Facultad de Ciencias

Diagnóstico de nódulos de pulmón mediante segmentación con modelos UNet y YOLOv8 entrenados con el dataset LIDC-IDRI

Diagnosis of lung nodules through segmentation using UNet and YOLOv8 models trained with the LIDC-IDRI dataset.

Trabajo de Fin de Máster
para acceder al

MÁSTER EN DATA SCIENCE

Autor: Abel Amado González Bernad

Director\es: Pablo Martínez Ruiz del Árbol
y Sergio Sánchez Cruz

Septiembre - 2023

Resumen

El cáncer es una de las enfermedades con más incidencia y mortalidad de la sociedad actual. Las instituciones médicas y expertos responsables de lograr diagnósticos precisos encuentran un enorme valor en el uso de *software* que pueda asistir la detección de posibles tumores en imágenes médicas. Por lo tanto, los avances en inteligencia artificial, especialmente los modelos basados en *deep learning*, están desempeñando un papel cada vez más importante en tareas de alta complejidad, como la detección de patologías a partir de una simple imagen. En este estudio se ha entrenado un modelo de *deep learning* con una arquitectura UNet con el dataset LIDC-IDRI, el cual es el repositorio abierto más grande de imágenes médicas etiquetadas. Este conjunto de datos contiene imágenes TAC de 1018 pacientes con los correspondientes nódulos pulmonares identificados por radiólogos. El modelo entrenado consigue detectar y segmentar las regiones tumorales de imágenes TAC. Además, se proporcionan métricas que evalúan su rendimiento de acuerdo a criterios tanto 2D como 3D. Por último, se compara su desempeño con uno de los modelos de procesamiento de imágenes más conocidos y recientes: YOLOv8, entrenado, de nuevo, con una adaptación del dataset LIDC-IDRI.

Palabras clave: Cáncer, TAC, UNet, LIDC-IDRI, python, deep learning, visión artificial, nódulos, pulmón, segmentación y CNN.

Abstract

Cancer is one of the diseases with the highest incidence and mortality rates in today's society. Medical institutions and experts responsible for achieving precise diagnoses find significant value in the use of software that can assist in the detection of possible tumors in medical images. Therefore, advances in artificial intelligence, especially deep learning models, are playing an increasingly important role in tasks of high complexity, such as the detection of pathologies from a single image. In this study, a deep learning model with a UNet architecture was trained using the LIDC-IDRI dataset, which is the largest open repository of labeled medical images. This dataset contains CT scan images from 1018 patients with corresponding lung nodules identified by radiologists. The trained model successfully detects and segments tumor regions in CT scan images. Additionally, metrics are provided to evaluate its performance based on both 2D and 3D criteria. Finally, its performance is compared to one of the most well-known and recent image processing models: YOLOv8, which was trained once again using an adaptation of the LIDC-IDRI dataset.

Key words: Cancer, CT scan, UNet, LIDC-IDRI, python, deep learning, computer vision, nodules, lung, segmentation and CNN.

Índice

1. Introducción

1.1. Perspectiva y motivación

El cáncer es una enfermedad que ha afectado a la humanidad durante siglos y sigue siendo uno de los mayores desafíos de salud en todo el mundo. A medida que avanza el siglo XXI, los avances médicos y científicos han proporcionado un mayor conocimiento sobre el cáncer y sus diversas formas. Según la Organización Mundial de la Salud (OMS), se estima que el cáncer fue responsable de aproximadamente 9,5 millones de muertes en 2020, aproximadamente 1 sexto de las que se registran [1]. A su vez, la cantidad de casos diagnosticados va en aumento con el paso de los años. En parte, esto es debido a los avances en el diagnóstico y su accesibilidad, así como a una mayor concienciación pública sobre la importancia de una detección temprana.

El cáncer es un concepto amplio que puede clasificarse según a qué parte de cuerpo está afectando. Si bien es cierto que cada parte del mundo presenta una estadística distinta, especialmente de acuerdo a su nivel de desarrollo económico, los casos más comunes en millos por año son de mama, 2,26; de pulmón 2,21; colorrectal, 1,93; de próstata 1,41; de piel (distinto del melanoma) 1,20 y gástrico 1,09. Sin embargo, la tasa de mortalidad sigue una ordenación distinta siendo de mayor a menor: pulmón con 1,8 millones de defunciones anuales; colorrectal con 916 000; hepático con 830 000; gástrico con 769 000 y de mama con 685 000 en el año 2020 [1]. En España, en 2020 el cáncer supuso más muertes que las enfermedades infecciosas (incluida la COVID-19) [2].

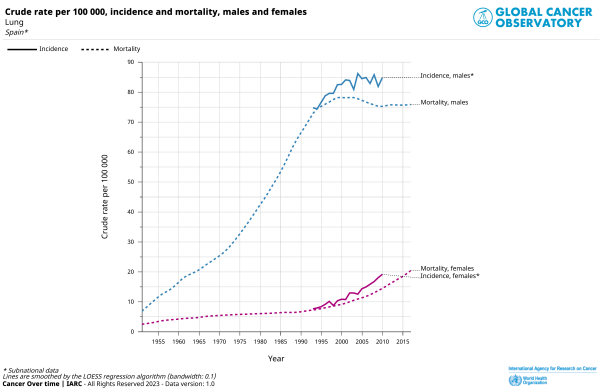


Figura 1.1: Casos registrados diagnosticados y fallecidos de cáncer de pulmón en España separado por sexos normalizados a casos cada 100, 000 habitantes [3].

Atendiendo a la Figura 1.1, se aprecia un aumento de casos de cáncer de pulmón con el paso de los años. A su vez, en concreto para este tipo de cáncer en se aprecia una mayor cantidad de casos en hombres que en mujeres.

Como es bien conocido, la probabilidad de supervivencia en los pacientes con cáncer aumenta considerablemente en caso de un diagnóstico temprano. Cuando este es detecta-

do en sus etapas iniciales, hay más opciones de tratamiento disponibles y las posibilidades de recuperación son mayores. Por ejemplo, el cáncer de ovario detectado en sus primeras etapas tiene una tasa de supervivencia a tres años por encima del 90 % en los países desarrollados [3], mientras que, en etapas más avanzadas, esta tasa disminuye significativamente.

Además, la detección temprana no solo salva vidas, sino que también reduce los costos asociados al tratamiento y a la necesidad de mayor asistencia al paciente en etapas avanzadas. Según [4], los costos del tratamiento del cáncer resultan en promedio cuatro veces más baratos que el cuidado necesario en etapas más avanzadas de la enfermedad. Esto subraya la importancia de la inversión en programas de detección temprana y la promoción de exámenes de detección, como pruebas de laboratorio de análisis de sangre y orina; biopsia, endoscopia; pruebas genéticas o imágenes médicas como radiografías, tomografías computerizadas (TC), resonancias magnéticas, ecografías, ...

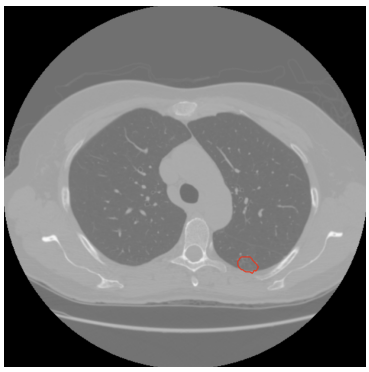


Figura 1.2: Ejemplo de una de las imágenes de una tomografía computerizada, TC, de tórax. Notar el nódulo indicado en el pulmón derecho [5].

Es en este último respecto en el que consiste este trabajo. En las imágenes médicas a vista de ojo experto puede identificarse bultos, texturas y patrones que pueden corresponderse con tumores, o en general, tejidos anómalos. Gracias a estas imágenes, los radiólogos pueden dar un diagnóstico sin haber realizado ninguna prueba invasiva al paciente. De esta forma, típicamente el experto debe revisar una a una las imágenes de cada paciente. En el caso de las tomografías computerizadas, TC, un escaneo supondría tener que revisar decenas de imágenes siendo cada una de ellas una sección del cuerpo. Y, como es razonable, para el ojo no experto no resulta evidente la tarea de identificar posibles tumores.

Por todo ello, resulta de gran interés el desarrollo de *software* que facilite el análisis de las imágenes al experto permitiendo un diagnóstico más rápido y certero. Sin embargo, aunque las decisiones de los expertos suelen ser mayoritariamente unánimes en cuanto a los diagnósticos, hay veces que aparecen ambigüedades, a la vez que toda persona está limitada por sus capacidades, conocimientos y vulnerable a ilusiones y equivocaciones.

Resulta evidente entonces la utilidad de disponer de una segunda opinión que reduzca la subjetividad del diagnóstico. A este respecto, los modelos de visión artificial basados en *deep learning* pueden elaborar conclusiones a partir de las imágenes tomadas de un paciente. En cierto sentido ofrecen un diagnóstico por inspección visual. El modelo podría clasificar la gravedad de la patología, por ejemplo, benignidad o malignidad, o incluso detectar dónde hay un tumor solo a partir de las imágenes tomadas directamente del paciente.

Actualmente existen ciertos *softwares* que sirven como herramienta para el análisis informático de las imágenes médicas. Estos son los programas CAD, *computer-aided diagnosis*. Están equipados con técnicas de visión artificial tradicional, análisis por computadora y los nuevos modelos de inteligencia artificial basados en *deep learning* suponiendo una asistencia en el diagnóstico.

Un ejemplo de ello es AmCAD-UT® (AmCAD Biomed, Taiwán) el cual es un *software* muy intuitivo que ofrece una estación de trabajo que gestionar y mostrar las imágenes a la vez que proporciona de forma semiautomática cierta delimitación de tejidos, distinguiendo bordes de estructuras, anomalías, vascularización, calcificaciones, ... Incluso ofrece una clasificación de malignidad en caso de tratarse de tumores [8].

Las técnicas de *machine learning* basadas en *deep learning* están en gran auge hoy día, consiguiendo métricas muy prometedoras en tareas de visión artificial. De ahí la motivación de este trabajo, centrado en la detección y segmentación de nódulos de pulmón a partir de imágenes de tomografías computerizadas, TC.

1.2. Cáncer, tumores y nódulos

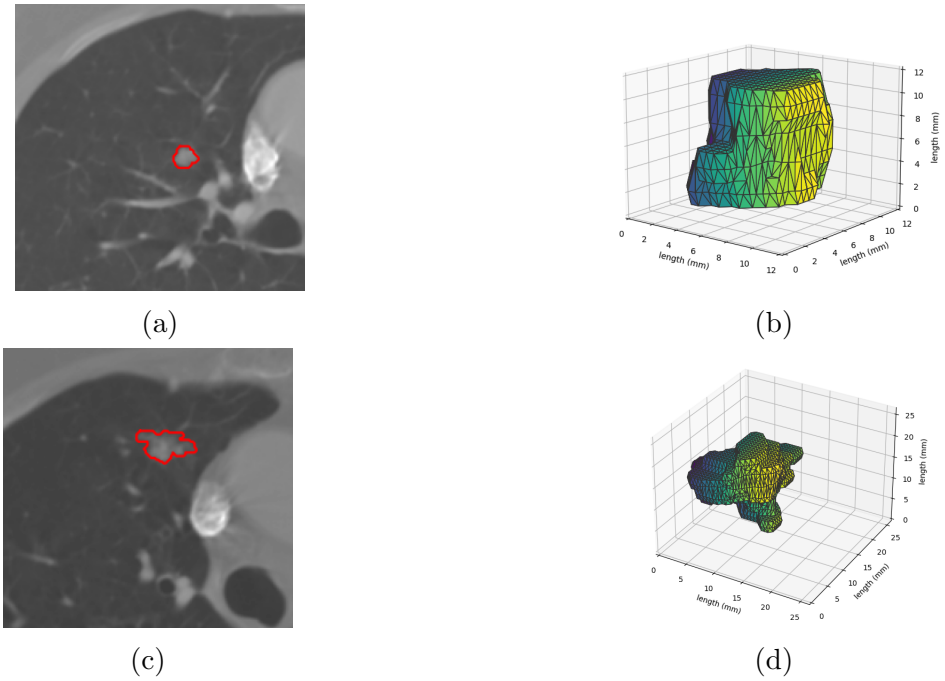
El cáncer recoge un gran número de enfermedades que se caracterizan por un desarrollo descontrolado de células anormales que pueden infiltrarse en otros tejidos, destruir tejido sano e incluso llegar a propagarse por el organismo.

De manera constante y natural las células del cuerpo se multiplican por un proceso llamado división celular. De esta forma se reponen las células dañadas, se genera tejido nuevo y se reemplazan células muertas. Puede ocurrir que cierta célula dañada, en vez de seguir este ciclo, se multiplique cuando no debería y crezca una masa de tejido anómalo. Estos crecimientos se denominan tumores que pueden clasificarse en cancerosos (malignos) o no cancerosos (benignos).

Esta clasificación de los tumores recoge una diferencia fundamental a la hora de evaluar la gravedad que supone para la salud. Los malignos pueden diseminarse por el organismo y propagar su desarrollo anómalo y descontrolado, derivando en otras consecuencias y apareciendo nuevas enfermedades, de ahí que sean cancerosos. En cambio,

los benignos, son desarrollos anómalos de células pero contenidos en un solo tejido. Es un volumen restringido sin capacidad de diseminación por el organismo. Por ello, estos últimos son menos graves, y muchas veces se ignoran, no requiriendo tratamiento. Aun así, se recomienda su vigilancia ya que pueden presentar evidencias de malignidad con el tiempo.

A nivel de diagnóstico, la textura, el tamaño y la forma permiten a los expertos realizar la clasificación de un tumor como benigno o maligno. Entre ellas, la presencia de espículas, es decir, una forma alejada de la redondez, con salientes. Un patrón poco homogéneo, con borde impreciso y difuso supone también la posibilidad de diseminación y por tanto malignidad, Figura 1.3.



(a) (b) (c) (d)

Figura 1.3: Ejemplo de nódulo de pulmón con signos de benignidad (a) y malignidad (c) tomados del dataset LIDC-IDRI [5].

A su vez, los radiólogos atienden también a la textura, es decir, a la presencia de estructura interna. Resulta evidente en imágenes médicas la calcificación de un tumor, es decir, el depósito de sales de calcio. Al igual que los huesos son fácilmente apreciables, la calcificación de un tumor resulta en un aumento de contraste respecto a tejido colindante no calcificado.

Comúnmente, los principales factores de riesgo del cáncer son el tabaco, siendo la principal causa de cáncer de pulmón en otros; el alcohol; exposición a luz solar, pudiendo producir cáncer de piel; obesidad; alimentación, siendo un factor importante el exceso de carnes rojas y carnes procesadas; enfermedades sexuales; edad avanzada; infecciones; hormonas; inflamaciones crónicas; inmunodeficiencias; radicación de alta energía; ciertas

sustancias químicas y los antecedentes familiares [6].

Por último, concretamente en el caso de los pulmones, este tejido anómalo es apreciable en tomografías computerizadas como el caso de la Figura 2.1 y reciben el nombre de nódulos (o masas si son superiores a 3 cm). Son tejidos que no deberían estar ahí pero que no siempre son sinónimo de cáncer. A lo largo de este trabajo estos bultos pulmonares serán referidos como nódulos en todo momento.

2. Imágenes médicas

La utilización de imágenes médicas en sus diversas técnicas ha revolucionado el campo de la medicina, proporcionando a los profesionales de la salud una herramienta muy útil para el diagnóstico y el seguimiento de diversas enfermedades y patologías. Estas técnicas permiten obtener representaciones visuales del interior del cuerpo, ofreciendo una visión detallada de estructuras anatómicas, tejidos y órganos. Gracias a su versatilidad, se emplean en una amplia variedad de situaciones clínicas, lo que las convierte en una pieza fundamental medicina moderna.

Las imágenes médicas son tomadas mediante dispositivos especializados, como tomógrafos computarizados (CT), resonancias magnéticas (MRI), ecografías, radiografías, entre otros. Cada técnica tiene sus propias características y aplicaciones específicas, lo que permite a los médicos seleccionar la más adecuada según las necesidades de cada paciente.

Figura 2.1: Tres ejemplos de imágenes médicas: radiografía de rayos X, una tomografía computerizada, TC, de tórax [5] y una ecografía [7].

Mediante la toma de imágenes médicas, los profesionales pueden detectar lesiones, tumores, fracturas, malformaciones y diversas anomalías de manera precisa e incluso en etapas tempranas, lo que facilita un diagnóstico más preciso y oportuno. La principal ventaja que presentan es que no requiere una intervención invasiva. Estas técnicas permiten obtener información de gran valor sin necesidad de cirugía, lo que reduce riesgos, preparaciones del paciente y del experto, tiempo de recuperación y costos asociados.

2.1. Rayos X: Tomografía computerizada TC y formato DICOM

Uno de los principales ejemplos de la toma de imágenes médicas son las tomografías computerizadas. Estas parten los principios de una radiografía simple, los rayos X. Estos son parte del espectro de la radiación electromagnética presentando una longitud de onda entre 10^{-8} y 10^{-11} m, es decir, del orden los armstrongs, Å y una frecuencia entre 10^{16} y 10^{20} Hz.

Esta radiación, al tener una longitud de onda menor a la luz visible presenta fotones (partículas de la radiación electromagnética) de mayor energía, confiriéndole una interacción radiación-materia distinta. Principalmente, la propiedad que les dota de utilidad en las imágenes médicas es la penetración que tiene en el tejido biológico. Estos pueden atravesar el cuerpo, ser absorbidos por este o dispersados.

Según la densidad de los tejidos, su espesor, los átomos que tengan sus moléculas y la propia energía de los rayos X, estos serán absorbidos en mayor o menor cantidad. De forma que los huesos serían el tejido más radiopaco del cuerpo, ofreciendo gran absorción traduciéndose en mayor intensidad de blanco en la radiografía. Por el contrario, el aire, o tejidos grasos y blandos presentan menor absorción apreciándose en tonos más oscuros en la imagen.

Ahora bien, las tomografías computerizadas, TC, son una sofisticación de esta idea, siendo la segunda técnica basada en rayos X más usada en medicina. Estas consiguen reconstruir el interior del cuerpo, pero en las 3 dimensiones espaciales tomando distintas imágenes correspondiéndose cada una con una sección del cuerpo, Figura 2.2.

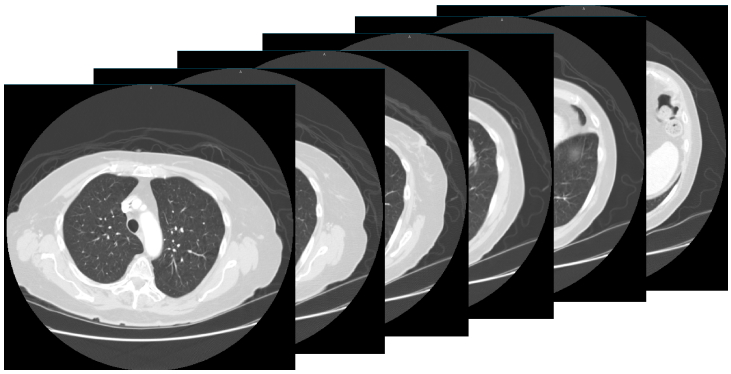


Figura 2.2: Imágenes o *slices* (secciones) de una tomografía computerizada de tórax de un paciente. Proporciona información de todo el volumen en 3D a lo largo del cuerpo [5].

