Detección de anomalías pulmonares producidas por COVID-19, a través de tomografías computarizadas axiales con un enfoque de aprendizaje de maquina

Flores-Silva P.

Abstract—Se desarrolló un sistema detector de tres anomalías desarrolladas por pacientes con COVID-19, a saber, consolidación pulmonar, derrame pleural y opacificación del vidrio en tomografías computarizadas. Con resultados bastante aceptables. Esto fue posible gracias a la segmentación y análisis de las propiedades de textura de las tres anomalías y zonas de pulmón sano presentes en cien tomografías computarizadas axiales de pulmón. El método de evaluación se realiza de forma cualitativa debido a la dificultad de la definición de una función de error. Finalmente, durante el desarrollo se encontraron diversos puntos que no fueron estudiados, sin embargo, se discuten algunos métodos para el mejoramiento de la obtención de datos, análisis y clasificación.
NOTA: El código desarrollado para este artículo así como las imágenes resultantes puede encontrarse en la siguiente dirección web: https://github.com/Pedri0/COVID-19-Lung-Images-Classifier.

Index Terms—Visión computacional, Imágenes, EigenFaces, Detector de Rostros, PCA, SVD, Linear Algebra.

1 Introducción

En Diciembre del año 2019 el gobierno chino dio a conocer la existencia de una nueva cepa del coronavirus (CoV), que afecta a humanos, en el poblado de Wuhan. La Organización Mundial de la Salud nombró este nuevo virus como **SARS-CoV-2** debido a su semejanza con el virus **SARS-CoV** [1] cuya aparición data a principios del 2003.
Debido a su reciente aparición, el origen de este virus, así como los mecanismos asociados a su patogenicidad no son claros aún. A pesar de ello se ha encontrado, mediante estudios genómicos, la semejanza con el virus **SARS-CoV** y varios coronavirus que afectan a murciélagos [1].
La enfermedad desarrollada al contraer este nuevo virus se conoce como **COVID-19** que es la abreviación de *coronavirus disease 2019*. Los síntomas comunes del COVID-19 son fiebre, tos seca, fatiga, perdida del olfato y dolor de cuerpo. En algunos pacientes infectados los síntomas pueden ser más severos como fiebre alta, tos severa, dificultad para respirar, los cuales a menudo son síntomas de neumonía. Mientras que algunos otros infectados no presentan ningún síntoma [2].

El nuevo coronavirus ha llamado la atención e incluso actualmente ha provocado preocupación entre la comunidad científica y la población en general, debido a su rápida propagación, su alta tasa de contagio y la relativa alta tasa de mortalidad. A pesar de que se sabe que estos factores son elevados, no se conoce con exactitud dichos valores [2].
Hasta el día 06.06.20 la Organización Mundial de la Salud reporta $6,663,304$ casos confirmados y $392,802$ muertes en el mundo por COVID-19. Esta cifra puede considerarse muy baja comparada con la población total, sin embargo, debe considerarse que aquellas cifras pueden ser mucho menores a las reales debido a que la cantidad de pruebas disponibles en el mundo no son suficientes para estudiar a cada ser humano que habita en el planeta. Por otro lado, en la república mexicana han surgido rumores de que la cifra de muertos por COVID-19 se ha alterado a la baja siendo reportadas como muertes por neumonía atípica [3].

De acuerdo a [4] los síntomas severos producidos, tales como dificultad para respirar, dolor o presión en el pecho, perdida del habla o movimiento están altamente relacionados con el desarrollo de complicaciones pulmonares como neumonía, síndrome de distrés respiratorio agudo, septicemia y daños duraderos a los pulmones.
Cuando se presenta **Neumonía**, los pulmones se llenan de líquido y se inflaman provocando dificultad para respirar. Para algunos pacientes, los problemas para respirar llegan a ser tan severos de tal forma que se requiere de la asistencia con un ventilador u oxígeno. La neumonía causada por COVID-19 tiende a presentarse en ambos pulmones y puede llegar a ser severa [5]. El progreso de la neumonía produce el llenado de los alveolos de líquido que termina escapando de los vasos sanguíneos, eventualmente, la dificultad para respirar aumenta y puede conducir al síndrome de dificultad respiratoria aguda (SDRA), una forma de insuficiencia pulmonar. Los pacientes con SDRA a menudo no pueden respirar por sí solos y pueden necesitar de ventilación para ayudar a circular el oxígeno en el cuerpo [5].

• *P. Flores-Silva estudiante del PCIC en la UNAM.*
E-mail: flosipan@ciecias.unam.mx

Otra complicación debida al COVID-19 es la septicemia, que ocurre cuando una infección en un determinado órgano se propaga a través de la sangre [5].

La obtención de tomografías computarizadas (CT) de los pulmones de un paciente de COVID-19 con síntomas severos, puede revelar al menos tres anomalías en el pulmón. La primera de ellas se conoce como **Derrame pleural** que es la acumulación de exceso de líquido entre las capas de la pleura fuera de los pulmones. La pleura son membranas delgadas que recubren los pulmones y el interior de la cavidad torácica y actúan para lubricar y facilitar la respiración. Normalmente, una pequeña cantidad de líquido está presente en la pleura [6]. La segunda de ella se conoce como **consolidación pulmonar** y ocurre cuando el aire que generalmente llena las vías respiratorias en los pulmones se reemplaza con un sólido o líquido, se diferencia de la primera en la zona que ocupa el líquido [7]. La última se conoce como **opacificación del vidrio** es un término descriptivo que se refiere a un área de mayor atenuación en el pulmón en la tomografía computarizada con marcas bronquiales y vasculares preservadas. Es un signo específico con una etiología amplia que incluye infección, enfermedad intersticial crónica y enfermedad alveolar aguda [8].

En el presente artículo se trabajará con tomografías computarizadas axiales de pacientes con COVID-19 con la finalidad de entrenar una maquina de vectores de soporte de tal forma que una vez entrenada, dada una tomografía detecte y segmente las tres anomalías, anteriormente descritas, presentes en dichas tomografías. Debido a que el contenido teórico es amplio se decidió dividir la introducción y el desarrollo experimental en dos partes, siendo la primera de ellas lo relacionado a la obtención de datos de entrenamiento, mientras que la segunda tratará sobre el método empleado para la clasificación.

1.1 Parte I: Datos de entrenamiento

1.1.1 La escala HU y su relación con las tomografías computarizadas

En la actualidad las imágenes médicas han tomado gran relevancia en el ámbito médico, ya que gracias a ellas es posible detectar patologías o anomalías en diferentes partes del cuerpo, además son útiles para soportar decisiones, diagnósticos y terapia médicos.
Existe una gran variedad de imágenes médicas que se obtienen empleando distintos fenómenos físicos o químicos. De acuerdo a lo que se quiera estudiar se suelen elegir alguno de estos métodos.
Para el caso de tomografías computarizadas, que se basa en la reconstrucción de la imagen a partir de la detección de rayos X dispersados, suele usarse una medida cuantitativa de la *densidad de radio*, conocida como **unidades de Hounsfield**, para la interpretación de las mismas. El coeficiente de absorción de la radiación dentro de un tejido se utiliza durante la reconstrucción por TC para producir una imagen en escala de grises. La densidad física del tejido es proporcional a la absorción del haz de rayos X. La unidad de Hounsfield, también conocida como la unidad de CT, se calcula a partir de una transformación lineal del coeficiente de atenuación lineal de referencia del haz de rayos X, donde el agua se define arbitrariamente como cero unidades de Hounsfield (HU) y el aire se define como -1000 HU . La transformación lineal produce una escala Hounsfield que se muestra como tonos grises. El tejido más denso, con mayor absorción de rayos X, tiene valores positivos y parece brillante, mientras que el tejido menos denso, con menos absorción de rayos X, tiene valores negativos y parece oscuro [9]. Bajo condiciones ambientales de temperatura a $0°C$ y una presión de 10^5 Pa, el aire tiene valor de -1000 HU, mientras que para huesos muy densos se encuentran valores aproximadamente de $\pm$2000 HU y para metales arriba 3000 HU. Tomando en cuenta estos valores y el objeto que se quiere estudiar (cuerpo humano), generalmente las imágenes de CT son imágenes a 12 bits [10].
Existen, al menos, dos formatos preferidos para almacenar imágenes médicas, éstos son el formato DICOM y el formato NIFTI. El análisis se hará sobre imágenes en este último del cual existen dos formatos de almacenamiento: single y dual [11].
El formato dual almacena información en un par de archivos: una cabecera (.hdr) que contiene los metadatos y un archivo de datos (.img) que contiene la información de la imagen.
El formato single almacena los datos en un solo archivo (.nii) el cual contiene la información de la cabecera seguida de los datos de imagen.
En este trabajo los datos de las cabeceras no fueron tomados en cuenta debido a que no contenían información útil. Sin embargo, estos datos pueden llegar a ser relevantes. Un estudio detallado del formato NIFTI puede encontrarse en [12].

