisado aprende a identificar procesos y patrones complejos sin necesidad de que un ser humano le proporcione orientación y supervisión a lo largo del proceso de aprendizaje. [1]

Con ambos métodos de aprendizaje, la visión artificial ha permitido al área de la medicina procesar y analizar datos de relevancia médica para mejorar la gestión sanitaria y facilitar la realización de diagnósticos para el paciente. Por ejemplo, un modelo donde se ha ayudado al área de la salud se menciona en la referencia [2] que habla sobre desarrollo y evaluación de métodos para producir etiquetas a partir de informes radiológicos y que concuerden mejor con las imágenes etiquetadas por radiólogos. Otro ejemplo más reciente se menciona en la referencia [3] el cual hace revisión de métodos de procesamiento de imágenes para el diagnostico de COVID-19.

Siguiendo la misma línea, este trabajo explora algoritmos y técnicas que pueden ayudar a descubrir estructuras en un conjunto de imágenes. Este conjunto se tomará como la base de datos, donde se hará una extracción de puntos de interés a cada elemento de dicha base, para poder agruparlos a partir de ciertos criterios y por último minarlos. El minado encontrará patrones, tendencias o reglas que expliquen el comportamiento de los datos que se han recopilado. De tal manera que podremos resaltar estructuras en las imágenes que permitirán al usuario ver si existe algún comportamiento de interés que se deba analizar.

El proceso se repetirá para tres grupos que tienen diferentes cantidades de puntos de interés. La razón, para ver cómo influye la cantidad de estos puntos con los resultados del minado. Es decir, a mayor cantidad de puntos de interés, mejor calidad de estructuras encontradas. Algunos de estos resultados serán mostrados a radiólogos para solicitar una identificación de lo que pueda estar en las estructuras.

El conjunto de imágenes estará dado por la base de datos que se le proporcionó al proyecto COVIDx CT [4], hospedado en la plataforma kaggle [5–13] y que corresponden a 194,922 cortes por tomografía computarizada de 3,745 pacientes, todos con diagnóstico positivo de COVID-19 (es decir, RT-PCR, confirmado por radíologos.) en imágenes blanco-negro y formato PNG.

Y por ser una cantidad de más de 15 Gb de información que manejamos y guardamos, se utilizó el servidor del IIMAS de la UNAM para poder agilizar los experimentos.

1.1. Objetivos

El objetivo de esta tesis es identificar patrones dentro de una colección de imágenes de TC con diagnóstico de COVID-19. Que puedan brindar información útil a especialistas o personas interesadas en el tema.

Para lograrlo se deben cumplir los siguientes objetivos particulares:

1. Disminuir la base de datos original por un conjunto que sea posible manejar con el poder de cómputo que se tiene.

2. Seleccionar el parámetro más adecuado para realizar la extracción de características en el conjunto de imágenes que se tiene.

3. Realizar el minado de tópicos con diferentes parámetros.

4. Visualizar los tópicos de mayor relevancia.

1.2. Metas

Los objetivos particulares se pueden alcanzar cumpliendo las siguientes metas:

1. Crear un criterio que pueda descartar imágenes no relevantes de la base de datos.

2. Realizar experimentos con diferentes parámetros para la extracción de caracteristicas.

3. Realizar experimentos con diferentes parámetros para la obtención de tópicos.

4. Realizar un análisis sobre los tópicos obtenidos.

1.3. Organización de la tesis

La tesis se divide en 4 capítulos

- Capitulo 1: Estado del arte. Se presenta un antecedente de los trabajos y conceptos que se utilizaron.

- Capítulo 2: Metodología. Se desarrollan los procesos que se llevaron a cabo para cumplir las metas.

- Capítulo 3: Marco teórico. Se explica el contexto de la investigación.

- Capítulo 4: Datos y experimentación. Se muestran los experimentos y los resultados para el análisis.

- Capitulo 5: Conclusiones. Se muestran los resultados del análisis.

2 Estado del arte

El mundo que nos rodea lo percibimos a través de los sentidos. Donde los estímulos desencadenan sensaciones, pero la organización, interpretación y análisis depende del cerebro. Por ejemplo, la corteza cerebral tiene 30 áreas visuales localizadas en los lóbulos occipitales, parietal, temporal y frontal. Cada área extrae diferentes tipos de información de la señal de entrada visual; desde los rasgos más elementales como la orientación, hasta los rasgos más complejos como el movimiento. [14]

La descripción a grandes rasgos de este sentido, es la entrada de luz que reflejan los objetos a los ojos, llegando hasta a la retina y transformándose en un impulso eléctrico que viaja a través del nervio óptico para llegar al cerebro. En este lugar el impulso se reinterpreta en las áreas de corteza cerebral que le corresponde para brindar información de color, iluminación, entre otras características.

Con lo anterior, se podría decir que la visión es una tarea de procesamiento de información, sin embargo, es algo más complejo, ya que, para saber que es lo que hay a nuestro alrededor, nuestro cerebro debe ser capaz de representar esta información en

toda su abundancia como forma, movimiento, profundidad, etc. y no solo carcterísticas de una imagen. Por lo que el estudio de la visión no solo debe ser el cómo extraer de las imágenes los diversos aspectos del mundo que nos son útiles sino también una investigación sobre la naturaleza de las representaciones internas mediante las cuales captamos esta información y la utilizamos como base para las decisiones sobre nuestros pensamientos y acciones. [15] Esta dualidad, de representación y procesamiento de información está en el corazón de los problemas que se relacionan con la inteligencia artificial y en específico, la rama de visión por computadora.

2.1. Visión por computadora

La visión por computadora o visión artificial utiliza una mezcla de herramientas y métodos de la matemática, física y programación para procesar, analizar y comprender una imagen, regresando información numérica o simbólica que puede ser tratada por un ordenador y así obtener casi cualquier cosa; desde modelos 3D, posición de la cámara, reconocimiento de objetos, agrupación y búsqueda de contenido. También puede incluir la deformación de las imágenes, la eliminación de ruidos y la realidad aumentada. [15]

2.2. Librería OpenCV y la función SIFT

Para llevar a cabo todo lo anterior existen diferentes bibliotecas y paquetes que trabajan según la tarea deseada, por ejemplo, OpenCV [16] es una biblioteca de código

abierto que permite manipular imágenes y videos para realizar tareas que van desde la detección automática de rostros, hasta enseñarle a un robot a reconocer objetos.

OpenCV contiene más de 500 funciones que abarcan muchas áreas y, debido a que la Visión por computadora y el aprendizaje automático ("Machine Learning") a menudo van de la mano, OpenCV también contiene una biblioteca completa de uso general de aprendizaje automático; la cual se centra en el reconocimiento de patrones estadísticos y el agrupamiento.

En esta tesis nos centraremos en una de esas funciones que permiten extraer características relevantes en imágenes para después ser manipuladas y crear cierta aplicación. Con características relevantes hacemos referencia a la información importante que hay en la imagen. Por ejemplo, los puntos de las esquinas, los puntos de los bordes, los puntos brillantes en las áreas oscuras y los puntos oscuros en las áreas brillantes. El nombre de dicha función que usaremos es: *Scale Invariant Feature Transform* o sólo *SIFT* [17]

SIFT es un método que obtiene puntos de interés en una imagen, mediante la difuminación con funciones gaussianas [17] que después describe mediante un histograma orientado de gradientes. Y lo hace de manera en que la localización y descripción presenten una invarianza a la orientación, posición y escala. Cada punto característico queda definido por un vector de características de 128 elementos y se obtiene información de su posición en coordenadas de la imagen, la escala a la que se encontró y

la orientación dominante de la región alrededor de dicho punto. Ver Imágenes 2.1 y 2.2

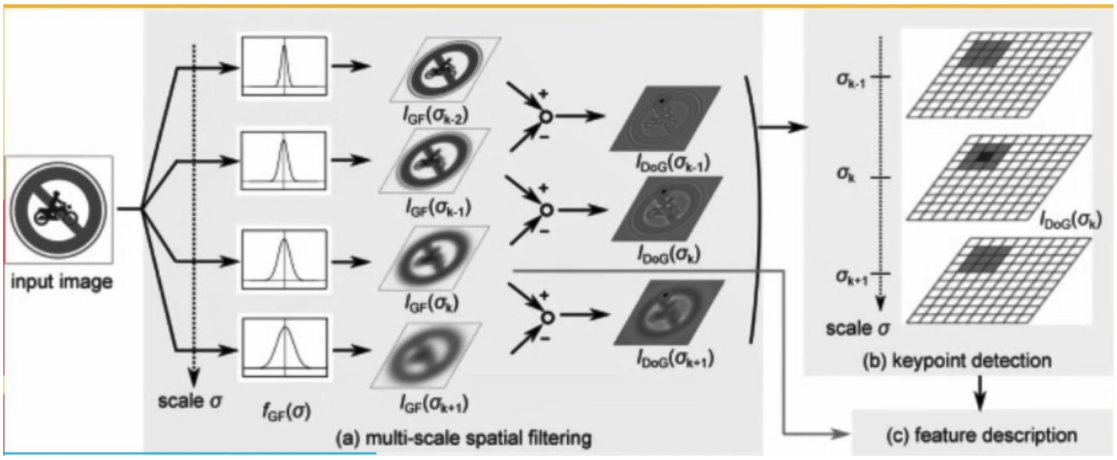

Figura 2.1: Detección de puntos de interés y la difuminación con funciones gaussianas [18]

.

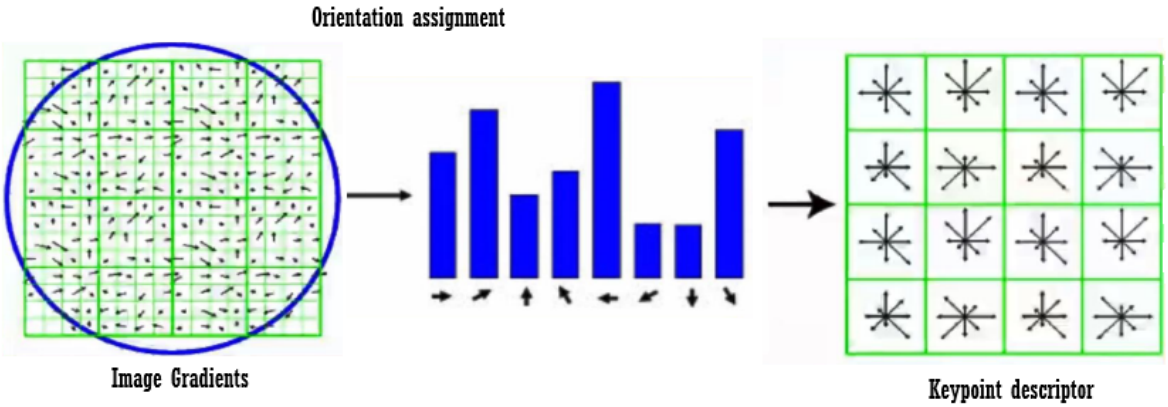

Figura 2.2: Cálculo del histograma orientado de gradientes y del vector de características Imagen tomada y modificada de [18]

SIFT fue propuesto para imágenes en escala de grises por lo que el vector de características de 128 elementos que define cada pixel contiene información sobre cómo

se distribuyen los niveles de intensidad alrededor de cada punto de interés que se obtuvo previamente. [19]

En resumen, el algoritmo *SIFT* consta de dos partes, la primera es la obtención de los puntos característicos o puntos de interés y la segunda la descripción de la región alrededor de cada punto de interés. Dando como resultado final la figura 2.3 Para obtener más información sobre el tema ver la referencia [19]

Figura 2.3: Puntos de interes o keypoints en una imagen [20]

Por otro lado, un ejemplo de cómo *SIFT* se ocupa para determinar la similitud entre dos imágenes se puede consultar, en la referencia [21] en donde este algoritmo busca descriptores en una entrada de 2 imágenes que después analiza para determinar si existen coincidencias, en algunos casos demostró no ser asertivo al no marcar detalles en ciertas imágenes, lo cual no es de extrañar ya que para una buena comparación y detección comúnmente se necesita la ayuda de más algoritmos.

Una segunda aplicación que utiliza el algoritmo de *SIFT* es la referencia [22], en

donde se utilizan los descriptores obtenidos por *SIFT* en imágenes reales y modificadas para hacer una comparación, encontrar similitudes y así detectar la falsificación de copias y movimientos. La utilidad de este algoritmo es grande porque podría identificar imágenes digitales falsas en la red, y así ayudar a desmentir y no proveer información incorrecta, lo cual se ha convertido en un problema hoy en día.

Esta misma coincidencia de descriptores, también se utiliza en video como lo menciona la referencia [23], donde la búsqueda de similitudes ataca un problema de seguimiento visual, en el cual, la apariencia del objeto rastreado y los alrededores de la escena cambian durante el tiempo.

SIFT no solo ha ayudado como herramienta para imágenes y videos generales sino también para imágenes médicas. Aunque se ha topado con ciertas dificultades como el no generar un buen número de descriptores, ya que generalmente este tipo de imágenes tienen grandes regiones de poca variación e intensidad, como lo publica en [24]. Sin embargo, no todo es malo porque a pesar del número bajo de descriptores ha ayudado a la creación de otro tipo de tareas como la corrección de regiones rotadas en imágenes médicas, publicado en el artículo [25] y la recuperación de imágenes como lo menciona el artículo [26].

Actualmente, existen otras técnicas que se pueden ocupar para la extracción de puntos de interés basados en métodos de aprendizaje profundo ("Deep Learning"). Los cuales se enfocan en la creación de redes neuronales y se entrenan con un conjunto de datos de imágenes que funcionan como puntos de referencia. Este proceso es clave

importante del algoritmo. Por esa razón este proyecto eligió *SIFT* debido a que no necesita una base de datos de ajuste.

2.3. Librería *Scikit-learn* y *k-means*

Otra librería de software libre es *scikit-learn*. La cual cuenta con algoritmos de clasificación, regresión, agrupamiento y reducción de dimensionalidad. Todas estas herramientas son simples y eficientes para el análisis de datos y modelado estadístico.

De las herramientas anteriores la que más nos interesa es la de agrupamiento (clusterización) de datos no etiquetados. La librería *Scikit-learn* ofrece diferentes algoritmos, cada uno tiene dos variantes una que implementa el método para aprender los centroides localizados en el tren de datos y una función que, dados los datos del tren, devuelve una matriz de etiquetas correspondientes a los diferentes centroides [27].

Existen otros detalles que hacen única la elección del algoritmo, por ejemplo: la cantidad de datos, los tamaños y la cantidad de los grupos que deseamos, si se utiliza geometría plana, entre otros aspectos.

En esta tesis utilizamos el algoritmo *k-means* que nos permite obtener la cantidad de agrupamientos que deseamos. *K-means* tiene como objetivo la partición de un conjunto de n elementos en k grupos en el que cada elemento pertenece al grupo cuyo valor medio es más cercano [28]. Además de que regresa cada elemento con una

etiqueta que marca a qué centroide le pertenece. Figura 2.4 [29]

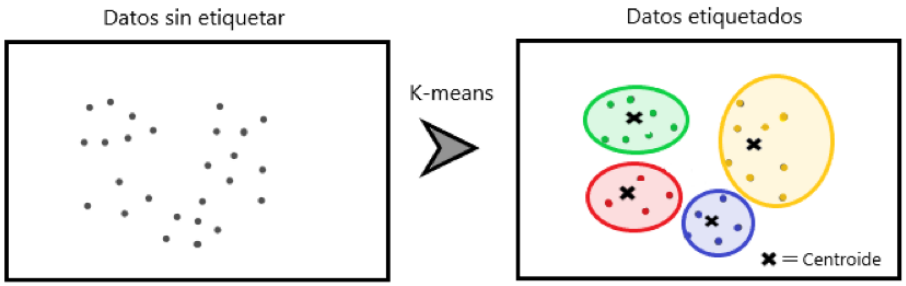

Figura 2.4: Como funciona *K-means*

El algoritmo consta de los 4 pasos siguientes:

- **Inicialización (Valor de K).** Se le dice al algoritmo cuantos centroides se desean (k), según sea el algoritmo, uno puede encontrar este valor de k o bien asignarlo desde un inicio.