Para la toma de estas imágenes se realiza el proceso conocido como tomografía axial computerizada, TAC. El paciente se tumba en la plataforma de la máquina. Esta dispone de generadores de rayos X y sensores para detectarlos en una estructura toroidal

dispuesta alrededor de la plataforma. Dentro de esta estructura el generador de rayos X va desplazándose realizando circunferencias entorno al paciente consiguiendo incidencias del haz de rayos X desde todos los ángulos en plano donde está el toro. Este proceso permitiría la toma de una sola imagen correspondiéndose con una sola sección del interior del cuerpo, *slice*. Así pues, de manera sucesiva, la estructura toroidal se desplazada para repetir del proceso y reconstruir una sección del cuerpo contigua. El proceso completo suele durar unos minutos, obteniendo un conjunto de imágenes, o *slices*, de tamaño 512×512 píxeles. Este proceso se ilustra en la Figura 2.3.

Figura 2.3: Proceso de toma de imágenes mediante una tomografía axial computerizada, TAC. Notar la rotación solidaria de la fuente de rayos X y el sensor, y el desplazamiento del toro para obtener las diversas secciones, *slices* [9].

En cuanto al almacenamiento digital de los TAC y, en general, las imágenes médicas, se guardan en formato `dicom`. Estas son las siglas *Digital Imaging and Communications in Medicine*, Imágenes Digitales y Comunicaciones en Medicina y es un estándar utilizado en todo el mundo. Estas son registradas con los dispositivos de toma de imágenes, algunos de ellos aquí nombrados como resonancias magnéticas, tomografías computarizadas, radiografías y ecografías. Además de la propia imagen, son asociados otros datos relevantes acerca de cómo se tomó la imagen, la información del paciente, anotaciones de los médicos, datos de la clínica o la fecha y hora en que se tomó la imagen entre otros.

Existe una librería de Python para gestionar este tipo de archivos llamada `pydicom`. Con ella pueden abrirse las imágenes de manera muy similar a la librería de visión por computadora más conocida `opencv`, a la vez que acceder a los atributos del objeto entre los que se encuentran datos como `PatientAge`, `PatientID`, `PatientName`, `PatientSex` ó `StudyDate`.

3. Introducción a la inteligencia artificial.

La Inteligencia Artificial (IA) es un concepto amplio, un campo de la informática que se ha vuelto cada vez más prominente en los últimos años debido a los avances tecnológicos, la creciente disponibilidad de datos y el aumento en la capacidad de procesa-

miento. Inteligencia artificial, estrictamente, recoge cualquier sistema que pueda realizar tareas de manera autónoma, imitando las capacidades humanas, yendo desde un razonamiento basado en una lógica sencilla, un cálculo matemático hasta el aprendizaje de conceptos abstractos, toma de decisiones, interpretación de imágenes o reconocimiento de emociones [10].

Actualmente el concepto de IA ha sido tomando por el aprendizaje máquina, conocido como *machine learning*, el cual consiste en plantear un algoritmo que aprenda mediante ejemplos, es decir, construir un modelo a base de nutrirlo con datos. Así pues, esta subdisciplina se centra en el desarrollo de modelos que permiten a las máquinas aprender de los datos sin definir una programación explícita. En lugar de seguir instrucciones específicas, los modelos de *machine learning* utilizan datos para identificar patrones y relaciones de cierta complejidad, lo que les permite realizar predicciones y tomar decisiones basadas en el aprendizaje previo. El conjunto de datos de los que se sirve un modelo para aprender se denomina *dataset*.

Uno de los ejemplos más comunes de IA y *machine learning* son los asistentes virtuales, como Siri de Apple o Google Assistant. Estos asistentes utilizan procesamiento del lenguaje naturalpara responder a las preguntas de los usuarios de manera muy eficiente.

Resulta útil distinguir dentro del *machine learning* modelos según su tipo de aprendizaje: supervisado o no supervisado. El aprendizaje supervisado es el más frecuente, consistiendo en ofrecer al modelo los datos de entrada y los resultados a los que debe llegar. De esta forma, el algoritmo se ajusta y aprende las reglas que llevan a esos resultados. Por el contrario, el aprendizaje no supervisado solo requiere los datos de entrada del modelo y no la conclusión a la que debe llegar. Estos últimos están orientados a evidenciar patrones que por lo general son difíciles de encontrar, no conociéndose el resultado apriori.

Es posible nombrar varios algoritmos de *machine learning* ampliamente utilizados que resultan de gran eficacia para resolver ciertos problemas y son útiles como punto de partida. El ejemplo más simple sería la regresión lineal, el cual es un algoritmo de aprendizaje supervisado utilizado para predecir valores numéricos continuos a partir de datos de entrada de manera lineal. Modelos más sofisticados pueden realizar una clasificación, es decir, categorizar los datos de entrada en una clase u otra. Ejemplos de ello pueden ser árboles de decisión, *k-near neighbour*, o métodos basados en *ensembles* como el conocido *random forest*. En tareas de agrupación de datos en conjuntos de manera no supervisada (*clustering*) se tendrían métodos como reglas de asociación o *k-means*. Los algoritmos de vector soporte, SVMs, *Support Vector Machine*, también consiguen resolver problemas de regresión y clasificación con gran éxito.

Sin embargo, se han quedado sin nombrar dentro del *machine learning* los algoritmos basados en *deep learning*. Estos han ganado gran relevancia en los últimos años

debido a su gran capacidad para aprender representaciones jerárquicas de los datos mediante el uso de redes neuronales artificiales (similar al funcionamiento biológico de una cerebro, *brain-based*) con múltiples capas. Su avance ha sido impulsado en gran medida por la disponibilidad de grandes cantidades de datos y el aumento en la capacidad de procesamiento, lo que ha permitido entrenar redes neuronales más profundas, complejas y potentes.

Estas redes son especialmente eficaces aprendiendo a identificar características y patrones altamente complejos y abstractos en los datos, lo que ha permitido logros sobresalientes en tareas como reconocimiento de imágenes, procesamiento del lenguaje natural, juegos estratégicos, interpretación de señales, ... Además, presentan un alto rendimiento en grandes conjuntos de datos y la habilidad de generalizar bien a nuevos casos una vez que ha sido entrenado adecuadamente.

3.1. Generalidades del *Deep Learning*

Los modelos de *Deep Learning* están detrás de los logros más sonados en la actualidad, algunos como ChatGPT de OpenAI [12] de procesamiento de lenguaje natural; YOLO de Ultralytics de procesamiento de imagen; Stable Diffusion de Stability AI de generación de imágenes, entre otros muchos. Están detrás de la visión artificial de los coches autónomos reconociendo el entorno y las señales de tráfico en tiempo real, de la interpretación del audio en dispositivos controlables por voz e incluso en diagnósticos médicos en programas CAD. Estos modelos consiguen realizar tareas que hace unos años parecían imposibles para un ordenador, alcanzando además precisiones que superan a algoritmos previos en problemas de gran complejidad e, incluso, en muchas ocasiones, al nivel humano.

Estos modelos se teorizaron en 1980, pero ha sido en los últimos 10 años cuando han cobrado gran importancia debido principalmente a dos motivos.

- Como se advertía, al englobarse dentro del *machine learning*, requiere datos que sirvan al modelo de ejemplos para realizar su aprendizaje. Es en la actualidad cuando más está destacando **la gran disponibilidad de los datos**, registrándose constantemente en todos los aspectos.
- Los cálculos necesarios para propagar las señales entre neuronas hasta llegar al resultado requiere gran capacidad de procesamiento. Son especialmente útiles en esta tarea las tarjetas gráficas, GPUs (*Graphics Processing Unit*), ya que su arquitectura permite realizar gran cantidad de operaciones sencillas simultáneamente. Hoy día los ordenadores disponen de **mayor capacidad de procesamiento** y en su mayoría de tarjetas gráficas.

Los modelos basados en *deep learning* parten de un algoritmo computacional similar al funcionamiento del cerebro: redes neuronales artificiales. A grandes rasgos, el modelo cuenta con gran cantidad de neuronas dispuestas en distintas capas. Estas reciben los datos de entrada y propagan sus valores a todas las neuronas de la capa siguiente interconectándose hasta llegar al final y ofrecer una respuesta, de ahí el concepto de *brain-based*, o basado en el funcionamiento del cerebro. El ejemplo más simple que recoge esta idea es el perceptron multicapa, Figura 3.1.

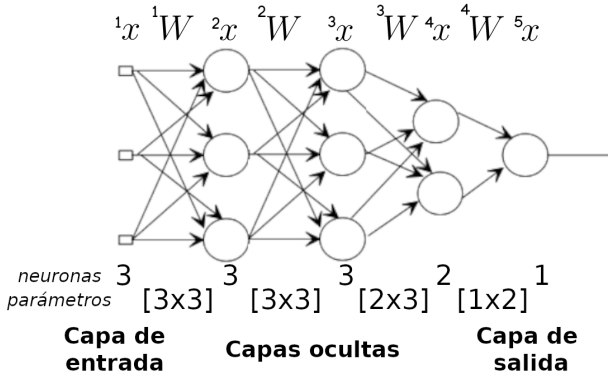


Figura 3.1: Esquema de un perceptrón multicapa, donde ^kx son las neuronas de una capa, k, y ^kW los pesos que las conectan con la siguiente. El concepto es similar a la interconexión de neuronas en un cerebro biológico.

Los datos de entrada pueden ser variados, desde una serie de valores medidos de alguna magnitud física, píxeles de una imagen, texto tokenizado, valores de una señal que va cambiando con el tiempo como un archivo de audio, etc. La cantidad de datos en cada muestra define el número de neuronas que tiene la capa de entrada, ya que sus neuronas tomarán esos valores. La señal entre neuronas se propagará entre las capas ocultas de la red hasta llegar a la última capa, Figura 3.1. La cantidad de neuronas de la última capa define la cantidad de valores de salida siendo esta dependiente del tipo de problema que se pretenda resolver.

- Regresión: El modelo debe obtener uno o varios valores (una o varias neuronas de salida) siendo estos continuos. Algunos ejemplos son el predecir la cantidad de precipitación que caerá mañana en una zona, la temperatura que hará, precios en la bolsa, estimar la edad de alguien a partir de una foto o las previsiones de venta de una empresa. Así pues, la red tendrá en su última capa tantas neuronas como valores deba estimar.
- Clasificación: El modelo debe asociar los datos de entrada con una clase, es decir, debe calificarlo en un tipo u otro. Un ejemplo sería clasificar una imagen según qué animal aparece, concluir qué sentimiento presenta principalmente un texto escrito por alguien o analizar si habido fraude fiscal o no en función de los movimientos

financieros. Típicamente, en estos casos la red presenta en su capa de salida tantas neuronas como clases, n_{clases}, pueda concluir tomando valores en el rango $[0, 1]$ representando la probabilidad o confianza de encontrarse ante esa clase u otra.

Ahora bien, el proceso de propagación de la señal entre neuronas consiste en conectar cada neurona, ${}^{k}x_i$ con todas las de la capa siguiente, ${}^{k+1}x_j$, de tal forma que una sola neurona, ${}^{k+1}x_j$, recibe el valor de todas las anteriores, ${}^{k}x$. Cada una combina dichos valores de una manera muy concreta de acuerdo con la Ec. 3.1.

$$ {}^{k+1}x_j = f(\sum_i {}^{k}W_{ij}\,{}^{k}x_i + \theta_j) = f({}^{k}W_j \cdot {}^{k}x + \theta_j) \tag{3.1} $$

De esta forma, cada neurona, ${}^{k+1}x_j$, de una capa, $k+1$, recibe el valor de todas las de la capa anterior, ${}^{k}x_i$, multiplicadas por un peso, W_{ij}. A su vez, a esta suma se le añade un valor llamado *bias* o sesgo, θ_j. Hasta aquí se intuye que la propagación de los valores entre neuronas de principio a fin es un conjunto de multiplicaciones y sumas, es decir, transformaciones lineales. Esto matemáticamente limitaría la capacidad de predicción del modelo a un simple ajuste lineal, lo cual no requeriría más de una capa para lograrlo. Ante esta situación resulta indispensable introducir alguna transformación no lineal en el proceso, siendo este el papel de la función f de la Ec. 3.1. Estas introducen no-linealidades y denominan funciones de activación, Tabla 3.1.

Función	Definición	Derivada	Rango	Gráfica
Sigmoide	$\sigma(x) = \frac{1}{1+e^{-x}}$	$\sigma'(x) = \sigma(x)(1-\sigma(x))$	$(0,\ 1)$	
ReLU	$\mathrm{f}(x) = \max(0, x)$	$\mathrm{f}'(x) = \begin{cases} 0, & \text{si } x < 0 \\ 1, & \text{si } x \geq 0 \end{cases}$	$[0, \infty)$	
Leaky ReLU	$\mathrm{f}(x) = \begin{cases} ax, & \text{si } x < 0 \\ x, & \text{si } x \geq 0 \end{cases}$	$\mathrm{f}'(x) = \begin{cases} a, & \text{si } x < 0 \\ 1, & \text{si } x \geq 0 \end{cases}$	$(-\infty, \infty)$	
Tanh	$\mathrm{Tanh}(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$	$\mathrm{Tanh}'(x) = 1 - \mathrm{Tanh}^2(x)$	$(-1, 1)$	
Softmax	$\mathrm{f}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$	No es relevante para el cálculo	$(0, 1)$	No tiene

Tabla 3.1: Principales funciones de activación. Estas introducen no linealidades en las operaciones que realiza la red.

Todo este proceso descrito en el que se va calculando capa tras capa los valores que debe recibir cada neurona en función del valor de las anteriores y de los pesos asociados se conoce como *forward propagation* o propagación hacia delante. Como cabe esperar, en las redes más grandes con gran cantidad de capas ocultas (de ahí el nombre de *deep*

learning) el número de parámetros puede estar entorno a los millones, por ello, en muchas ocasiones es necesaria cierta potencia de cómputo y una memoria principal del ordenador considerable.

Como se advertía, la red requiere un proceso de aprendizaje en el que analice los ejemplos de un *dataset* y aprenda una manera de llegar a los resultados correspondientes o etiquetas *labels*, ajustando y encontrando los valores de W_j y θ_j óptimos que minimizan su error. Este proceso de *training*, es decir, el ajuste paramétrico, consta de varias fases.

- 1. Elección de hiperparámetros: Son aquellos parámetros que caracterizan al modelo, que no son ajustados en el proceso de entrenamiento y que son definidos antes de comenzar la corrección del resto de parámetros. Por ejemplo, podría incluirse en esta categoría la cantidad de capas ocultas que tiene la red o qué funciones de pérdida usar, es decir, elecciones que no van a cambiar durante el entrenamiento. Destacar en este punto la partición del dataset en *train*, *validation* y *test*. Esto resulta fundamental para asegurar la eficacia en la práctica del modelo entrenado tanto en *deep learning* como en el *machine learning* en general. El conjunto de *train* suele representar la mayor parte de los ejemplos del total del *dataset* disponible siendo aquellos ejemplos de los que el modelo va a aprender. Cada vez que ha procesado todos los datos del *train* se dice que ha completado una época. Los entrenamientos requieren de múltiples épocas para llegar a encontrar los parámetros óptimos. Los conjuntos de *validation* y *test* son conjuntos más reducidos y tienen en común que el modelo nunca aprenderá de ellos, pero sí servirán para medir la bondad del modelo. En concreto, el *validation* permite poner a prueba al modelo tras cada época, proporcionando el error que comete al predecir sobre datos que no ha visto nunca. De esta forma se dispone de un seguimiento su rendimiento durante el entrenamiento. Este conjunto también ayuda a realizar una posterior optimización del resto de hiperparámetros tratando de obtener el mejor modelo minimizando el error en este conjunto. Por último, el *test* es una última prueba al modelo final, de nuevo, con ejemplos que nunca ha visto.

- 2. Forward propagation: Como se ha advertido, los cálculos realizados para obtener el resultado a partir de los datos de entrada se denomina *forward propagation*. Esto es necesario durante el entrenamineto ya que para ajustar sus parámetros debe comparase la predicción del modelo con los resultados esperados, es decir, las etiquetas, *labels*. Típicamente, este paso se realiza con varios ejemplos vez, es decir, un *batch* de casos.

- 3. Cálculo del error: Disponiendo ya del resultado de un *batch*, y_{output}, y sus respectivas etiquetas, y_{label}, se plantea aquí la comparación entre ambas. Si son muy distintas y no coinciden, el error debe ser alto, si el resultado es acertado el error

debe ser bajo. Aunque esto sea evidente, no es trivial la elección de la métrica que cuantifica esta diferencia. La función que calcula este error se denomina función de pérdida o *loss*. A pesar de que todas definen el error, algunas son más representativas que otras según el tipo de problema que deba resolver la red. Por ejemplo, para una regresión es razonable usar el MSE (*Mean Squared Error*, en cambio para una clasificación es adecuada la CE, *Cross Entropy*, Table 3.2. La elección de una métrica de error acertada influirá en la rapidez del proceso de ajuste de parámetros y la calidad final del modelo.

Función	Definición	Rango
Cross Entropy	$H(y,\hat{y}) = -\sum_{i=1}^{n} y_i \log \hat{y}_i$	$[0,\infty)$
Mean Squared Error	$\text{MSE}(y,\hat{y}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$	$[0,\infty)$
Mean Absolute Error	$\text{MAE}(y,\hat{y}) = \frac{1}{n}\sum_{i=1}^{n}\lvert y_i - \hat{y}_i\rvert$	$[0,\infty)$
Binary Cross Entropy	$BCE(y,\hat{y}) = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log \hat{y}_i + (1-y_i)\log(1-\hat{y}_i)]$	$[0,1]$
Intersection over Union	$\text{IoU}(y,\hat{y}) = 1 - \frac{\text{TP}}{\text{TP+FP+FN}}$	$[0,1]$

Tabla 3.2: Principales funciones de pérdida en deep learning. Estas funciones miden la discrepancia entre las predicciones y los valores reales.