1.1.2 Texturas

La textura es un concepto vago, regularmente atribuido a la percepción humana. Cada persona tiene su propia interpretación de la naturaleza de la textura. No existe una definición matemática de la textura [13]. Por lo que si nos referimos al diccionario *Oxford Concise English Dictionary* encontramos una definición de textura como (traducida del ingles) *Textura: sust. disposición de hilos etc. en tejido textil. sensación característica debido a esto; disposición de pequeñas partes constituyentes, estructura percibida (de piel, roca, tierra, tejido orgánico, obra literaria, etc.); representación de estructura y detalle de objetos en el arte*. Si cambiamos hilos por píxeles, la definición podría aplicarse a imágenes. Esencialmente, la textura puede ser lo que nosotros mismos definimos como puede ser (*texture can be what we define it to be*) [13]. Como definición podemos considerar a la textura como definida por la base de datos de imágenes que los investigadores utilizan para probar sus algoritmos. Muchos investigadores de texturas han utilizado una base de datos de imágenes de texturas producidas por artistas y diseñadores [13], un ejemplo de ello es la base de datos de Brodatz [14]. De una manera alternativa se define la textura como una cantidad para la cual los algoritmos de extracción de texturas proporcionan resultados significativos. Un extracto interesante de Karru [15] postula: *La respuesta a la pregunta ¿Hay alguna textura en la imagen? depende no solamente de la imagen estudiada, si no que también del objetivo para el que se utiliza la textura de la*

imagen y los rasgos de textura que se extraen de la misma..
Debido a que la textura no tiene una definición única, existen muchas formas de describirla, estas formas pueden dividirse en básicamente tres grandes ramas: *Estructurales*, *Estadísticas*, *Combinaciones*. Como ya se mencionó, este trabajo se centrará únicamente en uno los métodos estadísticos.

1.1.3 Enfoque Estadístico: Matriz de Coocurrencia

El enfoque estadístico más famoso es la *matriz de coocurrencia* o GLCM por sus siglas en inglés (Gray Level Coocurrence Matrix). Este enfoque fue el primer método para describir y clasificar la textura de imágenes propuesto por Haralick [13]. La matriz de coocurrencia contiene elementos que son cuentas del número de pares de píxeles para niveles específicos de gris cuando están separados por alguna distancia y con alguna inclinación relativa. Para dos niveles de gris b_1 y b_2, la matriz de coocurrencia C se define como:

$$C_{b_1,b_2} = \sum_{x=1}^{N}\sum_{y=1}^{N}(P_{x,y} = b_1) \wedge (P_{x',y'} = b_2) \quad (1)$$

donde $P_{x,y}$ es la entrada en la posición (x, y). x', y' es el offset definido por la distancia d y la inclinación θ dada por:

$$x' = x + dcos(\theta) \quad (2)$$
$$y' = y + dsen(\theta) \quad (3)$$

donde $\theta \in (0, 2\pi)$.
Cuando se aplica la ecuación (1) a una imagen, se obtiene una matriz cuadrada, simétrica, cuyas dimensiones corresponden a la cantidad de niveles de gris en la imagen.
Esencialmente, la matriz de coocurencia mide las relaciones espaciales entre los niveles de gris. Para generar resultados de rápido cálculo, comúnmente, se reduce el número de los niveles de gris a través del escalado del brillo de toda la imagen, reduciendo así las dimensiones de la matriz de coocurrencia, pero esto reduce la habilidad discriminatoria [13].
Para describir las propiedades de la textura Haralick y sus colaboradores definieron un conjunto de 28 características que pueden obtenerse a partir de la matriz de coocurrencia. Posteriormente, Conners y Harlow encontraron que de esas 28 características solo cinco de ellas son suficientes para describir la textura [16]. Éstas cinco son:

- Energía: provee información de la homogeneidad de la imagen; tiene valores pequeños cuando los pares de niveles de gris son bastante similares y valores altos en caso contrario. Definida matemáticamente como
$$\sum_{i=1}^{G}\sum_{j=1}^{G}P(i,j|d,\theta)^2$$
donde esta notación es análoga con (1) y G son los niveles de gris.
- Entropía: mide el desorden de la GLCM. Definida matemáticamente como
$$-\sum_{i=1}^{G}\sum_{j=1}^{G}P(i,j|d,\theta)log_2(P(i,j|d,\theta))$$
- Correlación: mide la dependencia lineal del nivel de gris entre píxeles (relativo a cada uno de ellos) en las posiciones especificadas; tiene valores altos cuando los valores están distribuidos uniformemente en el GLCM y valores bajos en caso contrario. Definida matemáticamente como
$$\sum_{i=1}^{G}\sum_{j=1}^{G}P(i,j|d,\theta)\left[\frac{(i-\mu_i)(j-\mu_j)}{\sqrt{\sigma_i^2\sigma_j^2}}\right]$$
- Homogeneidad local o momento de diferencia inversa: Es alto cuando se encuentra los mismos pares de píxeles. Definida matemáticamente como
$$\sum_{i=1}^{G}\sum_{j=1}^{G}\frac{P(i,j|d,\theta)}{1+(i-j)^2}$$
- Inercia o Contraste: Mide las variaciones locales presentes en la imagen. Definida matemáticamente como
$$\sum_{i=1}^{G}\sum_{j=1}^{G}(i-j)^2P(i,j|d,\theta)$$

Una de las grandes desventajas del enfoque GLCM es la alta dimensionalidad de la matriz y la alta correlación de las características de Haralick.
El método GLCM ofrece resultados buenos para situaciones simples donde las texturas son visualmente separables de una manera sencilla. Además, el algoritmo GLCM es fácil de implementar y ha sido empleado en una gran variedad de aplicaciones con buenos resultados. Sin embargo, debido a su gran dimensionalidad, la GLCM es muy sensible al tamaño de las muestras de textura que son procesadas [16].

1.2 Parte II: Clasificación

1.2.1 Filtro Sobel u Operador Sobel

Una imagen en escala de grises puede ser representada como una función $I(x, y)$ que evalúa la intensidad de un píxel en una posición (x, y) del píxel. Muchas de las tareas de visión computacional requieren, como entrada, del calculo de las derivadas parciales de la imagen, esto es, $\partial I(x, y)/\partial x$, $\partial I(x, y)/\partial y$, en las direcciones x e y. Estas derivadas son regularmente abreviadas como $I_x(x, y)$ e $I_y(x, y)$. El operador Sobel es un método popular para calcular dichas derivadas [17]. La figura 1 muestra el efecto de aplicar el operador Sobel a la imagen original de Lena. En ambas imágenes los píxeles blancos representan bordes.

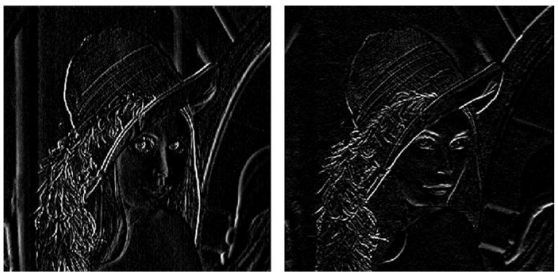

Fig. 1. Izquierda: $I_x(x, y)$. Derecha: $I_y(x, y)$. Obtenidas al aplicar el operador Sobel. Tomada de [17]

La figura 2 muestra que el operador Sobel también tiene la capacidad de extraer los bordes de una imagen. Para ello

es posible calcular la norma de la derivada X e Y de cada píxel, es decir,

$$\sqrt{\left(\frac{\partial I(x,y)}{\partial x}\right)^2 + \left(\frac{\partial I(x,y)}{\partial y}\right)^2} \tag{4}$$

Para utilizar este método de detección de bordes, el programa debe calcular las derivadas x e y de toda la imagen, lo que resulta en dos imágenes intermedias (fig 1). Posteriormente deben ser combinadas usando la norma para obtener la imagen resultante (fig 2).

Fig. 2. Combinación de $I_x(x,y)$ e $I_y(x,y)$ a través de la norma l_2. Tomada de [17]

Las derivadas pueden encontrarse fácilmente mediante las matrices de coeficientes definidas como,

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \tag{5}$$

$$G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 \end{bmatrix} \tag{6}$$

Al aplicarse sobre la imagen de entrada.
Imágenes ruidosas crean bordes falsos sobre la imagen resultante, por lo que un filtro Gausiano puede ser aplicado para disminuir el ruido y evitar este problema [17].

1.2.2 Maquinas de Soporte Vectorial (SVM)

Si se tienen L puntos de entrenamiento, donde cada entrada $\vec{x}_i$ tiene D atributos y pertenece a una de las dos clases $y_i = -1$ o $+1$ los datos de entrenamiento son de la forma:

$$\{\vec{x}_i, y_i\} \tag{7}$$

donde $i = 1, 2, ...L$, $y \in \{-1, 1\}$ y $x \in \mathbb{R}^D$.
Se asume que los datos son linealmente separables, es decir, es posible dibujar un hyperplano en las gráficas de $x_1, x_2, ..., x_D$ que separe las dos clases. Este hiperplano puede ser descrito por $\vec{w} \cdot \vec{x} + b = 0$. Donde $\vec{w}$ es el vector normal al hiperplano y $\frac{b}{||\vec{w}||}$ es la distancia perpendicular del hiperplano al origen.
Los Vectores de soporte son las muestras más cercanas al hiperplano de separación y el objetivo de las SVM es orientar este hiperplano de tal manera que esté lo más lejos posible de los miembros más cercanos de ambas clases. La figura 3 muestra muestra un ejemplo en dos dimensiones.