- **Asignación de elementos a los centroides.** A partir de aquí, el algoritmo inicia sus cálculos y a cada elemento de los datos se le asigna el centroide más cercano.

- **Actualización de centroides.** Como los centroides iniciales fueron asignados arbitrariamente, el algoritmo los actualiza tomando como nuevo centroide la posición del promedio de los elementos pertenecientes a dicho grupo. El paso anterior y éste, se repiten varias veces.

- **Criterio para detenerse.** Al ser un proceso iterativo debe llegar un momento en el que se deba detener y esto sucede cuando los centroides no cambian, se alcaza un número máximo de iteraciones o bien existe un umbral que se toma como la distancia mínima.

Lo anterior se muestra en el diagrama de la figura 2.5:

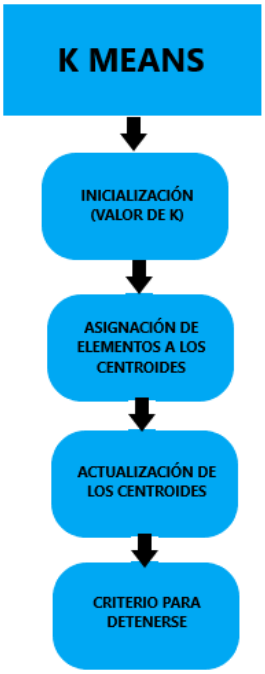

Figura 2.5: Diagrama *k-means*

Un ejemplo de *k-means* es el que se presenta en la referencia [30], donde *k-means* es uno de los algoritmos que se utiliza para modelar y ayudar a los auditores a realizar la tarea de detección de estados financieros fraudulentos. Tomando como valor de k igual a 47 razones financieras.

También, existen proyectos que utilizan los resultados de *SIFT* como datos de entrada para *k-means* para así obtener agrupamiento de descriptores. Un ejemplo de esto, es el que se muestra en el artículo de la referecia [31]. En el que dichos algoritmos ayudan al reconocimiento de huellas de un nudillo de dedo para que pueda ser aplicado a la autenticación y verificación.

Otro ejemplo que utiliza ambas funciones es el proyecto de guía móvil de museos, que permite que los teléfonos con cámara reconozcan las pinturas en las galerías de arte como lo menciona el artículo de la referencia [32].

Lo anterior señala que la unión de algoritmos hacen un método más eficiente. Por eso razón, el siguiente algoritmo tendrá mucha relevancia para esta trabajo.

2.4. Algoritmo *Hashing*

Un texto ya sea un ensayo, reporte, articulo, o cualquier otro siempre sigue una estructura. En todos existe un cuerpo textual formado por párrafos internos que suelen explicar las ideas principales, las cuales contienen palabras claves con las que se puede identificar el asunto o tema que se va a tratar.

A partir de esto, es como el algoritmo *Min-Hashing* trabaja. Mediante la generación de particiones múltiples y aleatorias de vocabulario de un cuerpo textual, para encontrar conjuntos de palabras de alta concurrencia y agruparlas para producir tópicos finales. Entiéndase por tópicos la asignación de una palabra clave que engloba el tema o la idea principal.

El descubrimiento de tópicos en cuerpos textuales es útil para tareas de procesamiento de lenguaje natural como clasificación, resumen y traducción de textos.

El algoritmo *Min-Hashing* fue desarrollado por Fuentes & Meza y aplicado en el proyecto de *Topic Discovery in Massive Text Corpora Based on Min-Hashing* [33], en el cual se busca obtener tópicos en una base de datos de 5 millones de textos de Wikipedia. También, fue aplicado en el proyecto *Minado de tópicos usando min-hashing en tesis UNAM y su visualización* [34] el cual hace estudios de tópicos sobre las tesis que se han hecho en la UNAM en el área de cómputo para observar cuales han sido los temas que se han tratado, han sido más llamativos para la comunidad y han conservado su presencia a lo largo del tiempo.

En esta tesis, extrapolamos este método a temas de visión por computadora. Donde la información brindada por imágenes podría construir "párrafos" creando un "cuerpo" que ya no sería textual, pero realizaría la misma función que es agrupar y producir tópicos sobre las imágenes, como muestra la figura 2.6.

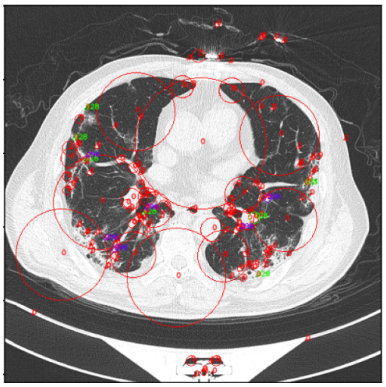

Figura 2.6: Topicos en una imagen

Para ello juntaremos los algoritmos *SIFT*, *k-means* y *Min-Hashing* para aplicar minado a las imágenes y obtener estructuras que después puedan ser analizadas para determinar si existe algún patrón relevante.

3 Marco Teórico

En 1895 Wilhelm Roentgen encontró que radiación altamente penetrante de naturaleza desconocida es producida por electrones acelerados que chocan contra la materia, llamándolos rayos x. Roentgen obtuvo la primera radiografía al "fotografiar"la mano izquierda de su esposa sobre una placa de metal, volviendo a los rayos x tan famosos, que dos meses después de su descubrimiento se comenzaron a utilizar en el área de la medicina. [35]

Desde ese tiempo varios médicos aplicaron de forma empírica la radiación resultante de los rayos x como herramienta de diagnóstico. La cual se basa en los diferentes grados de absorción en los tejidos. Por ejemplo, debido a su contenido de calcio el hueso resulta ser mucho más opaco a los rayos x que el músculo, que a su vez es más opaco que la grasa. En algunas ocasiones para mejorar el contrastre, a los pacientes se les administra un medio de contraste como el sulfato de bario para mostrar el sistema digestivo. También, se pueden inyectar otros compuestos en el torrente sanguíneo para permitir que se estudie el estado de los vasos sanguíneos.

En la actualidad este proceso se ha mejorado, dando paso a la imagenología médica para el apoyo clínico en diagnóstico y tratamiento de enfermedades. Esta mejora ha

sido por medio de la tecnología dando diferentes modalidades, un ejemplo de ellos es la tomografía computarizada (CT), que con la ayuda de una computadora combina una serie de exposiciones de rayos x del paciente tomadas desde diferentes direcciones para obtener imágenes transversales muy delgadas de las partes del cuerpo que se examinan. [35] permitiendo el análisis de una manera no invasiva de las estructuras internas del cuerpo humano.

Esta tecnología se ha vuelto indispensable en el área de la medicina, ya que con una tomografía, un experto en el tema puede determinar si existe alguna anomalía sólo con la observación, debido a que reconoce la estructura de un organismo sano a uno que no lo es.

De esta manera damos paso al siguiente proyecto, el cual esta interesado en la búsqueda de patrones en un grupo de tomografías con padecimiento de COVID-19. Todo ello mediante un algoritmo de visión por computadora.

4 Metodología

Este proyecto consta de 4 etapas, la primera es la extracción de caracteristicas, la segunda es el agrupamiento, la tercera es el minado de tópicos y la última es la visualización de tópicos o estructuras. Como se muestra en la figura 4.1.

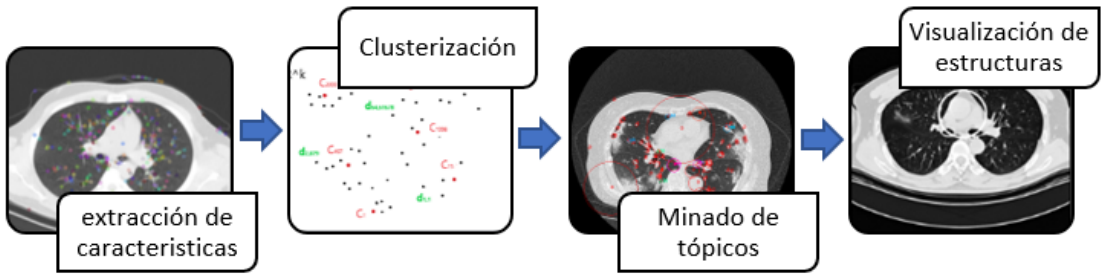

Figura 4.1: Etapas del proyecto

4.1. Extracción de características

El primer paso es hacer la extracción de características en las imágenes del dataset. Para ello, se creo un *script* en python que utiliza la función *SIFT* de OpenCV y sus parámetros. Este *script* obtiene puntos de interés o *keypoints* (figura 4.2), los cuales son puntos relevantes de la imagen que permiten obtener una descripción alrededor de cada uno de ellos, es decir, permite tener una descripción de la región local,

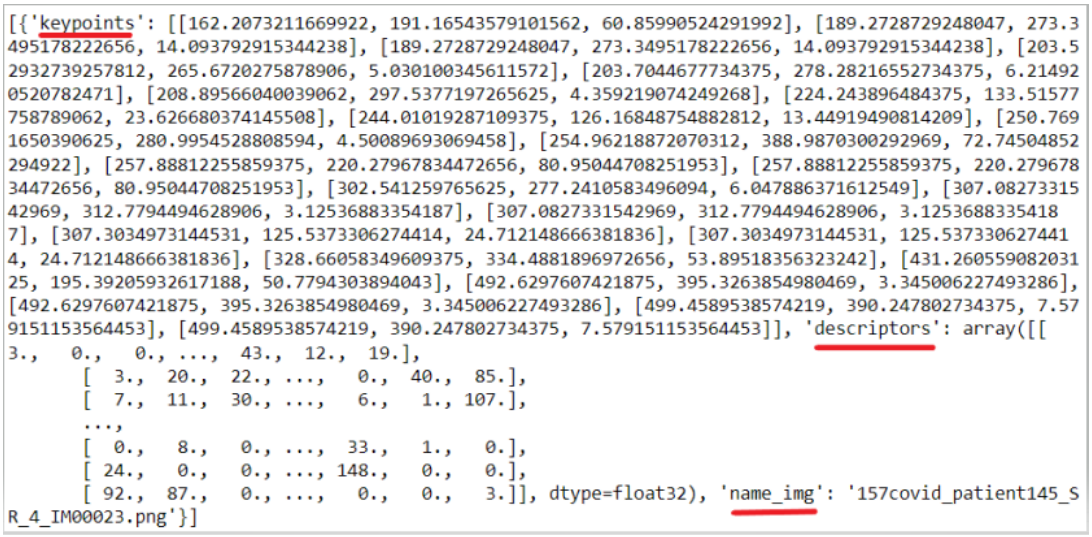

Figura 4.2: Descriptores, *keypoints* y nombre de las imágenes guardados en un archivo *pickle*

que después se transforma en un vector numérico (descriptor) de 128 entradas. Esta información de *keypoints*, descriptores y nombres de cada imagen se guardan como diccionario en un archivo *pickle* como se ve en la figura 4.2.

En la figura 4.2 los *keypoints* representan un conjunto de vectores de 3 entradas. Mientras que el descriptor es un vector de 128 entradas y se muestra como un arreglo en "descriptor". Por último en "name"se muestra el nombre de la imagen.

4.2. Agrupamiento

Con la información anterior en el archivo *pickle*, el paso siguiente es la clusterización. Entonces, ejecutamos el *script* de clusterización que funciona con el algoritmo de *k-means*. Dicho algoritmo es un método de agrupamiento que tiene como objetivo la partición de un conjunto de n observaciones en k grupos, en el que cada observación pertenece al grupo cuyo valor medio es más cercano. Eso quiere decir, que una lista

de n datos se divide en k grupos y el criterio es minimizando la suma de distancias entre cada objeto y el centroide de su grupo o también llamado cluster. Este criterio se repite varias veces para asi poder asegurar los ”mejores *clusters*”. Figura 4.3

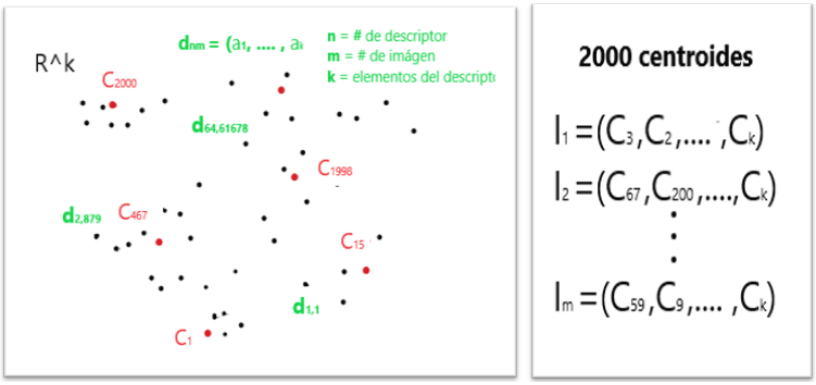

Figura 4.3: Como funciona *k-means*

Al ejecutar el script de clusterización, se especifica el número de *clusters* (k) que se desea tener, para este caso y por la cantidad de imagenes se tomó k igual a 2,000. Por lo que nuevamente se obtiene un archivo *pickle*, que tiene como información el nombre de la imagen y los centroides que fueron identificados. Tal y como muestra el ejemplo de la figura 4.4.

Nombre de la imagen ID	Centroides
Imagen 1	[127, 56, 72, 95, 113, 104, 45, 1003, 189, 1989, ... , 153, 549, 324]
Imagen 2	[13, 189, 32, 191, 123, 6, 82, 65, 82, 190, 35, 456, ... , 234, 45, 34]
Imagen 3	[102, 34, 1999, 23, 764, 457, 98, 578, 199, 145, 34, ... , 245, 92, 24]
:	:
:	:
Imagen 33,897	[100, 50, 103, 6, 67, 234, 92, 92, 456, 1899, 2, 45, 83, ... , 34, 345, 3]
Imagen 33898	[23, 768, 1546, 22, 83, 12, 5, 67, 890, 345, 23, 956, ... , 245, 98, 23]
Imagen 33 899	[234, 1764, 23, 456, 87, 88, 88, 98, 34, 223, 987, 3, ... , 23, 45, 456]
:	:
:	:
Imagen 67,793	[2, 12, 567, 879, 34, 23, 789, 1567, 34, 789, 312, 45, ... 45, 56, 234]
Imagen 67,794	[34, 567, 98, 1909, 185, 1935, 12, 567, 466, 26, 984, ... , 78, 456, 13]

Figura 4.4: Información que se guarda en el archivo *pickle*, centroides y nombres de las imágenes

4.3. Minado

El siguiente paso es preparar el archivo que será minado y la forma de hacerlo es ejecutando el *script* SHMdocument.py. El cual hará coincidir los archivos que se obtuvieron en la extracción y clusterización, al tomar la información del archivo anterior y haciendola coincidir con las imágenes y *keypoints* que le corresponden. Además, los descriptores en las imágenes serán remplazados por los centroides (etiquetas) y se toma la frecuencia con la que aparece cada centroide en una imagen, para obtener al final un archivo como el ejemplo de la figura 4.5.

Por otra parte, cuando se realiza el minado lo que se busca es ver las imágenes que tienen coincidencias, por lo que se necesita que el archivo anterior se encuentre de manera inversa. Es decir, en vez de tener una lista de centroides con su frecuencia para cada imagen, se debe tener una lista de imágenes con su frecuencia para cada

	Centroides(etiquetas):Frecuencia
16	127:2 56:3 72:15 95:5 113:3 104:3 1003:8 189:7 1989: 3 ... 153:7 549:9 324:6
54	13:4 189:19 32:5 191:2 123:6 6:7 82: 3 65:4 82:23 190:15 35:4 456:7 ... 234:3 45:5 324:7
78	102:4 34:93 1999:23 764:89 457:8 98:67 578:3 199:26 145:17 34:45 ... 245:4 92:56 24:6
:	:
:	:
45	100:4 50:34 103:4 6:34 67:45 234:4 92:36 92:36 456:24 1899:23 2:19 45:34 83:2 ... 34:2 345:4 3:12
112	23:4 768:45 1546:45 22:4 83:4 12:11 5:4 67:56 890:5 345:6 23:9 956:7 ... 245:6 98:67 23:4
37	234:5 1764:5 23:6 456:7 87:98: 88:5 88:5 98:16 34:6 223:56 987:67 3:4 ... 23:67 45:4 456:3
:	:
:	:
234	2:3 12: 45 567:8 879:45 34:4 23:3 789:29 1567:8 34:3 789:29 312:24 45:6 ... 45:6 56:89 234:54
23	34:3 567:45 98:5 1909:45 185:56 1935:78 12:4 567:89 466:4 26:2 984:45 ... 78:56 456:34 13:18

Figura 4.5: Cantidad de centroides, centroides y sus frecuencias

centroide. Dando una lista de 2,000 elementos con n cantidad de imágenes cada uno. Figura 4.6

Para realizar el paso de invertir el archivo, se ejecuta la línea de código que aparece en el apéndice B.4.