- 4. Backwards propagation: Una vez estimado el error cometido por el modelo en un *batch* debe realizarse la corrección de sus parámetros. Se parte de una métrica escalar de error estimada a la salida del modelo, por ello, el cálculo de cuánto debe corregirse el valor de cada parámetro, $\Delta^k w_{ij}$, comienza por la última capa, propagándose hacia las primeras capas, de ahí el nombre de *back propagation*. De esta forma, el ajuste de cada parámetro depende de cuánto afecta este al error del resultado. Este es un cálculo bastante costoso ya que, como se advertía, los modelos de *deep learnign* pueden tener millones de parámetros. Matemáticamente, la dependencia que tiene la función de *loss*, $\mathcal{L}$, con respecto a cada parámetro, ${}^k w_{ij}$, se refleja en una derivada. Así pues, la corrección de cada uno, $\Delta^k w_{ij}$, toma la forma dada por la Ec. 3.2.

$$\Delta^k w_{ij} = -\eta \frac{\partial \mathcal{L}}{\partial^k w_{ij}} \tag{3.2}$$

Siendo ${}^k w_{ij}$ un peso que recibe el valor de la neurona i de la capa k al propagarse a la neurona j de la siguiente capa, $k+1$. La derivada parcial de la Ec. 3.2 puede extenderse en más derivadas parciales de acuerdo con la regla de la cadena ya que este parámetro influye en las capas siguientes hasta llegar a la última, apareciendo en el proceso la Ec. 3.1. Por este motivo, resulta útil definir funciones de

activación, f, con derivadas fácilmente computables, Tabla 3.1. Destaca el factor de proporcionalidad η, el cual regula la cantidad de cambio que sufre el parámetro, de ahí su nombre *learning rate*. A su vez, la manera en la que se calcula el ajuste de los parámetros puede sofisticarse definiendo en los hiperparámetros el optimizador. Un añadido puede ser el término de memoria de cambios en *batches* anteriores como el *momentum*, α, Ec. 3.3, o corregir la función de *loss* con un término de regularización, λ castigando un crecimiento excesivo de los parámetros, Ec. 3.4.

$$\Delta^k w_{ij}(t) = -\eta \frac{\partial \mathcal{L}}{\partial^k w_{ij}} + \alpha \Delta^k w_{ij}(t-1) \tag{3.3}$$

$$\mathcal{L}' = \mathcal{L} + \lambda \sum_{k,i,j} {}^k w_{ij}{}^2 \tag{3.4}$$

Al final se trata de minimizar el valor de una función que implícitamente depende de tantos parámetros, n_p, como tenga la red, ${}^k w_{ij}$, tal que,

$$\mathcal{L}(y_{output}({}^k w_{ij}), y_{label}) : \mathbb{R}^{n_p} \to \mathbb{R} \tag{3.5}$$

Este procesamiento de un *batch* y el consiguiente ajuste por *back propagation* es un proceso iterativo, que se repite con todos los *batches* del dataset. Las correcciones de los parámetros y la minimización de la función de pérdida es una optimización realizada de manera discreta, recibiendo el nombre de *gradient descent* o descenso de gradiente. Puede visualizarse en la Figura 3.2.

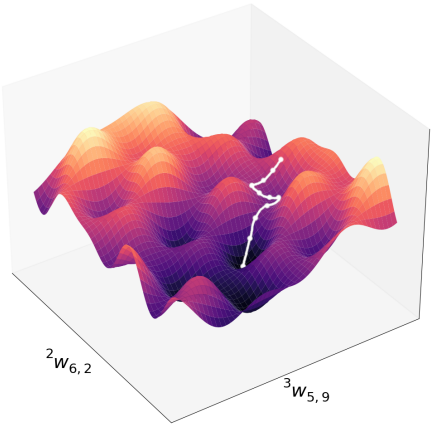


Figura 3.2: Representación de un función de *loss* para sólo dos parámetros de la red, ${}^2w_{6,2}$ y ${}^3w_{5,9}$, junto a una curva optimizando la función hacia su mínimo mediante *batch gradient descent*.

- 5. Validation: Tras haber procesado todos los *batches* del dataset, el modelo ha entrenado con todo el dataset una vez. A esto se le llama época. De esta manera, el entrenamiento consiste en realizar épocas sucesivamente e ir procesando el dataset varias veces. Esto aboca a reducir en cada una de ellas el error cometido en el conjunto de datos de *train*. Este conjunto puede ser limitado y no ser representativo de todos los casos posibles. En tal caso, es posible que el modelo haya aprendido

a acertar excesivamente bien en este conjunto, estimando patrones y reglas que lo alejan de la capacidad de generalizar y ser útil en otros datos que nunca haya visto. Por tanto, resulta necesario realizar pruebas o validaciones al modelo tras cada época. Para ello está el dataset de *validation*, evidenciando el desempeño que tendrá realmente ante nuevos casos. Tras cierto número de épocas cuando el error en el *validation* sea mínimo, el estado del modelo es guardado, evitando el llamado *overfitting* en el que el modelo llega a "aprenderse" los datos del conjunto de *train* dejando de aproximar la regla general. En tal caso, se apreciaría un continuo descenso del error en el *train* y un aumento del mismo en el *validation*, perdiéndose la generalidad y robustez del modelo.

Tras completar un entrenamiento son guardados los pesos o parámetros óptimos del mismo, siendo estos los resultados del ajuste.

3.2. Sistemas de visión basados en inteligencia artificial

En los últimos años el *deep learning* a revolucionado la visión artificial debido a la llegada de modelos muy potentes de procesamiento de imágenes los cuales resuelven tareas antes inalcanzables como la detección de objetos muy diversos o la propia generación de imágenes.

La visión artificial o visión por computadora lleva años resolviendo problemas de inspección visual. Antes de la llegada del *deep learning* a la visión, la única solución era programar una lógica explícita de procesado de imágenes que consiguiese extraer las características que definen aquello que se pretende detectar destacando umbralizaciones de intensidad, color y área; búsqueda de patrones; detección de bordes; necesidad de fuertes preprocesados, etc. Por tanto, se requerían cálculos ciertamente sofisticados y específicos para una tarea y condiciones concretas. El algoritmo era muy sensible a cambios visuales como iluminación, posición, perspectiva o tamaño.

Es aquí donde resultan muy útiles los modelos de *deep learning* los cuales consiguen, como se advertía, estimar una regla general que resuelve el problema extrayendo unas características de alto nivel y ciertamente abstractas que le permiten aproximar la solución. De nuevo, esto es aprendido mediante el entrenamiento con datos de ejemplo, en este caso imágenes.

Los problemas que resuelven los modelos de visión son los siguientes, Figura 3.3.

- Clasificación: Se trata de asociar una imagen a una clase, es decir, concluir si toda la imagen es de un tipo u otro.
- Detección: Estimar qué hay en la imagen y donde está (instancia u objeto), es decir, clasifica secciones en las que hay algo a detectar diciendo además dónde se ubica

en la imagen. Típicamente esto se indica con un *bounding box*, en la que se incluye coordenadas (x, y) y ancho y alto (w, h). Esto es computacionalmente más costoso que la simple clasificación.

- Segmentación semántica: Consiste en ofrecer una máscara, es decir, una imagen del mismo tamaño que la tomada como input pero binaria (valores $[0, 1]$) definiendo qué zonas de la imagen refieren a un mismo concepto. En este tipo de segmentación no se distingue entre objetos individuales de la misma clase en una imagen, sino que se centra en la categorización a nivel de píxel.
- Segmentación de instancias: En añadido a la detección en la que se distinguen objetos individuales (instancias u objetos) y se proporciona posición y tamaño del objeto, aquí se añade el perfilado o contorno de la detección. En este enfoque, esto va un paso más allá que la segmentación semántica, ya que, como la detección, cada objeto se trata como único y se le asigna un identificador.
- Otros: Cada vez son más las tareas que resuelven los modelos de procesamiento de imágenes basados en *deep learning*. Algunos ejemplos serían: generación de imágenes, aumento de resolución, detección de anomalías, estimación de pose, reconocimiento de emociones a partir de expresiones faciales, entre otras.

Figura 3.3: Tipos de problemas en visión artificial. Notar las diferencias en complejidad de los resultados que ofrece.

En este trabajo se propone un problema de segmentación semántica como eje principal mediante un modelo con arquitectura *U-Net*. También se comparan los resultados obtenidos con el modelo *YOLOV8-seg* el cual resuelve una segmentación por instancias.

Ahora bien, es necesario entender qué estructura y valores definen una imagen. Estas están compuestas por píxeles, es decir, una partición discreta del plano, siendo la unidad mínima de información. Así pues, una imagen es una matriz con una altura, n_h, y una anchura, n_w, de píxeles, es decir, una resolución $(n_h \times n_w)$. Cada uno de los píxeles tiene un color. Si la imagen es en escala de grises, un solo canal definirá la intensidad de gris teniendo la imagen $(n_h \times n_w \times 1)$ valores. Si es una imagen a color, son necesarios tres canales para definir cada píxel, habiendo entonces $(n_h \times n_w \times 3)$ valores. Estos tres

canales significan cantidades distintas según el formato: RGB (el conocido rojo, verde y azul), BRG, HSV, etc.

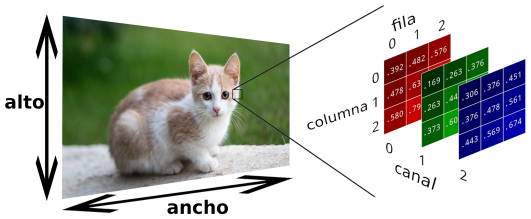


Figura 3.4: Una imagen es una matriz con información espacial y tantos canales como colores se combinen para dar lugar a cada píxel.

Ahora bien, las neuronas dispuestas en las capas densas de un perceptron multicapa resultan indistinguibles, es decir, a priori una ordenación distinta de los datos de entrada no tendría importancia, en cambio a la hora de procesar imágenes no sería conveniente. Es razonable intuir que parte de la información que presentan los píxeles viene dada por su ubicación respecto a los demás, es decir, hay correlación de algún tipo entre píxeles vecinos ya que se trata de información espacial, Figura 3.4. Resulta necesaria una manera de propagar los valores de las neuronas sin perder esa información. Por ello, los modelos de *deep learning* de procesamiento de imágenes proponen unas transformaciones de los valores distintas a las descritas para el perceptron multicapa.

El principal cambio radica en la presencia de capas convolucionales. El cálculo realizado en ellas involucra la operación de convolución. Esta implica superponer una pequeña matriz ("filtro" o "kernel"), $w_{i,j}$, en una región de la imagen y realizar una multiplicación punto a punto entre los elementos del filtro y los píxeles correspondientes en la imagen. Luego, se suma el resultado de estas multiplicaciones y se coloca el resultado en la posición correspondiente en una nueva matriz, llamada "mapa de características" o "capa de activación", $C(\nu, \mu)$, siendo esta, de nuevo, una matriz con información espacial, Figura 3.5 y Ec. 3.6. Repitiendo el proceso moviendo el filtro por toda la imagen en pequeños pasos de tamaño conocido como *stride* se va construyendo el mapa de características siguiente.

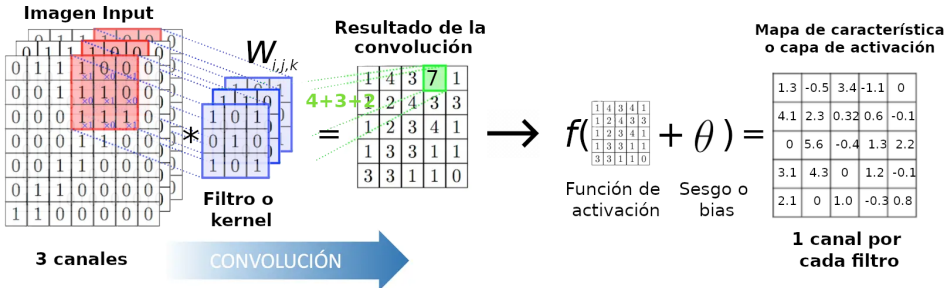


Figura 3.5: Esquema del proceso de convolución y generación de un mapa de características con un filtro.

$$C(\nu,\mu) = f\left(\left(\sum_k \sum_j \sum_i I_k(x+i, y+j)\cdot W_{i,j,k}\right) + \theta\right) \tag{3.6}$$

De esta manera, en la nueva "imagen de características", $C(\nu,\mu)$, habrá tantos canales como filtros hubiese aplicados a la capa de activación anterior. Esta operación es definida por ciertos parámetros.

- Tamaño del kernel: Es la dimensión de las matrices que convolucionan sobre las imágenes o capas de activación. Al ser una convolución discreta, típicamente es un numero impar, dejando claro el centro de la misma. En el ejemplo de la Figura 3.5 el kernel tiene un tamaño de 3×3.
- *Stride*: Es la cantidad de píxeles o índices que se desplaza el kernel tras cada convolución. Un *stride* grande dará lugar a un mapa de características de menor tamaño.
- *Padding*: A la hora de realizar la convolución en los bordes de la imagen puede ser necesario extender la imagen más allá de sus dimensiones originales ya que, si no, parte del kernel no tendría píxeles con los que realizar la operación. Esta extensión de la imagen puede realizarse de varias maneras, como espejando la imagen en el borde, dando valores nulos o replicando el valor del píxel del borde.
- Función de activación: De nuevo, es necesaria una transformación no lineal, por ello, tras sumar la convolución realizada en un punto de todos los canales de la imagen o mapa de características y sumarle el sesgo o *bias* se aplica la función de activación, siendo esta alguna de las ofrecidas en la Tabla 3.1.
- Parámetros a ajustar: En las capas convolucionales los parámetros a ajustar no son otros que los pesos que presentan los *kernels* o filtros y sus sesgos. Cada canal de la imagen de partida tiene asociada una matriz, k, de dimensiones $i \times j$, estas componen un filtro, $w_{i,j,k}$, con una cantidad de parámetros $i \times j \times k$ y un parámetro más que hace de sesgo o *bias*, θ. A su vez, puede haber varios filtros dando lugar a cada uno de los canales del mapa de características siguiente.

De esta manera, la finalidad principal de las capas convolucionales es aprender a extraer características relevantes de las imágenes. En lugar de tratar cada píxel de manera aislada, estas consideran relaciones espaciales y patrones locales en las imágenes, independientemente de su posición. En añadido, junto a las capas convolucionales se suelen considerar operaciones de *pooling*, las cuales reducen las dimensiones de los mapas de características. Estas realizan una media de cierta sección (*meanpooling*) o toman su máximo (*maxpooling*), por ejemplo. Con esta idea en mente, las redes neuronales convolucionales, CNN, reducen la dimensionalidad del problema llevándolo a conceptos de más alto nivel a medida que avanzan las capas, reduciendo su extensión i, j, pero aumentando

la cantidad de canales k como se aprecia en la Figura 3.6. Las estructuras que realizan esta labor de extracción de carácteristicas son denominadas *backbones*, destacando VGG, ImageNet, AlexNet, ResNet o Inception [14].

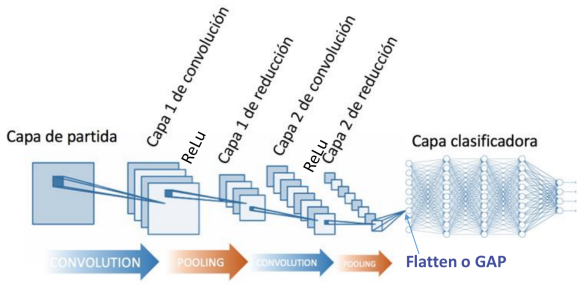


Figura 3.6: Esquema del *backbone* de una red neuronal convolucional, CNN. Notar el aumento de la cantidad de canales, pero la disminución el tamaño espacial de los mapas de activación, *downsampling*.

4. Aplicación de sistemas de ML a la detección de nódulos pulmonares: el dataset LIDC-IDRI

En el paradigma del aprendizaje automático, los datos desempeñan un rol primordial, siendo esenciales para el diseño, desarrollo y validación de modelos. La naturaleza misma del aprendizaje automático implica la extracción y generalización de patrones a partir de datos de entrenamiento, lo que instaura a estos últimos como el componente fundamental. En los últimos años el *machine learning* se ha ganado una notable presencia en cuestiones médicas, radiología y diagnóstico en general [15], mostrando una gran eficiencia y robustez en condiciones reales en la práctica, y aportando gran valor a los programas CAD (*Computer-Aided Detection*) [16]. Por tanto, resulta de gran utilidad disponer de amplios datasets con ejemplos de pruebas médicas, en este caso imagenes TAC, junto a un diagnóstico tomado como "verdad" ofrecido por el conocimiento experto, un radiólogo por ejemplo, es decir, los datos (imagenes) y las etiquetas (diagnóstico).

La iniciativa de ofrecer al público datasets con casos diagnosticados vino de la mano de la *Japanese Society of Radiological Technology* (JSRT) motivada por disponer de ejemplos con fines educativos, de entrenamiento e investigación. De esta manera, se ofreció un dataset de 247 radiografías de torax en el año 2000 [17], siendo esta prueba radiológica la más frecuente. Con el tiempo, numerosas instituciones se han sumado en colaboraciones para componer datasets de mayor contenido y calidad, siempre respetando la anononimización de los pacientes. Uno de los pasos importantes fue el uso del formato DICOM (*Digital Imaging and Communications in Medicine*), exigiendo a las imágenes estandarización y un control de calidad de los datos.

Ahora bien, el dataset más grande de etiquetado en medicina es *the Lung Image Database Consortium Image Collection* (LIDC-IDRI) ofrecido por *The Cancer Imaging Archive* (TCIA) [5], un proyecto mantenido por instituciones y hospitales estadounidenses junto a contribuciones de centros de todo el mundo. Su desarrollo comenzó en 2001 mediante la colaboración de cinco instituciones: Weill Cornell Medical College, University of California, Los Angeles, University of Chicago, University of Iowa y University of Michigan [16]. Este, inicialmente, era un repositorio de imágenes de tomografías computerizadas de torax, junto a metadatos de la toma de las imágenes (ancho de las *slices*, algoritmo de reconstrucción, etc.), información del paciente (edad, género, etc.) e información médica del diagnóstico.

Más adelante en 2004 el *National Institutes of Health* (FNIH) con la finalidad de acelerar el avance científico en cuestiones sanitarias crearon IDRI, (*Image Database Resource Initiative*). Este proyecto unió a dos nuevos centros sanitarios y a ocho compañías de imagen médica. Esto aportó 300 nuevos casos con sus respectivas tomografías computerizadas, siendo anotado, curado y preprocesado de la misma manera que los casos ya presentes en LIDC, denominándose el dataset en este punto como LIDC-IDRI *Database*.

El LIDC-IDRI *Database* contiene 1018 escáneres de tomografías computerizadas, CT, de 1018 pacientes distintos en formato DICOM, `.dcm`. Un escaneado da lugar a cierta cantidad de *slices* o imágenes del tórax, n_s, dependiente del espaciado entre ellas. Típicamente estas suelen tomarse cada pocos mm, reconstruyendo el tórax con unas cientos de fotos, n_s. A su vez, cada imagen resulta ser tener un solo canal, es decir, están en escala de grises, dando su intensidad información de la opacidad que tienen unos tejidos u otros. Cada una de estas *slices* tiene un tamaño constante de 512 píxeles. El propio formato DICOM, guarda en sus metadatos la escala con la que interpretar las distancias sobre el plano de las imágenes, (x, y) y la profundidad (distancia entre *slices*), z. Las variaciones en otros parámetros de la toma de imágenes de un paciente a otro es debida a que cada caso no fue registrado para la construcción de este dataset, sino que es resultado de la agrupación de escáneres realizados por dispositivos y configuraciones distintas en distintos centros [16].