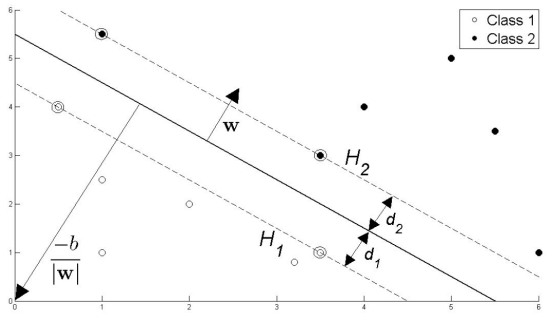

Fig. 3. Hiperplano divisor de dos clases linealmente separables. Tomada de [18]

La figura 3 prueba que implementar una SVM se reduce a encontrar las variables $\vec{w}$ y b tales que nuestro conjunto de entrenamiento pueda ser descrito por:

$$\vec{x_i} \cdot \vec{w} + b \geq +1 \quad siy_i = +1 \tag{8}$$

$$\vec{x_i} \cdot \vec{w} + b \leq -1 \quad siy_i = -1 \tag{9}$$

Las dos ecuaciones anteriores se reducen a:

$$y_i(\vec{x_i} \cdot \vec{w} + b) - 1 \geq 0 \tag{10}$$

Si se consideran los puntos más cercanos al hiperplano (vectores soporte, son los círculos en la figura 3). Entonces, los dos planos H_1 y H_2 y los puntos pueden ser descritos por las ecuaciones:

$$\vec{x_i} \cdot \vec{w} + b = +1 \quad H_1 \tag{11}$$

$$\vec{x_i} \cdot \vec{w} + b = -1 \quad H_2 \tag{12}$$

De la figura definimos d_1 y d_2 como la distancia del plano H_1 y H_2 al hiperplano separador, donde el hiperplano es equidistante a H_1 y H_2, es decir $d_1 = d_2$. A esta cantidad se le conoce como margen SVM. Con el fin de orientar el hiperplano de tal forma que se encuentre los mas lejos posible de los vectores de soporte, es necesario maximizar el margen. De la figura, es fácil demostrar que el margen es igual a $\frac{1}{||\vec{w}||}$, por lo que la tarea consiste en maximizar este ultimo valor bajo la constricción dada por la ecuación (10) o equivalentemente, minimizar el valor de $||\vec{w}||$ bajo las condiciones de (10). A partir de la minimización de $||\vec{w}||$ es posible predecir el valor óptimo de b. La deducción de todas las ecuaciones se encuentra en [18]. En este trabajo no se incluyen ya que no es el objetivo.
Las SVM pueden extenderse a modo que se consideren más de dos clases. Esta extensión es muy natural; únicamente se ajusta un plano considerando la existencia de únicamente dos clases: Una clase separada contra el resto. Este proceso se sigue hasta haber analizado por separado cada clase.

1.2.3 Superpíxeles

Muchos de los algoritmos en visión computacional usan una malla simétrica o cuadrícula de píxeles como representación adyacente de una imagen. Por ejemplo, los modelos estocásticos de imágenes, tales como los campos aleatorios de Markov [19]. Regularmente la simetría de dichas mallas se ajusta a una simetría cuadrangular sin cambios en sus dimensiones.
Sin embargo, una cuadricula de imágenes no es una representación natural de las escenas del mundo real, es más

bien un artefacto con el cual es posible procesar una imagen de una manera muy sencilla. Debido a esto se busca una forma más natural y eficiente para representar objetos significativos dentro de la imagen y éstos, al igual que con las cuadrículas, deben ser agrupadas mediante algún proceso que las identifique correctamente. A estos objetos representativos los llamamos superpíxeles.
De una manera más compacta definimos un superpíxel como un grupo de píxeles similares en color y otras propiedades de bajo nivel, donde este grupo es un conjunto de una sola partición [20]. Estos superpíxeles deben tener las siguientes propiedades [19], [20]:

- El cálculo de los superpíxeles debe ser computacionalmente eficiente de tal forma que reduzca la complejidad de las imágenes de cientos de miles de píxeles a solo unos cientos de superpixeles.
- Deben ser eficientes en cuanto a la representación de la imagen.
- Deben ser perceptualmente representativos: cada elemento de un superpíxel debe ser parecido con los otros elementos del mismo conjunto. Esta similitud puede ser por color, textura, entre otros.
- El conjunto de superpíxeles debe ser compacto,completo y disjunto, es decir, debido a que los superpíxeles son el resultado de la segmentación de una imagen, cada píxel de la imagen debe estar contenido en un solo elemento del conjunto. Todos los superpíxeles sin translaparse (este hecho es obvio al considerar la constricción anterior) deben cubrir toda la imagen.
- La cantidad de superpíxeles generados debe ser ajustable por el usuario.
- Adherencia de frontera: los superpíxeles deben preservar las fronteras o bordes de la imagen. La definicion de frontera dependerá de la imagen.
- Conectividad: Los superpíxeles deben representar conjuntos conectados de píxeles.

Existen al menos 28 algoritmos para obtener los superpíxeles de una imagen dada que pueden dividirse en siete diferentes categorías según la metodología que se emplee para obtener los resultados deseados [20]. De una manera breve se explicara el algoritmo Simple Linear Iterative Clustering (SLIC) perteneciente a la categoría basada en clusterizacion.

1.2.4 Algoritmo SLIC

El algoritmo SLIC se basa en la técnica de clusterizacion ya que está inspirado en los algoritmos como k-means. Debido a que este tipo de algoritmos es iterativo es necesario realizar un post-procesado de la imagen con el fin de reforzar la conectividad de los superpíxeles.
Este algoritmo genera superpíxeles al clusterizar píxeles basados en la similitud y proximidad del plano de la imagen. Esto se realiza en el espacio (l, a, b, x, y) definido por el vector de color (l, a, b) del espacio de color CIELAB y el vector de posiciones (x, y).
El espacio de color CIELAB es usado normalmente para describir todos los colores que percibe el ojo humano. Los tres parámetros representan la luminosidad (l), la posición entre valores del rojo y verde (a) y la posición entre el amarillo y el azul (b). El algoritmo puede resumirse de la siguiente manera [21]:

- Inicializa los centros de los clusters $C_k = (l_k, a_k, b_k, x_k, y_k)^T$ mediante el muestreo de píxeles en pasos regulares S de la cuadrícula
- Perturba los centros de los cluster en una vecindad de $n \times n$, a la posición del gradiente mínimo
- De manera iterativa hasta cumplir cierta condición de paro: Para cada centro de cluster C_k asigna los píxeles mas similares de una vecindad cuadrada de tamaño $2S \times 2S$ al rededor de el centro del cluster usando una métrica previamente definida. Y redefine los centros de los clusters según lo calculado anteriormente, calcula un error residual definido como la distancia entre los centros anteriormente calculados y los nuevos

2 Procedimiento experimental y Resultados

El código desarrollado para este articulo consiste en cuatro libretas de Jupyter Python. Nombradas de la siguiente manera: **PreprocessingImagesAndGetData.ipynb**, **SegmentingLungs.ipynb**, **TestingData.ipynb** y **TrainWithLessSamples.ipynb**. Debido a que el código desarrollado puede llegar a ser largo para cada libreta y además dichas libretas, con documentación dentro de las mismas, se encuentran publicas en GitHub (ver liga en el apartado abstract), se procurara colocar la menor cantidad del mismo en el presente trabajo.
En las siguientes subsecciones se mencionará la libreta que contiene el código de la tarea a la que se está mencionando.
El conjunto de imágenes empleadas durante todo el desarrollo experimental puede encontrarse en http://medicalsegmentation.com/covid19/. En el apartado Download Data. Únicamente no fue considerado el archivo *CSV file connecting slice no. with SIRM case no.*.
Hasta el día 07.06.20 los datos siguen disponibles en la liga.

El conjunto de datos consiste de Cien tomografías axiales de entrenamiento de pacientes con COVID-19. Cien imágenes mascara donde se marcan al menos una de las tres anomalías previamente discutidas. Cabe aclarar que es posible relacionar ambas a través del índice de cada imagen, ya que siempre coinciden. Las imágenes mascara tienen valores del entre $[1, 3]$, donde 1 representa a la consolidación pulmonar, 2 corresponde a derrame pleural y 3 a la opacificación del vidrio.
Se adjuntan otras diez tomografías axiales como datos de prueba y no se cuentan con sus respectivas mascaras

2.1 Parte I: Datos de entrenamiento

La carga de los datos de cada archivo provisto a la libreta requiere de la instalacion del paquete NiBaBel (https://nipy.org/nibabel/) ya que las imágenes se encuentran en formato NIfTI. Acceder a los datos requiere el siguiente código

```
import nibabel as nib
dataTrain = nib.load('tr_im.nii.gz')
dataMaskTrain = nib.load('tr_mask.nii.gz')
```

Listing 1. script para leer imagenes en formato NIfTI.

Para ambos casos, la variable contiene cien imágenes de 512×512 píxeles en escala HU. Para acceder a alguna de las cien imágenes es necesario crear una nueva variable con la siguiente instrucción.

```
z = nombreVariable.get_fdata()[:,:,i]
```

Listing 2. script para acceder a la imagen i.

nombreVariable se refiere al nombre de la variable que fue designado para cargar los datos NIfTI. Nueve tomografías de entrenamiento se muestran en la figura 4.

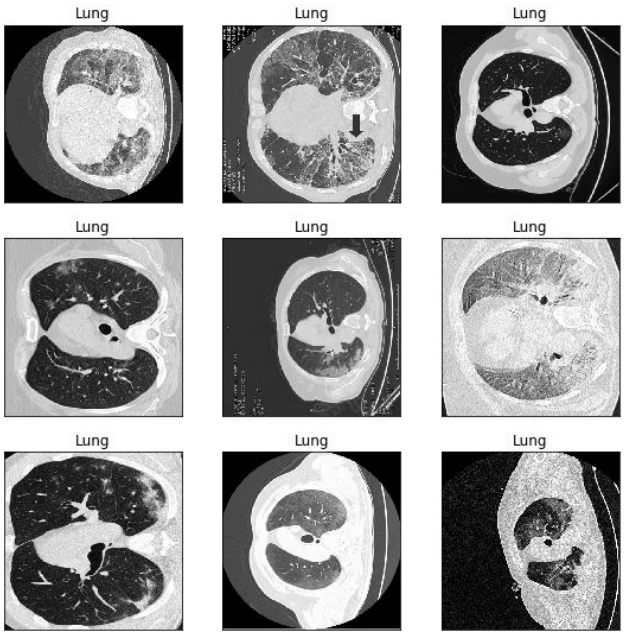

Fig. 4. Nueve tomografías o imágenes de entrenamiento

Note que cada imagen parece diferir en sus alrededores por lo que es posible que este conjunto de datos haya sido construido a partir de los datos obtenidos de diferentes escáneres.
Una imagen de entrenamiento y su respectiva imagen mascara se muestran en la figura 5.