	Imágenes:Frecuencia
16	imagen1:2 imagen34:3 imagen89:15 imagen789:3 imagen1023:4 imagen30343:45 ... imagen398:22
54	imagen356:3 imagen857:56 imagen98:9 imagen45798:34 imagen33:2 ... imagen12:3 imagen9843:3
78	Imagen23:56 imagen90:8 imagen45678:8 imagen23454:7 imagen33:6 imagen66344:23 ... imagen34:4
:	:
:	:
45	Imagen456:4 imagen89:3 imagen7862:2 imagen34:38 imagen56897:98 imagen273:9 ... imagen23:9
112	Imagen32:3 imagen89:7 imagen66:9 imagen980:9 imagen983:3 imagen312:45 ... imagen34:22
37	Imagen23:4 imagen8976:78 imagen345:98 imagen6754:8 imagen45:2 imagen3456:34 ... imagen47:5
:	:
:	:
234	Imagen344:78 imagen78:4 imagen9765:3 imagen270:3 imagen39:4 imagen24563:9 ... imagen34:56
23	Imagen98:2 imagen1:98 imagen23:8 imagen67:2 imagen76:9 imagen:89:23 imagen78:9 ... imagen4:2

Figura 4.6: Cantidad de imagenes, imagenes y sus frecuencias

Una vez que se tiene el archivo en la forma que se necesita se comienza con el minado. Para ello se hace uso del algoritmo *min-hashing*, el cual permite descubrir patrones a partir de datos diádicos a gran escala. *SMH* se basa en *Min-Hashing* para

extraer de manera eficiente las relaciones más allá de los pares que se agrupan para formar los patrones finales descubiertos. [36]. Esto es enfocado a texto pero aquí lo extrapolaremos a imágenes.

SMH utiliza las funciones hash y los parámetros l y r. Donde las funciones asignan un valor fijo a un dato de cualquier longitud y los parámetros l y r son número de tuplas y longitud de tupla con un rango de [0,9,000] y [2,9]. Para ilustrar el proceso del algoritmo se tiene el siguiente ejemplo:

Supongamos en la lista de la figura 4.6. existen 20,000 imágenes que contienen un mismo centroide. Entonces, si el parámetro r es igual a 2 y l igual a 100, lo que hará el algoritmo es utilizar la función *hash* en la primera tupla (l=1) e identificará si el valor *hash* coincide para otros centroides en al menos dos imágenes y si encuentra coincidencia, los centroides se guardaran en un contenedor o "bucket". Este proceso se repite para la siguiente tupla, pero con la condición de que en esta tupla sus elementos hayan hecho una permutación y así sucesivamente hasta llegar al valor de l. De esta forma, los centroides que fueron altamente co-ocurrentes en todo el proceso son lo que se llama tópicos y a la colección de tópicos es lo que se le conoce como minado. Figura 4.7

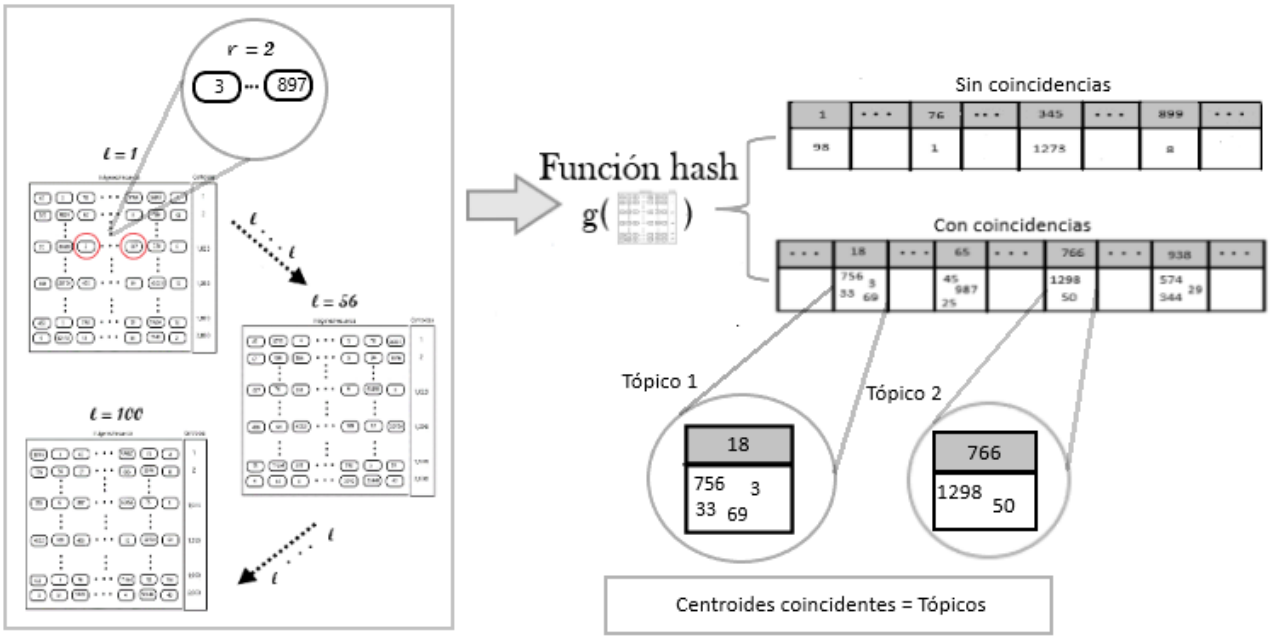

Figura 4.7: Proceso de minado

Cada valor de l y r generan un archivo y este se crea ejecutando la línea de código que aparece en el apéndice B.5. y si además, hacemos correr el valor de l y r en los rangos que corresponden, se puede formar una tabla con la cantidad de tópicos que se encontraron. (tablas 14, 15 y 16)

En este proyecto la tabla de tópicos corre con los valores de 2 a 9 para r y con valores de 500, 1,000, 5,000 y 9,000 para l. Por lo que se obtiene un total de 32 archivos con terminación *.models*

4.4. Visualización

Ahora, los tópicos encontrados en los archivos se pueden visualizar mediante el *notebook* visualizacion.ipynb. El cuál permite ver si los tópicos descubiertos cumplen

con una clasificación o patrón. Esta clasificación se realiza a las imágenes y es a criterio del observador.

El *notebook* visualización.ipynb es interactivo y está dividido en 5 partes:

- **visualización de tópicos**

  El código carga los tópicos y elementos de la figura de los archivos que se obtuvieron en los pasos anteriores. De esa manera, al ejecutar el código, aparecerá una caja opciones que permitirá seleccionar el archivo de minado que se desee. Una vez hecho esto, se oprime *run interact* y aparecerá el número de tópicos y el nombre del archivo con terminación *.models*. Figura 4.8

Figura 4.8: Celda interactiva y su resultado

- **Contenido de tópico**

  Como su nombre lo dice nos permite ver el contenido de los tópicos o dicho de otra forma los id de los centroides junto con sus pesos. Figura 4.9

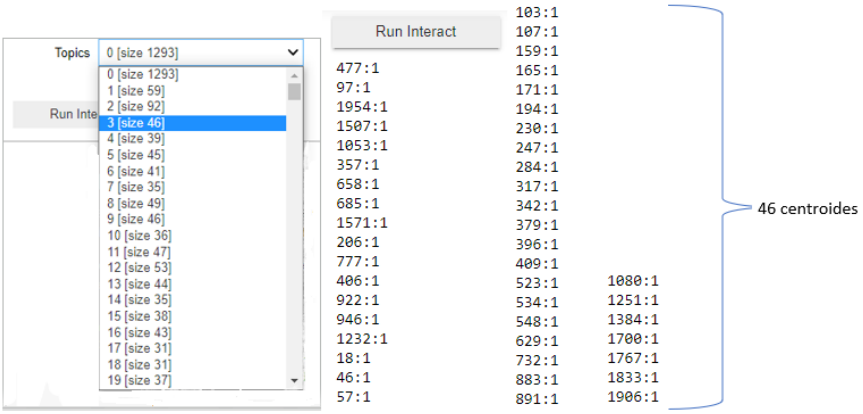

Figura 4.9: Elección del tópico 3 y los 46 centroides que le corresponden

- **Elegir tópico**

  Esta parte contiene una celda interactiva, donde la caja de opciones nos permite elegir el tópico, la cantidad de imágenes en donde buscará el tópico seleccionado y un umbral ("threshold") que dice que tan precisa es la búsqueda. Y una vez que se eligió lo anterior se oprime *run interact* y desglosa el número de imágenes que cumplieron con esos requisitos. Figura 4.10

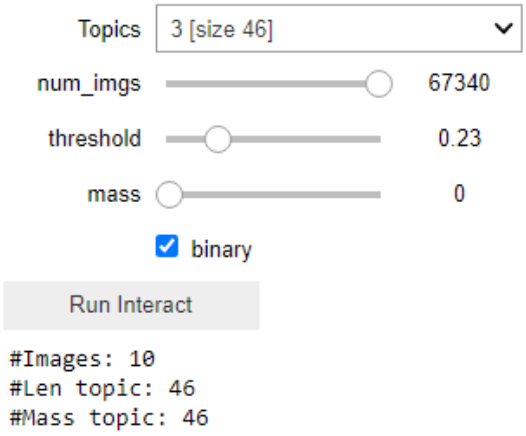

Figura 4.10: Las 10 imágenes que le corresponden al tópico 3

- **Visualizando ejemplos del tópico**

  Esta parte muestra ejemplos de imágenes que contiene el tópico seleccionado en la sección anterior. Figura 4.11

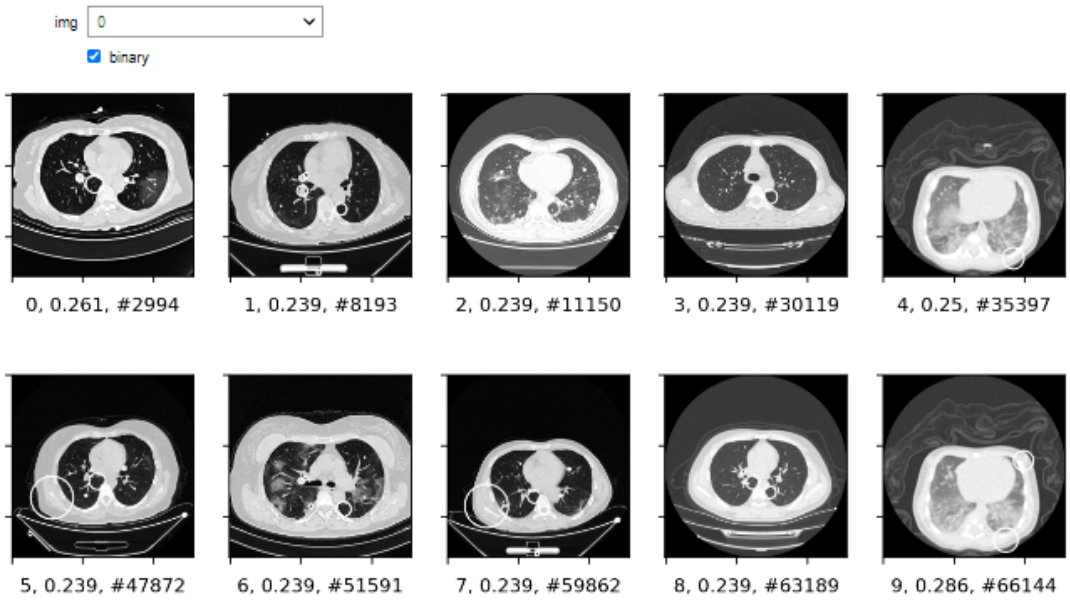

Figura 4.11: Visualización de las 10 imágenes

- **Ver una imagen**

  Esta última parte corresponde a la visualización de una sola imagen junto todos sus posibles tópicos según sea el umbral elegido. Figura 4.12

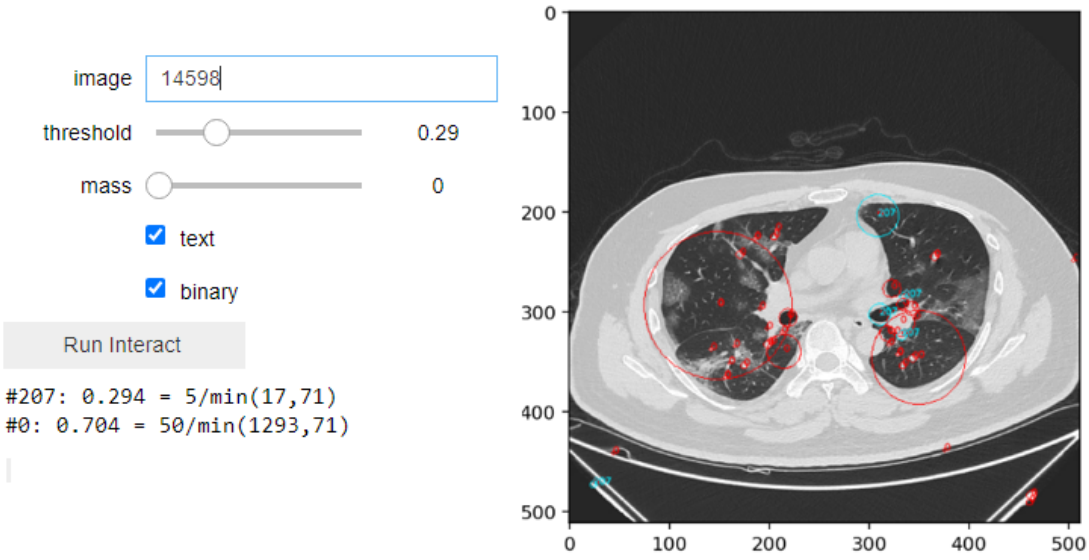

Figura 4.12: Visualización de una sola imagen y sus tópicos

Con todo lo anterior, el usuario puede hacer un análisis de tópicos y seleccionar aquellos que compartan una estructura relevante y así descubrir un patrón.

5 Datos y experimentación

El dataset que utilizamos para trabajar en esta tesis se encuentra en el proyecto COVIDx CT [4] de la plataforma Kaggle, cuenta con 194,922 imágenes que pertenecen a 3,745 pacientes, donde cada imagen corresponde a un corte de tomografía computarizada (TC) y están separadas por 20 nombres diferentes, como se ve en la tabla 5.1.

La primera prueba a la que se sometió el dataset fue al script extractor, el cual debe crear un archivo *pickle* con una lista de *keypoints*, descriptores y nombre de cada imagen. Sin embargo, se obtuvo un error de memoria. Por lo que, como primera solución se dividió el conjunto de datos en 4 partes cada uno con un aproximado de 48,000 imágenes. Se seleccionó uno de ellos, aplicamos el *script* extractor y se obtuvo el mismo error.

	No. de imágenes		No. de imágenes
normalcd1	117	normal1	2,556
normalcd2	171	norma12	1,863
normalcd3	489	norma13	784
137covid	1,889	norma14	2,055
157covid	393	norma15	1,161
cdunnormal	451	HUST	45,912
cdnormal4	129	NCP	31,070
normal	45,758	LIDC	3,999
radiopaedia	4,057	CP	39,009
volume	11,816	coronacases	1,243

Tabla 5.1: Distribución de imágenes.

Por esa razón, se tomó la decisión de seleccionar un conjunto más pequeño de 21,357 imágenes, que corresponden a los cortes con menor cantidad de imágenes. Repetimos el proceso del *script* y esta vez se creó un archivo *pickle* de 7.9 GB con una lista de 15,505,691 descriptores, esto nos dice que cada imagen tiene en promedio 724 descriptores, por lo que todo el conjunto de datos podría crear un archivo *pickle* de 71 Gb con un alrededor de 139,551,219 descriptores, haciendo comprensible el error de memoria.

