Christian Nahuel Botta

Localización de Región de Interés (ROI) en imágenes DICOM mediante técnicas de Deep Learning

2023

Instituto: Ingeniería y Agronomía

Carrera: Ingeniería en Informática

Universidad Nacional Arturo Jauretche

Instituto de Ingeniería y Agronomía

Ingeniería en Informática

PRÁCTICA PROFESIONAL SUPERVISADA

Informe Final

Localización de Región de Interés (ROI) en imágenes DICOM mediante técnicas de Deep Learning

---

Christian Nahuel Botta

Florencio Varela, 8 de agosto de 2023

ESTUDIANTE

Nombre y Apellido: *Christian Botta*

Correo electrónico: *chrisbotta1@gmail.com*

ORGANIZACIÓN DONDE SE REALIZA LA PRÁCTICA PROFESIONAL SUPERVISADA

Nombre de la institución: *Universidad Nacional Arturo Jauretche*

Dirección: *Av. Calchaquí 6200, Florencio Varela, (1888) Buenos Aires, Argentina*

Teléfono: *+54 11 4275 6100*

Sector: *Programa TICAPPS (Tecnologías de la Información y la Comunicación en Aplicaciones de Interés Social), Instituto de Ingeniería y Agronomía*

TUTOR ORGANIZACIONAL

Nombre y Apellido: *Dr. Ing. Martín Morales*

Correo electrónico: *martin.morales@unaj.edu.ar*

DOCENTES SUPERVISORES

Nombre y Apellido: *Dr. Ing. Marcelo Cappelletti*

Correo electrónico: *mcappelletti@unaj.edu.ar*

Nombre y Apellido: *Ing. Lucas Olivera*

Correo electrónico: *lolivera@unaj.edu.ar*

COORDINADOR DE LA CARRERA DE INGENIERÍA INFORMÁTICA

Nombre y Apellido: *Dr. Ing. Morales, Martín*

Correo electrónico: *martin.morales@unaj.edu.ar*

Resumen

Actualmente, la medicina ha experimentado un notable avance gracias a la aplicación de tecnologías de inteligencia artificial, en particular, el aprendizaje profundo. Una de las áreas más destacadas es la visión artificial aplicada a la localización de regiones de interés en estudios médicos. Esta tecnología permite a los profesionales de la salud detectar anormalidades o patologías en imágenes médicas de manera precisa y eficiente. En este proyecto, se explorarán técnicas avanzadas de Deep Learning para la localización de Regiones de Interés (ROIs) en imágenes de formato DICOM. Se abordarán tres aspectos fundamentales: detección, clasificación y segmentación, utilizando para ello dos arquitecturas populares dentro del ámbito de las Redes Neuronales Artificiales: YOLO y U-Net.

Palabras Clave: Inteligencia Artificial, Aprendizaje Profundo, Redes Neuronales Artificiales, Visión Artificial, Regiones de Interés, DICOM, Detección, Clasificación, Segmentación, YOLO, U-Net

Abstract

Currently, medicine has experienced a remarkable advancement thanks to the application of artificial intelligence technologies, particularly deep learning. One of the most prominent areas is artificial vision applied to the localization of regions of interest in medical studies. This technology enables healthcare professionals to detect abnormalities or pathologies in medical images with precision and efficiency. In this project, advanced deep learning techniques will be explored for the localization of Regions of Interest (ROIs) in DICOM format images. Three fundamental aspects will be addressed: detection, classification, and segmentation, using two popular architectures within the field of Artificial Neural Networks: YOLO and U-Net.

Keywords: Artificial Intelligence, Deep Learning, Artificial Vision, Regions of Interest, DICOM, Detection, Classification, Segmentation, YOLO, U-Net

Dedicatorias y Agradecimientos

Dedicado

- A mi Familia, por su inquebrantable apoyo y amor. Gracias a su constante ánimo y paciencia, he logrado alcanzar mis objetivos.
- A mis Amigos y Compañeros de estudio. Aquellos que estuvieron presentes a lo largo de este viaje.

Agradecimientos

- A mis Tutores, Marcelo Cappelletti y Lucas Olivera, por el acompañamiento y su guía experta durante mi camino en la investigación y el aprendizaje.
- A la Universidad Nacional Arturo Jauretche por brindarme la oportunidad de ser parte de la educación pública y la formación integral que ha sido fundamental en mi desarrollo académico y personal.

Índice General

Índice de Tablas

Índice de Figuras

CAPÍTULO I

Introducción

El presente trabajo forma parte de la Práctica Profesional Supervisada (PPS) y se presenta como requisito para la obtensión del título de Ingeniería en Informática otorgado por el Instituto de Ingeniería y Agronomía de la Universidad Nacional Arturo Jauretche (UNAJ). Como desarrollo de la misma, se propone la investigación de distintas técnicas de aprendizaje profundo (Deep Learning) para identificar Regiones de Interés (ROI: Region of Interes) en imágenes médicas bajo el estándar de Imagenología Digital y Comunicaciones en Medicina (DICOM).

El objetivo de esta práctica es explorar y desarrollar técnicas de visión artificial y Deep Learning para la localización precisa de las ROI en imágenes DICOM. Para ello, se analizarán diferentes arquitecturas de redes neuronales y métodos de preprocesamiento de imágenes, con el fin de obtener resultados robustos y confiables. A su vez, se evaluará el desempeño de los modelos propuestos utilizando métricas de precisión, sensibilidad y especificidad, comparándolos con métodos tradicionales de localización manual.

Desde un punto puramente informático, en este trabajo se estudia el comportamiento de redes profundas. Para ello, se aplicarán Redes Convolucionales (CNN) a problemas de detección, clasificación y segmentación con el objetivo de conseguir resultados suficientemente precisos para su uso. Estas redes se entrenarán desde cero, con conjuntos de datos extraídos de fuentes públicas para tal fin y se evaluarán sus desempeños acorde a los resultados obtenidos.

1.1. Motivación

El análisis de estudios biomédicos hace referencia al uso de varios métodos y técnicas que se emplean para obtener imágenes del cuerpo humano, que luego son utilizadas por profesionales de la salud para diagnosticar y tratar a los pacientes. El procesamiento de dichas imágenes es primordial a la hora de identificar distintas patologías.

En estudios médicos, las anomalías y enfermedades pueden resultar ser difíciles de detectar para un especialista sin la ayuda de herramientas de análisis de imágenes. Las imágenes DICOM se utilizan ampliamente en el campo de la imagenología médica porque ofrecen información detallada sobre pacientes, estudios y resultados acerca de la caracterización de la fisiología y anatomía de diversos órganos o partes del cuerpo humano. La identificación precisa de las ROI en estas imágenes es esencial para facilitar el diagnóstico temprano, la planificación de tratamientos o el seguimiento de la evolución de las

enfermedades. Sin embargo, debido a la complejidad y variabilidad de las estructuras anatómicas presentes en dichas imágenes, la localización manual de las ROI puede resultar una tarea tediosa y propensa a errores.

En el campo de la ingeniería en informática, el procesamiento de imágenes médicas desempeña un papel fundamental en la labor de reconocimiento de enfermedades. Con el desarrollo continuo de las técnicas de aprendizaje profundo, el análisis de este tipo de imágenes se ha convertido en un campo activo de investigación. El estudio de diferentes tareas de reconocimiento de patrones, como las ROI, buscan brindar información que contribuya a encontrar indicios sobre posibles afecciones médicas.

En los últimos años, las técnicas de Deep Learning han demostrado un gran potencial en el análisis de imágenes médicas. Estos enfoques basados en redes neuronales artificiales han logrado avances significativos en la detección, clasificación y segmentación automática de estructuras de interés. Mediante el entrenamiento de modelos de Deep Learning con grandes conjuntos de datos etiquetados, es posible enseñar a la red a reconocer y localizar automáticamente las ROI en imágenes DICOM. La implementación de las Redes Neuronales Convolucionales (CNNs) representan una gran oportunidad para solventar estos desafíos.

1.2. Objetivo General

El objetivo principal del presente trabajo es identificar las Regiones de Interés (ROI) en imágenes médicas mediante el protocolo de Imagenología Digital y Comunicaciones en Medicina (DICOM: Digital Imaging and Communication On Medicine), a través de diferentes técnicas de aprendizaje automático y visión artificial, en pos de proporcionar información relevante partiendo de estudios clínicos.

1.3. Objetivos Específicos

Se determinaron los siguientes objetivos específicos para alcanzar el objetivo general:

- Investigar y comparar las distintas técnicas de aprendizaje profundo para identificar cuál es la más adecuada en detectar las ROI en imágenes médicas.
- Seleccionar y preprocesar un conjunto de imágenes DICOM representativas para el entrenamiento y prueba del sistema.
- Diseñar e implementar distintas arquitecturas de redes neuronales adecuadas para localizar regiones de interés en imágenes DICOM.
- Entrenar las redes neuronales utilizando el conjunto de imágenes DICOM preprocesadas y evaluar su rendimiento mediante métricas de evaluación relevantes.

- Optimizar los parámetros de las redes neuronales para mejorar su precisión y reducir el tiempo de procesamiento.
- Validar el sistema mediante nuevas imágenes DICOM y comparar los resultados obtenidos.
- Realizar un análisis de los resultados y sus implicaciones en el campo del procesamiento de imágenes médicas y la aplicación de redes neuronales artificiales para la detección de regiones de interés en imágenes DICOM.