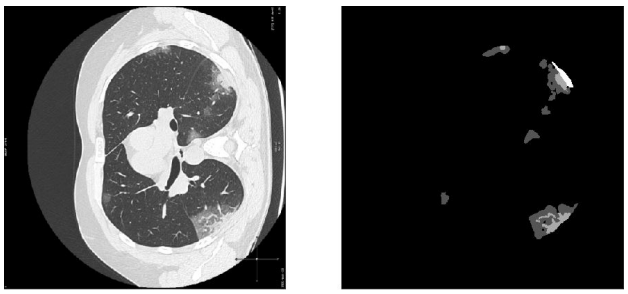

Fig. 5. Izquierda: Imagen de entrenamiento Derecha: mascara de la misma imagen de entrenamiento

Para la obtención de las texturas para cada una de las tres anomalías es necesario obtener las imágenes de segmentación de cada clase por separado. Para ello se requiere de una modificación de cada mascara dejando únicamente el valor de la clase y dividiendo sobre el mismo valor, con el fin de tener una mascara de una clase determinada con valor 1. Se encontró que la manipulación de los valores de la mascara, aun realizando una copia de la misma, modificaba la mascara original por lo que era necesario volver a cargar a memoria los datos de las mascaras cada vez que se realizaba alguna manipulación sobre las mismas. Una vez realizado lo anterior basta con multiplicar entrada a entrada la imagen de entrenamiento con su respectiva mascara con una clase dada para obtener la zona segmentada según la clase. Este proceso claramente se hace iterativamente para las cien imágenes de entrenamiento y a lo más 300 imágenes mascara para una clase dada. Se refiere a lo más 300 mascaras ya que no todas las mascaras contienen las tres anomalías. La figura 6 muestran nueve segmentaciones para la consolidación pulmonar, mientras que 7 muestra nueve segmentaciones para el derrame pleural y 8 muestra nueve segmentaciones para la opacificación del vidrio.

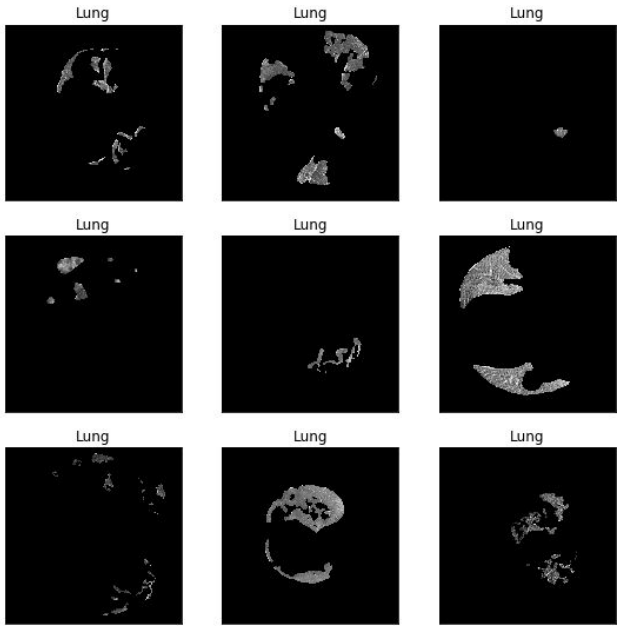

Fig. 6. Segmentaciones para la consolidación pulmonar

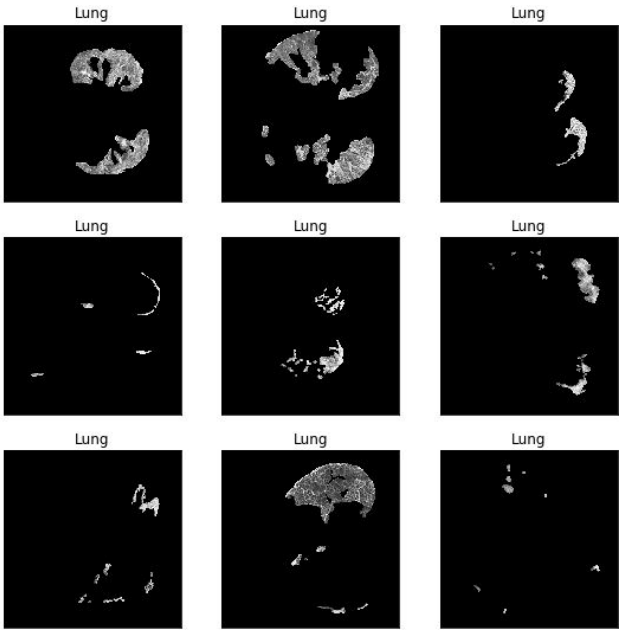

Fig. 7. segmentaciones para el derrame pleural

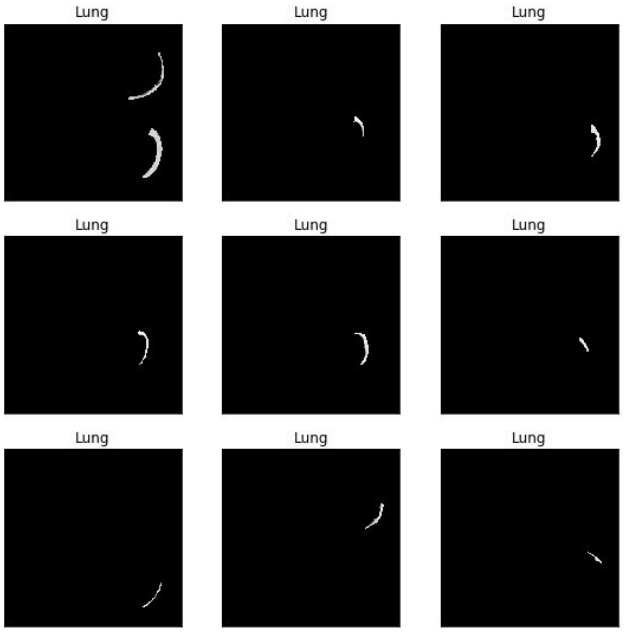

Fig. 8. Segmentaciones para la opacificación del vidrio

Un análisis rápido reveló que las tres anomalías no se superponen en las imágenes de entrenamiento.
Una vez obtenidas las imágenes se procedió a tratar de obtener las características de textura para cada una de ellas. Sin embargo debido a que la función ya implementada de Skimage greycomatrix requiere de una imagen de entrada del tipo $uint$. Como ya se mencionó una imagen de CT queda bien caracterizada con 12 bits, por lo que se procedió a convertir la imagen al tipo $uint16$. A pesar de ello el método no funcionó debido a un error de memoria, ya que el análisis de texturas ocupa, para este dataset al menos 32 GB (ver figura 9).

```
In [15]:
#instantiate the texture properties of each type (1,2,3) and preprocesing it using the function scale
#from scikit learn
type1 = getTextureProperties(listForType1)
type2 = getTextureProperties(listForType2)
type3 = getTextureProperties(listForType3)
#create dataframes from the texture properties
type1DataFrame = pd.DataFrame(type1)
type2DataFrame = pd.DataFrame(type2)
type3DataFrame = pd.DataFrame(type3)
#Add the classes to every dataFrame
type1DataFrame['Class'] = 1
type2DataFrame['Class'] = 2
type3DataFrame['Class'] = 3
#read the data from healthy Lungs
healthy = pd.read_csv('healthy.csv',delimiter=',',header=None)
#Add column class to healthy dataFrame
healthy['Class'] = 0
#create a list. With this list we can concatenate all dataframes
frames = [healthy, type1DataFrame, type2DataFrame, type3DataFrame]
#concatenate all dfs
data = pd.concat(frames)
#set X variable as the obtained texture data
X = data.iloc[:,0:6].to_numpy()
#set Y variable as the classes
Y = data['Class'].to_numpy()

---------------------------------------------------------------------------
MemoryError                               Traceback (most recent call last)
<ipython-input-15-2e26b7a8841d> in <module>
      1 #instantiate the texture properties of each type (1,2,3) and preprocesing it using the function scale
      2 #from scikit learn
----> 3 type1 = getTextureProperties(listForType1)
      4 type2 = getTextureProperties(listForType2)
      5 type3 = getTextureProperties(listForType3)

<ipython-input-14-52c60e2d9696> in getTextureProperties(maskedImages)
      4         GLCM = greycomatrix(maskedImages[i],distances=[1],angles=[0],symmetric=False,normed=True,levels=65535)
      5         GLCM = GLCM[1:,1:,:,:]
----> 6         contrast = greycoprops(GLCM,'contrast')
      7         dissimilarity = greycoprops(GLCM,'dissimilarity')
      8         homogeneity = greycoprops(GLCM,'homogeneity')

C:\ProgramData\Anaconda3\lib\site-packages\skimage\feature\texture.py in greycoprops(P, prop)
    222
    223     # normalize each GLCM
--> 224     P = P.astype(np.float64)
    225     glcm_sums = np.apply_over_axes(np.sum, P, axes=(0, 1))
    226     glcm_sums[glcm_sums == 0] = 1

MemoryError: Unable to allocate 32.0 GiB for an array with shape (65534, 65534, 1, 1) and data type float64
```

Fig. 9. Error de memoria al analizar las texturas de todas las imágenes del tipo $uint16$

Por esta razón se optó por desarrollar una función que transforma una imagen dada a un tipo dado, esta función es una transformación de la forma:

$$I = \left(\frac{MaxI-MinI}{MaxE-MinE}\right)(E) + \left(MaxI - \left(\frac{MaxI-MinI}{MaxE-MinE}\right)(MaxE)\right) \quad (13)$$