Con lo anterior llegamos a la observación de que el número de descriptores para una sola imagen es demasiado grande, tal y como se muestra en las figuras 5.1:

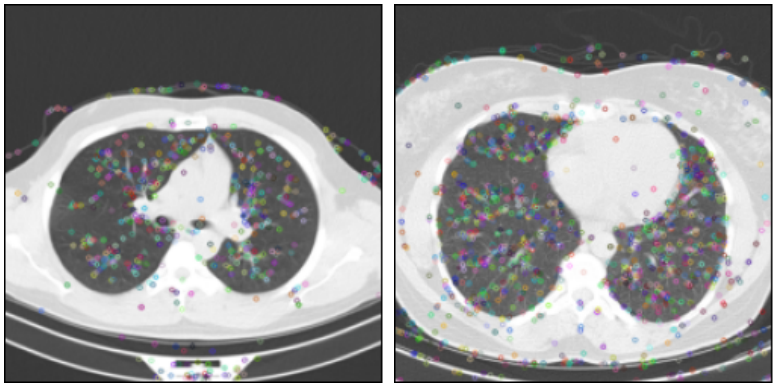

Figura 5.1: Imágenes con 608 descriptores (izquierda) y 1,184 descriptores (derecha)

Ambas imágenes muestran que la mayoría de los descriptores caen dentro del cuerpo, sin embargo, también hay una gran acumulación fuera de él y en las líneas de contorno, lo que vuelve a la información menos confiable.

Para mejorar esa parte se moderó la acumulación de descriptores. Por lo que se tomó una muestra de 3 imágenes, donde aparece el corte axial de tórax para poder ver que la mayoría de los descriptores estuvieran dentro del cuerpo y no en los alrededores, de tal modo que la información importante no se pierda. Figura 5.2

Figura 5.2: Imágenes de muestreo

Para disminuir el número de descriptores de forma general, debemos variar los parámetros de la función *SIFT*. Los parámetros que se pueden reajustar son: el **contrastThreshold** y el **edgeThreshold** que permiten la localización de los puntos de interés.

El contrastThreshold es un umbral de contraste utilizado para filtrar puntos clave débiles en regiones semiuniformes (de bajo contraste), cuyo valor por default es 0.03.

Por otro lado, el edgeThreshold también es un umbral utilizado para filtrar entidades similares a bordes, cuyo valor por default es 10.

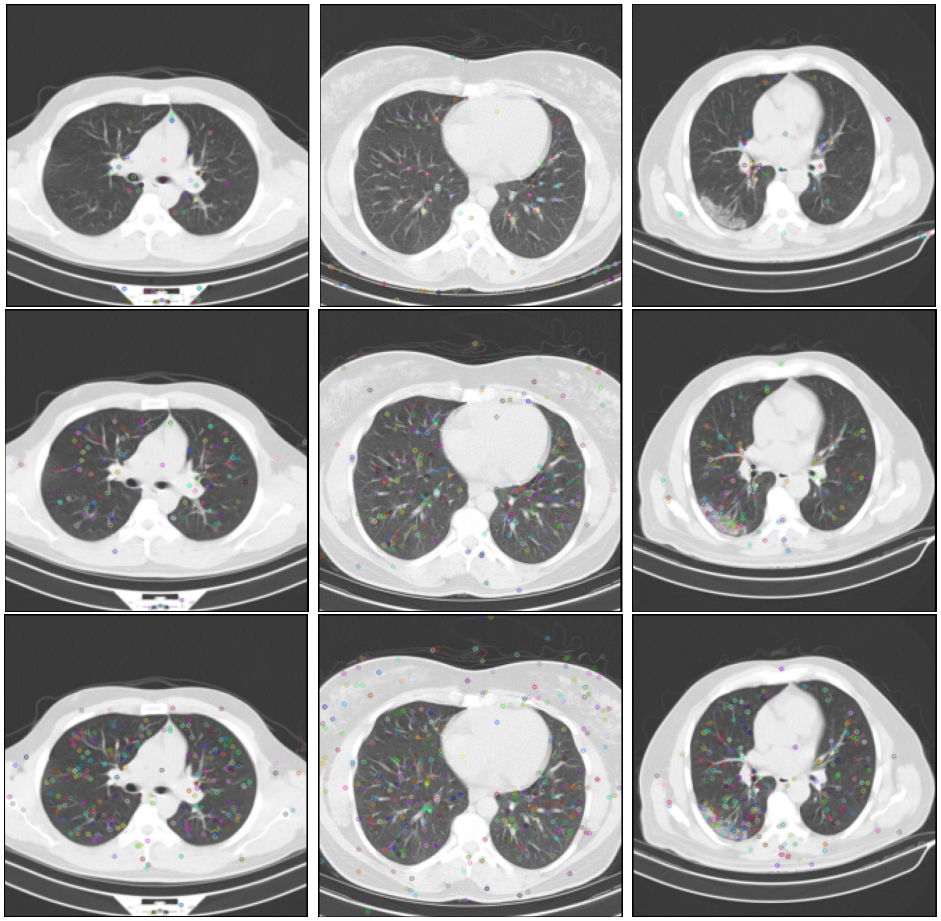

Figura 5.3: Arriba: variación de contrastThreshold (0.13), En medio: variación de edgeThreshold (2), Abajo: variación de contrastThreshold (0.02) y edgeThreshold (2)

Entonces, en la muestra de 3 imágenes aplicamos los cambios de contrastThreshold y edgeThreshold, primero dejando fijo uno de ellos (con valor por default) mientras variamos el segundo. Después, variamos ambos parámetros al mismo tiempo y se obtuvo lo siguiente:

Si dejamos fijo el edgeThreshold = 10 y variamos el contrastThreshold de 0 a 0.20 notamos que el número de descriptores disminuye, no sólo en cantidad sino también en las partes no deseadas como las líneas y el contorno de la mesa. Haciendo que el número de descriptores mínimo que se puede alcanzar sea de una a dos cifras.

Por otra parte, si dejamos fijo el contrastThreshold = 0.03 y variamos el edgeThreshold de 1 a 20, el número de descriptores aumenta. Para el valor 0 obtenemos el número más alto de descriptores, pero sin sentido ya que toda la imagen se llena de ellos. Para el valor 1 obtenemos cero descriptores por lo que el valor mínimo (de tres cifras) y no nulo se alcanza en 2. En este caso también disminuye considerablemente la cantidad de descriptores en las líneas de contorno y el contorno de la mesa, el único problema es que al ser el valor mínimo tres cifras, una imagen puede tener 400 descriptores o más.

Por último, para el caso donde variamos ambos parámetros, se observa que los descriptores disminuyen pero la mayoría se localizan en la parte blanca de las imágenes que corresponde a los huesos. Este resultado hace que no sea la mejor opción.

Con el análisis anterior, se decidió que la mejor forma de disminuir y moderar descriptores es variando el contrastThreshold. Por lo que el siguiente paso es elegir el valor adecuado para este.

Para escogerlo se tomó en cuenta la comparación entre las imágenes y sus descripto-

res, a partir de valores de contrastThreshold de 0.10 a 0.18, así como el archivo creado por el extractor con los parámetros anteriores de tal modo en que sea manejable para el *script* de clusterización y no obtener error al ejecutarlo.

Comenzamos con todo el conjunto de datos y con un valor de contrastThreshold de 0.10, para este caso no se logra crear un archivo *pickle* de descriptores debido a que sigue siendo una cantidad grande. Por lo que la siguiente prueba se decidió empezar con un contrastThreshold de 0.18 a todo el conjunto de datos e ir bajando de valor. El resultado fue un archivo de 8.4 Gb con una lista de 16,478,380 descriptores. Un detalle que resalta de este archivo es que no todas las imágenes tienen descriptores, sólo logra identificar 194,903 imágenes eso quiere decir, que quedan afuera 19 imágenes y se están identificando en promedio 85 descriptores por cada imagen.

Además, como 0.18 es el contrastThreshold limite que deseamos tomar, (debido a que si sigue creciendo los descriptores disminuyen y con ello información relevante) se pensó que la mejor decisión es disminuir el conjunto de datos.

Los criterios para disminuir el contenido de imágenes fueron: retirar imagenes que mostraran una misma tonalidad en la escala de grises o bien, que no tengan un contraste alto como se ve en las figuras 5.4. Los grupos que se retiraron fueron: normal, radiopaedia, normal2, CP, NCP, coronacases y cdnormal4. Dando como total un dataset de 67,794 imágenes.

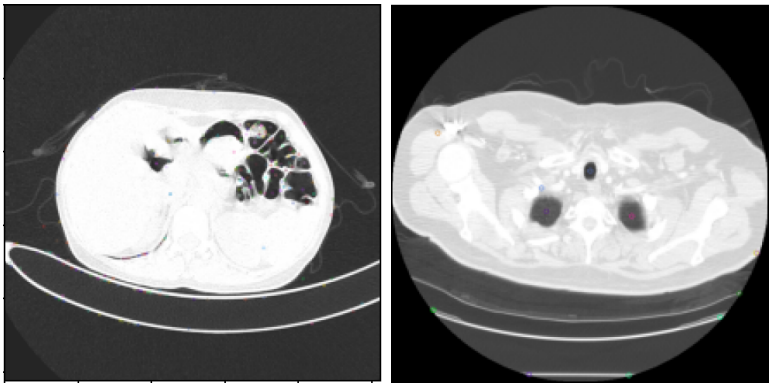

Figura 5.4: Tipo de imágenes eleminadas del dataset

Con este nuevo tamaño del conjunto de datos, se realiza el procedimiento de encontrar el mejor contrastThreshold, para escogerlo nuevamente se toma en cuenta la comparación entre las imágenes y sus descriptores a partir de valores de contrastThreshold de 0.10 a 0.18, así como el archivo creado por el extractor con los parámetros anteriores de tal modo que sea manejable para el script de clusterización y no obtener error al ejecutarlo.

Entonces, se crearon los archivos de descriptores para los valores de contrastThreshold igual a 0.10 y 0.11, con el que se obtuvo un error de memoria. Mientras que para valores de 0.12 a 0.18 no hubo ningún problema y se logró obtener la tabla 2, la cual contiene información de cada archivo *pickle* como: contrastThreshold, Número de descriptores, número de descriptores por imagen, tamaño del archivo y si da error o no al entrar al *script* de clusterización.

constrastT.	No. descriptores	Descriptores por imágen	Tamaño del archivo	Error
0.12	17,449,605	257	8.9 GB	SI
0.13	13,740,371	203	7.0 GB	NO
0.14	10,609,197	156	5.4 GB	NO
0.15	8,304,154	122	4.3 GB	NO
0.16	6,770,079	100	3.5 GB	NO
0.17	5,696,595	84	2.9 GB	NO
0.18	4,819,645	71	2.5 GB	NO

Tabla 5.2: Tabla de caracteristicas del conjunto de datos.

Esta tabla, nos permite hacer la comparación entre cada valor de contrastThrehold. Además, de observar que el archivo *pickle* de descriptores ya no tiene problemas para crearse. Ahora el único problema que tenemos es que si el valor de constrastThreshold es 0.12 no nos permite ejecutar el *script* siguiente, debido a un error de memoria por lo que autómaticamente queda descartado de la elección. Mientras que en los valores de 0.13 a 0.18 no tenemos ese error, por lo que la elección se intentó hacer por la cantidad de descriptores y su distribución de ellos en los siguientes grupos de imágenes figura 5.5 y fgura 5.6.

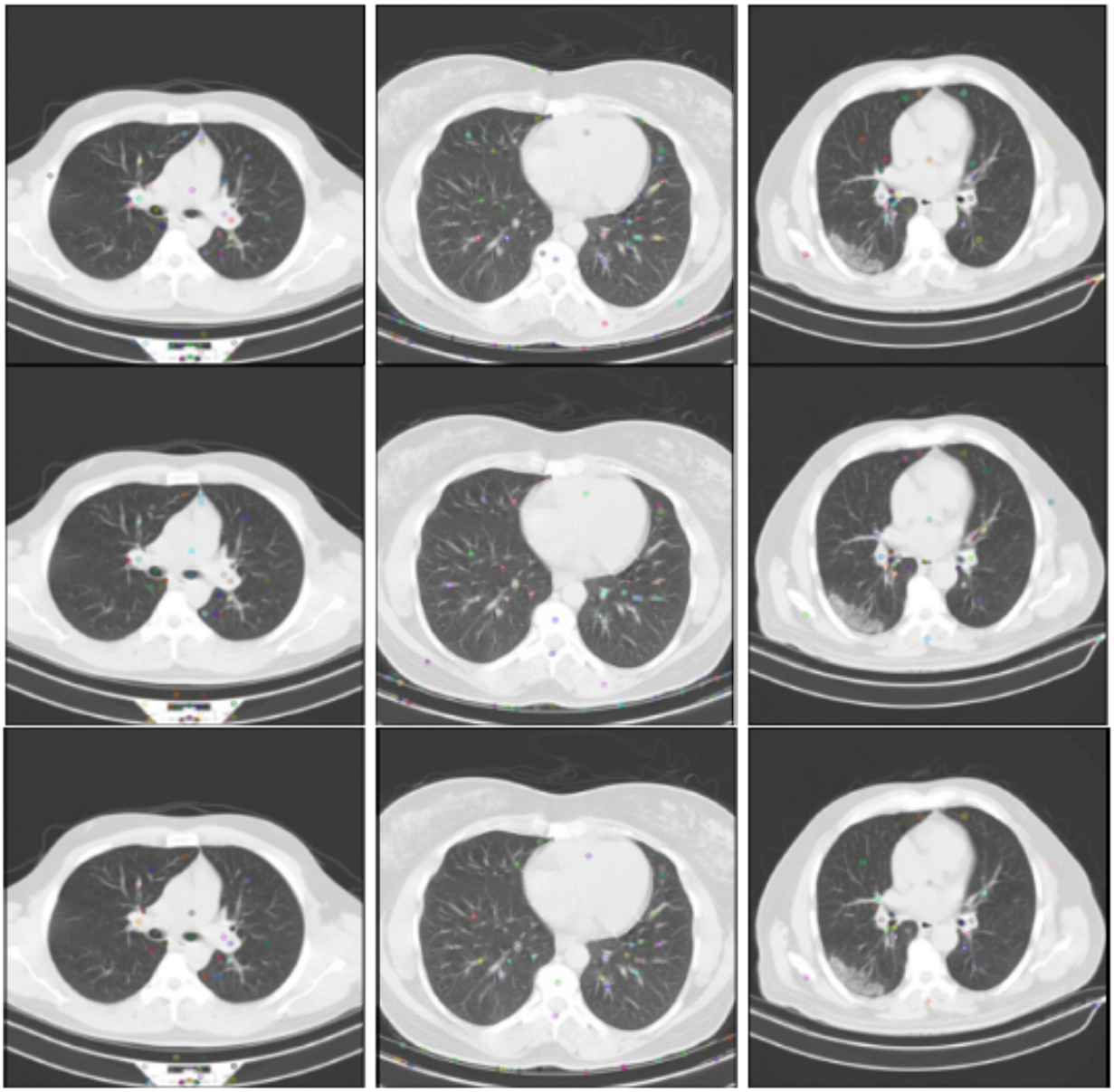

Figura 5.5: Arriba: ContrastThreshold = 0.13, (62, 104, 48) descriptores. En medio: ContrastThreshold = 0.14, (57, 77, 42) descriptores. Abajo: ContrastThreshold = 0.15, (49, 64, 37) descriptores