1.4. Tareas Ejecutadas

El desarrollo de la PPS se llevó a cabo durante seis meses. Durante ese período, la metodología empleada comprendió las siguientes etapas:

1.4.1. Estudio de herramientas basadas en aprendizaje automático

En primer lugar, se estudiaron herramientas de aprendizaje automático, con el propósito de adquirir conocimientos y capacidades específicas sobre los últimos avances referidos a esta temática (fundamentos, evolución, características, ventajas y desventajas, aplicaciones, estructura, eficiencias, etc.).

El estudio se centró en las tareas más complejas dentro del dominio de la visión artificial: la detección, la clasificación y la segmentación semántica de imágenes médicas. En un principio, se analizaron dos topologías de redes neuronales artificiales llamadas U-net y Yolo. Ambos tipos fueron desarrollados para la detección de objetos en imágenes, por lo tanto, sus arquitecturas pudieron servir como base para las definiciones de modelos futuros.

1.4.2. Recolección de datos

Esta etapa consistió en la búsqueda de los datos necesarios para llevar a cabo la investigación, los cuales pudieron provenir de diferentes sectores: sensores, páginas web, bases de datos públicas, etc. Particularmente, en el marco del proyecto, se tomaron muestras de dos datasets de tomografías de tórax (TC) haciendo posible el estudio de enfermedades pulmonares y la composición de la estructura de los pulmones.

1.4.3. Tratamiento y procesado de datos

Una vez completada la fase anterior, se procesaron los datos recolectados con el objetivo de obtener el mayor rendimiento posible. Se trató de un proceso de limpieza de los datos en

bruto, para lo cual se realizaron diferentes tareas, como, por ejemplo: la reducción de la dimensión, la normalización, la detección de valores atípicos, un análisis estadístico y gráfico de los datos, análisis de datos faltantes, entre otras operaciones. La recopilación de datos y su procesamiento constituyeron quizás las fases más importantes del análisis. Al finalizar esta etapa, se contó con un conjunto de datos "limpios", que contenía la suficiente información significativa y relevante del problema bajo estudio, lo que permitió ser analizado por las etapas siguientes con la mayor precisión posible.

Los estudios médicos que contenían imágenes, típicamente se encontraban bajo el estándar de comunicación DICOM (Digital Imaging and Communication On Medicine). Por esa razón, antes de lanzar el proceso de entrenamiento, fue necesario procesar todos los datos obtenidos y convertir dichas imágenes en un arreglo de píxeles. Una vez que se hizo la extracción, se realizaron varias transformaciones: afinidad y reducción del tamaño de la imagen. Por último, las imágenes se normalizaron con el objetivo de obtener un mejor rendimiento durante la fase de entrenamiento.

1.4.4. Evaluación de los modelos de predicción

Después del entrenamiento y testeo de las redes neuronales, se evaluó y comparó el desempeño de cada modelo predictivo en términos de parámetros estadísticos. Para esto, se utilizaron herramientas como matrices de confusión y diferentes métricas, como la exactitud, la precisión o la sensibilidad, entre otros. Esto resultó muy útil para visualizar la calidad de los modelos en función de las predicciones realizadas.

1.4.5. Análisis de los resultados y difusión de los mismos

Finalmente, se elaboraron las conclusiones correspondientes al trabajo desarrollado, remarcando el propósito del mismo, orientado al campo de la informática, la medicina y, en concreto, al análisis de imágenes médicas.

CAPÍTULO II

Estado del Arte

En la actualidad existen diversas maneras de abordar el problema de localización de ROI en imágenes médicas. Es por ello que, iniciamos la exploración del estado del arte a partir de la búsqueda de investigaciones y artículos académicos relacionados con la identificación de las ROI, el procesamiento de imágenes DICOM y los modelos de aprendizaje profundo. Con esto se pretende mostrar, en un principio, las técnicas y topologías de redes neuronales convolucionales (CNNs) empleadas para la extracción de características de las imágenes y qué resultados se obtuvieron al respecto, para tener noción de las metodologías empleadas y encabezar la propia implementación.

2.1. Análisis de las ROIs

En términos generales, las regiones de interés (ROI) hacen referencia a un conjunto de píxeles dentro de una imagen que son de particular interés para su análisis o estudio. Estas regiones se seleccionan porque contienen información importante sobre una condición específica. Por ejemplo, dentro del ámbito medicinal, puede ayudar en el diagnóstico de enfermedades, evaluar la eficacia de un tratamiento o llevar a cabo investigaciones médicas.

Las ROI se identifican mediante diversas técnicas, una muy utilizada es la segmentación, que consta de procesos automáticos o semiautomáticos que permiten delimitar y distinguir las regiones de interés en una imagen médica. Estas técnicas utilizan características como la intensidad de los píxeles, la textura, la forma o la ubicación espacial de las estructuras de interés.

Una vez que se ha identificado una ROI, se pueden extraer datos cuantitativos o cualitativos de esa región para su análisis. Esto puede incluir medidas de tamaño, forma, densidad, textura u otras características relevantes para el estudio médico en cuestión.

Las ROI se utilizan en diversas aplicaciones médicas, como por ejemplo:

- *Detección de tumores*: Las ROI pueden ayudar en la detección de tumores en diferentes partes del cuerpo, como cáncer de mama, cáncer de pulmón, cáncer de colon, entre otros.
- *Evaluación de enfermedades cardiovasculares o neurológicas*: Las ROI pueden utilizarse para analizar imágenes de resonancia magnética (MRI) o tomografía computarizada (CT) del corazón o del cerebro. Esto puede ayudar en la detección de

enfermedades cardíacas (enfermedad coronaria, anomalías estructurales, aneurismas, malformaciones vasculares) como también enfermedades neurológicas (tumores cerebrales, accidentes cerebrovasculares, esclerosis múltiple, enfermedad de Alzheimer, trastornos del movimiento).

- *Seguimiento de enfermedades renales*: La detección y seguimiento de ROI en imágenes de resonancia magnética (MRI) o tomografía computarizada (CT) de los riñones pueden ser valiosos para la evaluación de enfermedades renales, como tumores renales, quistes, infecciones o cálculos renales.

- *Monitoreo de enfermedades pulmonares*: La identificación y segmentación de ROI en imágenes de radiografías o tomografías computarizadas (CT) de los pulmones pueden ser útiles para evaluar enfermedades pulmonares como neumonía, tuberculosis, enfisema y cáncer de pulmón.

2.2. Visión Artificial

La visión artificial, también conocida como visión por computadora, combina el procesamiento de imágenes, la inteligencia artificial y los métodos de aprendizaje automático para examinar y comprender el contenido visual de las imágenes. Su objetivo principal es imitar cómo las personas ven, entienden e interactúan con su entorno para que luego pueda usarse en una variedad de contextos, incluida la medicina.

La visión artificial puede desempeñar un papel fundamental en este proceso al proporcionar herramientas y algoritmos avanzados para la detección y segmentación automatizada de las ROI. Es posible extraer características particulares de imágenes médicas y realizar análisis cuantitativos para localizar e identificar objetos utilizando técnicas de visión por computadora.

Las redes neuronales artificiales y los algoritmos de aprendizaje automático se utilizan en técnicas de visión artificial para entrenar modelos que puedan reconocer y categorizar las ROI en imágenes médicas. Estos modelos son capaces de detectar patrones sutiles, características distintivas y anomalías que son difíciles de ver a simple vista.

En este contexto, las tareas más significativas de la visión por computadora son la detección, la clasificación y la segmentación. A continuación, se describen brevemente cada una de estas tareas y su importancia:

- *Detección*: La detección de la ROI implica identificar y localizar áreas de interés en una imagen médica. Esta implicación trata de encontrar y ubicar objetos como tumores, órganos o estructuras anatómicas.

- *Clasificación*: La clasificación es el proceso de dar una etiqueta o categoría a una región específica de una imagen después de que se haya encontrado una ROI. La

clasificación puede determinar si una ROI detectada es benigna o maligna, por ejemplo, en el caso de imágenes de cáncer de mama.

- *Segmentación*: La segmentación es el proceso de dividir una imagen en distintas regiones o estructuras, separando y delimitando el ROI del resto de la imagen. Al permitir la extracción precisa de la ROI, la segmentación puede ayudar a medir el tamaño, la forma o la ubicación exacta de una estructura.

2.3. Búsqueda y Selección de Artículos

Iniciamos la investigación con la búsqueda de artículos e investigaciones con el tema principal planteado a través de diferentes bibliotecas digitales tales como: Google Académico, Semantic Scholar, IEEE Xplore, Pappers With Code y PubMed. Algunas palabras claves que fueron utilizadas son: “Región de interés”, “procesamiento de imágenes DICOM”, “redes neuronales convolucionales”, “visión artificial”, “Deep Learning”, entre otras.

Para la selección de los artículos se optó por escoger aquellos que estén actualizados, que hayan sido ampliamente revisados y que cuenten con una sólida base de citaciones en la comunidad científica. Esto asegura que la investigación se encuentre en la vanguardia del conocimiento y que los resultados presentados sean respaldados por un riguroso proceso de revisión y evaluación académica.