Donde I es la nueva imagen, $MaxI$ es el valor máximo de escala de gris de la imagen I, $MinI$ es el valor mínimo de escala de gris de la imagen I. E es la imagen de entrada, $MaxE$ es el valor máximo de escala de gris de la imagen E, $MinE$ es el valor mínimo de escala de gris de la imagen E. Esta transformación garantiza un orden en los valores originales en escala HU. A pesar de ello si es posible que exista perdida de información debido a una transformación a menor escala. Este caso sucede al transformar la imagen a $uint8$, sin embargo, con este método es posible obtener las características de textura.
Al analizar la textura de cada segmentación obtenemos tres documentos llamados **type1.csv**, **type2.csv** y **type3.csv** que corresponden a las características de textura de consolidación pulmonar, derrame pleural y opacificación del vidrio respectivamente.
Debido a que en las tomografías de prueba se buscarán las anomalías en la imagen, es necesario distinguir entre éstas tres y zonas del pulmón que no corresponden a ninguna de dichas afecciones que consideraremos como pulmón sano. Para obtener los datos de pulmón sano se reutilizaron los datos de entrenamiento y sus respectivas mascaras con segmentación de pulmón extra. Esta técnica se describe a continuación. Cabe aclarar que se buscaron bases de datos de pulmón sano, sin embargo no se encontraron. Además, debido a que el procesamiento de las imágenes de entrenamiento nos indica una perdida de información debido a que en la pagina, https://medium.com/@hbjenssen/covid-19-radiology-data-collection-and-preparation-for-artificial-intelli (referida por la pagina donde se obtiene el conjunto de datos), se menciona que originalmente las imagenes se encontraban en un formato .jpg y fueron escalados. Este escalamiento, si se considera otro conjunto de datos de

pulmón sano, podría llegar a provocar demasiados errores debido a este cambio de escala.

El segmentado de los pulmones pudo realizarse gracias al código presentado por el usuario **Ankasor** para el *Data Science Bowl 2017*. El código de la libreta de python **SegmentingLungs.ipynb** es una adaptación de dicho código por lo que los créditos corresponden a dicho usuario.

Este proceso no requiere de la carga de las mascaras. Dado que las imágenes se encuentran en escala HU sabemos que los pulmones se encuentran entre [-700,-600] HU por lo que restringir en una variable la imagen a valores menores de -300 HU no asegura que la variable contendrá los valores del pulmón. Posteriormente se eliminan los bordes conectados contenidos mediante las función clear_border de scikit-image. A partir de está última se crean etiquetas empleando la función label de scikit-image. Para cada un de estas etiquetas es posible obtener sus características como el área y sus coordenadas. Por lo que se considera aquellas etiquetas cuya area sea mayor que dos y cumpla que el área de la región analizada sea mayor que los momentos HU de los datos definidos por el área de las etiquetas, en otro caso aquellas etiquetas tienen valor cero. (ver código más abajo). Una vez obtenidas las etiquetas quye cumplen las condiciones se aplica la función binary_dilatation de scipy.ndimage, esto con el fin de obtener la región que cubre los pulmones como hueso, grasa, entre otros. Todo este proceso se realiza en la función generate_markers(image) cuyo código se muestra a continuación.

```
#function to get the lungs and its surroundings (
    white part wich corresponds to bones or body)
def generate_markers(image):
    #Creation of the internal Marker
    marker_internal = image < -300
    marker_internal = segmentation.clear_border(
    marker_internal)
    marker_internal_labels = measure.label(
    marker_internal)
    areas = [r.area for r in measure.regionprops(
    marker_internal_labels)]
    areas.sort()
    if len(areas) > 2:
        for region in measure.regionprops(
    marker_internal_labels):
            if region.area < areas[-2]:
                for coordinates in region.coords:
                    marker_internal_labels[
    coordinates[0], coordinates[1]] = 0
    marker_internal = marker_internal_labels > 0
    #Creation of the external Marker
    external_a = ndimage.binary_dilation(
    marker_internal, iterations=10)
    external_b = ndimage.binary_dilation(
    marker_internal, iterations=55)
    marker_external = external_b ^ external_a
    #Creation of the Watershed Marker matrix
    marker_watershed = np.zeros((512, 512), dtype=np
    .int)
    marker_watershed += marker_internal * 255
    marker_watershed += marker_external * 128

    return marker_internal, marker_external,
    marker_watershed
```

Listing 3. script para obtener las imagenes Internal external y watershed Marker.

Al procesar una solo imagen se obtienen las tres imágenes mostradas en 10. Internal Marker corresponde a las etiquetas que cumplen las condiciones. External marker corresponde a la dilatación de la imagen anterior sin considerar la zona de dicha imagen y Watershed Marker corresponde a la unión de las dos imágenes anteriores donde cada una de ellas tendrá diferente escala de gris.

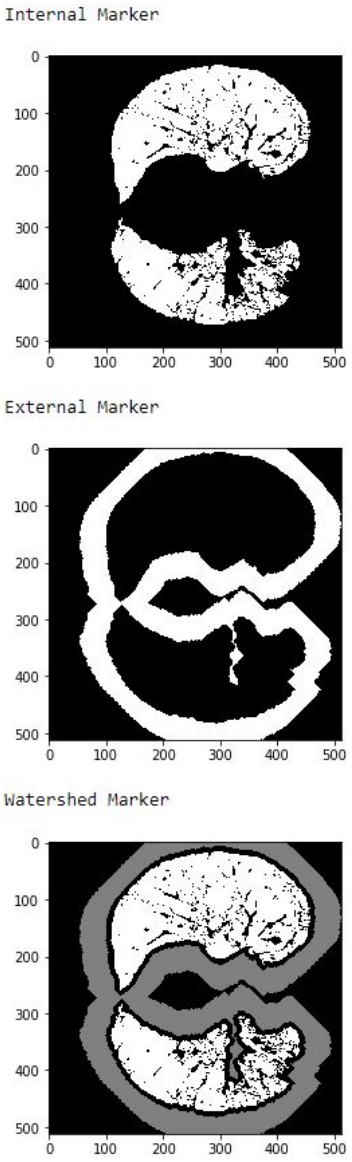

Fig. 10. Internal, External y Wateshed Marker para una imagen de entrenamiento. Note que el trío preserva las dimensiones de la imagen original

Las primeras 16 imagenes Watershed del conjunto de entrenamiento se muestran en la figura 11.

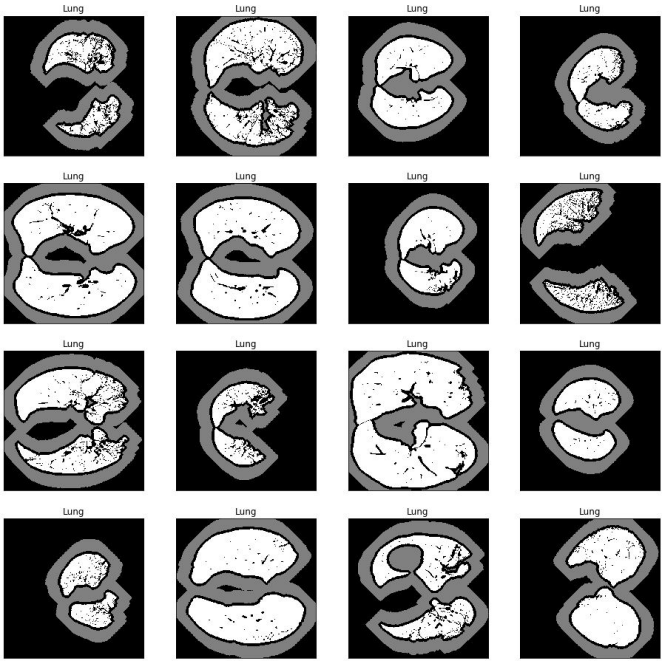

Fig. 11. Internal, External y Wateshed Marker para una imagen de entrenamiento. Note que el trío preserva las dimensiones de la imagen original

Puede observarse que las imágenes Watershed no son suficientes para segmentar el pulmón, ya que hay zonas donde se pierde información. Para Ankasor propone un algoritmo basado en el operador de Sobel, en las imágenes Watershed y un filtro basado en la dilatación binaria que ofrece resultados sorprendentes. El código se muestra a continuación.

```
#function to segment the lungs
def seperate_lungs(image):
    #Creation of the markers as before:
    marker_internal, marker_external,
    marker_watershed = generate_markers(image)

    #Creation of the Sobel-Gradient
    sobel_filtered_dx = ndimage.sobel(image, 1)
    sobel_filtered_dy = ndimage.sobel(image, 0)
    sobel_gradient = np.hypot(sobel_filtered_dx,
    sobel_filtered_dy)
    sobel_gradient *= 255.0 / np.max(sobel_gradient)

    #Watershed algorithm
    watershed = morphology.watershed(sobel_gradient,
     marker_watershed)

    #Reducing the image created by the Watershed
    algorithm to its outline
    outline = ndimage.morphological_gradient(
    watershed, size=(3,3))
    outline = outline.astype(bool)

    #Performing Black-Tophat Morphology for
    reinclusion
    #Creation of the disk-kernel and increasing its
    size a bit
    blackhat_struct = [[0, 0, 1, 1, 1, 0, 0],
                       [0, 1, 1, 1, 1, 1, 0],
                       [1, 1, 1, 1, 1, 1, 1],
                       [1, 1, 1, 1, 1, 1, 1],
                       [1, 1, 1, 1, 1, 1, 1],
                       [0, 1, 1, 1, 1, 1, 0],
                       [0, 0, 1, 1, 1, 0, 0]]
    blackhat_struct = ndimage.iterate_structure(
    blackhat_struct, 8)
    #Perform the Black-Hat
    outline += ndimage.black_tophat(outline,
    structure=blackhat_struct)

    #Use the internal marker and the Outline that
    was just created to generate the lungfilter
    lungfilter = np.bitwise_or(marker_internal,
    outline)
    #Close holes in the lungfilter
    #fill_holes is not used here, since in some
    slices the heart would be reincluded by accident
    lungfilter = ndimage.morphology.binary_closing(
    lungfilter, structure=np.ones((5,5)), iterations
    =3)

    #Apply the lungfilter (note the filtered areas
    being assigned -2000 HU)
    segmented = np.where(lungfilter == 1, image,
    -2000*np.ones((512, 512)))

    return segmented, lungfilter, outline, watershed
    , sobel_gradient, marker_internal,
    marker_external, marker_watershed
```

Listing 4. script obtener las imagenes del pulmon segmentadas.