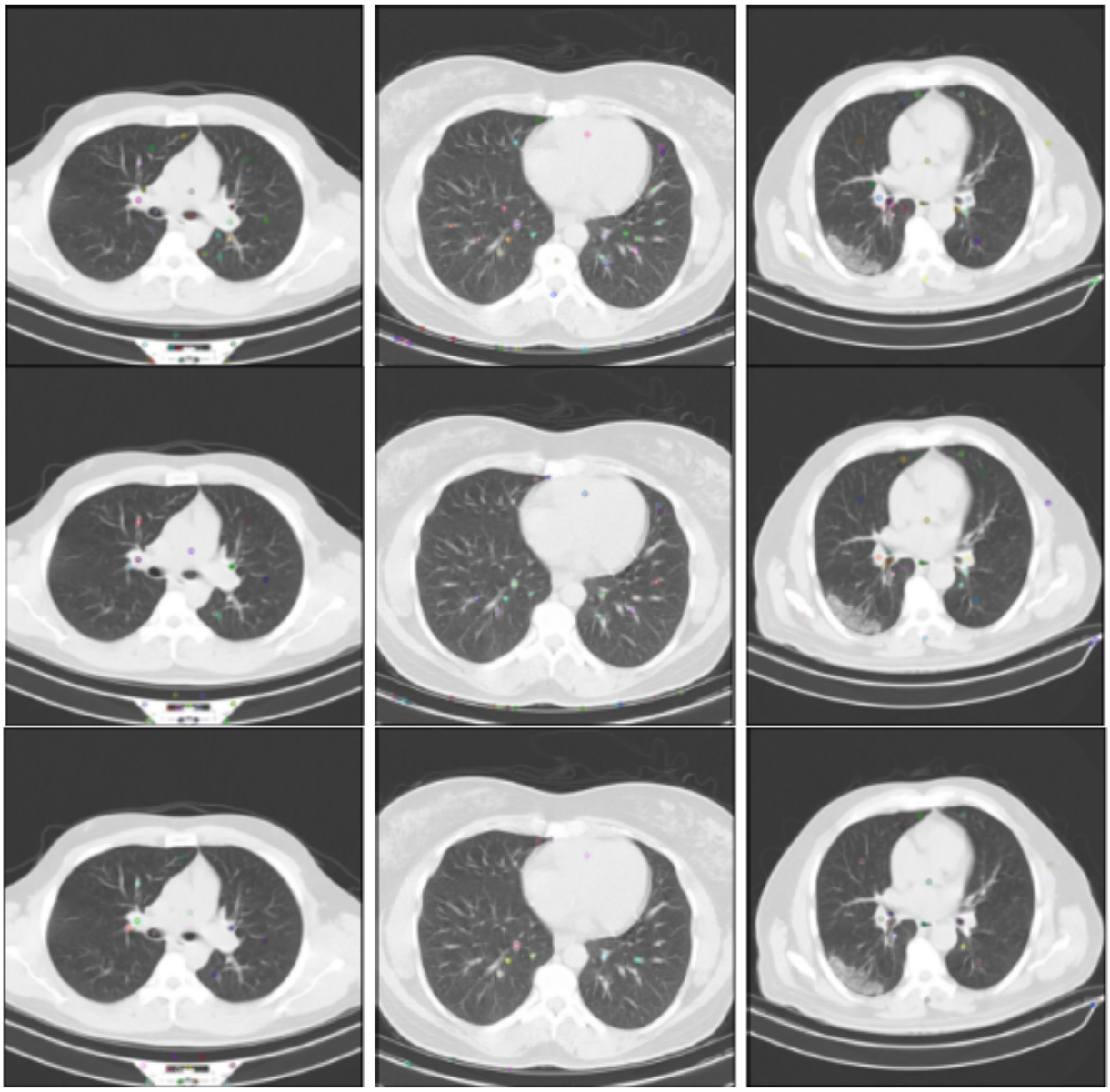

Figura 5.6: Arriba: ContrastThreshold = 0.16, (45, 53, 35) descriptores. En medio: ContrastThreshold = 0.17, (34, 38, 32) descriptores. Abajo: ContrastThreshold = 0.18, (33, 19, 23) descriptores

En las imágenes anteriores observamos que la distribución aún sigue siendo muy parecida. Por lo que se tomó el criterio de que cada imagen debería tener máximo 99 descriptores, lo cual evitaría que el rango de diferencia entre la cantidad de descriptores por imagen sea mayor que 100, es decir, que una imagen tenga 300 descriptores y otra 700.

Ahora, si nosotros necesitamos que cada imagen tenga a lo más 99 descriptores, se tiene que el número de descriptores no debe ser mayor que 6,711,606. En la tabla 5.2, los valores que entran en ese rango son: contrastThreshold de 0.17, 0.18 y tal vez 0.16 que supera por 58,493 descriptores por lo que la decisión se tomó en la comparación de imágenes finales, es decir, ejecutamos todos los *scripts* y *notebooks* para cada valor de constrastThreshold (0.16, 0.17 y 0.18) y observamos con cual se obtienen los mejores patrones.

Por otra parte, en la misma tabla 5.2 anotamos la característica de si existe error al ejecutar el *script* de clusterización. Sólo uno de ellos lo tuvo y fue descartado. Para los demás que fueron aceptados y no rechazados por el número de descriptores, aplicamos el proceso restante. Este proceso, consiste en ejecutar el *script* SHMdocument para generar lo que llamamos *Nip Corpus*. Seguido de esto, sacamos el inverso al *Nip Corpus* y por último obtenemos la tabla que muestra el número de tópicos que corresponden al variar los parameros r y l del algoritmo *min-hashing*.

De este modo, los casos que nos interesan son los siguientes:

Tabla para un valor de contrastThreshold igual a 0.16

l \ r	2	3	4	5	6	7	8	9
100	16	9	8	3	0	0	0	0
1,000	398	36	17	10	8	4	4	1
5,000	2,292	157	51	19	12	11	7	7
9,000	3,506	298	56	28	14	9	12	10

Tabla 5.3: Tabla de número de topicos variando l y r de 0.16.

Tabla para un valor de contrastThreshold igual a 0.17

l \ r	2	3	4	5	6	7	8	9
100	12	7	4	1	0	0	0	0
1,000	272	29	13	10	6	9	2	1
5,000	1,754	95	44	27	15	11	7	3
9,000	2,912	188	42	30	18	17	7	5

Tabla 5.4: Tabla de número de topicos variando l y r de 0.17.

Tabla para un valor de contrastThreshold igual a 0.18

l \ r	2	3	4	5	6	7	8	9
100	23	8	5	4	1	2	2	0
1,000	213	30	17	9	9	6	3	2
5,000	1,388	83	34	25	14	13	10	6
9,000	2,563	135	40	29	14	12	10	10

Tabla 5.5: Tabla de número de topicos variando l y r de 0.18.

Las tablas anteriores son el número de tópicos que se obtienen al variar los parámetros r y l. El primero se encuentra en un rango de [2,9] y el segundo de [0,9,000]. En ellas vemos que el número de tópicos aumenta, si el parámetro l lo hace y el número de tópicos disminuye si el parámetro r aumenta. Entonces, haciendo una descripción de lo que fue más destacado en cada caso se obtuvo lo siguiente:
La Tabla 5.3 muestra un número de tópicos de 0 a 3,506 Donde las 3 estructuras más relevantes fueron las siguientes:

- Estructura 1 aparece en #Tópicos: 4 de la tabla 5.3 y corresponde al tópico 2. También se repite para los #Tópicos:12 y 157 de la tabla 5.3 y que corresponden a los tópicos 11 y 90. Donde la zonas marcadas se encuentran en la parte central del torax muy cercana al pecho y cuyo elemento dentro de las circunferencias con mayor repetición es la arteria aorta. Figura 5.7

- Estructura 2 aparece en #Tópicos: 17 de la tabla 5.3. y corresponde al tópico 4.

También se repite para los #Tópicos: 12, 10 y 14 de la tabla 5.3 y corresponden a los tópicos 8, 4 y 5. Donde las zonas marcadas se encuentran a los costados de los órganos del centro y cuyos elementos dentro de las circunferencias se les puede llamar nódulos, ya que pueden ser tejidos de diversa naturaleza como ganglios, estructura de vasculatura, conglomerados entre otros. Figura 5.8

- Estructura 3 aparece en #Tópicos: 19 de la tabla 5.3. y corresponde al tópico 8. También se repite para los #Tópicos 12, 14 y 157 de la tabla 5.3 y corresponden a los tópicos 5, 5, 89. Donde las zonas marcadas se encuentran en el área de pulmón y cuyos elementos dentro de las circunferencias, sólo para algunas imágenes son partes difusas de color gris a las que se les conoce como vidrio despulído. Figura 5.9

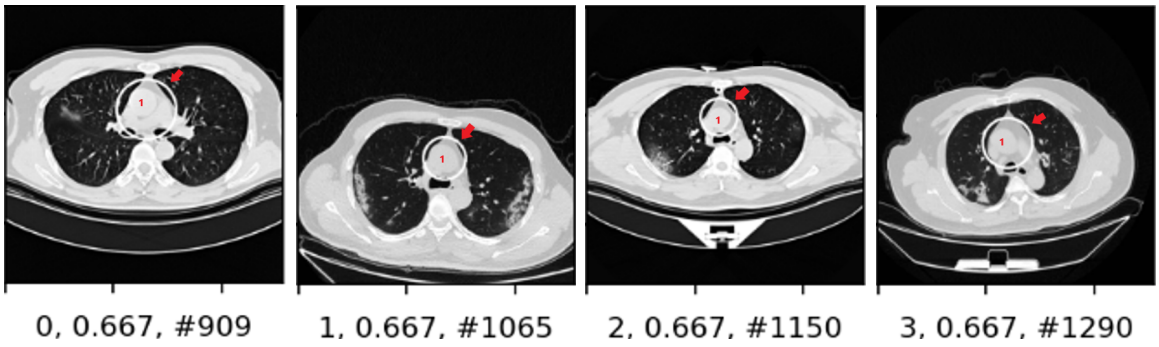

Figura 5.7: **Estructura 1.** El elemento con mayor repetición en dicha estructura es la arteria aorta.

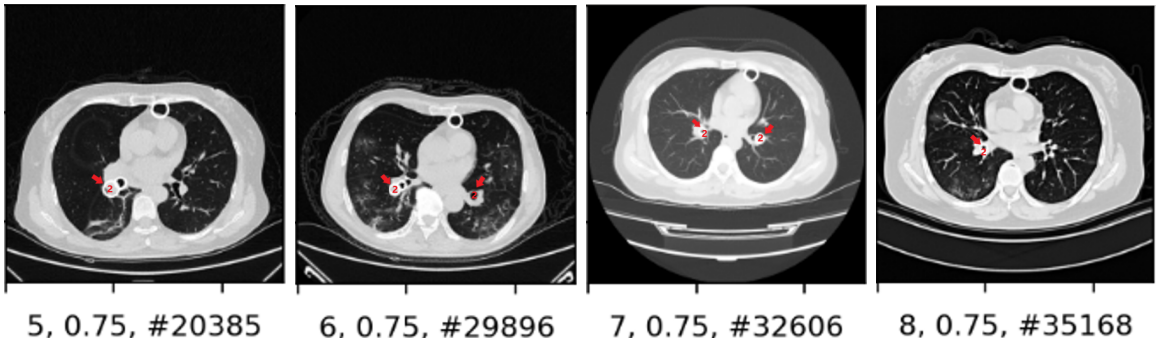

Figura 5.8: **Estructura 2.** Se les puede nombrar como nódulos, ya que se observan tejidos de diversa naturaleza como ganglios, estructura de vasculatura, conglomerados entre otros.

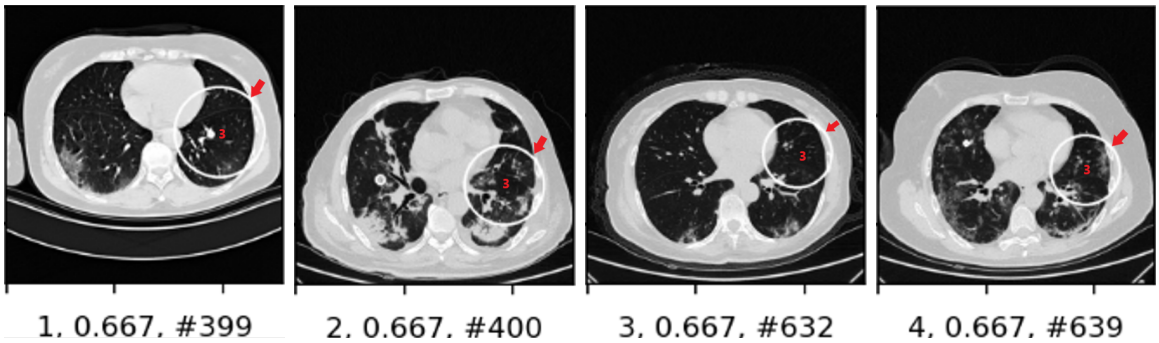

Figura 5.9: **Estructura 3.** El elemento con mayor repetición en dicha estructura es la zona difusa y se conoce como vidrio despulído.

Para la tabla 5.4 se muestra un número de tópicos de 0 a 2,912. Donde las 2 estructuras más relevantes fueron las siguientes:

- Estructura 1 aparece en #Tópicos: 12 de la tabla 5.4 y corresponde al tópico 8. También se repite para los #Tópicos: 29 y 1754 de la tabla 5.4 y que corresponden a los tópicos 21 y 1453. Donde las zonas maracadas se encuentran en la parte central del torax. Sin embargo, no todos los elementos dentro de la circunferencia son iguales y en algunas se repite el ventrículo izquierdo. Figura 5.10

- Estructura 2 aparece en #Tópicos: 18 de la tabla 5.4. y corresponde al tópico 1. También se repite para los #Tópicos: 7, 12 y 27 de la tabla 5.4 y corresponden a los tópicos 2, 2 y 0. Donde las zonas marcadas estan en la mesa de exploración, no son tan relevantes pero se repiten mucho. Figura 5.11

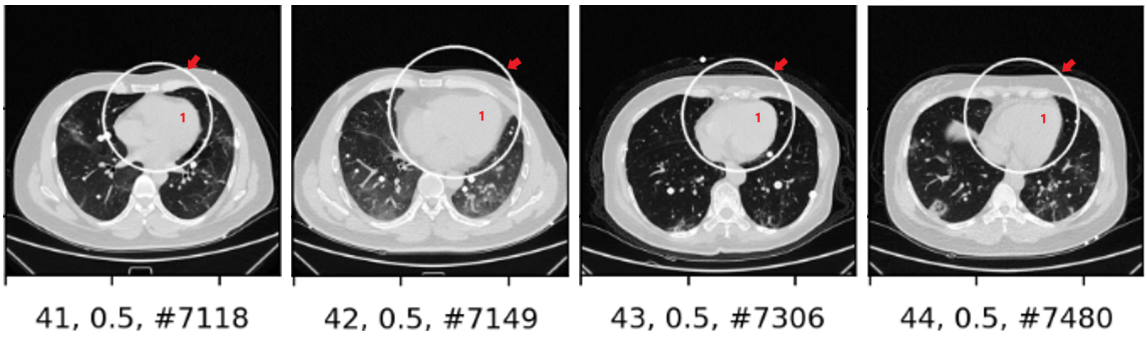

Figura 5.10: **Estructura 1.** los elementos dentro de las circunferencias no son todos iguales. Sin embargo, se repite el ventrículo izquierdo.

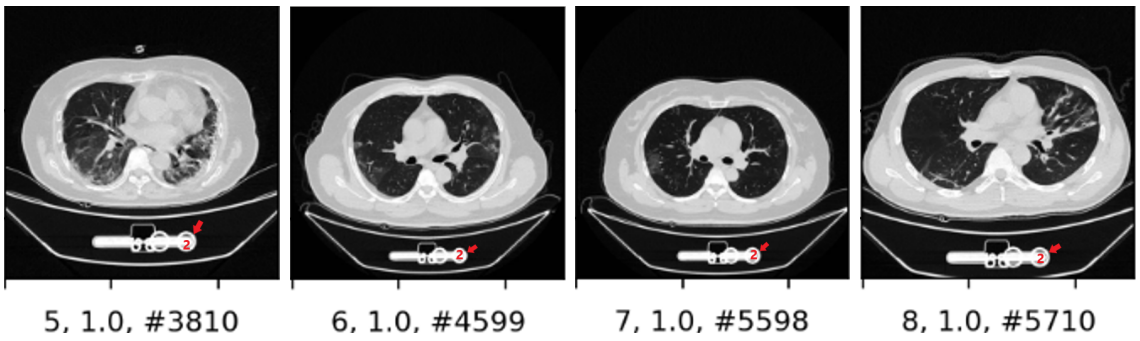

Figura 5.11: **Estructura 2.** pertenece a la mesa de exploración y no al cuerpo.