Por otro lado, además de las imágenes de las tomografías, resulta de vital importancia las anotaciones del diagnóstico, es decir, la etiquetas que indican los posibles nódulos propuestas por los expertos. En esta tarea fueron involucrados 4 radiólogos de distintos centros que, aunque acordaron el criterio de etiquetado, cada uno presenta su cierta subjetividad. El proceso de etiquetado resumidamente consistió en una primera fase "ciega" de etiquetado, donde cada radiólogo identificó los nódulos, estimando su ubicación y bordes junto a la clasificación en masas < 3 mm o > 3 mm. Tras esto, se realizó una segunda fase, esta vez "no ciega", en la que cada radiólogo se le indicaba su anotación anterior y la de algún compañero, pudiendo corregir la definida en la primera fase. A su vez, en esta fase

se estimaban parámetros médicos de cada nódulo como esfericidad, solidez, espiculación, malignidad, benignidad, etc. Toda esta información de posicionamiento de los nódulos se almacenó en archivos XML, de extensión `.xml`, formato convertido en estándar para el intercambio de datos, especialmente en entornos web. Así pues, considerando nódulo como aquello que ha sido etiquetado por al menos un radiólogo se tienen un total de 7371 nódulos en todo el dataset.

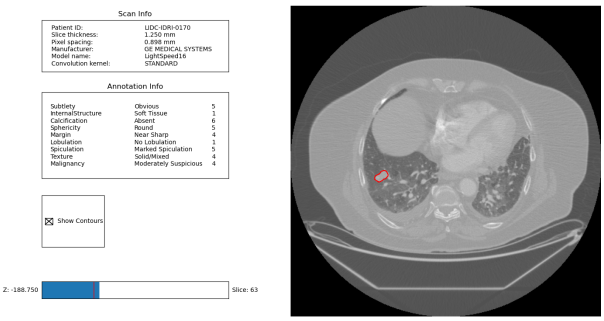


Figura 4.1: Ejemplo de anotación de una *slice*: parámetros médicos y ubicación del nódulo con su contorno. Se corresponde con la *slice* 63 del paciente con ID LIDC-IDRI-0170.

En cuanto a la licencia de uso del dataset, como se puede apreciar en la página de TCIA [5], es CC BY 3.0. (*Creative Commons Attribution*). Es un tipo de licencia que permite a los usuarios compartir, adaptar y utilizar el conjunto de datos, incluso con fines comerciales, siempre y cuando se atribuya el crédito correspondiente al creador original. En el contexto de CC BY 3.0, el número 3.0 se refiere a la versión específica de la licencia *Creative Commons* que se aplica. Se trata pues de un dataset totalmente abierto al público con el objetivo de impulsar y facilitar la investigación y desarrollo. Desde la propia página web del dataset, el propio *The Cancer Imaging Archive* (TCIA) [5], anima a realizar estudios con él y publicarlos [18].

4.1. Acceso al dataset: NBIA y `pyLIDC`

El dataset LIDC-IDRI está disponible al público a través de la web de *The Cancer Imaging Archive* (TCIA) [5]. Desde el navegador es posible seleccionar aquellos pacientes de interés. Cada uno recibe un identificador único que sigue la estructura LIDC-IDRI-XXXX, donde XXXX es un número del 0001 al 1018 (uno para cada CT del dataset). A su vez, es posible abrir un visualizador (*OHIF Viewer TCIA ALPHA*, similar a un software CAD, que facilita la inspección visual de la CT. Tras seleccionar los pacientes de interés siguiendo una dinámica similar a "carro de la compra" se puede aceptar la descarga y se obtiene un fichero de formato, `.tcia`, con la *query* u orden de descarga de los casos seleccionados. Este archivo es referido como *manifest*) y permite, mediante el programa NBIA *Data Retriever* el cual es un software desarrollado por Instituto Nacional del Cáncer (NCI), la descarga de datos médicos del repositorio. Este dispone de una interfaz en la

que seleccionar el fichero de descarga, `.tcia`, a la vez que realizarlo programáticamente desde terminal ejecutando `/opt/nbia-data-retriever/nbia-data-retriever --cli 'path/to/manifest-1683454688195.tcia' -d /path/to/save/folder`. Este proceso puede apreciarse en la Figura 4.2. El dataset completo presenta un tamaño en el disco duro de 137 GB.

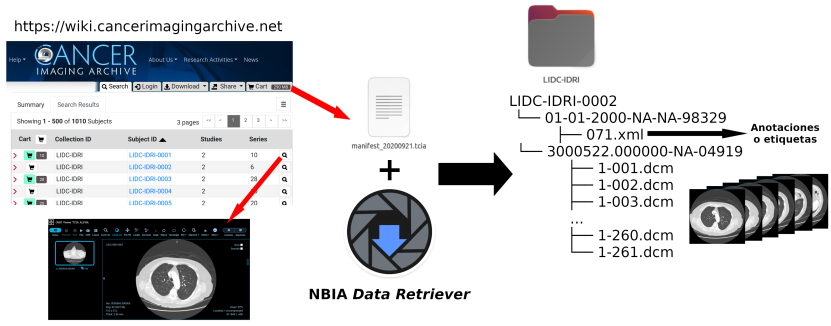


Figura 4.2: Proceso de adquisición del dataset LIDC-IDRI [5].

Ahora bien, para manejar imágenes médicas de python existe la librería `pyDICOM` [1]. Esta, de manera muy similar a la librería más conocida de visión artificial, `opencv`, permite leer, gestionar y mostrar imágenes con extensión `.dcm`, `pydicom.dcmread("path/to/image/dicom.dmc")`. También ofrece los metadatos asociados al paciente y a la toma de la imagen siendo estos atributos del objeto de python. En el dataset LIDC-IDRI, los campos que serían identificadores del paciente desde el punto de vista de la privacidad están vacíos. En añadido, resulta de mayor comodidad la librería de python `pyLIDC` [2]. Está pensada para facilitar el acceso y gestión de este dataset en particular de manera programática. Para hacer uso de ella es necesario establecer en la raíz del ordenador un archivo oculto con el nombre `/home/user/.pylidcrc`, en el cual está la dirección en la que se ha descargado el dataset y donde irá a leer los datos la librería `pyLIDC`. Esta funciona internamente con `SQLAlchemy`, de tal forma que las *queries* se realizan mediante un mapeo objeto-realacional (ORM), lo cual permite desde lenguajes de programación basados en objetos como python trabajar con bases de datos relacionales SQL (*Structured Query Language*). Es posible realizar consultas por tomografías mediante el objeto `pylidc.Scan`, por anotaciones con `pylidc.Annotation`, realizar la operación `join` entre ambos, acceder a las imágenes y visualizarlas, abrir una pantalla interactiva para recorrer las *slices*, visualizar una composición 3D del volumen etiquetado como nódulo, obtener *bounding boxes*, calcular las máscaras que definen las regiones con tumor, etc. Un ejemplo de alguna de estas posibilidades se ofrece en el *script* de la sección Anexos. Notar que es posible definir la forma del nódulo según cuántos radiólogos la han considerado como tal, véase `clevel=0.5` en la agrupación de anotaciones realizada al final del *script*.

[1] Documentación de la librería de python `pyDICOM`: https://pydicom.github.io/
[2] Documentación de la librería de python `pyLIDC`: https://pylidc.github.io/

4.2. Trabajos previos con el dataset LIDC-IDRI

El dataset LIDC-IDRI es un valioso conjunto de ejemplos de tomografías computerizadas de tórax acompañado de las etiquetas fruto del diagnóstico de 4 radiólogos. Por ello, es razonable el gran número de estudios que parten de este dataset, especialmente en el campo de la inteligencia artificial.

Desde el punto de vista del *machine learning*, son variadas las tareas que aspiran a resolver los modelos obtenidos en la literatura [19]. Una primera distinción entre estudios previos puede ser la naturaleza del modelo propuesto, es decir, si la extracción de características está basada en *deep learning* o, modelos que requieren de una ingeniería de extracción de características previa. Entre estos últimos se incluyen técnicas como *support vector machines* [20] [21], métodos basados en *ensembles* como *random forest* [22], arboles de regresión [23] realizando una extracción de características geométricas tanto 2D como 3D o basados en el conocimiento anatómico (*skeletonization* [24]. Todas ellas tienen en común la necesidad de preprocesar la imagen, llegando a segmentar los pulmones (algunos proponen *thresholding* o umbralización), posteriormente realizan otro tratamiento de *thresholding*, filtrado por forma, evaluando y clasificando cada candidato con alguno de los modelos nombrados.

Muchos de los artículos aquí nombrados resumen la bondad de su modelo mediante el coeficiente DICE. En el ámbito de las imágenes, da cuenta de la similitud entre el área segmentada, a_s, y etiquetada, a_l, de las segmentaciones Ec. 4.1.

$$DICE = 2\frac{a_s \cap a_l}{a_s + a_l} \tag{4.1}$$

Los estudios más recientes exploran las capacidades del *deep learning* siendo esto parte del objetivo de este trabajo. Antes, conviene ahora distinguir qué tipos de tareas sea han tratado de resolver en la literatura. Algunos de ellos se limitan a realizar una tarea de clasificación binaria entre nódulo benigno o maligno como [25] o, más recientemente, [26] del 2023 consiguiendo precisiones entorno al $\approx 95\,\%$. El modelo recibiría un ROI (*region of interest*) tomada por ojo experto (radiólogo) recibiendo este una conclusión que le ayude a decidir sobre la gravedad del tumor que él ya ha detectado previamente. En concreto [26] realiza un recorte entorno cada tumor, lo cual es facilitado ya por la librería `pyLIDC` con el método `consensus(ann_clust, pad=[(64, 64), (64, 64), (0, 0)] )`, eliminan parte del tejido con estructura bronquial y evalúan clasificadores con *backbones* variados: VGG-16, VGG-19, DenseNet-121 and DenseNet-169 y ResNet.

Sin embargo, en este trabajo la tarea que se pretende resolver no es una clasificación de un tumor previamente identificado, el objetivo es detectarlos, siendo el modelo el que indique dónde debe dirigir la atención el radiólogo. De nuevo, hay muchas publicaciones

que ofrecen un algoritmo de detección de nódulos de pulmón apoyados en este dataset. De 2018 se tiene una aproximación a la tarea de segmentación mendiante una CNN, *convolutional neural network*, [28] la cual busca hiperparámetros mediante PSO (*Particle Swarm Optimization*). Esta es una técnica de optimización que realiza un muestreado de valores de manera aleatoria, lo cual, en muchas ocasiones, resulta muy eficiente a la ahora de recorrer rangos de valores en muchas dimensiones. En esa publicación consiguen una precisión del 97,62 % y una sensibilidad o *recall* del 92,20 %, ambas definidas en la sección 6.2, siendo valores muy altos. Destaca su esfuerzo por reducir falsos positivos, lo cual suele ser común en segmentación semántica de estructuras tan variables.

Por otro lado, en [27] de 2017, se propone un *Central Focused* CNN, *convolutional neural network*, para segmentar los nódulos. La red que usan clasifica cada píxel de cada imagen partiendo de un recorte de la misma y realizado convoluciones sobre ella en 2D y en 3D, es decir, con un kernel de 3D operando sobre un volumen y teniendo en cuenta así *slices* contiguas, Figura 4.3. Es razonable pensar que hay información médica en la morfología del bulto en las 3 dimensiones, no solo en una sección del mismo. Se consigue una sensibilidad del 92,75 % y un coeficiente de similaridad DICE de 80,02 % . Otros como [29] también optan por convoluciones en 3D, consiguiendo 96,64 % de sensibilidad y 71,43 % especificidad.

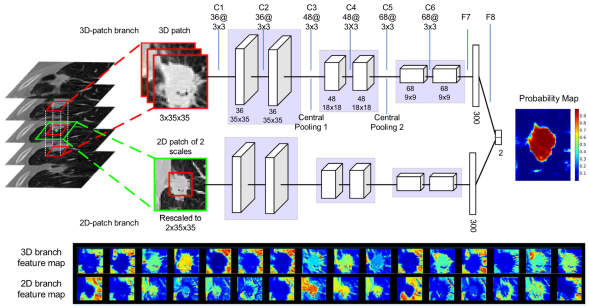


Figura 4.3: Clasificador de cada píxel mediante CNN en 2D y 3D en el modelo de segmentación semántica propuesto en [27]. Notar los mapas de características dados por las capas de activación tras las convoluciones realizadas por los kernels.

Este trabajo se centra en una arquitectura de modelo conocida como UNet como se explicará más adelante. En esta línea pueden encontrarse artículos que evalúan la eficacia de estos modelos para segmentar nódulos a partir de este dataset. Entre ellos puede nombrarse el estudio de 2021 [30] en el que se propone una UNet con capas convolucionales en 3D, junto al *backbone* ResNet50 alcanzando una precisión 77,8 % y un coeficiente DICE 83,04 %.

En cuanto a modelos que tomen como *input* una sola imagen, es decir, que presenten operaciones bidimensionales, es posible partir de artículos que desarrollan UNets para otras tareas en medicina. Por ejemplo, en [31] de 2015 de Olaf Ronneberger et al. se diseña y entrena una red con esta arquitectura al completo enfocada a segmentar células que conforman un tejido. Un paso más allá, volviendo al 3D es posible encontrar lo propio un año después, 2016, [32], compartiendo parte de los autores.

Por último, es posible nombrar una aproximación a una tarea muy similar a la aquí propuesta en la que se plantea la segmentación semántica de nódulos mediante una UNet 2D [33] de 2021. Aquí se consigue una precisión de 98,71 % y un coeficiente DICE de 75,0 %. Estos son métricas muy notables, pero hay que destacar, una vez más, el preprocesado que se hace. En este caso, la red no toma como *input* toda la imagen, sino que se realiza un recorte de la región de interés alrededor de cada nódulo, concretamente en una región de 64×64. Esto reduce la complejidad de la tarea ya que el modelo recibe una sección con la detección centrada, presentando una utilidad para el experto prácticamente de confirmación tras una previa identificación del posible nódulo.

5. Gestión y procesado de los datos: `processLIDC`

Un adecuado uso de los datos disponibles es vital en un proyecto de *machine learning*, por ello, para facilitar y automatizar el acceso a los pacientes del dataset, sus etiquetas y realizar el preprocesado de los datos se ha desarrollado un *software* de python: `processLIDC` [3]. En este destaca la clase `Patient(id_patient)`, la cual requiere solo un argumento para instanciarse: el identificador del paciente `id_patient: str`. Por ejemplo: `LIDC-IDRI-0674`, `LIDC-IDRI-0019`, etc.

La clase `Patient` al instanciarse realiza las *queries* necesarias a través de `pyLIDC` para obtener las imágenes y las etiquetas. Como se indicaba en la sección 4.1 Acceso al dataset: NBIA y `pyLIDC` puede definirse la región nodular de acuerdo a cuántos de los cuatro radiólogos lo consideraron como tal mediante el argumento `clevel` de la función `consensus()`. Se ha considerado un valor `clevel=0.5`, esto es, considerar aquellos píxeles que hayan sido identificados como nodulares al menos por dos radiólogos. Estas etiquetas o segmentaciones de los nódulos han sido tomadas para crear máscaras del tamaño de las seccione o *slices* de la tomografía, $512 \times 512 \times 1$ (imágenes en escala de grises: 1 canal). Las máscaras toman valor binario: 0 en los píxeles que no sean nodulares y 1 en los que sí. La instanciación de esta clase permite el acceso a los atributos `self.vol` (imágenes) y `self.mask` (etiquetas), Figura 5.1.

Figura 5.1: Atributos de la clase de python `Patient()` con las imágenes y las etiquetas (máscaras), $512 \times 512 \times 1$. Los píxeles blancos en las máscaras son los nódulos etiquetados.

[3] Repositorio de `processLIDC`: https://github.com/abelBEDOYA/processLIDC. Tabla 8.2 en Anexos.

Una vez se dispone de las imágenes es posible llevar a cabo un preprocesado de estas. Se ha considerado de utilidad realizar un escalado de los valores que definen la intensidad de gris de cada imagen de acuerdo a la Ec. 5.1.

$$t(x) = \begin{cases} 0 & \text{si } x < 0 \\ \log(x+1) & \text{si } x \geq 0 \end{cases} \tag{5.1}$$

Este escalado desplaza al 0 los valores negativos que no aportan información, siendo el espectro relevante las intensidades positivas. A su vez, el hecho de realizar una transformación logarítmica supone un aumento de contraste en las intensidades más reducidas, es decir, en tejidos más blandos y menos absorbentes. Sin embargo, supone la pérdida de contraste en las regiones de mayor intensidad, donde, presumiblemente, no es requerida la diferenciación de tejidos, por ejemplo, entre huesos y músculos, Figura 5.2.

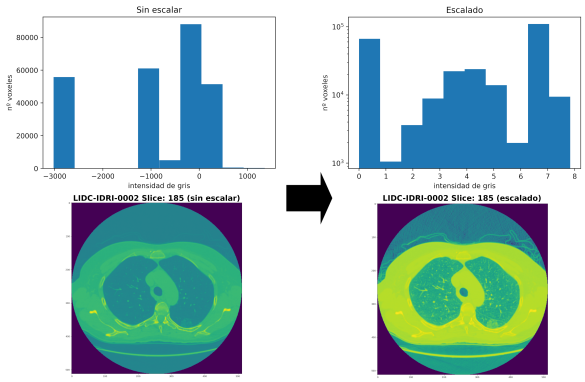


Figura 5.2: Resultado del escalado de los valores de intensidad de grises en las imágenes. Notar el nódulo del pulmón derecho cerca de la espalda, este se aprecia mejor en la imagen escalada.

Una vez realizada este proceso de ETL (Extraer, Transformar, Cargar) ya se dispone de las imágenes y las máscaras con los nódulos indicados fácilmente para ser usados en un entrenamiento o para ser visualizados. La clase `Patient()` permite visualizar selectivamente las *slices* a voluntad con `patient.imshow(slices=(184,185,186), scaled=True, labels=True)`. A su vez, también permite obtener una visión más general del paciente y el procesado llevado a cabo. Esta es una reconstrucción del cuerpo y los nódulos etiquetados a partir de las *slices* estimada con *thresholding* (umbralización). Este método toma aquellos voxeles que superen cierto umbral, o *threshold*, de intensidad. Se puede llamar a este plot dinámico mediante el método `patient.reconstruct_body(nodulos=True)`, Figura 5.3).