La Tabla 1 representa algunos de los artículos e investigaciones más relevantes que fueron consultados:

Tabla 1. Artículos e investigaciones destacados

#	Título	Arquitecturas utilizadas (CNNs)	Fuente
I	Automated Breast Cancer Detection and Classification in Full Field Digital Mammograms Using Two Full and Cropped Detection Paths Approach	YOLOv4, ResNet50, VGG16 y Inception V3	IEEE
II	A fully integrated computer-aided diagnosis system for digital X-ray mammograms via deep learning detection, segmentation, and classification	YOLO-Based, U-Net, FrCN y SegNet	PubMed
III	Automatic Liver Segmentation from CT Images Using Deep Learning Algorithms: A Comparative Study	U-Net y RestNet	Paper with code
IV	Efficacy evaluation of 2D, 3D U-Net semantic segmentation and atlas-based segmentation of normal lungs excluding the trachea and main bronchi	U-Net (2D - 3D)	Semantic Scholar

V	Classification of malignant lung cancer using deep learning	AlexNet y Google-Net	PubMed
VI	Left Ventricle Segmentation and Volume Estimation on Cardiac MRI using Deep Learning	U-Net, VGG, Google-Net y Resnet	PubMed
VII	Diagnosing Parkinson's disease in Early Stages using Image Enhancement, ROI Extraction and Deep Learning Algorithms	ResNet-34, VGG-19, ResNet-50 y AlexNet	IEEE
VIII	An Effective Sign Language Learning with Object Detection Based ROI Segmentation	YOLO	IEEE
IX	Localization and Edge-Based Segmentation of Lumbar Spine Vertebrae to Identify the Deformities Using Deep Learning Models	YOLOv5, RestNet y U-Net	Semantic Scholar
X	Fighting against COVID-19: A novel deep learning model based on YOLO-v2 with ResNet-50 for medical face mask detection	YOLOv2 y ResNet-50	Semantic Scholar
XI	An Improved Algorithm for Detecting Pneumonia Based on YOLOv3	YOLOv3	Semantic Scholar
XII	Diagnosis of Brain Tumor Using Light Weight Deep Learning Model with Fine-Tuning Approach	YOLOv5	Semantic Scholar
XIII	A Combined Approach for Accurate and Accelerated Teeth Detection on Cone Beam CT Images	YOLOv3 y Faster R-CNN	PubMed
XIV	Combination of UNet++ and ResNeSt to classify chronic inflammation of the choledochal cystic wall in patients with pancreaticobiliary maljunction	UNet++ y ResNeSt	PubMed

2.4. Elección de Modelos

A partir de los artículos consultados, se realizó énfasis en el análisis de dos modelos de redes neuronales ampliamente conocidas y estudiadas: YOLO (You Only Look Once) y U-Net. La elección de estas redes se basa en una revisión exhaustiva de investigaciones recientes que han demostrado su efectividad dentro del ámbito de estudios meramente clínicos.

En primer lugar, diversos estudios han explorado el uso de YOLO para la detección y clasificación de anomalías médicas, en particular en el ámbito de la detección de cáncer de mama. La investigación titulada "A fully integrated computer-aided diagnosis system for digital X-ray mammograms via deep learning detection, segmentation, and classification" describe un sistema de diagnóstico asistido por computadora (CAD) completamente integrado para mamografías de rayos X digitales. El sistema utiliza detección, segmentación

y clasificación basada en aprendizaje profundo para proporcionar un diagnóstico preciso del cáncer de mama. Se implementó la detección “Mass Detection” (detección en masa) basada en el algoritmo YOLO. Además, el artículo "Fighting against COVID-19: A novel deep learning model based on YOLO-v2 with ResNet-50 for medical face mask detection" demuestra la utilidad de YOLO en la detección de mascarillas médicas para combatir la propagación de enfermedades infecciosas, como el COVID-19.

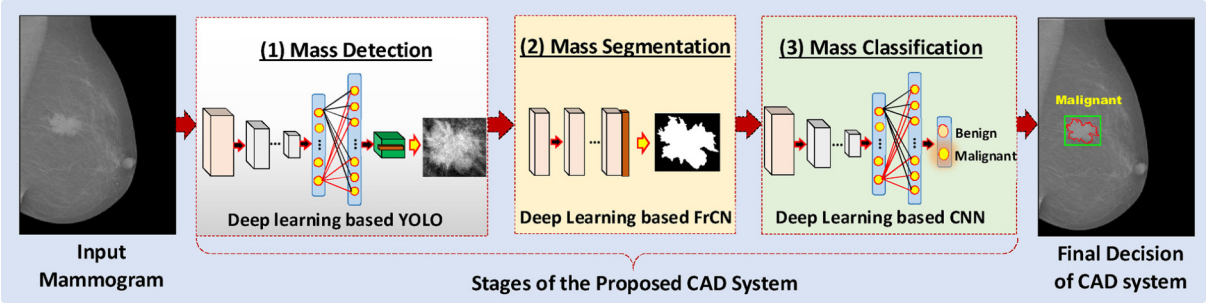


Figura 1. Diagrama esquemático del sistema de diagnóstico asistido por computadora (CAD) basado en aprendizaje profundo para detectar, segmentar y clasificar masas de cáncer de mama a partir de mamografías digitales de rayos X de entrada (Adaptado de Al-Antari, M. A., Al-Masni, M. A., Choi, M. T., Han, S. M., & Kim, T. S. 2018, https://doi.org/10.1016/j.ijmedinf.2018.06.003).

En segundo lugar, la red neuronal U-Net ha sido ampliamente utilizada para la segmentación precisa de estructuras anatómicas en imágenes médicas. El artículo "Automatic Liver Segmentation from CT Images Using Deep Learning Algorithms: A Comparative Study" presenta una investigación comparativa que evalúa diferentes algoritmos de aprendizaje profundo, incluyendo U-Net, para la segmentación automática del hígado en imágenes de tomografía computarizada (CT). Asimismo, el estudio "Left Ventricle Segmentation and Volume Estimation on Cardiac MRI using Deep Learning" se centra en la segmentación y estimación de volumen del ventrículo izquierdo del corazón utilizando U-Net y resonancia magnética cardíaca, proporcionando una herramienta automatizada para la evaluación del funcionamiento cardíaco.

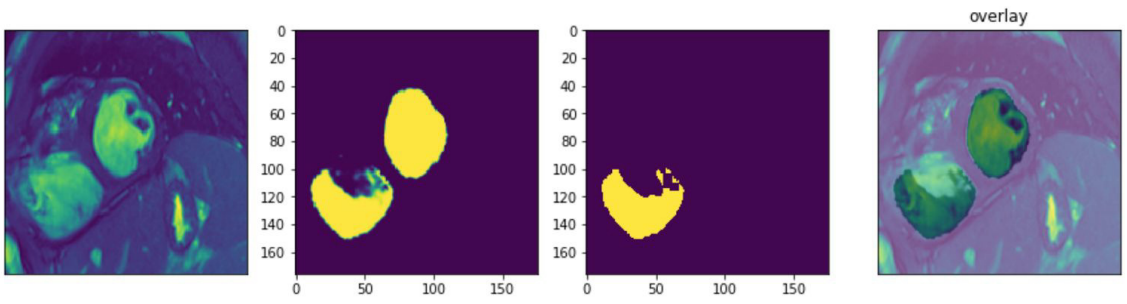


Figura 2. Segmentación del ventrículo utilizando la arquitectura U-Net (Adaptado de Abdelmaguid, E., Huang, J., Kenchareddy, S., Singla, D., Wilke, L., Nguyen, M.H., & Altintas, I. (2018). Left Ventricle Segmentation and Volume Estimation on Cardiac MRI using Deep Learning. ArXiv, abs/1809.06247)

Estas investigaciones subrayan la importancia de implementar redes neuronales como YOLO y U-Net en aplicaciones médicas y de diagnóstico. Las capacidades de detección precisa de YOLO y las capacidades de segmentación detallada de U-Net brindan beneficios significativos para mejorar el diagnóstico temprano, la clasificación precisa y la segmentación de regiones de interés en imágenes médicas.

CAPÍTULO III

Marco Teórico

3.1. Inteligencia Artificial

La Inteligencia Artificial (IA) es un campo de la informática que busca la creación de máquinas que puedan imitar el comportamiento inteligente. Este campo, que está cada vez más presente en los ámbitos de nuestra vida cotidiana, se incorpora en muchas herramientas de uso común para múltiples propósitos. Aunque los términos "Inteligencia Artificial", "Machine Learning" y "Deep Learning" a menudo se utilizan de manera intercambiable, no son lo mismo, pero sí están relacionados entre sí.