Por último, para la tabla 5.5 se muestra un número de tópicos de 0 a 2,563. Donde las dos estructuras más relevantes fueron:

- Estructura 1 aparece en #Tópicos: 30 de la tabla 5.5 y corresponde al tópico 25. También se repite para los #Tópicos: 34, 83 y 135 de la tabla 5.5 y que

corresponden a los tópicos 15, 57 y 75. Donde las zonas marcadas se encuentra en la parte central del torax muy cerca del pecho y cuyo elemento denrtro de las circunferencias con mayor repetición es la arteria aorta. Figura 5.12

- Estructura 2 aparece en #Tópicos: 2,564 de la tabla 5.5. y corresponde al tópico 2,559. También se repite para los #Tópicos: 9 y 10 de la tabla 5.5 y corresponden a los tópicos 6 y 6. Donde las zonas marcadas se trata en su mayoría del cuerpo vertebral. Figura 5.13

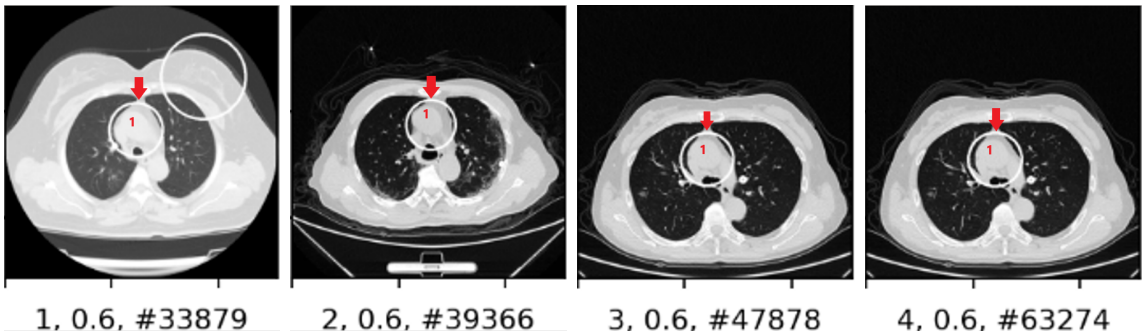

Figura 5.12: **Estructura 1.** el elemento con mayor repetición es la arteria aorta.

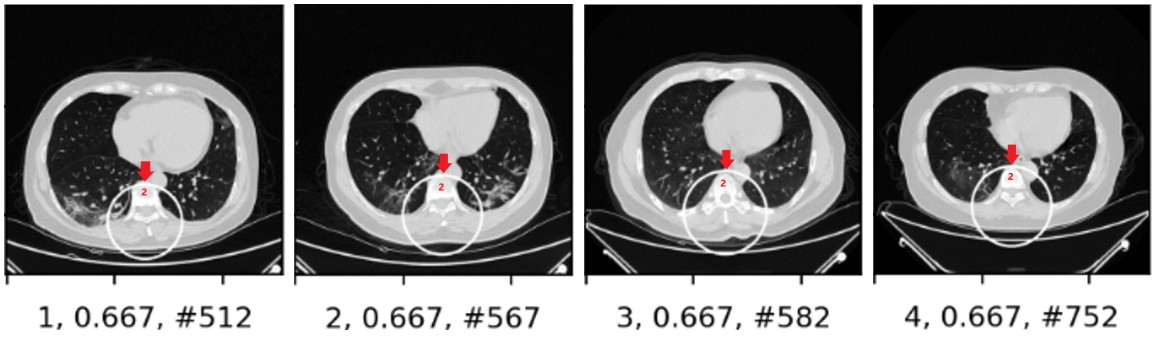

Figura 5.13: **Estructura 2.** La mayoria de las zonas marcadas son del cuerpo vertebral.

Como detalle a los experimentos se tiene lo siguiente:

- Las imágenes que obtuvieron descriptores de más de dos dígitos, permitió que el algoritmo tuviera más coincidencias, el problema es que se vuelve complicado para la computadora.

- En el caso de la visualización, el umbral partía de 0.49 y se modificaba si el número de imágenes era igual o mayor a 1,000.

- Para números de tópicos muy grandes, los primeros valores no tienen coincidencias en las imágenes, para los intermedios sí, pero no muestran patrones relevantes y en los del final se encuentran las mayores coincidencias.

- Para la tabla 17 no existió ninguna coincidencia en los números de tópicos de 0 a 11 y de 13 a 17. siendo los intervalos más largos para los tres casos.

- La mayor parte de los patrones encontrados fueron órganos y como detalle, la estructura 1 para los tres casos es muy similar, debido a que se encuentra al centro y muy cerca del pecho, pero los elementos que contienen no son los mismos.

- Juntando todos los casos hay un total de 6 patrones, es decir, hay características que se repiten y otras que solo le pertenecen a cada conjunto. En el apéndice A se muestran otros 2 casos que tienen la peculiaridad de repetirse pero no son estructuras como tal.

6 Conclusiones

Con el desarrollo anterior se puede llegar a las siguientes observaciones:
Generar archivos con una cantidad grande de descriptores resulta ser complicado, ya que para lograrlo, la base de datos se debe disminuir. Sin embargo, una base de datos pequeña no se podría explotar de manera eficiente ya que la comparación entre datos se ve muy limitada por lo que debe existir un equilibrio.
Ese equilibrio se intentó encontrar al hacer la base de datos un conjunto de 67,794 imágenes y al elegir tres valores diferentes de contrastThreshold que generarían una cantidad distinta de descriptores y *keypoints*. Que al final minaríamos de la misma manera para ser comparados. De este resultado podemos decir que, la cantidad de *keypoints* sí provoca un cambio en el número de tópicos encontrados. Donde hay que aclarar que el número de tópicos no es igual a el número de estructuras, pero sí existe una mayor posibilidad de encontrarlas. Por esa razón y al tener una estructura más, se toma como el mejor valor de contrastThreshold 0.16.

Este programa logró identificar parecidos entre imágenes, ya que las estructuras que se hallaron no siempre encerraron elementos iguales pero si parecidos. Como ejem-

plo, fue la estructura uno que está en el centro del torax y que se repite para los tres casos. Siendo para dos casos, la arteria aorta y para el otro el ventrículo izquierdo. Por otra parte, este programa también permitió encontrar igualdades entre imágenes, por ejemplo, la estructura dos de la tabla 5.4, que corresponde a la camilla en la que se coloca el paciente, se repite en varías imágenes. Esta estructura no tiene tanta relevancia como las partes que pertenecen al cuerpo, pero permite observar que el programa tiene mayor posibilidad de detectar puntos iguales en imágenes.

Los elementos que tienen nombre en las imágenes, fueron identificados por radiólogos. La mayoría fueron estructuras anatómicas como: la arteria aorta, nódulos, ventrículo izquierdo, cuerpo vertebral y estructuras conocidas como de "vidrio despulído". Este último suele ser un indicador de neumonitis. A pesar de este resultado, este algoritmo queda lejos de poder utilizarse como apoyo al dignóstico para COVID. Sin embargo, puede ser una introducción para un proyecto a futuro, donde la extracción de características sea utilizando una red neuronal. La participación un médico radiólogo desde el principio puede ayudar a la especificación de elementos necesarios para un diagnóstico y que el algoritmo pueda realizar busquedas específicas.

A Tópicos y patrones

Los patrones y tópicos encontrados por los 3 experimentos fueron los siguientes:

a. Marcas de tejido blando en los extremos superior e inferior izquierdo y derecho.

b. Detección de la parte central del torax muy cercana al pecho.

c. Detección de ramificaciones cerca de la parte central de torax.

d. Detección de zonas grises a los costados del centro del torax de gran tamaño.

e. Detección de imágenes iguales.

f. Deteción de partes del cuerpo vertebral.

De manera general, la siguiente tabla muestra los tópicos que se lograron destacar para cada patrón. En ella está el número de tópico seguido de un punto y del tópico relevante. Como recordatorio el número de tópicos es el que aparece en las tablas 5.3, 5.4 y 5.5 de datos y experimentación. Donde el color magenta corresponde a los datos de la tabla 5.3, el color azul a los datos de la tabla 5.4 y el color verde a los datos de la tabla 5.5

De lo anterior se obtienen las siguientes imagenes muestra:

.

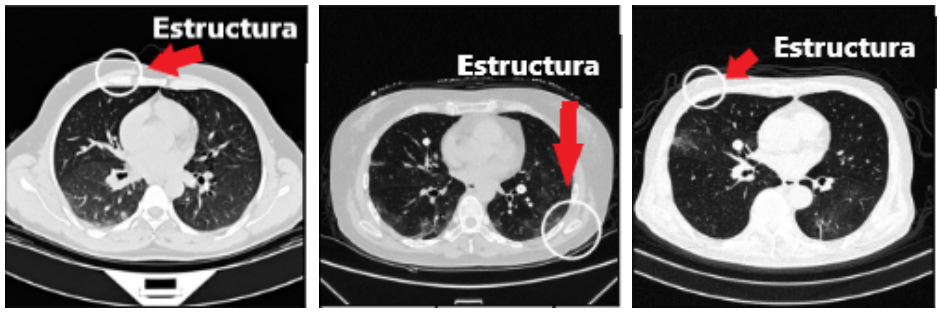

Figura A.1: Ejemplos de a uno por cada valor de contrastThreshold

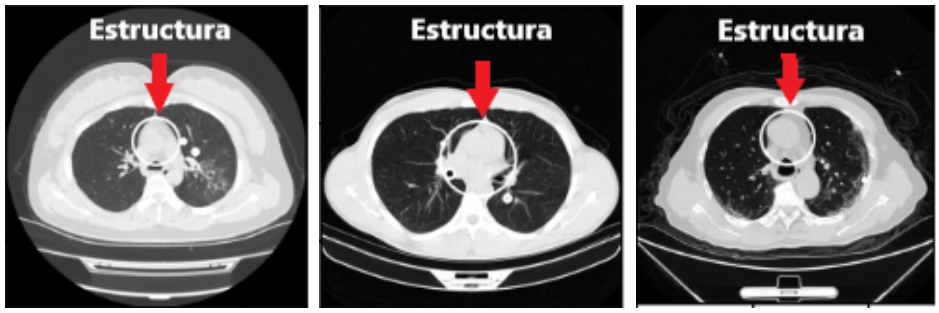

Figura A.2: Ejemplos de b uno por cada valor de contrastThreshold

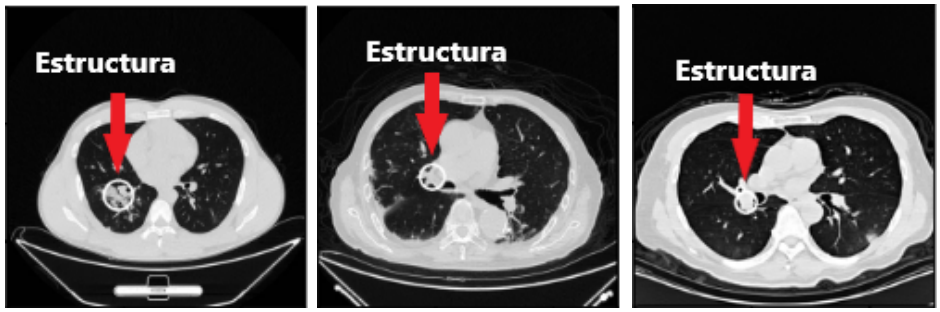

Figura A.3: Ejemplos de c uno por cada valor de contrastThreshold

Figura A.4: Ejemplos de d uno por cada valor de contrastThreshold

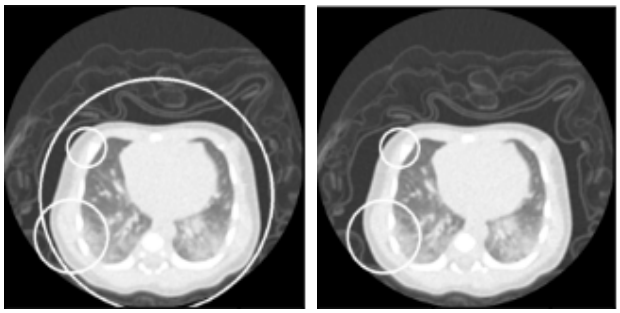

Figura A.5: Ejemplos de e uno por cada valor de contrastThreshold de 0.17 y 0.18

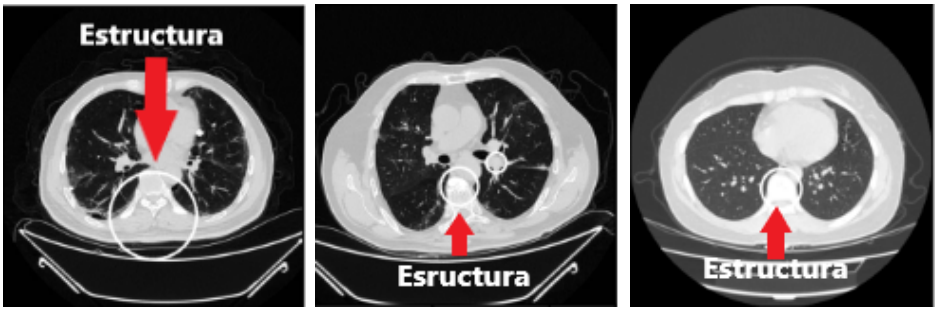

Figura A.6: Ejemplos de f uno por cada valor de contrastThreshold

Patrón	Tópicos que corresponden a cada patrón
a	3.1, 3.3, 4, 7, 8, 10, 12.10, 2912.211, 8.6, 17.2, 2565.2449
b	4, 12, 36.15, 157.90, 3506.1967, 12.8, 29.21, 1754.1653, 1754.1453, 1754.1053, 25.21, 29.28, 30.25, 34.15, 83.55, 135.75, 1338.187, 2565.579
c	10, 12.4, 14.8, 14.11, 14.13, 16.6, 19.5, 19.12 28.13, 398.349, 3506.3054, 2912.2211, 2.1, 29.16, 30.9 40.38, 2565.2009
d	12.5, 14.5, 16.15, 17.4, 19.8, 28.7, 36.21, 36.29, 51.21, 56.9, 56.25, 157.78, 157.81, 157.89, 157.90, 157.40, 298.115, 398.389, 3506.1434,3506.854, 1754.453, 1754.253, 12.11, 13.7, 40.24, 40.14, 83.39, 1338.1357, 1338.1317, 2565.2459, 2565.1519, 2565.1499
e	12.2 29.2 2912.611, 23.0, 23.2, 1338.987, 2565.669
f	298.149, 398.69, 398.387, 12.7, 27.0, 29.10, 29.18, 9.6, 13.12, 14.11, 14.7, 2565.2559, 2565.2339, 2565.2299, 2565.2289, 2565.1939, 2565.1859, 2565.1879, 2565.1089

Tabla A.1: Tópicos que corresponden a cada patrón.

B Scripts y notebooks

Todos los Scripts y Notebooks de esta tesis fueron tomados del proyecto: Descubrimiento de estructuras en imágenes de galaxias con MinHshing, k.means y SIFT. Realizados por Ricardo García García, Mayra Cid Oros y Jose Ángel Guadarrama Ramírez. Para más información visitar el repositorio que aparece en [37].