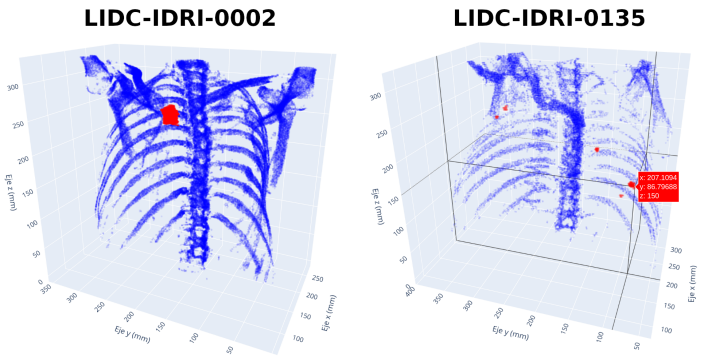


Figura 5.3: Interpretación del cuerpo y los nódulos etiquetados por los radiólogos a partir de las imágenes usando umbralización (*thresholding*).

6. Desarrollo del modelo

En esta sección se expone los modelos de *deep learning* con arquitectura *U-net* evaluados, su proceso de entrenamientos y cálculo de métricas del rendimiento del mismo. Todo el desarrollo aquí expuesto se ha realizado en ordenador con sistema operativo Ubuntu 22.04 con arquitectura x86_64 con una CPU de la marca Intel(R) Core(TM) modelo i9-10900 a 2.80GHz y una memoria RAM de 32 GB. Se disponía a su vez de una tarjeta gráfica (GPU) de la marca NVIDIA modelo GeForce RTX 3090 con una memoria VRAM de 24 GB. En cuanto al almacenamiento se dispuso de un disco de estado sólido, SSD, tipo NVMe (*Non-Volatile Memory Express*) de 1 TB. Este *hardware* ha resultado muy útil a la hora de resolver el cómputo exigido en los entrenamientos de los modelos, especialmente por la potencia de la GPU y su memoria, pudiendo cargar y procesar en ella *batches* con varias imágenes. Sin embargo, una vez obtenido el modelo entrenado, un ordenador más modesto con una gráfica menor o incluso computando con la CPU y cargando las imágenes en memoria RAM podría realizar una inferencia tardando unos pocos segundos.

En cuanto al *software* se ha desarrollado todo el código en un entorno de conda en lenguaje Python. El *framework* con el que se han evaluado los modelos propuestos es `pytorch`, versión `2.0.0+cu117` [4]

6.1. Aquitectura UNet

Los modelos con arquitectura UNet han demostrado su eficacia en tareas de visión artificial de segmentación semántica, detectando y delimitando con precisión objetos o regiones de interés en imágenes. Esta, como todo modelo de *deep learning* de procesamiento de imágenes, se basa en el uso de capas convolucionales. Consiguen reducir la

[4]Librerías y *frameworks* usados en Anexos. Tabla 8.1.

dimensión espacial de las capas de activación obteniendo mapas de características que representan cada vez conceptos de más alto nivel y complejidad. Esto se refleja en una aumento de la profundidad de las capas de activación, es decir, más canales. Este primer proceso es conocido como *downsampling* realizado por el *encoder* como se indica en la Figura 6.1 donde destacan capas convolucionales y operaciones de *max pooling*. De manera opuesta, tras el *downsampling* se sucede el proceso contrario, es decir, un aumento de la información espacial y una reducción de canales volviendo a generar mapas de características más extensos y de más bajo nivel, *upsampling*, realizado por el *decoder*. De esta forma, el *output* de la red es una imagen de las mismas dimensiones espaciales que la imagen original. Esta aspira a ser una máscara que delimite las regiones de interés, tomando valores en el rango $[0, 1]$. Para ello, la función de activación de la última capa es una sigmoide, Tabla 3.1. Como se aprecia en la Figura 6.1, esta estructura puede asemejarse a una forma de "U" de ahí su nombre, UNet. Destacar, además, las operaciones de concatenación, en la que se juntan mapas de características del proceso de *downsampling* con los del *upsampling*. De esta forma, la red puede tener en cuenta de manera más eficaz características anteriores así como detalles más finos para obtener una segmentación más precisa y fiel a la imagen original.

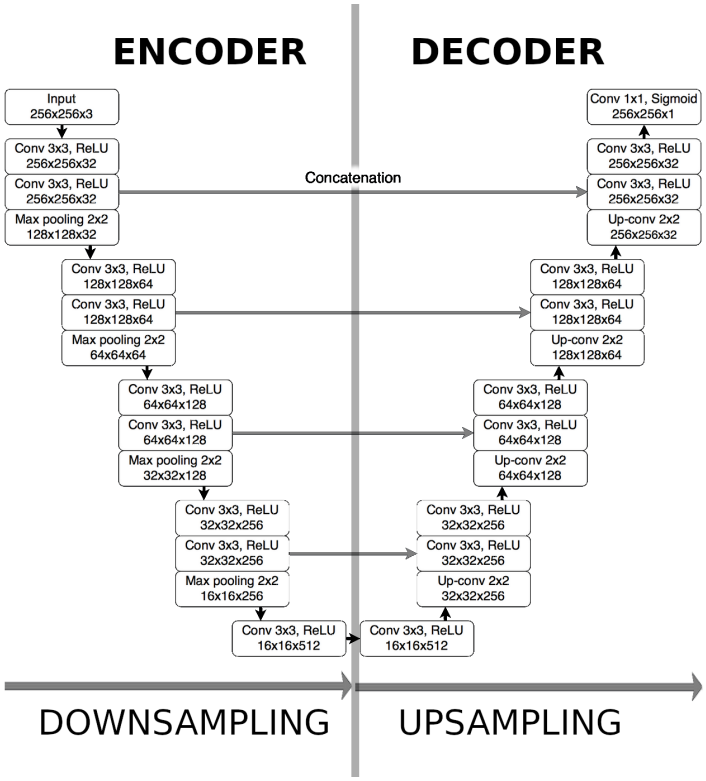


Figura 6.1: Esquema de la arquitectura de la UNet evaluada. Notar la fase de *downsampling* y *upsampling*, buscando reducir las dimensiones espaciales de los mapas de características y ampliarlos respectivamente, junto a las concatenaciones de mapas previos [34].

Se debe destacar que, en el proceso de *upsampling*, aparecen unas transformaciones que consiguen el efecto contrario de las capas convolucionales, estas son de convolución transpuesta, o *up-conv*. Como se ha explicado, se trata de aumentan la resolución espacial y reducir el número de canales, para ello, esta operación consiste en tomar cada píxel del mapa de activación como un peso por el que multiplicar el filtro o *kernel* imprimiendo sobre el nuevo mapa de características sus valores con un cierto *stride*. De esta forma, de cada píxel original se obtienen $h \times w$ valores, es decir, el tamaño del *kernel*.

6.2. Métricas de evaluación

Un entrenamiento de un modelo de *deep learning*, como se advertía en el apartado introductorio, es un proceso iterativo, en el que se procesa el dataset de *train* numerosas veces (épocas) realizando el ajuste de los parámetros del modelo. Tras cada época, el modelo se evalúa con un conjunto de datos que no ha visto nunca, el dataset de *validation*. Este permite realizar un seguimiento del progreso de aprendizaje intuyendo así su desempeño ante casos nuevos a medida que avanza el entrenamiento. Esto se refleja en la representación de las curvas de error frente al paso de las épocas, Figura 6.2.

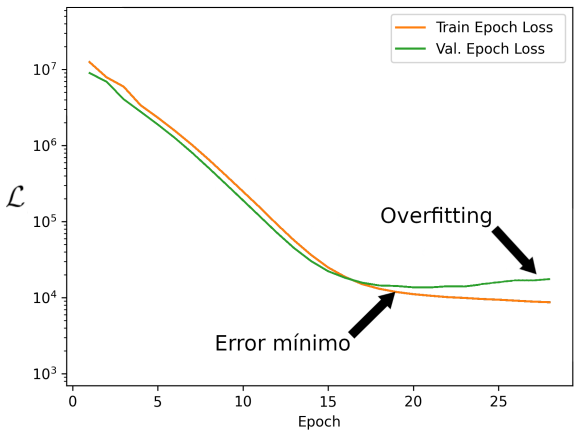
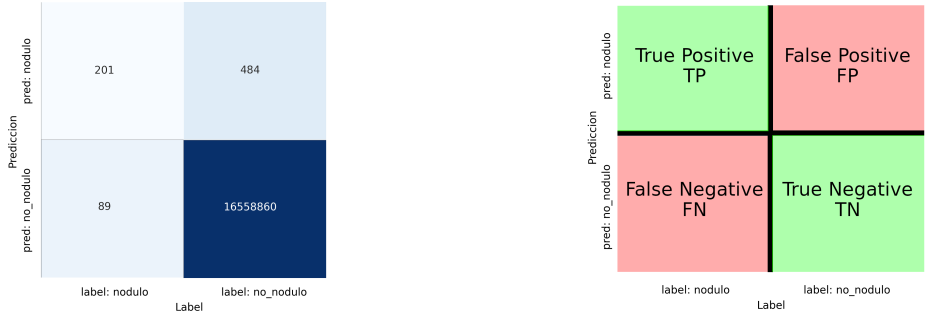


Figura 6.2: Curvas de error del conjunto de *train* y *validation* dado por la función de *loss*, $\mathcal{L}$, frente a las épocas de un entrenamiento. Notar el punto óptimo de mínimo error en la curva de *validation* antes del *overfitting*.

El error decrece en ambos conjuntos de datos hasta cierta cantidad de épocas, en el que el modelo es suficientemente potente como para seguir mejorando las predicciones en *train*. Sin embargo, el error del *validation* puede empezar a crecer. En tal caso, el aprendizaje se aleja de la regla o patrón que debería seguir para tener buen desempeño en unos casos que nunca a haya visto, es decir, pierde generalidad (*overfiting* o sobreajuste). El punto óptimo es, por tanto, donde el error en la validación sea menor. Por ello, el estado del modelo que se guarda (los pesos) es aquel en dicha época.

Por lo que respecta a un problema de detección con segmentación en imágenes

es posible definir una métrica que tenga en cuenta el rendimiento del modelo en esta tarea. Un ejemplo es la matriz de confusión. Esta permite comparar para un dataset dado (tipicamente *validation* o *test*) las predicciones del modelo con las etiquetas. Esta se define como una matriz de dimensiones $n \times n$ donde n es el número de clases siendo uno de los índices de la matriz la estimación del modelo y el otro las etiquetas. Si ambos coinciden con frecuencia porque el modelo acierta, los elementos de la diagonal de la matriz tomarán valores altos. Si no concuerdan (el modelo falla) se contribuye a los valores fuera de la diagonal pudiendo identificar qué clase el modelo confunde con cual. En este caso, el objetivo es detectar y segmentar nódulos de pulmón por ello solo hay una clase: nódulo. Sin embargo, el modelo puede detectar o no un nódulo dado, o incluso intuir uno donde realmente no lo hay. Para tener en cuenta esto se suele añadir una "clase" más: *background* o fondo. Se trata pues de un problema de clasificación binario. De esta manera, la matriz de confusión de este problema tomaría la forma indicada en la Figura 6.3.



(a) (b)

Figura 6.3: (a) Ejemplo de matriz de confusión del problema de detección de nódulos, enfrentando predicciones y etiquetas, o *labels*, en cada píxel. (b) Esquema general de una matriz de confusión de un problema de clasificación binario.

Atendiendo a la matriz de confusión de la Figura 6.3a como ejmplo, se aprecia un rendimiento bastante pobre del modelo en el que los valores fuera de la diagonal (FP y FN) son comparables a uno de los elementos de la diagonal. El criterio seguido para construirla consiste en enfrentar las predicciones y las etiquetas de cada píxel de las imágenes pudiendo tomar estos dos valores: nódulo o no nódulo (*background*), 1 ó 0 respectivamente. Este razonamiento permite entender la presencia de un alto valor en el elemento de *True Negative*, TN, ya que la mayor parte del espacio de una imagen es fondo, siendo los nódulos volúmenes pequeños (unos milímetros).

Ahora bien, este criterio basado en el valor de cada píxel individual para construir la matriz no evidencia de manera muy clara si el modelo ha detectado cierto tumor en su mayor parte o cómo agrupa los píxeles en los que se equivoca. Por ello, se plantea ahora

considerar cada nódulo a la hora de construir la matriz. Un primer enfoque puede ser en 2 dimensiones, es decir, un nódulo es definido como una sección en una imagen. Por ejemplo, si un mismo nódulo es intersectado por 5 *slices* computará como 5 distintos en la matriz. Esta distinción entre posibles nódulos en una imagen vendría dada por un modelo de segmentación de instancias, pero no en el *output* (máscara binaria) del modelo U-Net de segmentación semántica. Por ello, para poder distinguir, contar y ubicar nódulos se ha realizado una "clusterización" de la máscara, Figura 6.4. Para ello, el primer paso fue binarizarla realizando *thresholding* o umbralización, es decir, considerar como positivos solo aquellos píxeles que tienen una confianza superior a cierto umbral. Teniendo la máscara binaria, se realizó la "clusterización" mediante el método `cv2.findContours()` de la librería de visión artificial `opencv-python` el cual proporciona el perfilado de regiones separadas a partir de una máscara binaria. Esto permite referirse a un nódulo o a otro en la imagen, es decir, distinguirlos, Figura 6.4.

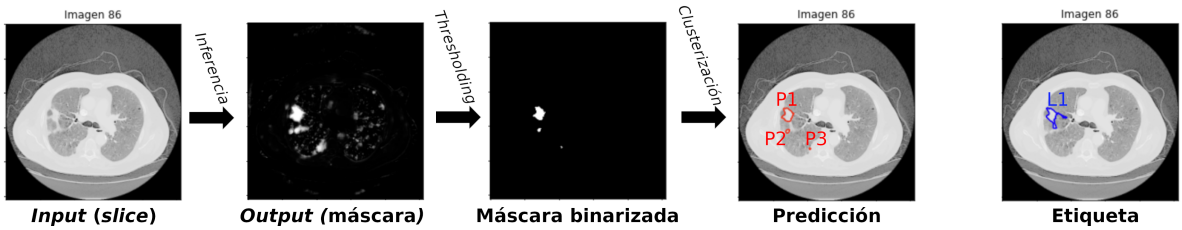


Figura 6.4: Proceso de "clusterizado" de nódulos a partir de la máscara ofrecida por la UNet.

Ahora bien, una vez ubicados los nódulos detectados es posible realizar la comparación con los que realmente hay (etiquetas). Para ello se propone calcular el coeficiente IoU (*intersection over union*) de cada uno de los nódulos predichos con cada uno de los etiquetados, Ec. 6.1,

$$\text{IoU} = \frac{a_{Li} \cap a_{Pj}}{a_{Li} \cup a_{Pj}} \tag{6.1}$$

donde a_{L1} es el área de un nódulo etiquetado y a_{P1}, Figura 6.5. De esta manera, una coeficiente IoU nulo o muy reducido permite concluir un fallo del modelo no habiendo acertado esa detección o, al menos, que no son asociables, y, por el contrario, un valor alto implica una correspondencia entre predicción y etiqueta, es decir, una detección correcta construyendo así una matriz de confusión.

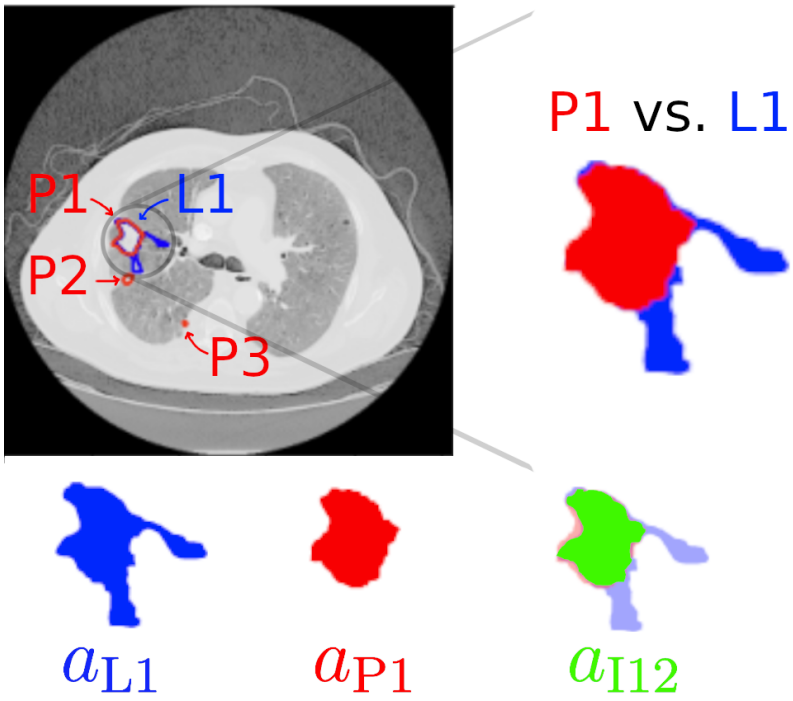


Figura 6.5: Ejemplo de comparación entre un nódulo detectado, P, y etiquetado, L, mediante IoU, *intersection over union*, donde es la a_{I12} es la intersección entre ambos.

Atendiendo a la Figura 6.5, se tiene la detección de tres nódulos siendo P2 y P3 claramente falsos positivos, con IoU= 0 con la etiqueta. El nódulo P1 puede considerarse un verdadero positivo teniendo un coeficiente IoU con L1 de 0,785. Se intuye pues la necesidad de definir un umbral de IoU para concluir esta asociación. Este valor será indicado en cada matriz calculada con este criterio.

Este procesado y cálculo está recogido en el fichero `matrix_2d.py` haciendo uso de la clase `Patient()` y ejecutable mediante `get_matrix_2d.sh` en el *software* `procesLIDC` [5].

Por último, es posible definir otro criterio con el que construir la matriz de confusión. Aunque los modelos aquí evaluados reciben una imagen 2D los nódulos tienen un significado en 3 dimensiones. Un nódulo es seccionado por varias *slices* apareciendo en varias imágenes. Como cabe esperar que este no presentará el mismo tamaño en todas ellas pudiendo estar etiquetado aun cuando la sección sea muy reducida. Por ello, es razonable plantear el criterio basado en IoU pero en 3 dimensiones tomando, en lugar del área, el volumen en la comparación. La Figura 6.6 ilustra esta idea.

[5]URL del repositorio `processLIDC` https://github.com/abelBEDOYA/processLIDC, Tabla 8.2 en Anexos.