Las primeras 16 imágenes de Sobel obtenidas por el código de Anksar se muestran en la figura 12.

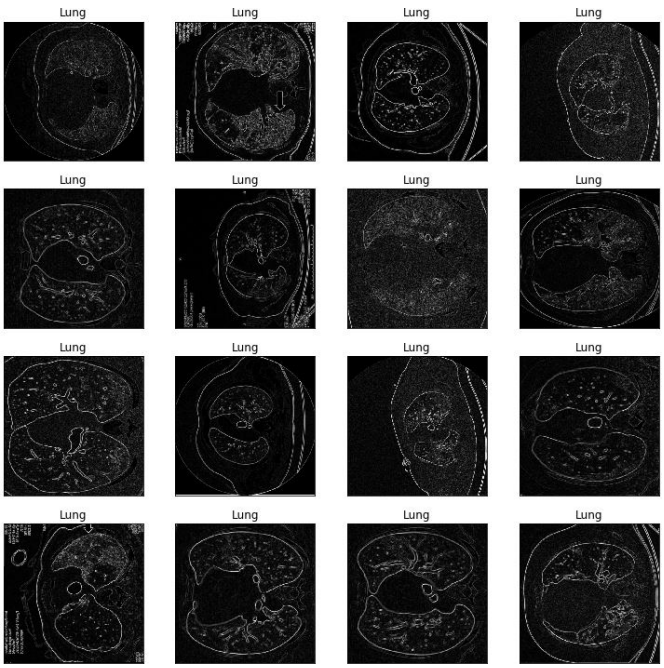

Fig. 12. Imagenes de Sobel para las primeras 16 imágenes de entrenamiento

Las primeras 16 imágenes segmentadas de pulmón obtenidas por el código de Anksar se muestran en la figura 13.

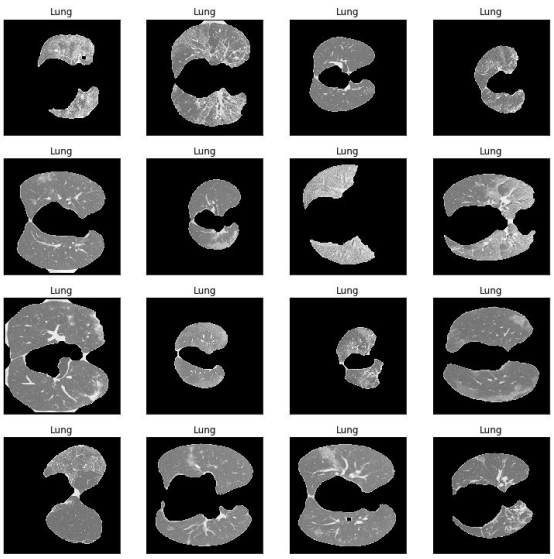

Fig. 13. Imágenes segmentadas de pulmón para las primeras 16 imágenes de entrenamiento

Puede observarse que el segmentado no es siempre perfecto. Un claro ejemplo de ello es la imagen de pulmón en la segunda fila, tercera columna. Al compararla con la imagen de Sobel observamos que existe una perdida de información. Esto se debe a los valores de intensidad en HU debidas a alguna de las enfermedades ya que debemos recordar que se obliga a las etiquetas contengan un umbral determinado, que hace que dichas regiones queden fuera de la consideración. A pesar de mover el valor de umbral a valores mayores no se obtienen mejores resultados. Este hecho nos agrega una limitante al modelo, que será discutida más adelante.
Una vez obtenidas las imágenes de segmentación, se toman las mascaras provistas para el entrenamiento y se igualan a 1 en los valores donde existe alguna clase de anomalía. Esto con el fin de tener una mascara general que nos indique las zonas donde el pulmón se encuentra enfermo. Una muestra de estas imágenes pude observarse en la figura 14.

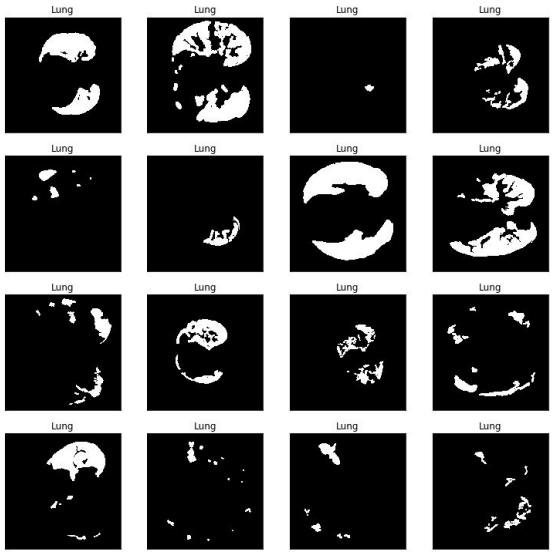

Fig. 14. Zonas de pulmón dañado o enfermo para 16 imágenes de entrenamiento

Para obtener las mascaras de zonas del pulmón sano basta con evaluar la resta de las imágenes de pulmón evaluada a dos valores de intensidad de gris (0,1) con las mascaras de pulmón enfermo. Dado que las mascaras de pulmón podrían no haber considerado alguna región que quizá exista en las mascaras de enfermedad, la resta de estos valores resultará en un valor de -1 y como sabemos que aquellas corresponden a zonas con daño, entonces aquellos valores se hacen cero para así asegurar que las mascaras contengan información de pulmón sano. Para obtener las mascaras reales (ver fig. 15) basta con multiplicar entrada a entrada las imágenes de entrenamiento con estas nuevas mascaras. Posteriormente de estas mascaras reales se extrae la información de textura y se guarda en un archivo llamado **healthy.csv**.

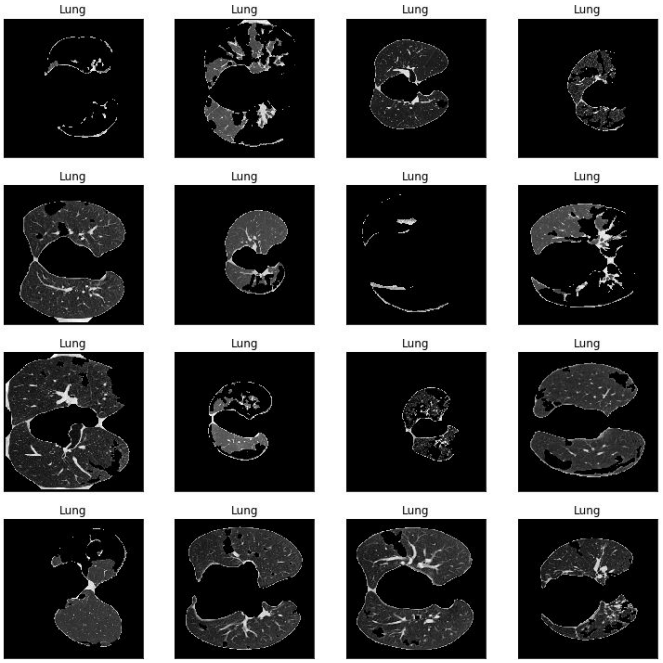

Fig. 15. Mascaras reales de pulmón sano para 16 imágenes de entrenamiento

2.2 Parte II: Entrenamiento y validación

Esta parte la podemos dividir en otras dos partes. La primera de ellas consiste en entrenar y validar únicamente con el conjunto de entrenamiento, esto claramente dividiendo este conjunto en dos para entrenar y validar. La segunda de ellas consiste en entrenar con las cien imágenes de entrenamiento y validar con las diez provistas. Esta se división se hace ya que para la segunda parte no hay una forma para comprobar la eficiencia de nuestro modelo.

2.2.1 Conjunto de entrenamiento únicamente

El código que se presenta en esta parte corresponde a la libreta **TrainWithLessSamples.ipynb**.
Es necesario cargar las imágenes de entrenamiento y sus respectivas mascaras. El conjunto de cien imágenes se divide en 80% entrenamiento y 20% para validación, es necesario conocer los índices originales de las imágenes originales ya que para validar es necesario obtener las mascaras originales de las imágenes de validación. Al igual que en la parte I, se

convierten todas las imágenes al tipo $uint8$ y se procede a obtener las mascaras originales de cada conjunto según su clase. Tal y como se hace antes. De esta manera es posible obtener las propiedades de textura para el conjunto de entrenamiento. Note que este paso se vuelve a hacer ya que en la parte uno se consideró un conjunto de entrenamiento de 100 imágenes. Una vez obtenidas las propiedades de cada una de las tres clases, los datos se almacenan en un dataFrame de pandas con el fin de agregar una columna de clases, que contendrá un número del 0 al 3 según su clase. Con estos datos se entrena una maquina de soporte vectorial con kernel sigmoide y pesos de $\{0 : 0.9, 1 : 1.4, 2 : 1.4, 3 : 0.97\}$ donde el numero a la izquierda de los dos puntos corresponde a la clase y el número a la derecha corresponde al parámetro peso de la maquina de soporte vectorial. Tanto el kernel y los pesos se pueden considerar hiperparámetros que fueron encontrados a base de prueba y error.