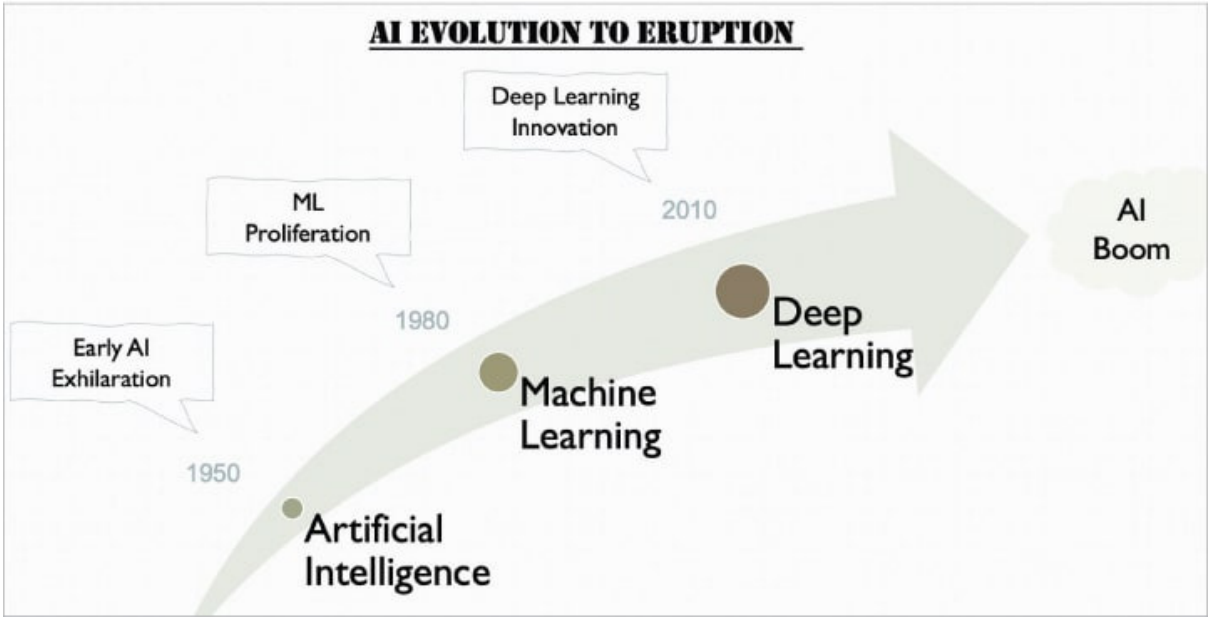


Figura 3. Evolución de la inteligencia artificial

La IA surgió en la década de 1950 como un subcampo dentro de la ciencia de la computación, que para ese entonces era una disciplina nueva. En sus inicios, la IA buscaba alcanzar la capacidad de procesamiento inteligente mediante sentencias y reglas lógicas, que eran programadas manualmente. Hasta el final de la década de 1980, este fue el paradigma predominante en el campo de la IA. Sin embargo, la implementación de Machine Learning cambió este paradigma, generando sistemas estocásticos y dotando a las máquinas de la habilidad de "aprender" a resolver tareas. Esto significó un salto en el rendimiento de los sistemas informáticos, brindando a sus usuarios la posibilidad de procesar grandes volúmenes

de información, analizar, tomar o recomendar decisiones y actuar en consecuencia, predecir y generar información inexistentes previamente.

A partir de la IA surgieron nuevos subcampos como la robótica, los sistemas expertos, la visión por computadora, el procesamiento de lenguaje natural, procesamiento de voz, machine learning, entre otras. Estos subcampos se convirtieron en los primeros grandes avances de la informática dentro de los procesos productivos, como por ejemplo, el uso de brazos mecánicos en las fábricas automotrices. Pero estos sistemas primitivos eran deterministas, repetían un proceso, una serie de pasos, que dotaban a estos sistemas de la sensación de que eran "inteligentes" o que se comportaban inteligentemente, pero ese comportamiento se adjudicaba al programador y no a la máquina.

Cada uno de estos subcampos se describe brevemente a continuación:

- *La robótica*: se ocupa del diseño, construcción, operación, estructura, manufactura, y aplicación de los robots. Incluye todos aquellos dispositivos que tienen interacción directa con el mundo físico.
- *La visión artificial*: aquí se encuentran aquellos software capaces de analizar información a través de imágenes o videos.
- *El lenguaje natural*: se encarga de entender, interpretar y manipular el lenguaje humano. Toma elementos prestados de muchas disciplinas, incluyendo la ciencia de la computación y la lingüística computacional, en su afán por cerrar la brecha entre la comunicación humana y el entendimiento de las computadoras.
- *La voz*: tiene el objetivo de procesar la voz humana con el fin de entenderla, interpretarla y actuar en consecuencia. Se encarga principalmente de "traducir" la voz en texto y viceversa.

La evolución de estos campos de la IA ha sido constante y sigue en desarrollo, marcando una línea de tiempo en la historia de la ciencia de la computación.

3.2. Aprendizaje Automático

El Machine Learning o Aprendizaje Automático (ML) se basa en una serie de algoritmos que se ejecutan en una máquina y permiten la elaboración de modelos que "aprenden" de manera automática sin estar específicamente programadas para tal fin. El "aprendizaje" de las computadoras se refiere a la capacidad para identificar patrones en grandes conjuntos de datos y a través de ellos tomar decisiones, o hacer una predicción acerca de comportamientos futuros de una situación utilizando un análisis estadístico.

ML está estrechamente relacionado con las estadísticas matemáticas, pero presenta algunas diferencias. Por una parte, el ML tiende a trabajar con conjuntos de datos tan grandes y complejos que para el análisis estadístico clásico sería muy poco práctico de llevar cabo.

Por otra parte, el aprendizaje automático está orientado a la ingeniería en donde las ideas se prueban empíricamente con mucha más frecuencia que teóricamente. ML experimentó un crecimiento exponencial durante la última década, esto en gran parte se debe al crecimiento de la capacidad de cómputo y al Big Data, procesamiento de datos a gran escala.

Al aplicar técnicas de ML se presenta un notable cambio en el paradigma de programación. El paradigma de programación tradicional o clásico tiene como entrada las distintas reglas y los datos, con ellos el programa arroja como resultado una respuesta; en cambio, en el paradigma propuesto por ML las entradas son los datos y las respuestas esperadas y el programa con estas entradas devolverá las reglas, que podrán ser aplicadas a nuevos datos para producir respuestas originales. Un sistema de aprendizaje automático es "entrenado" en lugar de ser "programado", se presentan muchos ejemplos relevantes de una tarea y el sistema encuentra estructuras estadísticas con estos ejemplos, con lo que eventualmente el sistema crea reglas para automatizar la tarea.

La IA comprende un ámbito muy grande. Un sistema de IA es aquel que tenga la capacidad de aprender, actuar y adaptarse a las situaciones que se le presenten. Dentro del ámbito de la IA se encuentra el ML como un subconjunto y este está compuesto de algoritmos cuyo rendimiento mejora cuando se procesa una mayor cantidad de datos. El aprendizaje automático es utilizado para modelar, identificar, optimizar, predecir, pronosticar y controlar el comportamiento dinámico de diferentes sistemas reales.

Las técnicas de aprendizaje automático se pueden clasificar en 3 categorías: aprendizaje supervisado, aprendizaje no supervisado y aprendizaje por refuerzo. Se denomina aprendizaje al proceso por el cual la máquina mejora sus capacidades para resolver un problema. Existen diferentes tipos de aprendizaje en función del tipo de problema al que se enfrenta y de la técnica que se utilice.

3.3. Aprendizaje Supervisado

El Aprendizaje Supervisado es una técnica de Machine Learning que se basa en aprender a partir de información histórica, experiencias conocidas o conjuntos de datos etiquetados. En este enfoque, para cada vector de variables de entrada, se dispone del vector de variables de resultados esperado. Es decir, para cada vector xi se dispone de la variable de respuesta yi, donde i = 1, 2, 3,…, n.

Este mecanismo de aprendizaje hace alusión a la forma de identificar patrones que poseen los seres humanos. Por ejemplo, si una persona sabe que para un dado input de valor 2 se obtiene como output un 4, y luego para un 3 el resultado es 6, y para un 4 el resultado es 8, ante la pregunta sobre cuál es el output si el valor ingresado es un 5, seguramente la respuesta será 10, dado que la persona aprendió que el output se obtiene a partir de multiplicar por 2 el input. El mismo razonamiento es el que se utiliza en esta técnica de aprendizaje.

El modelo en el aprendizaje supervisado intentará ajustar sus coeficientes para que la diferencia entre la estimación y el valor de respuesta original sea lo menor posible. Para ilustrar cómo funciona el aprendizaje supervisado, se puede poner un ejemplo en donde se desea clasificar fotos de perros y gatos. Se debe contar con un conjunto de imágenes de estos animales y se debe identificar previamente cuál es el animal que se encuentra en cada una de las imágenes y etiquetar dicha imagen. Una vez terminado el proceso anterior, este set de datos será utilizado por el algoritmo durante su entrenamiento para aprender las relaciones existentes entre las imágenes y el tipo de animal contenido en ella y, así posteriormente, clasificar nuevas imágenes no etiquetadas que contengan alguno de estos animales.

Con este tipo de aprendizaje se pueden realizar tareas de clasificación y de regresión. La tarea será de "clasificación" cuando el tipo de valor esperado en la predicción sea discreto, por ejemplo, cuando el modelo debe decidir entre dos categorías como pueden ser objetos "reciclables" y "no reciclables". En caso de haber más de dos clases se trataría de un modelo de clasificación multiclase.

Por otro lado, la tarea de "regresión" corresponde a modelos en donde el valor de salida es continuo, es decir, no se pueden clasificar los resultados en clases. En resumen, el aprendizaje supervisado es una técnica poderosa para resolver problemas de clasificación y regresión en Machine Learning.

Algunas de las técnicas más utilizadas en el aprendizaje supervisado incluyen:

- *Regresión lineal*: Esta técnica se utiliza para predecir una variable continua a partir de una o más variables independientes. Por ejemplo, se podría utilizar la regresión lineal para predecir el precio de una casa en función de su tamaño y ubicación.

- *Regresión logística*: Esta técnica se utiliza para predecir una variable categórica a partir de una o más variables independientes. Por ejemplo, se podría utilizar la regresión logística para predecir si un correo electrónico es spam o no en función de su contenido.