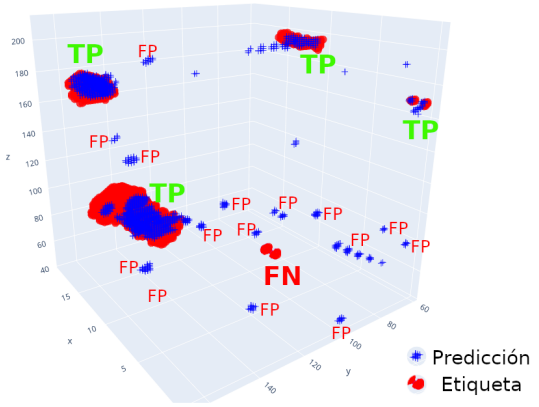


Figura 6.6: Representación 3D de los volúmenes etiquetados y detectados como tumor. Destacaría la alta presencia de falsos positivos, FP.

Atendiendo a la Figura 6.6 se aprecian las etiquetas y predicciones de nódulos en 3D. Cada uno de los *clusteres* apreciables computan como un nódulo (detectado o etiquetado). Se ha considerado de interés obviar las detecciones que tengan un volumen menor a 3 vóxeles, ya que la UNet tendía a ofrecer excesivos falsos positivos. A su vez, para no desvirtuar la cantidad de verdaderos positivos, en el caso de haber detectado nódulos distintos y, a la vez, contenidos en el volumen de un verdadero tumor, estos han sido computados como una sola IoU y, por tanto, siendo representados en la matriz como un solo verdadero positivo.

Por último, una vez calculada la matriz de confusión es posible calcular dos métricas que resumen el rendimiento del modelo: precisión, P, y *recall* o sensibilidad, R. Estas se definen de acuerdo a la Ec. 6.2.

$$P = \frac{TP}{FP + TP} \quad ; \quad R = \frac{TP}{FN + TP} \tag{6.2}$$

En otras palabras, la precisión refleja la probabilidad que el modelo tiene de acertar cuando afirma que hay un nódulo, mientras que el *recall* o sensibilidad resume qué probabilidad hay de que, habiendo un nódulo, el modelo lo detecte. De tal forma que la precisión es penalizada por los falsos positivos, FP, que cometa el modelo y el *recall* por los falsos negativos, FN.

6.3. Entrenamiento de la UNet

Por lo general, los modelos de visión requieren de miles de imágenes usadas como dataset de entrenamiento para poder ser eficaces resolviendo problemas. Esto resulta un problema cuando los ejemplos disponibles son limitados, lo cual sucede frecuentemente. Sin embargo, un modelo de visión entrenado en resolver cierta tarea puede ser reajustado

para "especializarlo" en otra similar. Este proceso, afortunadamente, no requiere de tal cantidad de imágenes. La interpretación que explica la posibilidad de mudar un modelo a distintas tareas con facilidad reconoce que el procesado de una imagen (en *deep learning*) es muy similar en un problema u otro, especialmente en las primeras capas. Es decir, la extracción de características de una imagen es un proceso común a los modelos, siendo específico de cada tarea en particular el procesado que elaboran las últimas capas, el cual, en muchas ocasiones, es una clasificación.

Por ello, es muy razonable partir de un modelo preentrenado en otra tarea (*transfer learning*) y realizar nuevamente un entrenamiento que especialice y ajuste la red al problema aquí propuesto: segmentar nódulos de pulmón. Se ha partido pues de un modelo UNet especializado en segmentación de anomalías en resonancias magnéticas del cerebro [34], Figura 6.1. Esta red toma tensores $256 \times 256 \times 3$ y devuelve una máscara de dimensiones 256×256. Para darle esta forma a las imágenes es posible llamar al método `patient.get_tensors(scaled=True)` de la clase `Patient`, el cual proporciona como tensor de `torch` la tupla `(images, masks)`.

En cuanto a los entrenamientos, en todo momento se ha tomado una partición de los 1018 pacientes de 90 % para el dataset de *train* y 10 % para el de *validation*. Este puede ejecutarse con algunos de los hiperparámetros descritos en el apartado 4.1. Generalidades del *Deep learning* mediante un comando por CLI, incluido en un ejecutable `.sh`,

```
python3 train.py --n_epochs 100 --batch_size 24 --val_split 0.1
--path2dataset 'path/to/LIDC-IDRI/' --path2savefiles '../trainings/train_1/'
--save_plots --save_epochs 1 --model_extension '.pt' --loss_type 4
```

Los entrenamientos se han llevado a cabo mediante el *script* `train.py` del repositorio `processLIDC`[6]. El proceso de cada época consiste en leer cada paciente y recorrer sus *slices* tomándolas en paquetes de tamaño *batch size* y realizando la corrección de parámetros (*backpropagation*). El optimizador usado es `Adam` y el *learning rate* se ha variado según el orden de magnitud de la función de pérdida.

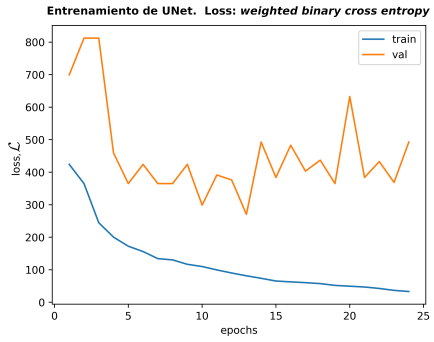
Es de gran importancia la elección de la función de pérdida, es decir, la métrica que indica cuánto se ha equivocado el modelo a la hora de dar un resultado. En este caso, se está ante un problema de segmentación semántica, por tanto, se trata de una clasificación de los píxeles de la imagen en nodular o no nodular, por ello, es razonable usar la *binary cross entropy*. Esta es típica en problemas de clasificación. Sin embargo, el proponer esta función de pérdida tal y como se define en la Tabla 3.2 suponía un rápido descenso del gradiente reduciendo el error, pero se producía una convergencia del modelo a dar tejido sano siempre y tener excesivos falsos negativos. En cierta medida,

[6]Repositorio de GitHub de `processLIDC`: https://github.com/abelBEDOYA/processLIDC, Tabla 8.2 en Anexos.

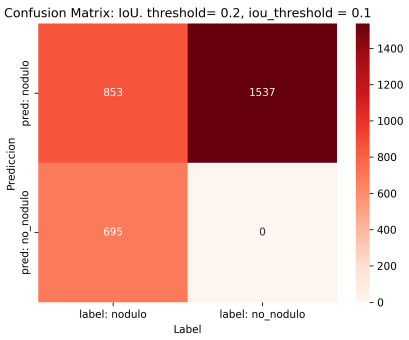
es razonable ya que los resultados que indiquen que no hay nódulos acertarán en la mayoría de casos y en la mayoría de la extensión de las imágenes, ya que los nódulos, en caso de haberlos, representan una superficie muy reducida. Para solucionar esto, se decidió eliminar del dataset la mayoría de las *slices* que no seccionen un nódulo (imágenes *background*) ya que estas representan más del 90 % de todas las imágenes. Así pues, se consideró solo el 2 % de las imágenes de *background*. En añadido, se modificó la función de pérdida tratando de añadir penalización cuando clasifique mal los píxeles etiquetados como nodulares definiéndose la función de *loss* como la 6.3.

$$\mathcal{L} = WBCE(y, \hat{y}) = \begin{cases} -[y \log \hat{y}_i + (1-y)\log(1-\hat{y})] & \text{si } y = 0 \\ -a[y \log \hat{y}_i + (1-y)\log(1-\hat{y})] & \text{si } y = 1 \end{cases} \tag{6.3}$$

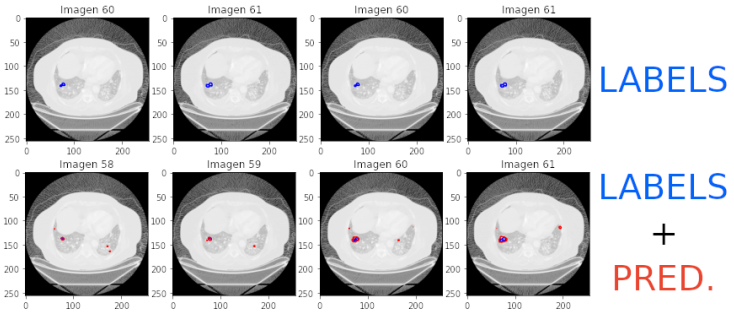
De acuerdo con la Ec. 6.3, donde y e $\hat{y}$ son etiqueta y predicción respectivamente, los píxeles que en la máscara etiquetada tengan un valor $y = 0$ recibirán un peso 1 mientras que aquellos que tengan valor $y = 1$, es decir, aquellos que sean nodulares recibirán un peso mayor tomándose $a = 40$. Esto se conoce como la *weighted binary cross entropy* (WBCE), la cual es propuesta en tareas de segmentación semántica en [31]. Realizando un entrenamiento de acuerdo a estas consideraciones se tiene la Figura 6.7.



(a)



(b)

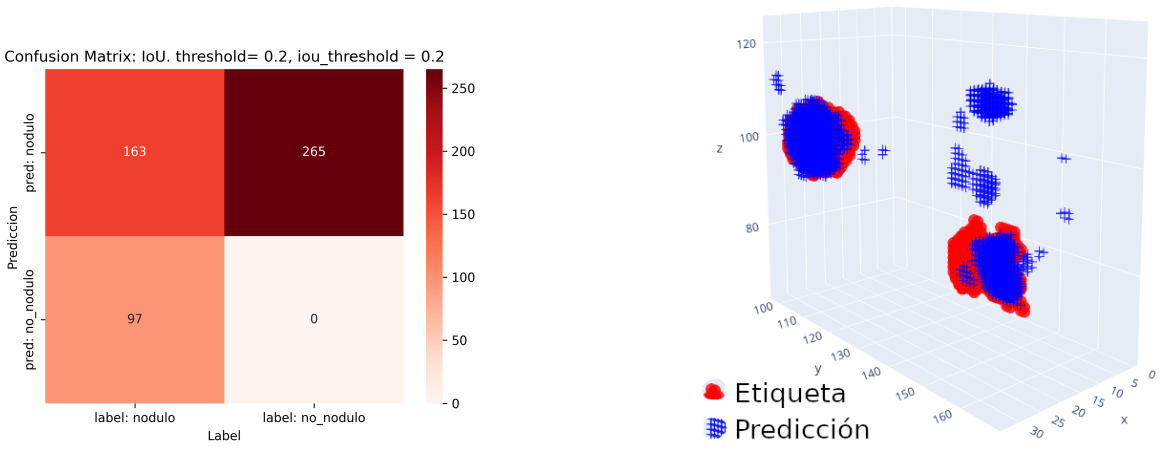


(c)

Figura 6.7: (a) Curvas de error del entrenamiento del modelo. (b) Matriz de confusión del *validation* con criterio IoU en 2D del modelo en el estado de la época 21. (c) Ejemplo de inferencia con la clase `Patient()` del paciente LIDC-IDRI-0170.

Atendiendo a la Figura 6.7 (a) se aprecian las curvas de error del dataset de *train* y *validation* durante el entrenamiento. Estas tienen comportamiento decreciente, aunque el descenso del error de *validation* cesa a las 15 épocas, oscilando a partir de entonces. Los pesos con mejor desempeño son los de la época 21, resultando en la matriz de confusión de la Figura 6.7 (b). Esta presenta un *recall* $R_{2D} = 55{,}1\,\%$ y una precisión $P_{2D} = 35{,}5\,\%$.

Por otra parte, es posible realizar el cálculo de la matriz de confusión, pero con el criterio 3D de la IoU (*Intersection over union*), Figura 6.8.



(a) (b)

Figura 6.8: (a) Matriz de confusión del modelo UNet calculada con el dataset de validación con un criterio 3D de IoU (*intersection over union*). (b) Representación de un ejemplo de predicciones junto a etiquetas del volumen de un paciente.

Ante este último criterio, Figura 6.8, aplicando un umbral de confianza de 0,2 y de IoU de 0,2, la precisión toma un valor $P_{3D} = 38{,}1\,\%$ y un *recall* de $R_{3D} = 62{,}7\,\%$. Destaca una mejoría en estos valores respecto al criterio 2D.

Por último, es conocido que parte del desarrollo que acompaña las técnicas de *deep learning* refieren a un proceso de experimentación en el que es necesario probar hiperparámetros y evaluar resultados. Así pues, son ofrecidos sólo los resultados que mejor evidencian el rendimiento de la UNet, obviando el resultado de otros entrenamientos. Por ello, hay que destacar las diversas pruebas que se han realizado tuneando hiperparámetros. Por ejemplo, el *learning rate* óptimo ha resultado ser 0,001. Valores superiores daban cambios muy bruscos y ruidosos en las curvas de *loss* en el *validation*. Valores inferiores daban lugar a una curva de *loss* estancada en cierto valor, no consiguiendo un aprendizaje tan bueno. Esto último es razonable pensando en un descenso del gradiente a pequeños pasos, lo cual es un proceso muy sensible a converger en mínimos locales de la función de *loss* no llegando a mínimos inferiores y mejor desempeño del modelo. Por su parte, el tamaño del *batch* que se toma para realizar una corrección de los parámetros influye, siendo el óptimo de 24 imágenes. Nombrar también las funciones de *loss* evaluadas,

entre las que destaca la *weighted binray cross entropy*. Otra, por ejemplo, basada en el coeficiente de *intersection over union* dificultaba la convergencia, protagonizando en la matriz de confusión los falsos negativos, FN.

6.4. YOLO: *You Only Look Once*

YOLO *(You Only Look Once)* es un modelo de *deep learning* de código abierto muy conocido de procesamiento de imágenes desarrollado por la empresa de *software* Ultralytics. YOLO se ha caracterizado por su velocidad de inferencia en tiempo real, su precisión y su facilidad a la hora de ser reentrenado y adaptado a nuevas tareas [13].

Desde la presentación original del modelo en 2015 [35] superando en métricas a modelos de detección con arquitectura RCNN (*Region-based Convolutional Neural Network*) [40] y *Fast* RCNN (*Fast Region-based Convolutional Neural Network*) [42] ha estado en constante revisión y evolución de versiones. Originalmente el modelo podía predecir imágenes cuadradas de 448×448 llegando ahora con las variantes más grandes a resoluciones de 1280×1280. En la segunda versión se añadieron técnicas de regularización como *dropout* y una nueva forma de estimar las posiciones de las detecciones: *anchor boxes* [36]. Nuevas mejoras fueron sucediendo, aumentando el tamaño del modelo. Por su parte, en su tercera versión destaca un *back bone* extractor de características que combinaba la arquitectura ResNet junto a otras facilitando una mejor propagación de los detalles finos así como el gradiente en la corrección de parámetros. YOLOv5 trajo consigo un crecimiento en su popularidad ya que fue ofrecido como modelo de código abierto en el *framework* de `pytorch` siendo este muy usado por usuarios y empresas de todo el mundo [37] [39].

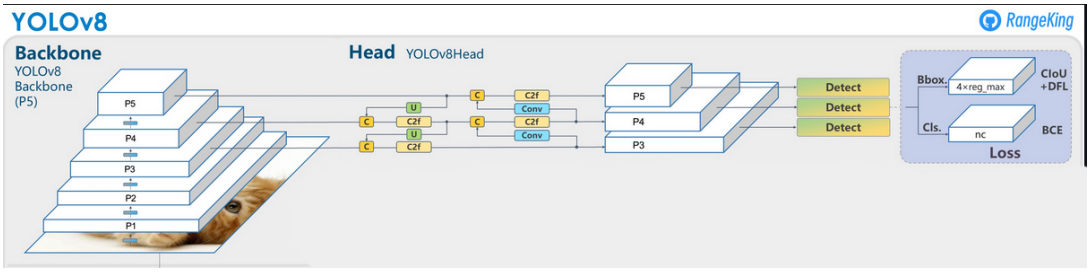


Figura 6.9: Esquema simplificado de la estructura del modelo de detección de objetos de YOLOv8 [38].

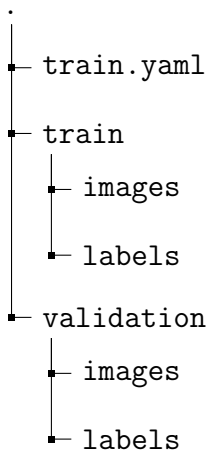
Desde su origen YOLO a proporcionado siempre la localización y tamaño de las clases detectadas en la imagen, de la forma x y *width height confidence*. Los cuatro primeros parámetros definen la *bounding box* que encierra al objeto detectado y la *confidence* refiere a la confianza de la detección para la clase con más seguridad, Figura 6.9. Sin embargo, muy recientemente (enero de 2023) se ha lanzado su última versión hasta la fecha: YOLOv8 [38] superando en rendimiento a sus predecesores. Esta ha incorporado

a su conjunto de modelos algunos destinados a la tarea de segmentación por instancias (entre otras). YOLOv8-segmentation es el que ha sido entrenado en este trabajo tratando de realizar la tarea de diagnosticar nódulos de pulmón.

6.5. Procesado del dataset para YOLOv8-segmentation

YOLO ofrece sus modelos junto a ciertos pesos ya entrenados con un gran dataset: COCO [43]. De esta manera, resulta relativamente fácil entrenar de nuevo especializando el modelo en una nueva tarea específica, de ahí su extendido uso por todo el mundo. El tomar unos pesos prentrenados se conoce como *transfer learning* como se advertía en anteriores secciones.

Así pues, para entrenar YOLOv8 en la tarea de segmentación de nódulos de pulmón se ha realizado un proceso de adaptación del dataset LIDC-IDRI. Este era accesible mediante `pyLIDC` o `pyDICOM` exclusivamente. Por ello, se ha transformado al formato que YOLO recibe. Esto es, imágenes en algún formato típico (`.jpg`, `.jpeg`, `.png`, ...) y sus correspondientes etiquetas en formato `.txt`. Estos deben distribuirse de la forma,



Gracias a `procesLIDC` se han procesado y guardado las imagenes en formato `.jpg` habiendo aplicado el aumento de contraste logarítmico propuesto anteriormente. Un fichero `train.yaml` contiene parte de la información del etiquetado e indica esta estructura de ficheros. En cuanto a las etiquetas, se ha creado un fichero `.txt` para cada imagen con su mismo nombre. En ellos se guarda la información del perfilado realizado por los radiólogos en el siguiente formato

```
$ cat LIDC-IDRI-0129_43.txt
0 0.6171875 0.68359375 0.609375 0.69140625 ... 0.607421875 0.69140625
0 0.60546875 0.689453125 0.60546875 0.701171875 ... 0.6015625 0.705078125
```

donde el primer número de cada fila representa la clase de la instancia. En este caso solo hay una: nódulo, por tanto, siempre es un 0. El resto de los números son las

parejas de valores (x, y) que definen el contorno de cada tumor. Estas siempre son dadas en relativo al ancho y alto de la imagen respectivamente.

Como cabe esperar, no todas las imágenes contienen algún nódulo, siendo la mayoría fondo (`.txt` vacios). Por ello, para balancear el dataset se han tenido en cuenta solo el 2 % de las imágenes vacías, representando ahora el 25 % de todo el dataset aproximadamente [7]. Estos tendrán `.txt` vacios.