Para correr el extractor.py se ejecuta la siguiente linea:

```
python extractor.py SIFT Ruta_imagenes Ruta_archivo_genera\Nombre
```

B.1. Código Extractor.py

```

#!/usr/bin/env python
# coding: utf-8
"""
Exporta una lista (archivo pickle) que contiene los keypoints,
descriptores y nombre del archivo para cada imagen encontrada
en el directorio.
"""
import sys
import os.path as path

modulos_path = path.abspath('../minIA')
if modulos_path not in sys.path:
    sys.path.append(modulos_path)

from utiles import lectura_img
from tqdm import tqdm
import argparse
import numpy as np
import cv2 as cv
import pickle
```

```
22
23 parser = argparse.ArgumentParser()
24 parser.add_argument("extr",
25     help='Extractor', choices=['SIFT', 'SURF', 'DELF'])
26 parser.add_argument("dir",
27     help='Ruta del directorio de imagenes')
28 parser.add_argument("dir_output",
29     help='Ruta del archivo de salida')
30 parser.add_argument("-auto",
31     help='Cálculo de parámetros automático', action="store_true")
32
33 parser.add_argument('-median_filter',
34     help='Filtro de mediana', default=False, type=bool)
35 parser.add_argument('-median_value',
36     help='Valores Impares', default=15, type=int)
37
38 parser.add_argument("-threshold",
39     help='Parametro de SURF', default=100, type=int)
40 parser.add_argument("-nOctaves",
41     help='Parametro de SURF', default=4, type=int)
42 parser.add_argument("-nOctaveLayers",
43     help='Parametro de SURF y SIFT', default=3, type=int)
44 parser.add_argument("-extended",
45     help='Parametro de SURF', default=False, type=bool)
46 parser.add_argument("-upright",
47     help='Parametro de SURF', default=True, type=bool)
48
49
50 parser.add_argument("-nfeatures",
51     help="Parametro de SIFT", default=0, type=int)
52 parser.add_argument("-contrastThreshold",
53     help="Parametro de SIFT", default=0.18, type=float)
54 parser.add_argument("-edgeThreshold",
55     help="Parametro de SIFT", default=10, type=float)
56 parser.add_argument("-sigma",
57     help="Parametro de SIFT", default=1.6, type=float)
58
59 args = parser.parse_args()
60
61 '''
62 Se planea que todos los extractores implementen esta clase, para que
63 el código principal (main) no tenga modificaciones y funcione igual
64 independientemente del método de extración.
65 '''
66 class Extractor(object):
67     def calculoDescriptores(self, imagen):
68         raise NotImplementedError('todas las subclases deben
69         sobrescribir')
70         #      [[ [x, y, size] , [descriptor], [nombreArch] ] ....[]]
71
72
73 class Sift(Extractor):
74     def __init__(self, auto, nfeatures,
75         nOctaveLayers, contrastThreshold, edgeThreshold, sigma):
76         if auto:
77             self.sift = cv.xfeatures2d_SIFT.create()
78         else:
79             self.sift = cv.xfeatures2d_SIFT.create(nfeatures,
80             nOctaveLayers, contrastThreshold, edgeThreshold, sigma)
81
82     def calculoDescriptores(self, imagen):
83         kps, descs = self.sift.detectAndCompute(imagen,None)
84         keypoints = list()
85         for kp in kps:
86             keypoints.append([kp.pt[0], kp.pt[1], kp.size])
```

```
        return  {'keypoints': keypoints, 'descriptors':descs}

class Surf(Extractor):
    def __init__(self, auto, threshold, nOctaves, nOctaveLayers,
                 extended, upright):
        if auto:
            self.surf = cv.xfeatures2d.SURF_create()
        else:
            self.surf = cv.xfeatures2d.SURF_create(threshold,
            nOctaves, nOctaveLayers, extended, upright)

    def calculoDescriptores(self, imagen):
        kps, descs = self.surf.detectAndCompute(imagen,None)
        keypoints = list()
        for kp in kps:
            keypoints.append([kp.pt[0], kp.pt[1], kp.size])
        return  {'keypoints': keypoints, 'descriptors':descs}

#Definición del tipo de extractor
if args.extr == 'SIFT':
    extractor = Sift(args.auto, args.nfeatures, args.nOctaveLayers,
    args.contrastThreshold, args.edgeThreshold, args.sigma)
elif args.extr == 'SURF':
    extractor = Surf(args.auto, args.threshold, args.nOctaves,
    args.nOctaveLayers, args.extended, args.upright)
else:
    #extractor = Delf()
    pass

def main(args=args):
    path_images = lectura_img(args.dir)
    path_pickle = path.abspath(args.dir_output+'_'+args.extr+
    '.pickle')
    descriptores = list()
    pickle_file = open(path_pickle, 'wb')
    for imagen in tqdm(path_images):

        if args.median_filter == True:
            img = cv.imread(imagen, cv.COLOR_HSV2BGR)
            image_gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
            img= cv.medianBlur(image_gray, args.median_value)
        else:
            img = cv.imread(imagen, cv.COLOR_BGR2GRAY)

        nom_img = path.split(imagen)[1]
        descs_img = extractor.calculoDescriptores(img)
        if descs_img['descriptors'] is not None:
            descs_img['name_img'] = nom_img
            descriptores.append(descs_img)

    pickle.dump(args, pickle_file, pickle.HIGHEST_PROTOCOL)
    pickle.dump(descriptores, pickle_file, pickle.HIGHEST_PROTOCOL)
    pickle_file.close()
    print('¡Listo! ' + args.extr)

if __name__ == '__main__':
  main()
```

Para correr el código de cluster.py se ejecuta la siguiente linea:

```
python cluster.py SIFT Archivo_extractor Ruta_archivo_genera\Nombre
```

B.2. Cluster.py

```
#!/usr/bin/env python
# coding: utf-8
"""
Genera el vocabulario de palabras visuales
Entrada:
    Archivo de descriptores originales (generado por el extractor)
    Número de clusters a generar
Salida:
    Archivo de descriptores clusterizados
"""

import pickle
import numpy as np
import pandas as pd
from sklearn.cluster import MiniBatchKMeans
import matplotlib.pyplot as plt
import argparse

parser = argparse.ArgumentParser()
parser.add_argument('descriptors',
  help = 'Ruta del archivo generado por el extractor')
parser.add_argument('cluster',
  help = 'Ruta del archivo a generar')
parser.add_argument('Nclusters', help = 'Número de clusters',type=int)
args = parser.parse_args()

def main(args):
    with open(args.descriptors, 'rb') as pickle_file:
        params = pickle.load(pickle_file) #Parámetros de extración
        desc_imgs = pickle.load(pickle_file) #Lista de descriptores
        b = [item['descriptors'] for item in desc_imgs] #"Vectores
        descriptores"
        c = [item['name_img'] for item in desc_imgs] #"Nombre de la
        imagén "
        #Longitud de cada vector descriptor
        hi=[]
        for i in range(0,len(b),1):
            hi.append(len(b[i]))
        #Lista de descripores con sus n entradas
        descriptores = b
        desc = np.array(descriptores)
        J=[]
        for D in desc:
            for i in D:
                J.append(i)
        #Centroides y etiquetas de cada vector descriptor
        print("\nDatos cargados correctamente, iniciando

        clusterización...\n")
        kmeans = MiniBatchKMeans(n_clusters=args.Nclusters,

        init='k-means++',
          n_init=10, max_iter=300, tol=0.0001 ).fit(J)
        centroids = kmeans.cluster_centers_
        etiquetas = kmeans.labels_
        la = kmeans.predict(J)
        #Suma de las longitudes de los vectores
        laSuma = 0
        nu=[]
```

```
        for i in hi:
            laSuma = laSuma + i
            nu.append(laSuma)
        #Se le agrega el 0 a la lista
        nu.insert(0,0)
        #Etiquetas agrupadas por por vector descriptor

        (listas de etiquetas)
        lab=[]
        for i in range(0,len(nu)-1):
            lab.append(etiquetas[nu[i]:nu[i+1]])
        # Cambiamos el tamaño del arreglo anterior
        my_array= np.array(lab)
        poo=my_array.reshape((len(lab),1))
        # Agregamos los nombres al arreglo
        cool=np.insert(poo, poo.shape[1], np.array((c)), 1)
        #Formato en pandas
        labels=pd.DataFrame(cool)
        #Se guarda en un archivo pickle
        with open(args.cluster, 'wb') as pickle_file:
            pickle.dump(labels, pickle_file)
main(args)
```

B.3. SMHdocument.py

Para correr el código de SMHdocument.py se ejecuta la siguiente linea:

```
python SMHdocument.py FRECUENCY Archivo_extractor Archivo_cluster
Ruta/Nombre_nuevo
```

```

#!/usr/bin/env python
# coding: utf-8

'''
Obtiene el documento necesario para SMH, asiciando el ID de cada
imagen con los ID's de los descriotores (centroides) de la imagen.

Entrada:
    Archivo de descriptores originales (generado por el extractor)
    Archivo de descriptores clusterizados

Salida:
    Documento de entrada para SHM
'''
import pickle
import pandas as pd
import numpy as np
from collections import Counter
from tqdm import tqdm
import argparse

parser = argparse.ArgumentParser()
parser.add_argument("magnitude", help='Selecciona la magnitud asociada
a los índices de la imagen',
  choices=['FRECUENCY', 'WEIGHT'])
parser.add_argument("original", help='Ruta del archivo de
descriptores originales')
```

```
parser.add_argument("cluster", help='Ruta del archivo de
descriptores clusterizados')
parser.add_argument("document_name", help='Ruta y nombre del archivo
a crear')
args = parser.parse_args()


def genDocumentFrecuency(desc_imgs,images_descr):
    """
    En cada imagen, toma cada descriptor de la imagen y cuenta el
    número de veces que aparece en él.
    """
    with open(args.document_name, 'w') as file:

        img = 0
        for descr in tqdm(images_descr[0]):
            cnt_caract = Counter(descr)
            row = str(len(cnt_caract))
            for car in sorted(cnt_caract.keys()):
                row+= ' '+ str(car) +':'+str(cnt_caract[car])
            file.write(row+'\n')
            img+=1


def genDocumentWeight(desc_imgs,images_descr):
    """
    En cada imagen, toma cada descriptor de la imagen y mide el
    tamaño asociado
    a cada uno de ellos, si se repiten los descriptores suma ambos
    tamaños.
    """
    with open(args.document_name, 'w') as file:
        #Obtiene lista de KP para cada descriptor
        img_keypoints = [x['keypoints'] for x in desc_imgs]
        img_centroides = list(images_descr[0])

        for centroides, keypoint in tqdm(zip(img_centroides,
        img_keypoints),
          total=len(img_centroides)):
            #Diccionarios con Centroide, peso en 0
            img = (dict( (cent, 0) for cent in set(centroides)))
            for cent, kp in  zip(centroides, keypoint):
                img[cent] += round(kp[2]) #Incrementa el tamaño
            row = str(len(img))+"
            "+str(img).replace(", "," ").replace(": ",":")[1:-1]
            file.write(row+'\n')

def main(args):
    with open(args.original, 'rb') as file_original:
        args_info = pickle.load(file_original)
        desc_imgs = pickle.load(file_original)
    with open(args.cluster, 'rb') as file_cluster:
        images_descr = pickle.load(file_cluster)
    #Verificación de simetría, no ejecutar si no son simétricos
    if (len(images_descr) != len(desc_imgs)):
        print("\nLos índices están desfasados, verificar la

        cardinalidad")
        print("Cluster: " + str(len(images_descr)))
        print("Original: " + str(len(desc_imgs))+"\n")
        exit()
    print("\nArchivos cargados correctamente, generando archivo...")
    if args.magnitude == 'FRECUENCY':
        genDocumentFrecuency(desc_imgs,images_descr)
    else:
        genDocumentWeight(desc_imgs,images_descr)
```

```
96
97 main(args)
98
```

B.4. Archivo inverso

Para obtener el archivo inverso se ejecuta la siguiente linea:

`smhcmd ifindex Archivo_generado_por_SMHdocument.py Archivo_nuevo_terminación_.ifs`

B.5. Minado del archivo inverso

Para el minado con diferentes valores de los parámetros r y l con $r \in [2, 10]$ y

$l \in [0, 9,000]$ se ejecuta la siguiente linea:

`smhcmd discover -r 2 -l 1000 Archivo_generado_SMHdocument.py Archivo_nuevo_+.models`

B.6. Notebook visualización.ipynb

El siguiente código se realiza en notebook

1° Celda

```
1  import sys
2  from os import path, mkdir, listdir, chdir, path
3  import pickle
4  import cv2 as cv
5  import ipywidgets as widgets
6  from ipywidgets import interact, interact_manual
7  import matplotlib.pyplot as plt
8  from skimage import io
9  import pandas as pd
10 import numpy as np
11 from IPython.display import display
12 from PIL import Image
13
14 modulos_path = path.abspath('../minIA')
15 if modulos_path not in sys.path:
16     sys.path.append(modulos_path)
17
18 from utiles import lectura_img
19 import random
20 random.seed(42)
21 from collections import Counter
```

2° Celda

```
#Recibe lista de keypoints de una imagen, crea los objetos KP
de openCV
def genKeyPoints( kp_img ):
    keypoints = list()
    for kp in kp_img:
        keypoints.append(cv.KeyPoint(kp[0], kp[1], kp[2]))
    return keypoints
```

3° Celda

Visualización tópicos

```
El siguiente codigo carga los tópicos y elementos de la figura
para mostrar los archivos; seleccionar de la primera caja de opciones
el archivo de minado y oprimir el botón: "run interact", por ejemplo:
index_size_inv_full_dataset_v2_SIFT.r4l1000000o09.models
```

4° Celda

```
@interact_manual
def load_topicos(
    topics_path=[(file,path.join("/media/working/radiografias
    /minhashing/16
    /",file))
    for file in listdir("/media/working/radiografias/minhashing/16/")
    if file.endswith(".models")],
    centXimg='/media/working/radiografias/descriptores/16
    /dataset_cluster',
    descr_kp='/media/working/radiografias/descriptores/16
    /dataset16_ext67_SIFT.pickle',
    image_dir='/media/working/radiografias/imagenes/2A_images
    /dataset'):
    global topics
    global len_topics
    global images_descr
    global desc_kp
    global etiquetas
    global dir_out
    global dir_img
    global topics_lenght

    topics = open(topics_path, 'r')
    topics= topics.readlines()
    topics_lenght=[]
    for line in topics:
        centroides = line.strip().split()[1:]
        centroides = [ int(cent.split(':')[0]) for cent in centroides]
        topics_lenght.append(len(centroides))

    len_topics=len(topics)

    pickle_file = open(centXimg,'rb')
    images_descr = pickle.load(pickle_file)
    pickle_file.close()

    pickle_file = open(path.abspath(descr_kp), 'rb')
    args = pickle.load(pickle_file)
```

```
    desc_kp = pickle.load(pickle_file)
    pickle_file.close()

    dir_img=image_dir

    print('#Topics: '+ str(len_topics))
    print('Model name:',topics_path)
```

5° Celda

```
def indicesTopicos (centroides, images_descr,threshold=0.7,
    threshold2=0.7, binary=True):tam_cents = len(centroides)
    centroides_= centroides
    img_index = 0
    images = list()
    for idesc,descrp in enumerate(images_descr[0]):
        descp_ = dict([(ic,i)
        for i,ic in enumerate(descrp)])
        key_points= [desc_kp[idesc]['keypoints'][kp]
        for kp in [ descp_[ic] for ic in descrp]]
        keypoints_ = genKeyPoints(key_points)
        new_descrp=Counter()
        for i,curKey in enumerate(keypoints_):
            if descrp[i] in new_descrp:
                new_descrp[descrp[i]]+= np.int(curKey.size)+1
            else:
                new_descrp[descrp[i]] = np.int(curKey.size)+1
        if binary:
            new_descrp=Counter(new_descrp.keys())

        inter=centroides_ & new_descrp
        overlap=sum(inter.values())
        /min(sum(centroides_.values()),sum(new_descrp.values()))
        if overlap >= threshold and sum(inter.values()) >= threshold2:
            images.append((img_index,[ (descp_[ic],v) for ic,v
            in inter.items()],
            overlap))
        #Overlaping
        #max_ = round( min(tam_descp, tam_cents)*.30 )

        #Requiere un 30%
        #posc_index = 0
        #posc = list()
        #for cent in descrp:
        #    if cent in centroides:
        #        posc.append(posc_index)
        #    posc_index += 1
        #if len(posc) > max_ : #No puede ser >= porque max_

        puede ser cero
        #    images.append((img_index ,posc))
```

```
        #Añade una tupla, del indice de la imagen y los indices
        de los KP
        img_index += 1
    return images
```

6° Celda

Ver contenido de tópico (opcional)

```
Seleccionar el tópico para ver el contenido, id de cluster y pesos
```