Para la clasificación de las imágenes de entrenamiento fue necesario encontrar la segmentación de los pulmones tal y como se describe en la parte uno, esto con el fin de asegurar el análisis de únicamente los pulmones. Para estas imágenes segmentadas se encontraron los superpíxeles para cada una con aproximadamente 50 unidades, y compacidad igual a 5. Nuevamente estos valores los consideramos hiperparámetros. La imagen 16 muestra una imagen de validación con sus respectivos superpíxeles (lineas amarillas).

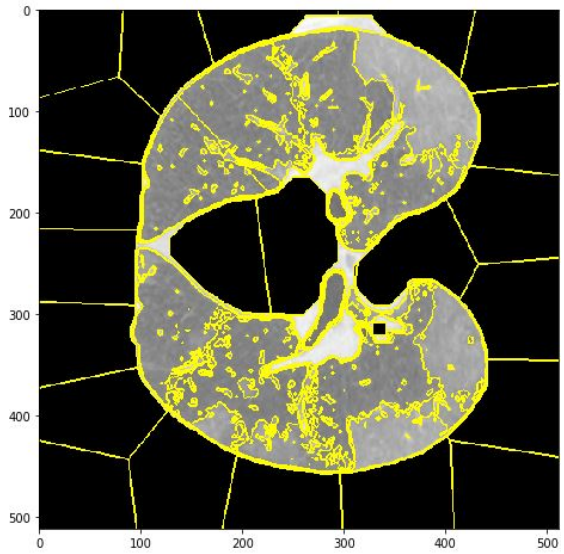

Fig. 16. Imagen de validación y sus superpíxeles

Dado que cada superpíxel contiene la información de un conjunto de la imagen y el algoritmo SLIC devuelve una matriz con las posiciones de dichos superpíxeles, siendo diferentes entre sí, basta con crear una función que itere sobre la imagen y sus respectivos superpíxeles y para cada uno de ellos se calculen las propiedades de la textura. Dado que no se conoce el orden de los superpíxeles, el cálculo de las propiedades de textura se hace para cada uno de ellos. Sin embargo dado que el fondo es negro, esperamos que el contraste y la disimilitud sean uniformes iguales a cero. De esta forma para un superpíxel dado, si dichas propiedades son diferentes a dicho valor sabemos que el superpíxel contiene información relevante y procedemos a clasificarlo, de acuerdo a sus propiedades con la SVM. Adicionalmente dado que estamos recorriendo la imagen sobre todos los superpíxeles es posible recrearla de acuerdo a la clasificación de la SVM. Para este caso cuando la imagen determina la clase sana en un determinado superpixel se pinta o se asignan los valores de dicho superpíxel a la imagen reconstruida como cero o negro. Para la clase 1 (consolidación pulmonar) se asigna el valor 128, los valores 192 y 255 corresponden a las clases 2 (derrame pleural) y 3 (opacificación de vidrio) respectivamente. El código de dicha función es como sigue.

```
#function to obtain the texture propierties of each
    masked image using the GLCM method
def wws(images,slicMatrix,predictor):
    data = []
    predictedImages = []
    #data = np.zeros((slicMatrix.shape[0],np.amax(
    segmentedSlic), 6),dtype=float)
    for i in range(len(images)):
        #auxdata = np.zeros((np.amax(segmentedSlic[i
    ,:,:]), 6),dtype=float)
        auxdata = []
        predictedImage = np.zeros((images[i].shape),
    dtype=np.uint8)
        for j in range(np.amax(slicMatrix[i,:,:])+1)
    :
            segment = np.where(slicMatrix[i,:,:]==j)
            auximage = np.zeros((images[i].shape[0],
    images[i].shape[1]),dtype=np.uint8)
            auximage[segment] = 1
            auximage = auximage*images[i]
            GLCM = greycomatrix(auximage,distances
    =[1],angles=[0],symmetric=False,normed=True)
            GLCM = GLCM[1:,1:,:,:]
            contrast = greycoprops(GLCM,'contrast')
            dissimilarity = greycoprops(GLCM,'
    dissimilarity')
            homogeneity = greycoprops(GLCM,'
    homogeneity')
            asm = greycoprops(GLCM,'ASM')
            energy = greycoprops(GLCM,'energy')
            correlation = greycoprops(GLCM,'
    correlation')
            if (contrast and dissimilarity and
    homogeneity !=0):
                auxvar = (contrast,dissimilarity,
    homogeneity,asm,energy,correlation)
                auxdata.append(auxvar)
                auxvar1 = np.array(auxvar)
                auxvar1 = auxvar1.reshape(1,-1)
                prediction = predictor.predict(
    auxvar1)[0]
                #print(prediction)
                if prediction == 0:
                    predictedImage[segment] = 0
                elif prediction == 1:
                    predictedImage[segment] = 128
                elif prediction == 2:
                    predictedImage[segment] = 192
                else:
                    predictedImage[segment] = 255
        auxdata = np.array(auxdata)
        #print(auxdata[:,:,0,0].shape)
        data.append(auxdata[:,:,0,0])
        predictedImages.append(predictedImage)
    return data, predictedImages
```

Listing 5. Funcion para la clasificacion de imagenes.

Los valores de cada clase previamente mencionados también se ajustaron para las mascaras proporcionadas inicialmente. Esto para poder comparar visualmente la eficiencia de la SVM. Los resultados de clasificación se muestran en la figura 17 y las mascaras provistas correspondientes a las

mismas imágenes de validación se muestran en la figura 18. El orden de aparición de ambas es el mismo y corresponden a una tomografía particular.

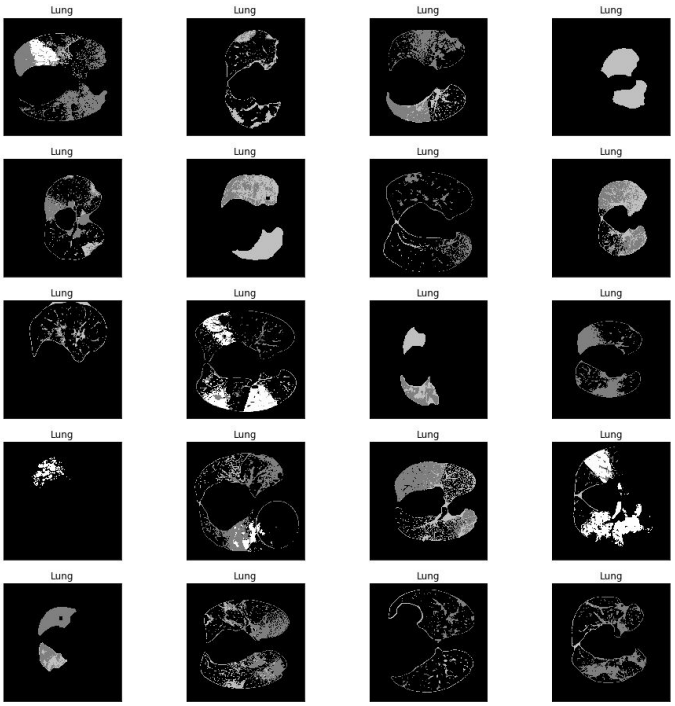

Fig. 17. Imágenes reconstruidas después de la clasificación por la SVM

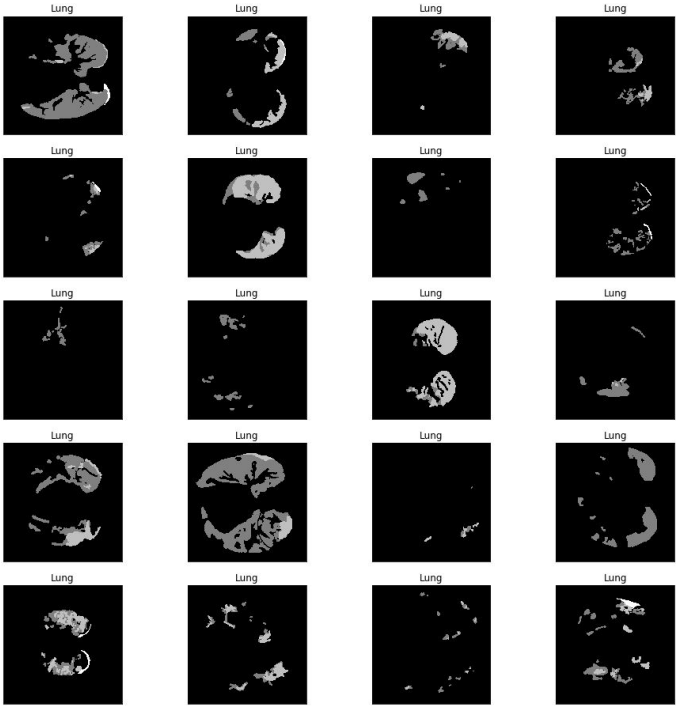

Fig. 18. Mascaras provistas elaboradas por un médico especialista

2.2.2 Conjunto de entrenamiento de 100 imágenes y Validación de 10 imágenes

El proceso a seguir en esta parte es exactamente la misma que se siguió para la parte *Conjunto de entrenamiento únicamente* con la diferencia que no se cuentan con mascaras con las cuales comparar realizadas por un especialista. Por lo que los resultados presentados en esta parte únicamente pueden compararse con las imágenes de validación, que para algunas zonas no se requiere ninguna especialización en medicina para intuir la clase de daño que presenta la imagen. Sin embargo, existirán casos para los cuales un no especialista no podrá diferenciar.