- *Árboles de decisión*: Esta técnica se utiliza para predecir una variable de salida a partir de una serie de reglas basadas en las variables de entrada. Por ejemplo, se podría utilizar un árbol de decisión para predecir si un cliente comprará un producto en función de su edad, ingresos y comportamiento de compra anterior.

- *Redes neuronales*: Esta técnica se utiliza para modelar relaciones complejas entre las variables de entrada y salida. Por ejemplo, se podría utilizar una red neuronal para predecir el precio de las acciones en función de una serie de indicadores económicos y financieros.

- *Máquinas de vectores de soporte (SVM)*: Esta técnica se utiliza para clasificar datos en dos o más clases. Por ejemplo, se podría utilizar una SVM para clasificar imágenes de perros y gatos, como se mencionó en el texto.

- *K-Nearest Neighbors (K-NN)*: Esta técnica se utiliza para clasificar un objeto basándose en las clases de sus vecinos más cercanos. Por ejemplo, se podría utilizar K-NN para clasificar un nuevo cliente en un segmento de mercado basándose en los segmentos de los clientes más similares.

Estas técnicas permiten a las máquinas aprender de los datos y hacer predicciones precisas, lo que puede ser útil en una amplia gama de aplicaciones, desde la detección de spam hasta la predicción de precios de viviendas y la clasificación de imágenes.

3.4. Aprendizaje No Supervisado

El Aprendizaje No Supervisado es una técnica de Machine Learning que, a diferencia del aprendizaje supervisado, no se basa en datos etiquetados y no se conoce cuál es el resultado que se está buscando. Este tipo de aprendizaje busca interpretar patrones más abstractos y se basa en métodos estadísticos para devolver el resultado. En definitiva, busca emular la manera en que se tiene de aprender el idioma natal.

En el aprendizaje no supervisado, los datos de entrenamiento no incluyen las etiquetas, por lo tanto, será el algoritmo el que buscará realizar la clasificación por sí mismo. Es decir, cada vector de observación xi no cuenta con la variable de respuesta yi, el modelo debe autoajustarse buscando patrones y relaciones de dependencia entre las variables observadas. Siguiendo con un ejemplo, en la clasificación de imágenes de perros y gatos, el sistema será capaz de distinguir si la imagen contiene un perro o un gato, pero no sabrá a cuál categoría pertenece hasta que se le indique.

Este tipo de aprendizaje presenta una de las principales fronteras de la Inteligencia Artificial. Es un tipo de aprendizaje que busca modelar una estructura subyacente o una distribución de datos teniendo como premisa "aprender" más sobre estos.

Existen varias técnicas de aprendizaje no supervisado, entre las que se incluyen:

- ➢ *Clustering*: Esta técnica se utiliza para agrupar datos similares en grupos, donde los datos dentro de un mismo grupo son más similares entre sí que con los de otros grupos. Un ejemplo de esto es la segmentación de clientes en marketing.
- ➢ *Reducción de dimensionalidad*: Esta técnica se utiliza para reducir el número de variables en un conjunto de datos, mientras se mantiene la mayor cantidad de información posible. Un ejemplo de esto es el análisis de componentes principales (PCA).
- ➢ *Detección de anomalías*: Esta técnica se utiliza para identificar datos que se desvían de la norma. Un ejemplo de esto es la detección de fraudes en transacciones financieras.

- ➢ *Asociación de reglas*: Esta técnica se utiliza para descubrir relaciones entre variables en grandes conjuntos de datos. Un ejemplo de esto es el análisis de cestas de la compra en el comercio minorista.

Algunas de las técnicas más utilizadas en el aprendizaje no supervisado incluyen:

1. *Autoencoders*: Son redes neuronales que se utilizan para aprender representaciones eficientes de los datos, a menudo con el objetivo de reducir la dimensionalidad. Los autoencoders son útiles en tareas de compresión de datos y eliminación de ruido.

2. *K-means*: Es un algoritmo de clustering que divide un conjunto de observaciones en K clusters, donde cada observación pertenece al cluster con la media más cercana. Es útil en tareas de segmentación de mercado, clasificación de documentos y análisis de imágenes.

3. *DBSCAN*: Es un algoritmo de clustering basado en densidad que puede descubrir clusters de forma y tamaño arbitrarios. Es especialmente útil cuando los clusters son de formas irregulares o cuando hay ruido en los datos.

4. *Análisis de correspondencias*: Es una técnica que se utiliza para visualizar la relación entre dos conjuntos de datos categóricos. Se utiliza a menudo en el análisis de encuestas y en la exploración de datos textuales.

5. *Aprendizaje semi-supervisado*: Es una técnica que combina elementos del aprendizaje supervisado y no supervisado. Se utiliza cuando se dispone de una gran cantidad de datos no etiquetados y una pequeña cantidad de datos etiquetados.

Estas técnicas permiten a las máquinas aprender de los datos sin necesidad de una guía explícita, lo que puede ser útil en una amplia gama de aplicaciones, desde la detección de anomalías hasta la recomendación de productos y la visualización de datos de alta dimensión.

3.5. Aprendizaje por Refuerzo

El Aprendizaje por Refuerzo es una técnica de Machine Learning que se basa en la metodología de premio-castigo. En este enfoque, los modelos de aprendizaje deben buscar los resultados sin conocerlos previamente. Ante una acción o resultado determinado, se otorgará una recompensa: si el resultado es el esperado, se otorgará una recompensa positiva, y en caso contrario, se otorgará una recompensa negativa.

La metodología de trabajo que utilizan estos algoritmos está compuesta por dos elementos, un entorno y un agente. El entorno le envía un estado al agente, este, basándose en su conocimiento, realizará una acción para responder a ese estado y esta acción es devuelta al entorno. Posteriormente, el entorno enviará al agente un nuevo estado, pero esta vez acompañado de una recompensa referente a la acción recibida del estado anterior. Con esta

recompensa, el agente actualizará su conocimiento y luego evaluará el nuevo estado. Este bucle continuará hasta que el entorno envíe la señal indicando la finalización del proceso.

Este método de aprendizaje es similar a cómo los seres humanos aprenden a jugar un nuevo juego, como por ejemplo, Mario Bros. Cada vez que el jugador recoge una moneda o un hongo, recibe un premio, y cada vez que es tocado por una tortuga, recibe un castigo. El jugador, indirectamente, interpreta estas situaciones intentando conseguir la mayor cantidad de premios y los mínimos castigos.

Algunos ejemplos de técnicas de aprendizaje por refuerzo incluyen:

- *Q-Learning*: Es un algoritmo de aprendizaje por refuerzo que busca aprender la política óptima que maximiza la recompensa total esperada. Por ejemplo, se podría utilizar Q-Learning para entrenar a un agente a jugar a un juego como el ajedrez o el Go.

- *Deep Q-Networks (DQN)*: Es una variante de Q-Learning que utiliza redes neuronales para aproximar la función de valor Q. Por ejemplo, Google's DeepMind utilizó DQN para entrenar a un agente a jugar a varios juegos de Atari a un nivel superhumano.

- *Policy Gradients*: Es una familia de algoritmos de aprendizaje por refuerzo que busca optimizar directamente la política del agente. Por ejemplo, se podría utilizar Policy Gradients para entrenar a un robot a realizar una tarea compleja como caminar o correr.

- *Actor-Critic Methods*: Son una familia de algoritmos de aprendizaje por refuerzo que combinan las ideas de policy gradients y value function approximation. Por ejemplo, se podría utilizar Actor-Critic Methods para entrenar a un agente a jugar a un juego de estrategia en tiempo real como StarCraft II.

3.6. Aprendizaje Profundo

El Deep Learning, o Aprendizaje Profundo, es una subcategoría del Machine Learning, y se caracteriza por su modelo de representación de capas de procesamiento no lineal. Estas capas aprenden de manera autónoma al ser alimentadas con grandes volúmenes de datos, y son responsables de descifrar patrones ocultos, formando una jerarquía de características desde un nivel de abstracción más bajo hasta uno más alto. Entre las herramientas principales del Deep Learning se encuentran las redes neuronales artificiales, incluyendo las redes neuronales artificiales profundas (DNN), las redes neuronales convolucionales (CNN) y las redes neuronales recurrentes (RNN).

El Deep Learning se diferencia de los algoritmos de aprendizaje poco profundos por la cantidad de transformaciones que se aplican a la señal a medida que se propaga desde la capa de entrada hacia la capa de salida. Aunque no hay un número mínimo de capas ocultas para

que un modelo sea considerado de aprendizaje profundo, generalmente se espera que tenga al menos más de dos capas intermedias.

Los modelos de Deep Learning han mejorado significativamente las técnicas de reconocimiento de voz, clasificación de objetos visuales, detección de objetos, entre otros. Buscan que una máquina pueda aprender a realizar tareas que son naturales para las personas, como aprender a través de ejemplos. En ocasiones, estos modelos pueden alcanzar una precisión que supera el rendimiento humano.

El auge del Deep Learning se ha dado principalmente en la última década, aunque sus primeras teorías se desarrollaron en la década de 1980. Su utilidad ha crecido exponencialmente debido a dos factores: la disponibilidad de grandes cantidades de datos y la necesidad de una gran potencia de cálculo computacional. El avance de las unidades de procesamiento gráfico (GPU) de alto rendimiento y la computación en la nube han permitido reducir considerablemente los tiempos necesarios para el entrenamiento de una red de Deep Learning.