7° Celda

```
@interact_manual
def inspect_topicos(Topics=[(f"{i} [size {z}]",i) for i,z in
zip(range(len_topics),
topics_lenght)], binary=True):
    global lista_imgs
    global num_topic
    global name_images

    num_topic= Topics
    centroides = topics[Topics].split()[1:]
    centroides = Counter(dict([ tuple(int(x) for x in
    cent.split(':')) for cent in centroides]))
    if binary:
        centroides = Counter(centroides.keys())
    for c,v in centroides.items():
        print(f"{c}:{v}")
    print('#Len topic: '+str(len(centroides)))
    print('#Mass topic: '+str(sum(centroides.values())))
```

8° Celda

Elegir tópico

```
Este código permite elegir el tópico, el número de imágnes dónde
buscar el código y el threshold a usar para considerar que un tópico
esta presente o no en la imagen (topicos con números pequeños usar
1.0)

Selecionar el tópico de la caja de opciones, poner el número
de imágenes al máximo, ajustar el threshold, si el archivo de minado
seleccionado incluye en su nombre size desmarcar el checkpoint
de binany; no moverle a la masa.
Oprimir el botón de "run interact".
```

9° Celda

```
@interact_manual
def choose_topicos(Topics=[(f"{i} [size {z}]",i) for i,z in

zip(range(len_topics),
topics_lenght)],
                   num_imgs= (0, int(len(images_descr)/1), 5),
                   threshold = (0.001, 1.0, 0.01),
                   mass = 0,
                   binary=True):
    global lista_imgs
    global num_topic
    global name_images

    num_topic= Topics
    centroides = topics[Topics].split()[1:]
    centroides = Counter(dict([ tuple(int(x) for x in cent.split(':'))
    for cent in centroides]))
    if binary:
        centroides = Counter(centroides.keys())
    lista_imgs = indicesTopicos (centroides, images_descr[0:num_imgs],
    threshold=threshold, threshold2=mass, binary=binary)

    imgs_index=[imgs[0] for imgs in lista_imgs]
    aux=0
    name_images=[]
    for ii,i in enumerate(imgs_index):
        imagen = path.abspath(dir_img + desc_kp[i]['name_img'])
        name_images.append((desc_kp[i]['name_img'],aux,i,
        lista_imgs[ii][1],lista_imgs[ii][2]))
        aux+=1

    print('#Images: '+str(len(lista_imgs)))
    print('#Len topic: '+str(len(centroides)))
    print('#Mass topic: '+str(sum(centroides.values())))
```

10° Celda

Visualizando ejemplos del tópico

```
Se muestran ejemplos de imágenes que contiene el tópico seleccionado
en la parte anterior
```

11° Celda

```
plt.rcParams["figure.figsize"] = (10,5)
plt.rcParams['figure.dpi'] = 160
```

```
3
4  @interact
5  def show_images_per_topic(img=[(aux,(img,aux,i,v,o)) for img,aux,
6  i,v,o in name_images],binary=True):
7      img,aux,i,v,o=img
8      if aux-5 < 0:
9          aux=5
10     if aux+5 > len(name_images):
11         aux=len(name_images)-5
12     min_=max(0,aux-5)
13     max_=min(len(name_images),aux+5)
14     fig, axs = plt.subplots(2,5)
15     for ii in range(min_,max_):
16         img,aux,i,vals,overlap=name_images[ii]
17         imagen = path.abspath(path.join(dir_img, img))
18         img = cv.imread(imagen,cv.IMREAD_GRAYSCALE)
19         key_points= [desc_kp[i]['keypoints'][kp] for kp,v in vals]
20         keypoints = genKeyPoints(key_points)
21         for j,curKey in enumerate(keypoints):
22             x=np.int(curKey.pt[0])
23             y=np.int(curKey.pt[1])
24             if binary:
25                 size=np.int(curKey.size)
26             else:
27                 size=vals[j][1]
28             img=cv.circle(img,(x,y),size,color=(255,0,0),
29             thickness=3, lineType=0, shift=0)
30
31         x=ii-min_
32         y=x%5
33         x=int(x/5)
34         axs[x,y].set_yticklabels([])
35         axs[x,y].set_xticklabels([])
36         axs[x,y].set_xlabel(f'{ii}, {overlap:0.3}, #{i}')
37         axs[x,y].imshow(img, cmap = 'gray') # Checar como cambiar
38         el color
```

12° Celda

Ver una imágen

```
1  Este código permite seleccionar una imágen y ver los tópicos
2  presentes en esa imágen.
3
4  Se escribe un número, se deteminar un threshold, no mover a la mass.
5  Sí se quiere que aparezca el número del tópico dejar seleccionado
6  text,
7  si el archivo de minado tiene size quitar la seleccion de binary,
```

```
oprimir "run interact"
```

13° Celda

```
@interact_manual
def choose_topicos(image=str(random.choice(range(1,len(images_descr))
)),
                   threshold = (0.01, 1.0, 0.01),
                   mass= 0,
                   text=True,binary=True):
    global lista_imgs
    global topics
    global name_images
    global images_descr
    cmap=plt.get_cmap("hsv")

    descrp=images_descr.iloc[int(image)][0]
    descp_ = dict([(ic,i) for i,ic in enumerate(descrp)])
    img=images_descr.iloc[int(image)][1]
    key_points= [desc_kp[int(image)]['keypoints'][kp]
    for kp in [ descp_[ic] for ic in descrp]]
    keypoints_ = genKeyPoints(key_points)
    new_descrp=Counter()
    for i,curKey in enumerate(keypoints_):
        if descrp[i] in new_descrp:
                new_descrp[descrp[i]]+= np.int(curKey.size)+1
        else:
                new_descrp[descrp[i]] = np.int(curKey.size)+1
    descrp = new_descrp
    if binary:
        descrp = Counter(descrp.keys())
    tam_descp = len(descrp)
    topics_in_image= []

    imagen = path.abspath(path.join(dir_img,img))
    img = cv.imread(imagen)
    gray=img
    keypoints=[]
    for itopic, topic in enumerate(range(len(topics))):
        color=tuple(int(c*256) for c in cmap(topic/len(topics))[:3])
        centroides = topics[topic].split()[1:]
        centroides = Counter(dict([ tuple(int(x) for x in

        cent.split(':'))
        for cent in centroides]))
        if binary:
            centroides = Counter(centroides.keys())

        inter=centroides & descrp
```

```
46          overlap=sum(inter.values())/min(sum(centroides.values()),
47          sum(descrp.values()))
48
49          if overlap >= threshold and sum(inter.values()) >= mass:
50              key_points= [(desc_kp[int(image)]['keypoints'][kp],v)
51              for kp,
52              v in [ (descp_[ic],v) for ic,v in inter.items()]]
53              keypoints_ = genKeyPoints([x for x,y in key_points])
54              for i,curKey in enumerate(keypoints_):
55                  x=np.int(curKey.pt[0])
56                  y=np.int(curKey.pt[1])
57                  if binary:
58                      size = np.int(curKey.size)
59                  else:
60                      size =key_points[i][1]
61                  gray=cv.circle(gray,(x,y),size,color=color,
62                  thickness=1, lineType=0, shift=0)
63                  if text:
64                      gray=cv.putText(gray,f"{topic}",(x,y),
65                      cv.FONT_HERSHEY_SIMPLEX ,0.3,color,1, cv.LINE_AA)
66              topics_in_image.append((itopic,overlap,sum(inter.values()),
67              sum(centroides.values()),sum(descrp.values())))
68
69      plt.imshow(gray, cmap=cmap)
70      topics_in_image = sorted(topics_in_image, key=lambda tup: tup[1])
71      for i,o,inter,cen,im in topics_in_image:
72          print(f"#{i}: {o:0.3} = {inter}/min({cen},{im})")
```

Bibliografía

[1] C. Sanchez, (15 de febrero de 2021). *Tipos de aprendizaje en la Inteligencia Artificial*. edsrobotics. Recuperado el 07 de Octubre de 2021 de https://www.edsrobotics.com/blog/tipos-aprendizaje-inteligencia-artificial/.

[2] S. Jain, A. Smit, S. Q. Truong, C. D. Nguyen, M.-T. Huynh, M. Jain, V. A. Young, A. Y. Ng, M. P. Lungren, and P. Rajpurkar, "Visualchexbert: addressing the discrepancy between radiology report labels and image labels," in *Proceedings of the Conference on Health, Inference, and Learning*, pp. 105–115, 2021.

[3] A. Bhargava and A. Bansal, "Novel coronavirus (covid-19) diagnosis using computer vision and artificial intelligence techniques: a review," *Multimedia tools and applications*, vol. 80, no. 13, pp. 19931–19946, 2021.

[4] J. Gunraj, (13 de septiembre de 2020). *COVIDx CT*. kaggle. Recuperado el 15 de Octubre de 2021 de https://www.kaggle.com/hgunraj/covidxct.

[5] K. Zhang, X. Liu, J. Shen, Z. Li, Y. Sang, X. Wu, Y. Zha, W. Liang, C. Wang, K. Wang, *et al.*, "Clinically applicable ai system for accurate diagnosis, quan-

titative measurements, and prognosis of covid-19 pneumonia using computed tomography," *Cell*, vol. 181, no. 6, pp. 1423–1433, 2020.

[6] P. An, S. Xu, S. Harmon, E. Turkbey, T. Sanford, A. Amalou, M. Kassin, N. Varble, M. Blain, V. Anderson, *et al.*, "Ct images in covid-19," *Cancer Imag. Arch*, 2020.

[7] S. A. Harmon, T. H. Sanford, S. Xu, E. B. Turkbey, H. Roth, Z. Xu, D. Yang, A. Myronenko, V. Anderson, A. Amalou, *et al.*, "Artificial intelligence for the detection of covid-19 pneumonia on chest ct using multinational datasets," *Nature communications*, vol. 11, no. 1, pp. 1–7, 2020.

[8] K. Clark, B. Vendt, K. Smith, J. Freymann, J. Kirby, P. Koppel, S. Moore, S. Phillips, D. Maffitt, M. Pringle, *et al.*, "The cancer imaging archive (tcia): maintaining and operating a public information repository," *Journal of digital imaging*, vol. 26, no. 6, pp. 1045–1057, 2013.

[9] M. Jun, G. Cheng, W. Yixin, A. Xingle, G. Jiantao, Y. Ziqi, and H. Jian, "Covid-19 ct lung and infection segmentation dataset (version verson 1.0)[data set]. zenodo," 2020.

[10] S. G. Armato III, G. McLennan, L. Bidaut, M. F. McNitt-Gray, C. R. Meyer, A. P. Reeves, B. Zhao, D. R. Aberle, C. I. Henschke, E. A. Hoffman, *et al.*, "The lung image database consortium (lidc) and image database resource initiative (idri): a completed reference database of lung nodules on ct scans," *Medical physics*, vol. 38, no. 2, pp. 915–931, 2011.

[11] M. Rahimzadeh, A. Attar, and S. M. Sakhaei, "A fully automated deep learning-based network for detecting covid-19 from a new and large lung ct scan dataset," *Biomedical Signal Processing and Control*, vol. 68, p. 102588, 2021.

[12] W. Ning, S. Lei, J. Yang, Y. Cao, P. Jiang, Q. Yang, J. Zhang, X. Wang, F. Chen, Z. Geng, *et al.*, "Open resource of clinical data from patients with pneumonia for the prediction of covid-19 outcomes via deep learning," *Nature biomedical engineering*, vol. 4, no. 12, pp. 1197–1207, 2020.

[13] S. Morozov, A. Andreychenko, N. Pavlov, A. Vladzymyrskyy, N. Ledikhova, V. Gombolevskiy, I. A. Blokhin, P. Gelezhe, A. Gonchar, and V. Y. Chernina, "Mosmeddata: Chest ct scans with covid-19 related findings dataset," *arXiv preprint arXiv:2005.06465*, 2020.

[14] S. T. Oliva and P. Pérez-Sust, "Sistema visual: la percepción del mundo que nos rodea," *Offarm: farmacia y sociedad*, vol. 27, no. 6, pp. 98–102, 2008.

[15] R. López, (s.f.). *Libro online de IAAR*. GitHub. Recuperado el 25 de octubre de 2021 de https://iaarbook.github.io/.

[16] *Página de OpenCV*. (s.f.) OpenCV. https://opencv.org/.

[17] *Introduction to SIFT (Scale-Invariant Feature Transform)*. (14 de julio de 2022) OpenCV. https://docs.opencv.org/4.x/da/df5/tutorial$_p y_s i f t_i$ntro.html.

[18] D. Lowe, *Charla de algoritmo, Transformación de características invariantes de escala SIFT*. (04 de febrero de 2019). [video] bilibili. https://www.bilibili.com/video/av42629442?from=searchseid=12347294699672568781.

[19] E. Alegre, M. Pajares, and A. de la Escalera Hueso, *Conceptos y métodos en visión por computador*. s.l., 2016.

[20] *Introduction to SIFT (Scale-Invariant Feature Transform)*. (14 de Julio 2022) Scale-space Extrema Detection. [Imagen] OpenCV. https://docs.opencv.org/4.x/da/df5/tutorial$_p y_s i f t_i$ntro.html.

[21] M. Murga Bravo and C. Villca, "Algoritmo sift para la detección de imágenes coincidentes," *Revista de Investigación Estudiantil Iluminate*, p. 36, 2017.

[22] H. Huang, W. Guo, and Y. Zhang, "Detection of copy-move forgery in digital images using sift algorithm," in *2008 IEEE Pacific-Asia Workshop on Computational Intelligence and Industrial Application*, vol. 2, pp. 272–276, IEEE, 2008.

[23] X. Hu, Y. Tang, and Z. Zhang, "Video object matching based on sift algorithm," in *2008 International Conference on Neural Networks and Signal Processing*, pp. 412–415, IEEE, 2008.

[24] D. Sargent, C.-I. Chen, C.-M. Tsai, Y.-F. Wang, and D. Koppel, "Feature detector and descriptor for medical images," in *Medical Imaging 2009: Image Processing*, vol. 7259, p. 72592Z, International Society for Optics and Photonics, 2009.

[25] J.-H. Kim and I.-H. Jang, "Correction of rotated region in medical images using sift features," *Journal of Korea Multimedia Society*, vol. 18, no. 1, pp. 17–24, 2015.

[26] L.-J. Zhi, S.-M. Zhang, D.-Z. Zhao, H. Zhao, and S.-k. Lin, "Medical image retrieval using sift feature," in *2009 2nd International Congress on Image and Signal Processing*, pp. 1–4, IEEE, 2009.

[27] scikit-learn developers. (2007-2022) *Clustering* scikit learn. https://scikit-learn.org/stable/modules/clustering.html.

[28] *K-medias*. (s.f.). Wikipedia. https://es.wikipedia.org/wiki/K-medias.

[29] Jeffares, A. (19 de Noviembre de 2019) *K-means: A Complete Introduction.* [Figura]. Towards Data Science. https://towardsdatascience.com/k-means-a-complete-introduction-1702af9cd8c.

[30] Q. Deng and G. Mei, "Combining self-organizing map and k-means clustering for detecting fraudulent financial statements," in *2009 IEEE International Conference on Granular Computing*, pp. 126–131, IEEE, 2009.

[31] A. Muthukumar and S. Kannan, "Finger knuckle print recognition with sift and k-means algorithm," *ICTACT Journal on Image and Video Processing*, vol. 3, no. 03, p. 583, 2013.

[32] B. Ruf, E. Kokiopoulou, and M. Detyniecki, "Mobile museum guide based on fast sift recognition," in *International Workshop on Adaptive Multimedia Retrieval*, pp. 170–183, Springer, 2008.

[33] G. Fuentes-Pineda and I. V. Meza-Ruiz, "Topic discovery in massive text corpora based on min-hashing," *Expert Systems with Applications*, vol. 136, pp. 62–72, 2019.

[34] López, A. (2021). *Minado de tópicos usando min-hashing en tesis UNAM y su visualización.* [Tesis de licenciatura]. UNAM.

[35] A. Beiser, *Concepts of Modern Physics (SIE).* McGraw-Hill Education, 1963.

[36] Fuentes, G. (22 de Febrero de 2013). *Sampled-MinHashing.* GitHub. Recuperado 27 de Octubre de 2021 de https://github.com/gibranfp/Sampled-MinHashing/blob/master/README.md.

[37] "García, r. (2020). *minIA.* github. recuperado 16 de mayo de 2021 de https://github.com/pluginrichi/ minia."