Al igual que en *Conjunto de entrenamiento únicamente* se usaron los mismos hiperparámetros, ya que a partir de ellos fue posible obtener resultados, visualmente más cercanos, a los esperados. De esta manera únicamente presentamos las imágenes de clasificación (fig 19) y las tomografías originales (fig 20), nuevamente el orden de aparición de ambas es el mismo. El código y sus resultados pueden encontrarse en la libreta **TestingData.ipynb**.

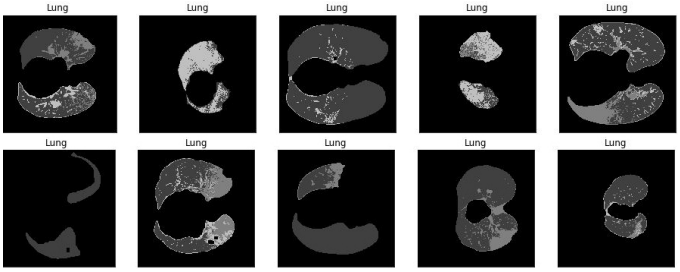

Fig. 19. Imágenes reconstruidas después de la clasificación por la SVM

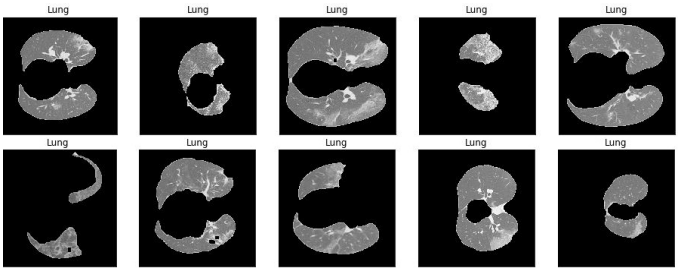

Fig. 20. Segmentación del pulmón de las tomografías originales

3 DISCUSIÓN

Los resultados de la segmentación de pulmón son aceptables, sin embargo, en algunos casos la segmentación no se realiza correctamente al no capturar la información del daño por alguna de las tres clases. Esto puede explicarse por el umbral definido en el algoritmo de Ankasor. A pesar de cambiar los valores de umbral no se consiguió resolver este problema. Esto crea una limitante al modelo que se propone en este artículo y es que no podría implementarse para la detección de anomalías sin la necesidad de un médico especialista, dejándolo únicamente como una herramienta de diagnostico complementaria. Debe notarse que esta mala segmentación se da únicamente para cuando el daño parece ser bastante grave. Para los casos en los cuales la intensidad de las anomalías presentes en las tomografías es similar a los valores de intensidad de un pulmón sano el algoritmo no tiene problemas para realizar su segmentación. En este punto cabe mencionar que el algoritmo de Ankasor fue escrito para detectar cáncer de pulmón cuyas diferencias en

las intensidades de las tomografías suelen ser significativas con respecto al problema que aquí se trata.

Por otro lado, al comparar, visualmente, los resultados obtenidos de la reconstrucción por la SVM y las mascaras reales (17 y 18) se encuentra que para algunos casos la detección llega a ser bastante similar considerándose una buena clasificación, mientras que en otros casos la clasificación o detección es incorrecta. Lamentablemente no se logró desarrollar una función métrica que ofrezca una cantidad numérica de clasificaciones y/o detecciones erróneas debido a que es complicado definirla sobre las mismas imágenes ya que se empleó el método de superpíxeles, que es complicado definir los mismos superpíxeles para dos imágenes cuya estructura difiere. Para el caso de la clasificación usando las diez imágenes provistas no existe un método especializado para determinar las mascaras reales sin la ayuda de un especialista, a pesar de ello, una comparación visual de las imágenes reconstruidas con las tomografías nos da un indicio de que la SVM logra distinguir algunas anomalías que son fácilmente detectables.

La obtención de características de textura para las tres anomalías y lo que se consideró como pulmón sano se realizó considerando la totalidad de la imagen y no mediante superpíxeles, por lo que podría considerarse que los datos tienen perdida de información por lo que un cambio de obtención de datos implementando superpíxeles podría aumentar la eficiencia de la SVM pero con un costo computacional más alto.

Si bien es cierto las mascaras provistas fueron examinadas por al menos un médico especialista debemos considerar que este trabajo manual puede llegar a ser cansado y podría contener algunos errores. Se ha demostrado que los médicos son propensos a cometer errores debido a diversos factores. De esta forma sería bastante enriquecedor trabajar en conjunto con un especialista con el fin de evitar errores y obtener información que unicamente se obtiene mediante la experiencia.

4 Conclusiones

A pesar de no contar con una métrica de clasificación, los resultados obtenidos se consideran, visualmente, aceptables. Por lo que se considera que el método aquí empleado es bueno y podría ser implementado como una herramienta de ayuda a los médicos para la detección de las anomalías de pulmón desarrolladas por pacientes con COVID-19. Durante el desarrollo experimental se encontraron múltiples complicaciones que no se estudiaron tales como la correcta segmentación de pulmón, el estudio de los hiperparámetros de la maquina de soporte vectorial así como los de los superpíxeles, el uso de superpíxeles para la obtención de los datos de entrenamiento, el análisis de las tomografías en tipo $uint16$, el uso de imágenes de pulmón verdaderamente sano y el uso de tomografías de pacientes con COVID-19 sin perdida de datos. Por lo que el presente artículo es solamente un estudio básico pero efectivo, que puede ser ampliamente mejorado al considerar dichas complicaciones.

References

[1] S. Khan, R. Siddique, M. Shereen, A. Ali, J. Liu, Q. Bai, N. Bashir, M. Xue *The emergence of a novel coronavirus (SARS-CoV-2), their biology and therapeutic options*, J. Clin. Microbiol., versión en linea, 2020.

[2] Hardward Health Publishing, *COVID-19 basics: Symptoms, spread and other essential information about the new coronavirus and COVID-19*, Harvard Medical School, https://www.health.harvard.edu/diseases-and-conditions/covid-19-basics. Consultado el 06.06.20

[3] N. Sánchez, *Gráfica: abrupto crecimiento de enfermedades respiratorias no diagnosticadas como Covid-19 en plena pandemia en México*, Nota Periodística para el sitio web infobae, https://www.infobae.com/america/mexico/2020/04/08/grafica-abrupto-crecimiento-de-enfermedades-respiratorias-no-diagnosticadas-c Consultado el 06.06.20

[4] OMS, *Q and A on coronaviruses (COVID-19)*, Sitio Web, https://www.who.int/emergencies/diseases/novel-coronavirus-2019/question-and-answers-hub/q-a-detail/q-a-coronaviruses#:~:text=symptoms. Consultado el 06.06.20

[5] P Galiatsatos, *What Coronavirus Does to the Lungs*, Articulo para Johns Hopkins Medicine, https://www.hopkinsmedicine.org/health/conditions-and-diseases/coronavirus/what-coronavirus-does-to-the-lungs. Consultado el 06.06.20

[6] Cleveland Clinic, *Pleural Effusion Causes, Signs and Treatment*, https://my.clevelandclinic.org/health/diseases/17373-pleural-effusion-causes-signs--treatment#:~:text=Pleural%20effusion%2C%20sometimes%20referred%20to,to%20lubricate%20and%20facilitate%20breathing.. Consultado el 06.06.20

[7] healthline, *Lung Consolidation: What It Is and How It's Treated*, LungConsolidation:WhatItIsandHowIt\OT1\textquoterightsTreated. Consultado el 06.06.20

[8] M. Assar, B. Amini *Ground-glass opacification*, Radiopaedia https://radiopaedia.org/articles/ground-glass-opacification-3?lang=us. Consultado el 06.06.20

[9] T. DenOtter, J. Schubert *Hounsfield Unit*, StatPearls, NCBI https://www.ncbi.nlm.nih.gov/books/NBK547721/. Consultado el 07.06.20

[10] F. Gaillard, K. Greenway *Hounsfield unit*, Radiopaedia https://radiopaedia.org/articles/hounsfield-unit. Consultado el 07.06.20

[11] Mathlab *niftiread*, Documentación https://www.mathworks.com/help/images/ref/niftiread.html. Consultado el 07.06.20

[12] M. Winkler *The NIFTI file format*, Brainder https://brainder.org/2012/09/23/the-nifti-file-format/. Consultado el 07.06.20

[13] M. Nixon, A. Aguado *Feature Extranction and Image Processing for Computer Vision*, 3rd ed. Elsevier Academic Press, 2012.

[14] P. Brodatz, *A Photographic Album for Artist and Designers*. Reinhold, USA.

[15] K. Karru, A. Jain, R. Bolle *Is there any texture in an image?*, Pattern Recog. 29 (9), 1437-1446.

[16] A. Humeau-Heurtier *Texture Feature Extraction Methods: A Survey*, IEEE Access, 7, 8975-9000, 2019.

[17] J. Bakos *Embedded Systems*, ARM Programming and Optimization, 1ra ed., 2016.

[18] T. Fletcher *Notas: Support Vector Machines Explained*, UCL University, 2008. https://cling.csd.uwo.ca/cs860/papers/SVM_Explained.pdf. Consultado el 07.06.20

[19] A. Anónimo *Superpixel: Empirical Studies and Aplications*, Chigago University, sin año. https://ttic.uchicago.edu/~xren/research/superpixel/. Consultado el 07.06.20

[20] D. Stutz, A. Hermans, B. Leibe *Superpixels: An Evaluation of the State-of-the-Art*, Computer Vision and Image Understanding, 2017.

[21] J. Darshita *Superpixels and SLIC*, Medium, 2019. https://medium.com/@darshita1405/superpixels-and-slic-6b2d8a6e4f08. Consultado el 26.04.20