En resumen, el Deep Learning es una forma especializada de Machine Learning. A diferencia de los modelos de Machine Learning, donde la extracción de características relevantes se hace de forma manual, en los modelos de Deep Learning, las características relevantes son extraídas directamente por el modelo. Además, en los algoritmos de Deep Learning, la precisión suele aumentar proporcionalmente con el aumento de la cantidad de datos que se alimenta al modelo, mientras que en el aprendizaje superficial existen problemas de convergencia. Por lo tanto, una ventaja fundamental de los modelos de Deep Learning es su capacidad para seguir mejorando a medida que aumenta el tamaño de los datos.

3.7. Redes Neuronales Artificiales

La técnica de las Redes Neuronales Artificiales (RNA) fue desarrollada a mediados del siglo pasado, pero su verdadero potencial ha comenzado a ser explotado recientemente, gracias al avance significativo en la capacidad de procesamiento de las computadoras.

Las Redes Neuronales Artificiales reciben su nombre debido a que su funcionamiento se inspira en el de las neuronas biológicas y su interacción en los seres vivos. Al igual que en el cerebro humano, donde las neuronas se interconectan y transmiten señales para aprender de la experiencia, las RNA se componen de unidades de procesamiento interconectadas, llamadas "nodos" o "neuronas artificiales", que trabajan juntas para aprender de los datos.

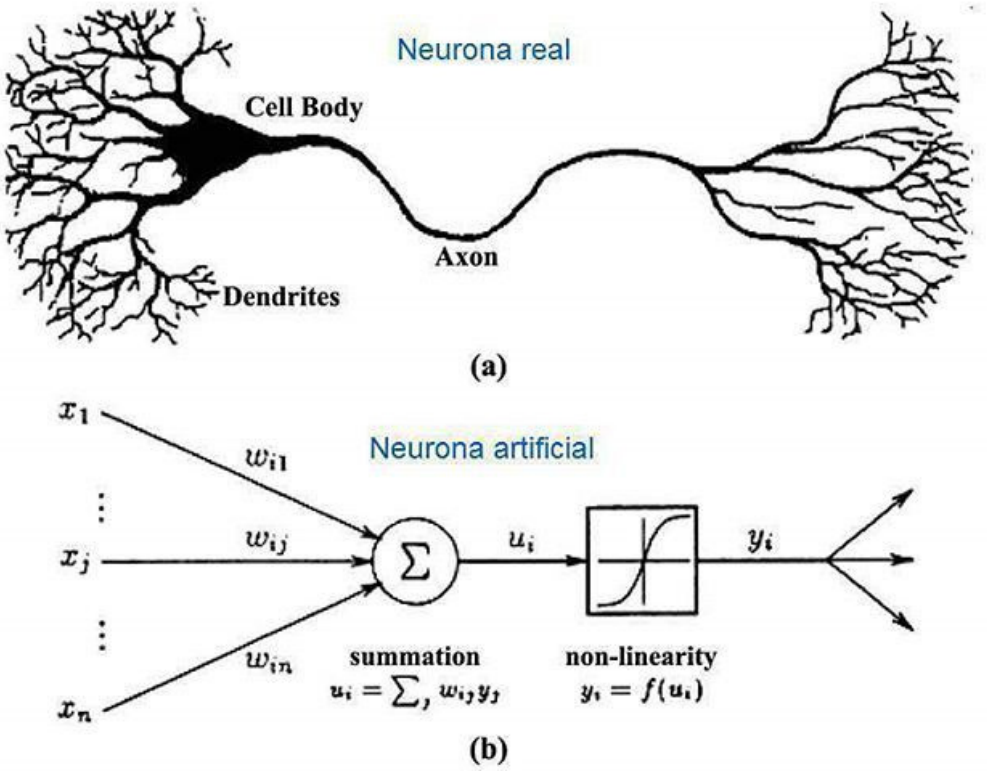


Figura 4. a) Diagrama de neurona biológica. b) Diagrama de neurona artificial

3.7.1. Arquitectura

El Deep Learning utiliza algoritmos que permiten la creación de modelos compuestos por múltiples capas de procesamiento. Estos modelos son capaces de aprender diferentes representaciones de datos, permitiendo múltiples niveles de abstracción y realizando transformaciones no lineales. A partir de las entradas, se generan salidas que se aproximan a las esperadas.

Una red neuronal, que es la base de estos modelos, se organiza en tres tipos de capas:

1. *Capa de entrada*: Esta es la capa que recibe los datos de entrada.
2. *Capas ocultas*: Estas pueden estar compuestas por una o más capas de neuronas. Cada capa puede tener una cantidad distinta de neuronas. Si la red solo contiene una capa de neuronas en la capa oculta, se considera una red neuronal simple. Si hay dos o más capas en la capa oculta, se dice que la red neuronal es profunda.
3. *Capa de salida*: Esta es la capa que devuelve la predicción realizada.

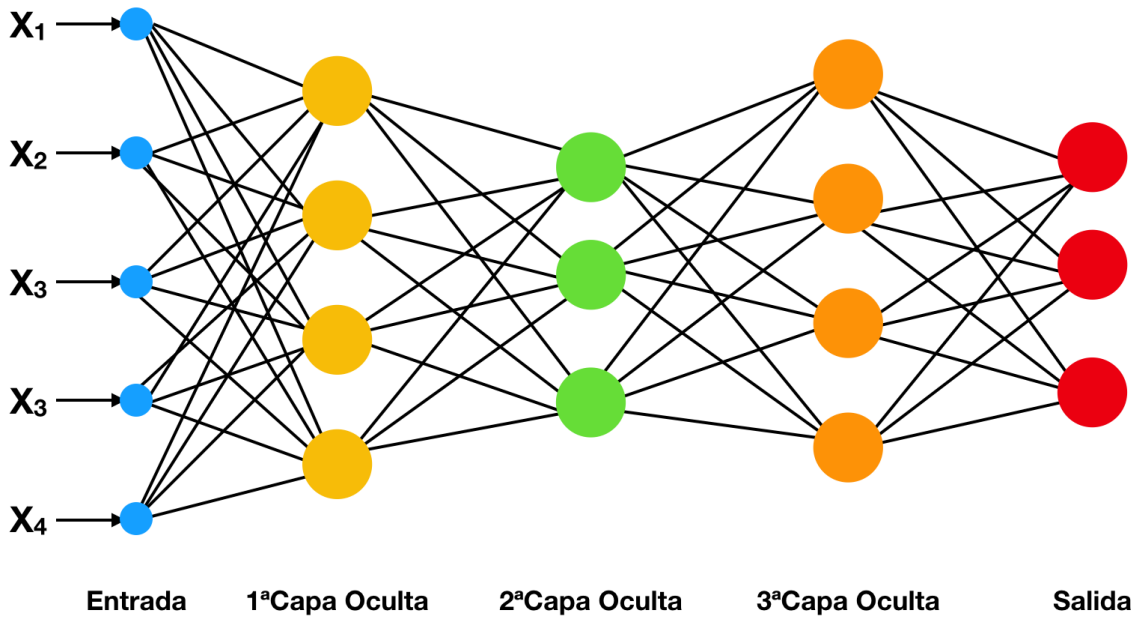


Figura 5. Arquitectura de una red neuronal con capas densamente conectadas

En la arquitectura más tradicional de las redes neuronales, cada neurona tiene una conexión con todas las neuronas de la siguiente capa. Esto se conoce como capas densamente conectadas.

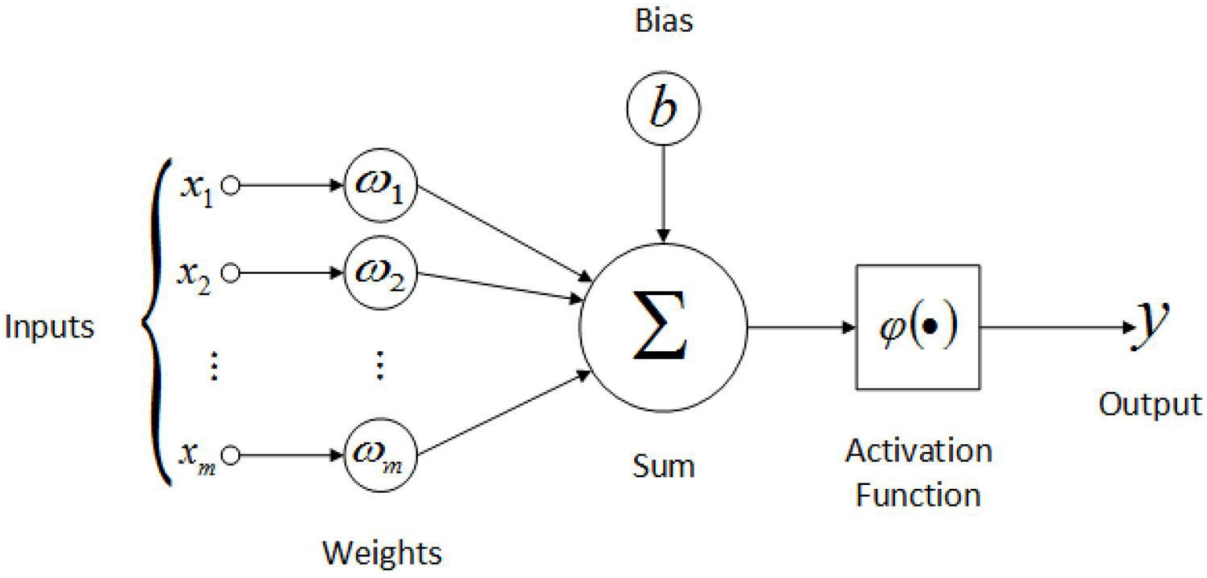


Figura 6. Arquitectura de un perceptrón

Para entender el funcionamiento de una red neuronal, podemos considerar la arquitectura más simple, es decir, una única neurona, también llamada perceptrón. Cada entrada de la neurona tiene un valor (xi) que se asocia a un peso (wi). Este peso se multiplica por el valor de la entrada. Los pesos son de suma importancia ya que son los valores que se irán ajustando

durante el entrenamiento de la red. Luego, se realiza la suma de todos los valores de entrada multiplicados por sus respectivos pesos. A esta suma se le aplica una función de activación (f) para obtener la salida de la neurona.