6.6. Entrenamiento de YOLOv8-segmentation

Disponiendo de la estructura y formato adecuados del dataset LIDC-IDRI es posible entrenar YOLOv8 segmentación con una linea de comando de terminal, indicando tamaño de modelo, `yolov8l-seg.pt`; tamaño de imagen de input, `512`; número de épocas, `100` y tamaño de *batch*, `8`.

```
yolo segment train model=yolov8l-seg.pt data=train.yaml imgsz=512
epochs=100 batch=8
```

Es posible resumir estos y otros hiperámetros en un archivo de configuración del entrenamiento. En caso de no indicarlo se toman valores por defecto. Se adopta un *learning rate* con valor de 0,01 y el optimizador Adam con momento 0,937. Se ha tomado el aumentado de imágenes que realiza YOLO por defecto.

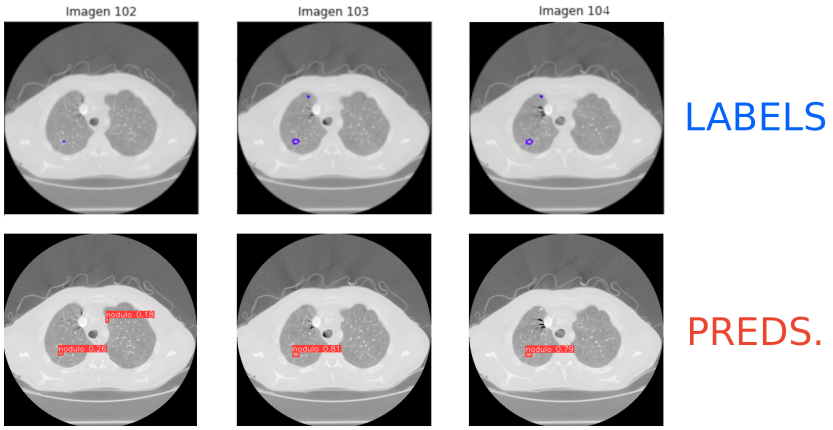
El resultado del entrenamiento son aquellos pesos correspondientes al estado del modelo en la época en la que el error en el dataset de validación era menor. A su vez, se estiman ciertas métricas que dan cuenta del rendimiento del modelo y la eficacia del entrenamiento a medida que se sucedían las épocas. Entre ellas está la matriz de confusión, calculada de manera equivalente a lo propuesto en el caso de la UNet mediante la evaluación de la *intersection over union*, IoU.

Ahora bien, en cuanto a segmentnación, YOLO dispone de varios modelos según su tamaño: `n`, `s`, `m`, `l` y `x`. Se ha realizado un entrenamiento partiendo de cada uno y el que mejor resultado ofrece es el `l`. Fue entrenado en 150 épocas como se aprecia en la Figura 6.10.

[7] Repositorio `yolo-lidc`: https://github.com/abelBEDOYA/yolo-lidc, Tabla 8.3 en Anexos.

(a)

(b)

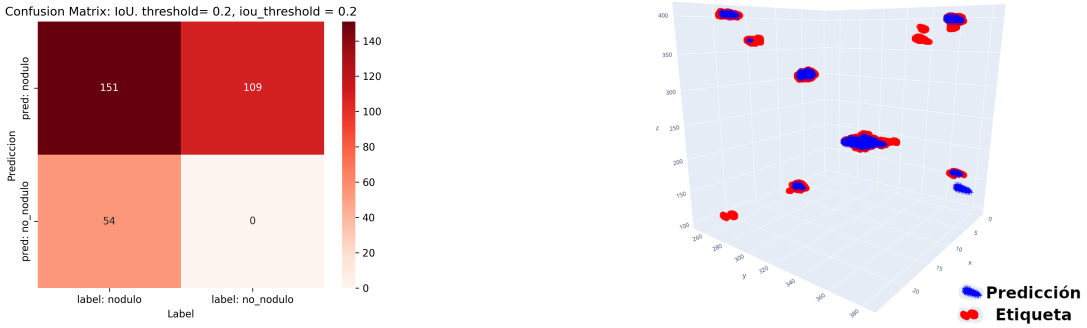


(c)

Figura 6.10: (a) Curvas de error del modelo del proceso de entrenamiento. (b) Matriz de confusión con criterio 2D de YOLO calculada con el dataset de validación, una confianza mínima de 0,2 y un threshold de IoU de 0,2. (c) Ejemplo de inferencia con YOLOv8l-seg. entrenado. Paciente: LIDC-IDRI-0011.

Se aprecian curvas de error muy ranzonables, reduciéndose tanto en *train* como en *validation*. Esto sucede en las tres métricas de error que considera YOLO: el error de las *bounding boxes* (box_loss), de la segmentación (seg_loss)y de la clase del objeto detectado (cls_loss). La matriz de confusión de cada sección de nódulo en 2D es calculada por el mismo programa de YOLO dando lugar a un rendimiento general mejor que el de la UNet, Figura 6.10. Ofrece una precisión $P_{2D} = 95{,}5\,\%$ y un *recall* de $R_{2D} = 57{,}6$.

A su vez, tomando los pesos ofrecidos en este entrenamiento es posible realizar el cálculo de la matriz de confusión, pero en base al criterio 3D aplicado con las predicciones de UNet, Figura 6.11.



(a) (b)

Figura 6.11: (a) Matriz de confusión 3D de YOLOv8l-segmentation. (b) Representación de un ejemplo de predicciones junto a etiquetas del volumen de un paciente.

Bajo el criterio IoU en 3D para calcular la matriz de confusión, Figura 6.11 la precisión adopta un valor $P_{3D} = 58{,}1\,\%$ y el *recall* de $R_{3D} = 73{,}6\,\%$.

7. Discusión

El entrenamiento programado de la UNet propuesta ha sido satisfactorio siendo evaluada con métricas basadas en el coeficiente *intersection over union* (IoU). La gestión y procesado del dataset por parte de la clase de python desarrollada (`Patient`) ha resultado vital para el entrenamiento. Las métricas tanto 2D como 3D reflejan que el modelo ha sido capaz de aprender a identificar y segmentar correctamente la mayor parte de los nódulos del dataset de validación, siendo pacientes que nunca ha visto antes.

Ahora bien, los resultados de la evaluación del rendimiento de la UNet entrenada están por debajo de los ofrecidos por la literatura. Si bien es cierto que detecta la mayor parte de los nódulos, los restantes son fugas, siendo en medicina especialmente importante el tener número de fugas muy reducido. Más evidente aun es la cantidad de falsos positivos que ofrece la red. Esto lleva a tener una precisión del 38,1 % y un *recall* del 62,7 % quedando lejos de los valores por encima de 90 % de [27], [30] o [33]. Sin embargo, hay que destacar que la red aquí toma toda la imagen y realiza una segmentación a partir de ella, sin ninguna identificación o recorte previo, procesando toda la sección, a diferencia de [33]. Por otro lado, varios de los modelos de *deep learning* de la literatura tienen una naturaleza 3D, analizando varias secciones a la vez tomando así información en los tres ejes espaciales. De esta manera, la red dispondría de la posibilidad de atender a mayor número de características para elaborar la detección [30]. Hay que nombrar también las métricas obtenidas por modelos no basados en *deep learning* en la literatura [20], [21], [22]. En ellos destaca una fuerte labor de ingeniería de características previa que prepara un *input* muy favorable, habiendo segmentado los pulmones, recortado solo relieves potencialmente

nodulares, etc.

Por otro lado, es posible razonar la gran cantidad de falsos positivos que presenta la red siendo un problema bastante común en segmentación semántica. Algún artículo como [28] se centra en reducirlos. En este caso, si se atiende a la función de *loss* propuesta: *binary cross entropy*, penaliza la equivocación en cada píxel, de tal forma que ante una reducida falsa detección el aumento del error será pequeño. Por ello, el modelo puede tender a detectar pequeños posibles nódulos ya que la penalización es muy reducida. Ante estas condiciones en las que la mayoría de los píxeles han sido acertados, la matriz de confusión de detección que sí distingue nódulos como cúmulos positivos detectados se ve perjudicada, ofreciendo una gran cantidad de falsos positivos. Esto se ve agravado ante la función de pérdida, *wbce*, que da más penalización a no detectar un píxel positivo que a detectar uno que no lo es. Frente a esta situación puede plantearse una nueva función de pérdida que tenga en cuenta la intersección que realmente hay con los tumores realmente etiquetados, es decir, una combinación entre la *wbce* y un valor proporcional al coeficiente IoU, Ec. 7.1.

$$\mathcal{L} = -c \sum_{i=1}^{n} [y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i)] + b \left(1 - \frac{TP}{TP + FP + FN}\right) \tag{7.1}$$

Donde b adopta un valor que es del orden de magnitud del valor que adopta la *wbce*. Sin embargo, el uso de esta función de pérdida no resultó en métricas mejores tras el entrenamiento, viéndose resentido especialmente el *recall*, $R = 36{,}7\,\%$. A su vez es posible discutir el peso, a, que se atribuye a la función de *loss* en los píxeles etiquetados como positivos, Ec. 6.3. En los entrenamientos se ha tomado $a = 40$, el cual es un valor con posibilidad de ser optimizado. Sí ha sido evidenciado que un valor $a = 1$, es decir, otorgar el mismo peso a todos los píxeles supone la convergencia a concluir toda la imagen como negativo. Otros valores inferiores daban peores resultados.

Por otro lado, es posible plantear posibles mejoras a la arquitectura de la red tomada [34]. Esta toma *inputs* de $256 \times 256 \times 3$. Destaca inicialmente la necesidad de ofrecer 3 canales aun cuando las imágenes de las TAC están en escala de grises, es decir, un solo canal. En [34] estas dos dimensiones extra refieren a la misma imagen, pero con un procesado distinto, tratando de evidenciar información médica complementaria. Sin embargo, aquí no se han realizado varios tratamientos, copiándose la misma imagen en los 3 canales. A su vez, la resolución de las imágenes ha sido reducida a la mitad, siendo originalmente de 512×512, perdiendo parte de la información que contenían. Ante esta situación, se ha propuesto una modificación de la estructura de la UNet, consiguiendo un *input* y *output* de 512×512. Sin embargo, tras realizar un entrenamiento se evidenció

la importancia de partir de un estado del modelo preentrenado. El hecho de modificar la arquitectura de la red impidió partir de los parámetros ya entrenados inicializándose de manera aleatoria y no convergiendo a ningún resultado favorable ofreciendo una curva de *loss* constante y ruidosa.

Un nuevo intento de mejora fue propuesto planteando una UNet distinta [44]. Esta es más potente, presentando una cantidad de parámetros mayor. Toma imágenes de la resolución original, aunque está pensada para RGB, de tal forma que su *input* es $512 \times 512 \times 3$ y su *output* es $512 \times 512 \times 2$. Notar que los canales de salida ahora son 2 debido a que la red realiza la predicción de la clase de interés: nódulo y también del fondo. Por ello, las etiquetas fueron adaptadas, construyendo ahora dos máscaras, una inversa a la otra. Tras el entrenamiento con una función de *loss wbce* las métricas de evaluación fueron peores que las de la red inicial, presentando un *recall* por debajo del 50 % en la mayoría de las ocasiones y una cantidad de falsos positivos superior a los ofrecidos por la UNet inicial.

Por otro lado, se ha entrenado el modelo de segmentación por instancias YOLOv8-segmentation [38], en concreto su tamaño *large*, *l*. Este es el que mejor resultados ofrecía en el "zoo" de modelos que ofrece siendo estos de menor a mayor tamaño: *n*, *s*, *m*, *l* y *x*. Los inferiores al *l* al ser menos potentes no conseguían el mismo rendimiento. Por el contrario, un modelo excesivamente grande, *x*, puede requerir una cantidad de datos mayor para entrenar, alcanzando con los disponibles un desempeño más modesto.

Ahora bien, es posible reunir las métricas de rendimiento de UNet y YOLOv8-segmentation alcanzadas en el dataset de validación para ser comparadas, Tabla 7.1.

	UNet		YOLOv8-seg	
	2D	3D	2D	3D
Precisión, P (%)	35.5	38.1	95.5	58.1
Recall, R (%)	55.1	62.7	57.6	73.6

Tabla 7.1: Valores de la precisión y el *recall* calculados con un criterio IoU 2D y 3D para UNet y YOLOv8-segmentation, con un *theshold* de confianza del 0,2.

Atendiendo a la Tabla 7.1 destaca un mejor desempeño de YOLOv8l-segmentation en todos los campos. Evaluando primeramente las métricas bajo el criterio 2D se aprecia una elevada precisión de YOLO del 95,5 %, es decir, presenta muy pocos fallos por falsos positivos. En otras palabras, una detección de YOLO será muy fiable, a diferencia de la UNet que alcanza un valor de 35,4 % con gran cantidad de falsos positivos. Esto es razonable ya que YOLO es un segmentador de instancias y no tiende tan fácilmente a detectar falsas regiones tumorales ya que es penalizado en mayor medida ante un falso positivo que la UNet, como se ha explicado. En cuanto al *recall* ambos ofrecen valores

similares, 55,1 % la UNet y 57,6 % YOLO. De nuevo, estos son valores considerablemente inferiores a las sensibilidades (*recall*) alcanzadas en la literatura.

En cuanto a las métricas obtenidas realizando el cálculo de la matriz de confusión bajo un criterio de *intersection over union*, IoU, en 3D, cabía esperar una mejora. Es razonable pensar que el volumen de un tumor es seccionado por varias *slices* presentando el nódulo un tamaño muy reducido en algunas de ellas siendo más difícil de detectar que en otras en las que su tamaño es mayor. En el criterio 2D el no haber detectado cierta sección de un tumor computa como falso negativo aun cuando el resto de las secciones de ese mismo tumor hayan sido detectadas. Por ello, el definir una IoU del volumen del tumor permite una asociación más completa y tolerante, computando ahora como +1 en algún elemento de la matriz el tumor completo y no cada una de sus secciones. Así pues, el *recall* es el principal beneficiado en este nuevo criterio teniendo YOLO un valor de 73,6 % y la UNet de 62,7 %. Se reduce la proporción de falsos negativos, presentando una sensibilidad mejor y más representativa del rendimiento de ambos modelos. En cuanto a la precisión, la UNet mejora subiendo hasta un 58,1 %. Sin embargo, este incremento puede venir justificado por un pequeño filtrado que se realiza a los clusters de vóxeles detectados positivos obviando aquellos que sean especialmente reducidos, propios de una segmentación presente en una sola *slice* y, por tanto, propenso a ser falso positivo. Por lo que respecta a YOLO, ningún filtro ha sido aplicado. Bajo este criterio 3D YOLO empeora su precisión con un 58,1 %, quizá por el hecho de que cada cluster 3D computa como un tumor detectado diferente juntando las secciones de tumor detectadas correctamente en un solo, reduciendo el total de verdaderos positivos a la vez que aumentando los falsos positivos. Aun así, está por encima de la UNet. Puede destacarse una virtud del modelo UNet sobre YOLO y es el hecho de resolver una segmentación semántica, la cual ofrece un mapa de confianzas que proporciona una estimación en todo punto de la imagen. Permite dar cuenta de qué regiones son potencialmente nodulares. Esto aporta más información al experto que una simple indicación de una instancia en la imagen, obviando el resto de regiones.

Hasta este punto, las etiquetas han sido tomadas como verdad absoluta. Si bien es cierto que se ha adoptado un valor de "acuerdo" entre radiólogos para definir las regiones tumorales como aquellas en las que al menos dos de ellos coincidían, no se ha nombrado la frecuencia con la que los 4 radiólogos concluyen lo mismo. Evaluar esto adecuadamente puede dar cuenta de qué rendimiento es esperable por parte de un experto en la tarea de detección. Para ello es posible atender a [45], en el que realizan un análisis del dataset LIDC-IDRI. En él se ofrece una estadística de concordancia entre las anotaciones de los 4 radiólogos. En la Figura 7.1 se aprecia que, en el criterio de concordancia entre radiólogos en el que se considera coincidencia si han anotado cierto tumor o bien como > 3 mm o como < 3, en más de la mitad no coinciden los 4. Teniendo un promedio

cada nódulo de ser etiquetado por aproximadamente 3,1 radiólogos. En caso de exigir que la consideración de su tamaño deba coincidir, este promedio baja a 2,7. Esto deja en evidencia la subjetividad e imprecisión que una persona, aunque sea experta puede añadir al problema.

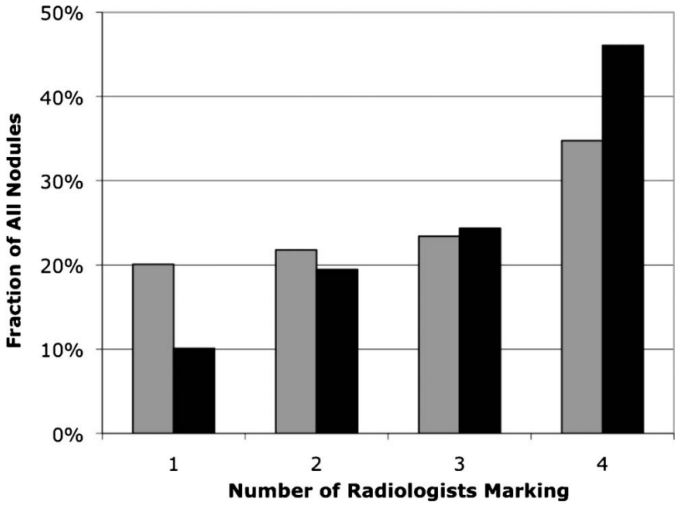


Figura 7.1: Gris: Porcentaje de nódulos anotados como < 3 mm, negro: Porcentaje de nódulos anotados como < 3 mm o > 3 mm, en función de la cantidad de radiógolos que concluyeron esa segmentación. Figura tomada de FIG. 4 de [45].

Por último, es posible discutir la especificidad de los modelos. En medicina resulta muy común el evaluar tanto sensiblidad (*recall*) como especificidad ("*recall* del fondo"). Esta última es la capacidad del modelo de estar ante un caso negativo y concluir negativo. Sin embargo, de acuerdo a los criterios seguidos para construir las matrices de confusión de detección de nódulos no tiene sentido preguntar por la cantidad de verdaderos negativos, TN (*true negative*). Para ello habría que definir claramente qué es un negativo. Sin embargo, no es evidente qué criterio habría que seguir, comprobar si el modelo las obvia y así contribuir al valor de TN. Más razonable sería de disponer de un conjunto de pacientes sanos y realizar una matriz de confusión de pacientes. En ella sí podrían computarse los cuatro valores TP, TN, FN y FP. En el caso del dataset LIDC-IDRI todos los pacientes presentan algún nódulo anotado. Por ello, en las matrices aquí estimadas el elemento correspondiente con TN no toma valor alguno.