Además de los pesos, las redes neuronales también utilizan un término llamado "bias", que es una constante que se suma a la entrada ponderada antes de aplicar la función de activación. El bias permite ajustar la salida de la neurona junto con los pesos.

Las funciones de activación son esenciales en las redes neuronales, ya que introducen no linealidades en el modelo, permitiendo que la red pueda aprender de datos más complejos. Algunas funciones de activación comunes incluyen la función sigmoide, la función ReLU (Rectified Linear Unit) y la función tanh (tangente hiperbólica).

3.7.2. Función de activación

La función de activación es un componente esencial en las redes neuronales, que se aplica a cada neurona de la red. Esta función recibe como entrada el resultado de la suma ponderada de las entradas de la neurona y realiza una transformación, produciendo un nuevo valor. Este valor es la salida de la neurona y se convierte en la entrada para las neuronas de la siguiente capa. La función de activación ayuda a determinar qué neuronas se activarán en la siguiente capa.

En el contexto del Deep Learning, las funciones de activación son no lineales. Si se optara por una función de activación lineal, la red se comportaría como una sola neurona y solo sería capaz de resolver problemas muy simples. Esto se debe a que la suma de múltiples funciones lineales da como resultado también una función lineal. Para evitar esto y permitir que la red aprenda de datos más complejos, se utilizan funciones de activación no lineales.

Existen varios tipos de funciones de activación no lineales. Entre las más tradicionales encontramos la función sigmoide y la función tangente hiperbólica. Sin embargo, en la actualidad, se utilizan con frecuencia funciones de activación como la Unidad Lineal Rectificada (ReLU) o sus derivados, como la función ELU (Exponential Linear Unit) y Leaky ReLU.

La función sigmoide transforma los valores de entrada a un rango entre 0 y 1, lo que la hace útil para problemas de clasificación binaria. La función tangente hiperbólica, por otro lado, transforma los valores de entrada a un rango entre -1 y 1. La función ReLU es popular en las redes neuronales profundas porque introduce no linealidad sin afectar a los valores positivos de entrada, pero establece todos los valores negativos en cero, lo que ayuda a mitigar el problema del desvanecimiento del gradiente. Las variantes de ReLU, como ELU y Leaky ReLU, intentan abordar el problema de las neuronas "muertas" que pueden ocurrir con ReLU al permitir pequeños valores negativos cuando la entrada es menor que cero.

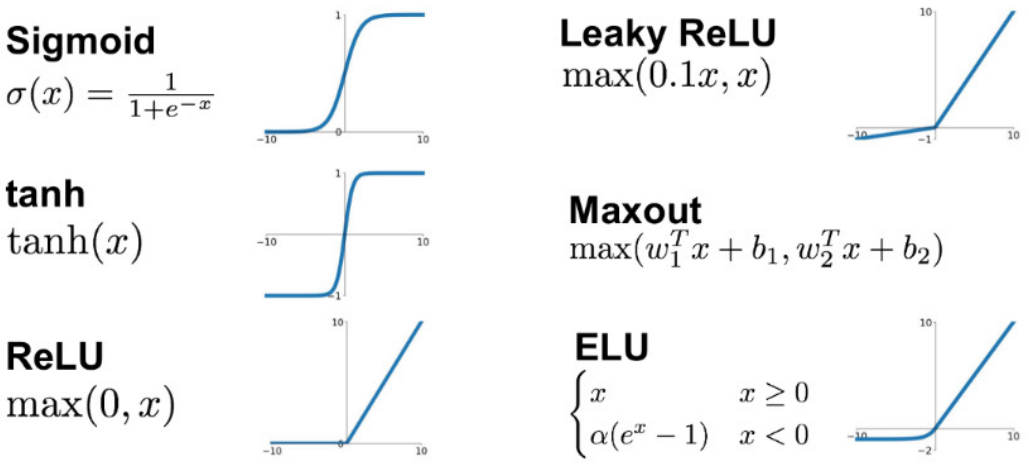


Figura 7. Comportamiento de las funciones de activación no lineales

Cada una de estas funciones de activación tiene un comportamiento único y se elige en función de las necesidades específicas del modelo de red neuronal.

3.7.3. Entrenamiento

El proceso de aprendizaje de una red neuronal es un procedimiento iterativo que implica ajustar los pesos de las neuronas para que los valores de entrada de la red permitan obtener la salida esperada. Este proceso es fundamental en el aprendizaje supervisado, donde para cada valor de entrada se conoce el valor de la salida esperada. De esta manera, en cada iteración del entrenamiento, los pesos de cada neurona se ajustan gradualmente de modo que, para todas las entradas, las salidas obtenidas sean las correctas.

Este proceso de aprendizaje se puede visualizar como un ciclo de ida y vuelta a través de las diferentes capas de la red neuronal. Los pasos más destacados de este ciclo son la propagación hacia adelante (forward propagation) y la propagación hacia atrás (backpropagation).

En la propagación hacia adelante, los datos de entrada se pasan a través de la red, desde la capa de entrada hasta la capa de salida, utilizando los pesos y bias actuales de la red. Luego, en la evaluación, se compara la salida obtenida con la salida esperada para calcular el error.

Posteriormente, en la propagación hacia atrás, este error se propaga de vuelta a través de la red, desde la capa de salida hasta la capa de entrada, ajustando los pesos y bias en el camino para minimizar el error. Finalmente, en la optimización, se actualizan los pesos y bias de la red utilizando un algoritmo de optimización, como el descenso de gradiente.

El siguiente diagrama ilustra este proceso

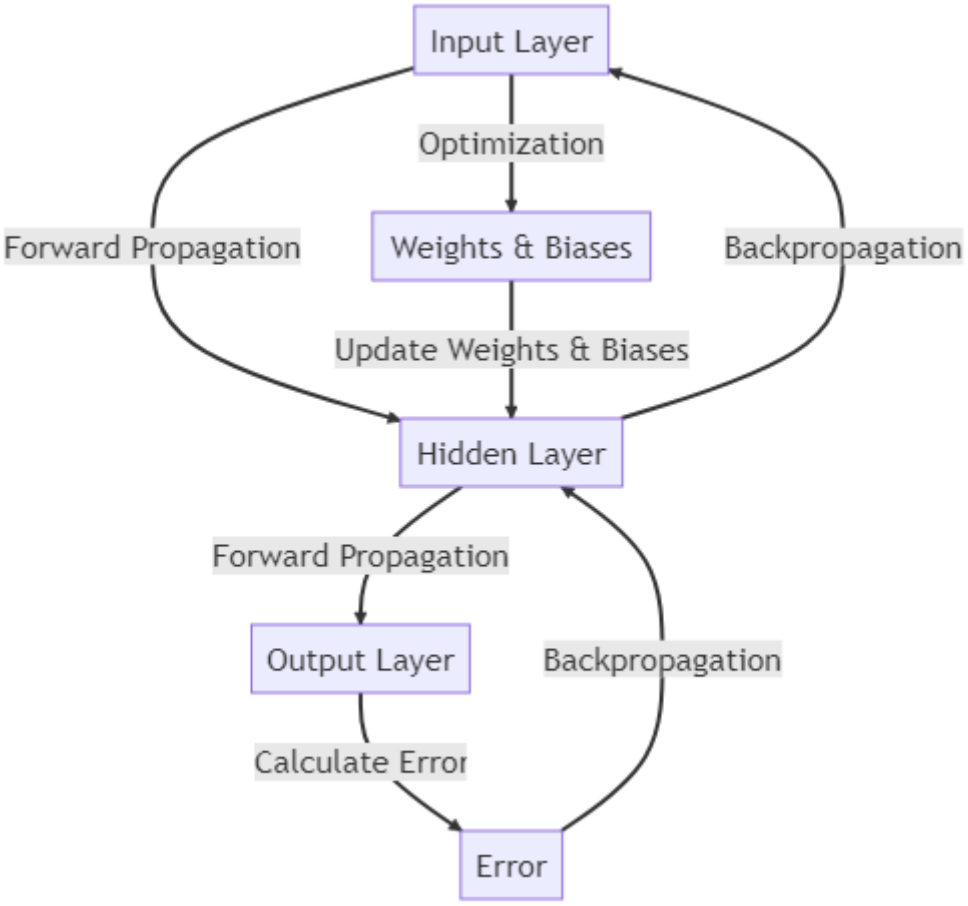


Figura 8. Diagrama de flujo de entrenamiento, Forward propagation, cálculo de error y Backpropagation

Forward Propagation

El entrenamiento de una red neuronal comienza con la propagación de los datos de entrada a través de toda la red, un proceso conocido como propagación hacia adelante o forward propagation. En este proceso, las neuronas realizan cálculos internos y propagan sus resultados a través de las capas de la red.