8. Conclusiones y trabajo a futuro

Se ha entrenado satisfactoriamente un modelo de *machine learning* basado en *deep learning* con arquitectura UNet para detectar nódulos en los pulmones a partir de imagenes TAC. El dataset usado para su entrenamiento y validación es LIDC-IDRI, el mayor conjunto de imágenes médicas etiquetadas abierto al público.

La construcción de la matriz de confusión bajo un criterio 3D basado en el coeficiente IoU refleja que el modelo UNet entrenado detecta el 62,7 % (*recall*) de los nódulos que recibe y acierta un 38,1 % (precisión) de las detecciones que realiza. Destacando el elevado número de falsos positivos, de ahí su baja precisión. También, se ha entrenado con el dataset LIDC-IDRI el modelo de segmentación por instancias YOLOv8l-segmentation, obteniendo un *recall* de 73,6 % y una precisión de 58,1 %. Este ofrece un rendimiento por encima al alcanzado con el modelo UNet.

En un trabajo a futuro es posible proponer funciones de pérdida nuevas para el entrenamiento de la UNet. Esta debe ser representativa del error cometido por el modelo ante una equivocación u otra. En esta tarea queda pendiente optimizar, por ejemplo, el peso recibido por el valor de *loss* de los píxeles etiquetados como positivos.

Por otra parte, la UNet entrenada procesa exclusivamente una imagen, aun cuando se intuye más información médica en la dimensión no analizada. El modelo podría realizar una extracción de características más completa, tal y como hacen otros modelos de la literatura si procesase en 3D.

Disponer de mayor cantidad de datos por lo general resulta beneficioso para el entrenamiento de los modelos. Sin embargo, no siempre es posible, teniendo que trabajar con una cantidad reducida. En tal caso, es bastante común (especialmente en modelos de procesado de imágenes) el realizar un aumentado de datos. Este proceso consiste en realizar modificaciones a las imágenes, generando ejemplos nuevos con condiciones variadas aumentado su cantidad y riqueza. Estas transformaciones no deben desvirtuar el significado de las etiquetas. Por ello, esta labor podría realizarse en un futuro junto a un conocimiento experto en materia de diagnóstico (radiólogo) que asegure un aumentado razonable.

La comparación fue realizada con el modelo de procesado de imágenes YOLOv8 motivado por su facilidad de uso, versatilidad en especializarse en nuevas tareas y su carácter novedoso. Sin embargo, hay otros modelos de segmentación ofrecidos al público que presumiblemente podrían alcanzar los rendimientos aquí ofrecidos, *Detectron2* de Facebook es un ejemplo.

En cuanto a la matriz de confusión, no se ha podido realizar una estimación de la especificidad del modelo, es decir, su robustez ante casos negativos. Queda pendiente

realizar una validación con pacientes sanos.

Por último, la matriz de confusión es una métrica simple que da cuenta del rendimiento del modelo de manera bastante completa. Sin embargo, podrían plantearse métricas más sofisticadas que reflejen la precisión de la segmentación de los nódulos, su forma, curvas de rendimiento en función de la confianza, entre otras.

Referencias

[1] World Health Organization (OMS). (s.f.). Cáncer. Recuperado de https://www.who.int/es/news-room/fact-sheets/detail/cancer

[2] Sociedad Española de Oncología Médica (SEOM). (2022). Nota de Prensa Día Mundial del Cáncer 2022 [PDF]. Recuperado de https://www.seom.org/images/seomcms/stories/recursos/NdP_Dia_Mundial_Cancer_2022.pdf

[3] International Agency for Research on Cancer (IARC). (s.f.). GLOBOCAN Overtime - Visualización de gráficos. Recuperado de https://gco.iarc.fr/overtime/en

[4] Robledo, M. C. G. Costos de la atención del cáncer de mama en México: análisis en dos escenarios de cobertura.

[5] Cancer Imaging Archive. (s.f.). Recuperado de https://wiki.cancerimagingarchive.net/

[6] Hospital Clínic de Barcelona. (s.f.). Causas y factores de riesgo del cáncer. Recuperado de https://www.clinicbarcelona.org/asistencia/enfermedades/cancer/causas-y-factores-de-riesgo

[7] Medimaging. (s.f.). Ultrasonido con contraste realzado para detectar cáncer de hígado. Recuperado de https://www.medimaging.es/ultrasonido/articles/294771681/ultrasonido-con-contraste-realzado-para-detectar-cancer-de-higado.html

[8] Universitat Autònoma de Barcelona. (s.f.). Programas CAD: Análisis de imágenes obtenidas por técnicas de diagnóstico médico. Recuperado de https://www.uab.cat/web/detalle-noticia/programas-cad-analisis-de-imagenes-obtenidas-por-tecnicas-de-diagnostico-medico-1
html?noticiaid=1345794625081

[9] OrigenSalud. (s.f.). Tomografía axial computarizada (TAC): ¿En qué consiste esta prueba? Recuperado de https://origensalud.com/tac-en-que-consiste-esta-prueba

[10] Saroop, A., Ghugare, P., Mathamsetty, S., & Vasani, V. (2021). Facial Emotion Recognition: A multi-task approach using deep learning. arXiv preprint arXiv:2110.15028.

[11] Castillo, E. (1998). Introducción a las redes funcionales con aplicaciones: un nuevo paradigma neuronal. Paraninfo.

[12] OpenAI. (s.f.). OpenAI Chat. Recuperado de https://chat.openai.com/

[13] Maji, D., Nagori, S., Mathew, M., & Poddar, D. (2022). Yolo-pose: Enhancing yolo for multi person pose estimation using object keypoint similarity loss. In Proceedings

of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 2637-2646).

[14] Elharrouss, O., Akbari, Y., Almaadeed, N., & Al-Maadeed, S. (2022). Backbones-review: Feature extraction networks for deep learning and deep reinforcement learning approaches. arXiv preprint arXiv:2206.08016.

[15] Doi, K. (2006). Diagnostic imaging over the last 50 years: research and development in medical imaging science and technology. Physics in Medicine & Biology, 51(13), R5.

[16] National Institute of Standards and Technology (NIST). (2016). LIDC-IDRI: The Lung Image Database Consortium image collection. Recuperado de https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=907229

[17] Shiraishi, J., Katsuragawa, S., Ikezoe, J., Matsumoto, T., Kobayashi, T., Komatsu, K. I., ... & Doi, K. (2000). Development of a digital image database for chest radiographs with and without a lung nodule: receiver operating characteristic analysis of radiologists' detection of pulmonary nodules. American Journal of Roentgenology, 174(1), 71-74.

[18] Cancer Imaging Archive. (s.f.). Analysis Results. Recuperado de https://www.cancerimagingarchive.net/analysis-results/

[19] Pehrson, L. M., Nielsen, M. B., & Ammitzbøl Lauridsen, C. (2019). Automatic pulmonary nodule detection applying deep learning or machine learning algorithms to the LIDC-IDRI database: a systematic review. Diagnostics, 9(1), 29.

[20] Akram, S., Younus Javed, M., Qamar, U., Khanum, A., & Hassan, A. (2015). Artificial neural network based classification of lungs nodule using hybrid features from computerized tomographic images. Applied Mathematics & Information Sciences, 9(1), 183-195.

[21] Alilou, M., Kovalev, V., Snezhko, E., & Taimouri, V. (2014). A comprehensive framework for automatic detection of pulmonary nodules in lung CT images. Image Analysis & Stereology, 33(1), 13-27.

[22] Jaffar, M. A., Zia, M. S., Hussain, M., Siddiqui, A. B., Akram, S., & Jamil, U. (2020). An ensemble shape gradient features descriptor based nodule detection paradigm: a novel model to augment complex diagnostic decisions assistance. Multimedia Tools and Applications, 79, 8649-8675.

[23] Lu, L., Tan, Y., Schwartz, L. H., & Zhao, B. (2015). Hybrid detection of lung nodules on CT scan images. Medical physics, 42(9), 5042-5054.

[24] Zhang, W., Wang, X., Li, X., & Chen, J. (2018). 3D skeletonization feature based computer-aided detection system for pulmonary nodules in CT datasets. Computers in biology and medicine, 92, 64-72.

[25] Al-Shabi, M., Shak, K., & Tan, M. (2022). ProCAN: Progressive growing channel attentive non-local network for lung nodule classification. Pattern Recognition, 122, 108309.

[26] Saied, M., Raafat, M., Yehia, S., & Khalil, M. M. (2023). Efficient pulmonary nodules classification using radiomics and different artificial intelligence strategies. Insights into Imaging, 14(1), 91.

[27] Wang, S., Zhou, M., Liu, Z., Liu, Z., Gu, D., Zang, Y., ... & Tian, J. (2017). Central focused convolutional neural networks: Developing a data-driven model for lung nodule segmentation. Medical image analysis, 40, 172-183.

[28] Da Silva, G. L. F., Valente, T. L. A., Silva, A. C., De Paiva, A. C., & Gattass, M. (2018). Convolutional neural network-based PSO for lung nodule false positive reduction on CT images. Computer methods and programs in biomedicine, 162, 109-118.

[29] Han, G., Liu, X., Zheng, G., Wang, M., & Huang, S. (2018). Automatic recognition of 3D GGO CT imaging signs through the fusion of hybrid resampling and layer-wise fine-tuning CNNs. Medical & biological engineering & computing, 56, 2201-2212.

[30] Yu, H., Li, J., Zhang, L., Cao, Y., Yu, X., & Sun, J. (2021). Design of lung nodules segmentation and recognition algorithm based on deep learning. BMC bioinformatics, 22, 1-21.

[31] Ronneberger, O., Fischer, P., & Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18 (pp. 234-241). Springer International Publishing.

[32] Çiçek, Ö., Abdulkadir, A., Lienkamp, S. S., Brox, T., & Ronneberger, O. (2016). 3D U-Net: learning dense volumetric segmentation from sparse annotation. In Medical Image Computing and Computer-Assisted Intervention–MICCAI 2016: 19th International Conference, Athens, Greece, October 17-21, 2016, Proceedings, Part II 19 (pp. 424-432). Springer International Publishing.

[33] Kumar, S. N., Bruntha, P. M., Daniel, S. I., Kirubakar, J. A., Kiruba, R. E., Sam, S., & Pandian, S. I. A. (2021, March). Lung nodule segmentation using unet. In 2021 7th International Conference on Advanced Computing and Communication Systems (ICACCS) (Vol. 1, pp. 420-424). IEEE.

[34] Buda, M., Saha, A., & Mazurowski, M. A. (2019). Association of genomic subtypes of lower-grade gliomas with shape features automatically extracted by a deep learning algorithm. Computers in biology and medicine, 109, 218-225.

[35] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 779-788).

[36] Chang, Y. L., Anagaw, A., Chang, L., Wang, Y. C., Hsiao, C. Y., & Lee, W. H. (2019). Ship detection based on YOLOv2 for SAR imagery. Remote Sensing, 11(7), 786.

[37] Ultralytics. YOLOv5 2020. Available online: https://github.com/ultralytics/yolov5 (accessed on 31 agosto 2023)

[38] Jocher, G.; Chaurasia, A.; Qiu, J. YOLO by Ultralytics. GitHub. 1 January 2023. Available online: https://github.com/ultralytics/ ultralytics (accessed on 31 agosto 2023).

[39] Hussain, M. (2023). YOLO-v1 to YOLO-v8, the Rise of YOLO and Its Complementary Nature toward Digital Manufacturing and Industrial Defect Detection. Machines, 11(7), 677.

[40] Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 580-587).

[41] Girshick, R. (2015). Fast r-cnn. In Proceedings of the IEEE international conference on computer vision (pp. 1440-1448).

[42] Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. Advances in neural information processing systems, 28.

[43] Common Objects in Context (COCO) Dataset. (s.f.). Recuperado de https://cocodataset.org

[44] Milesial. (s.f.). PyTorch U-Net Implementation. Recuperado de https://github.com/milesial/Pytorch-UNet

[45] Armato III, S. G., McLennan, G., Bidaut, L., McNitt-Gray, M. F., Meyer, C. R., Reeves, A. P., ... & Clarke, L. P. (2011). The lung image database consortium (LIDC) and image database resource initiative (IDRI): a completed reference database of lung nodules on CT scans. Medical physics, 38(2), 915-931.

Anexos

Ejemplo de uso de la librería `pyLIDC` para acceder a los pacientes del dataset LIDC-IDIR mediante consultas objecto-relacional.

```
import pylidc as pl
import numpy as np
## ID Paciente:
pid = 'LIDC-IDRI-0002'
## Hacer una consulta SQL como SQLAlchemy
# Tomografia del paciente 0002:
scan = pl.query(pl.Scan).join(pl.Annotation).\
        filter(pl.Scan.patient_id == pid).first()
# Primera anotacion del paciente 0002
ann = pl.query(pl.Annotation).join(pl.Scan).\
        filter(pl.Scan.patient_id == pid).first()
# Cantidad de nodulos:
print('Cantidad de nodulos:', len(scan.annotations))
## Tomar imagenes CT
vol = scan.to_volume()
print(np.array(vol).shape)  # (512, 512, 261)->alto, ancho, slices
## Tomar anotaciones:
nods = scan.cluster_annotations()
print('nods: ', np.array(nods).shape)
## Visualizacion interactiva:
ann.visualize_in_scan()
ann.visualize_in_3d()
from pylidc.utils import consensus
## Obtener la mascara dada por el nodulo 1 del paciente 0002:
ann_clust_14 = scan.cluster_annotations()[0]
cmask, cbbox, _ = consensus(ann_clust_14, clevel=0.5)
print(np.array(cmask).shape)  # (85, 90, 28)
```

Librería/Framework	Descripción
`numpy`	Librería para computación numérica, proporciona soporte para arrays multidimensionales y funciones matemáticas.
`pytorch`	*Framework* de aprendizaje profundo, *deep learning*, que facilita la creación y entrenamiento de modelos de redes neuronales.
`matplotlib`	Librería para la creación de gráficos y visualizaciones en 2D, utilizada en análisis de datos y generación de gráficos. Ha sido usada para las representaciones de las curvas de error y mostrar las detecciones sobre las secciones de las tomografías.
`seaborn`	Librería que se basa en `matplotlib` y proporciona una interfaz de alto nivel para crear visualizaciones estadísticas atractivas y informativas en Python. Usada para la representación de las matrices de confusión.
`plotly`	Librería para crear gráficos interactivos y visualizaciones de datos en la web, útil para generar gráficos dinámicos en aplicaciones y sitios web. Usada para la reconstrucción del cuerpo y la visualización de las detecciones en 3D.
`cv2`	OpenCV (Open Source Computer Vision Library) es una librería para visión por computadora que proporciona herramientas para procesar imágenes y vídeo. Usada para calcular la IoU entre detecciones y etiquetas en 2D.
`scipy`	Proporciona algoritmos y herramientas para la ciencia de datos, incluyendo optimización, estadísticas y procesamiento de señales. Usada para clusterizar cúmulos de detecciones en 3 dimensiones.
`pyDICOM`	Librería para el manejo de archivos `DICOM`, utilizados en imágenes médicas, como radiografías y resonancias magnéticas.
`pyLIDC`	Librería específica para el acceso y procesamiento de las imágenes CT (tomografía computarizada) pulmonares del dataset LIDC-IDRI. En esta librería se apoya el acceso al dataset que realiza `processLIDC`.
`Ultralytics`	*Framework* de código abierto que simplifica el entrenamiento y la implementación de modelos de visión por computadora, incluyendo detección de objetos, segmentación y clasificación. Se ha evaluado aquí YOLO, su modelo principal, la versión YOLOv8l-segmentation en concreto.

Tabla 8.1: Librerías y *frameworks* de Python usadas en el trabajo.

processLIDC	
Fichero	**Descripción**
`processLIDC.py`	Contiene la clase `Patient()`. Se puede instanciar solo indicando el ID del paciente. Permite leer y cargar todo lo referente a un paciente, además de preprocesarlo, visualizarlo a voluntad tanto en 2D como 3D, relizar inferencia con un modelo de `pytorch` y obtener los tensores necesarios para entrenar.
`train.py`	Ofrece las principales funciones para realizar el entrenamiento de un modelo de `pytorch` con el dataset LIDC-IDRI. Plantea la iteración de épocas, pacientes y *batches*. Permite elegir entre diversas funciones de *loss*, realiza la partición entre *train* y *val* y guarda los pesos o estados del modelo durante el entrenamiento a la vez que las curvas de error a medida que se suceden las épocas. Ofrece por terminal un *feedback* del progreso del entrenamiento con `tqdm`. Permite elegir hiperparámetros desde CLI, mediante la librería `argsparse` de Python.
`matrix_pixel.py`	Dados unos pesos de la UNet y un conjunto de pacientes calcula la matriz de confusión teniendo en cuenta la clasificación de cada píxel de las imágenes. Guarda la matriz resultante.
`matrix_2d.py`	Dados unos pesos de la UNet y un conjunto de pacientes calcula la matriz de confusión bajo un criterio de detección de objetos en 2D. Para ello clusteriza la máscara dada por la segmentación semántica de la UNet, compara detección y etiqueta de acuerdo al coeficiente IoU (*intersection over union*). Guarda la matriz de confusión resultante.
`matrix_3d.py`	Dados unos pesos de la UNet y un conjunto de pacientes calcula la matriz de confusión bajo un criterio de detección de objetos en 3D. Para ello clusteriza la máscara dada por la segmentación semántica de la UNet habiendo juntado las secciones del paciente, formando así un volumen procesado. Compara detecciones y etiquetas de acuerdo al coeficiente IoU (*intersection over union*), esta vez en 3D. Guarda la matriz de confusión resultante.

Tabla 8.2: *Software* de python desarrollado para facilitar la gestión, visualización y preprocesado del dataset LIDC-IDRI para entrenar un modelo UNet de segmentación semántica para la detección de nódulos de pulmón. Repositorio público en GitHub: https://github.com/abelBEDOYA/processLIDC

yolo-lidc	
Fichero	**Descripción**
`yolodataset.py`	Convierte el dataset LIDC-IDRI al formato necesario para entrenar YOLO, es decir, imágenes en formato `.png`, por ejemplo, y una fichero por imagen en formato `.txt` con las etiquetas. Realiza la separación automática de `train` y `validation`.
`matrix_3d.py`	Dados unos pesos de YOLO y un conjunto de pacientes calcula la matriz de confusión bajo un criterio 3D. Para ello inferencia y une secciones pudiendo clusterizar las detecciones en volúmenes. De esta manera calcula el coeficiente IoU (*intersection over union*) para cada detección y etiqueta en 3D, construyendo y guardado la matriz resultante.

Tabla 8.3: *Software* de python desarrollado para la adaptación del dataset LIDC-IDRI al formato necesario para entrenar YOLOv8-segmentation. Permite evaluar el modelo con un criterio 3D. Repositorio público en GitHub: https://github.com/abelBEDOYA/yolo-lidc.