Siendo X el vector de datos de entrada, se puede decir que $a(1) = X$. Por lo tanto, para obtener el resultado de la red, se realiza el siguiente cálculo en cada capa L de la red:

$$Z(L) = (W(L)\, a(L-1) + B(L)) \qquad \textit{(Ec. 1)}$$

$$a(L+1) = F(L) * (Z(L)) \qquad \textit{(Ec. 2)}$$

Donde L corresponde a la capa de la red en la que se está calculando su resultado, $W(L)$ son los pesos de la capa L, $B(L)$ es el bias y $F(L)$ es la función de activación de las neuronas de la capa L.

Una vez que esto ha sucedido en cada capa, el resultado de la capa final será considerado como el valor devuelto por la red neuronal. Este valor es el que se utiliza para comparar con el valor real y corregirlo si es necesario.

Cada dato de entrada es pasado por las capas intermedias de la red donde se realiza la transformación. El resultado obtenido es pasado como entrada de la siguiente capa. Cuando los datos cruzan toda la capa oculta, se pasan los valores de las transformaciones de la última capa oculta a la capa de salida para realizar la predicción. Este proceso de transformación y propagación se repite hasta que los datos de entrada hayan cruzado todas las capas de la red y se ha obtenido una predicción en la capa de salida.

Función de coste

En el proceso de aprendizaje supervisado de una red neuronal, se utiliza una función de coste o error para estimar el error y así poder comparar y medir si el resultado obtenido fue bueno o malo en comparación con el resultado esperado. Dado que los datos están etiquetados, se conoce el valor esperado. Idealmente, se espera que el error sea cero, es decir, que el valor predicho coincida con el esperado. Para lograr esto, a medida que se entrena el modelo, se ajustan automáticamente los pesos de las interconexiones de las neuronas hasta obtener buenas predicciones.

Entre las funciones de error más comunes para este tipo de arquitectura de redes neuronales se encuentran el error cuadrático medio (MSE), el error absoluto medio (MAE) y el error cuadrático logarítmico medio (*MSLE*).

El MSE es la más utilizada y representa la distancia promedio vertical u horizontal de los valores con respecto a la recta. Se representa con la siguiente ecuación:

$$MSE = 1/n \, \Sigma \, (Ri - Yi)^2 \qquad \textit{(Ec. 3)}$$

El MAE es un promedio de los errores absolutos y tiene la misma funcionalidad que el MSE, es decir, determinar el promedio de la distancia entre los valores y la recta. Su ecuación es la siguiente:

$$MAE = 1/n \, \Sigma \, |Ri - Yi| \qquad \textit{(Ec. 4)}$$

El *MSLE* es una variación del error cuadrático medio, donde sólo se preocupa de la diferencia porcentual entre los valores. Su ecuación es la siguiente:

$$MSLE = 1/n \, \Sigma \, (Log(Ri + 1) - Log(Yi + 1))^2 \qquad \textit{(Ec. 5)}$$

La función MSE es una de las más utilizadas en el campo de estudio debido a su facilidad de implementación en los lenguajes de programación y su bajo costo. Por lo tanto, en el presente trabajo la principal función de coste es MSE, aunque el sistema permite seleccionar las mencionadas previamente.

Back propagation

Tras calcular el error, la información se propaga hacia atrás a través de la red en un proceso conocido como backpropagation. Durante esta fase, la información se propaga desde la capa de salida hacia todas las neuronas de las capas ocultas que contribuyeron directamente a la obtención del valor de salida. Es importante tener en cuenta que cada neurona de la capa oculta solo recibe una fracción del valor total del error, basándose aproximadamente en la contribución relativa que cada neurona aportó al valor de salida. Este proceso se repite a lo largo de todas las capas.

Una vez finalizada la propagación hacia atrás, se ajustan los pesos de las conexiones entre las neuronas. El objetivo de este ajuste es minimizar el error en las futuras predicciones que realice el modelo.

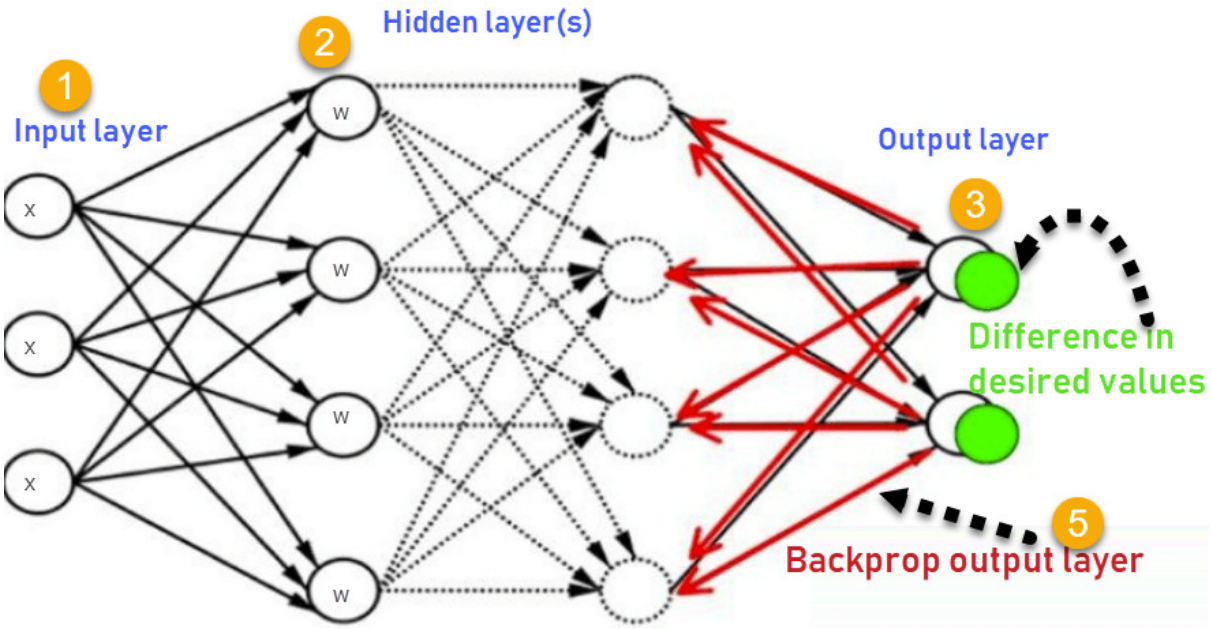


Figura 9. Ilustración back propagation

Una de las técnicas más utilizadas para lograr esto es el método del gradiente descendiente, también conocido como "gradient descent". Con esta técnica, los pesos se modifican en pequeños pasos basados en la tasa de aprendizaje (learning rate) y con la ayuda de la derivada parcial o gradiente de la función de pérdida. Esto permite determinar en qué dirección crece la función y, por lo tanto, utilizando el negativo del gradiente, se puede conocer la dirección en la que la función disminuye hacia un mínimo.

El objetivo es llegar al mínimo global de la función de error. Este trabajo se realiza generalmente en lotes de datos (batches) durante las sucesivas iteraciones (epochs) de entrenamiento con todos los datos que se pasan a la red como entradas. Estos conceptos se presentarán con un poco más de detalle en los siguientes apartados.

Optimización Gradient Descent

El optimizador "gradient descent" o "descenso del gradiente" es un algoritmo de optimización fundamental en el campo del aprendizaje automático y el aprendizaje profundo. Este algoritmo se utiliza para minimizar una función objetivo, como la función de pérdida o costo, ajustando iterativamente los parámetros del modelo. El gradiente es una generalización del concepto de derivada para funciones con múltiples variables. En el contexto del aprendizaje automático, nos interesa el gradiente de la función de pérdida con respecto a los parámetros del modelo. Cada componente del gradiente es la derivada parcial de la función de pérdida con respecto a un parámetro particular.

El proceso de descenso del gradiente implica calcular el gradiente de la función de pérdida y luego actualizar los parámetros del modelo en la dirección opuesta al gradiente. Esto se debe a que el gradiente apunta en la dirección de mayor incremento de la función, por lo que moverse en la dirección opuesta al gradiente reduce la pérdida. El mismo se realiza para cada capa de la red neuronal, comenzando desde la última capa y avanzando hacia la primera. Este proceso se conoce como backpropagation o retropropagación. Durante la retropropagación, se calculan las derivadas de la función de pérdida para cada capa oculta, teniendo en cuenta las derivadas de la función de la capa superior y la función de activación de la capa actual. Es importante destacar que las funciones de activación utilizadas en este proceso deben ser diferenciables.

Una vez que se han calculado los gradientes para todos los parámetros, se actualizan los valores de los parámetros. La magnitud del cambio en los parámetros está determinada por el gradiente y un hiperparámetro conocido como tasa de aprendizaje o "learning rate". La tasa de aprendizaje controla cuánto cambian los parámetros en cada paso del descenso del gradiente. Una tasa de aprendizaje más alta puede hacer que el aprendizaje sea más rápido, pero también puede hacer que el algoritmo sea inestable. Por otro lado, una tasa de aprendizaje más baja puede hacer que el aprendizaje sea más estable, pero también puede hacer que el aprendizaje sea muy lento.

En definitiva, el descenso del gradiente es un algoritmo de optimización que utiliza el gradiente de la función de pérdida para actualizar iterativamente los parámetros del modelo en la dirección que reduce la pérdida. Este proceso se realiza para cada capa de la red neuronal, lo que permite que el modelo aprenda de los errores y mejore su rendimiento en la tarea de aprendizaje.

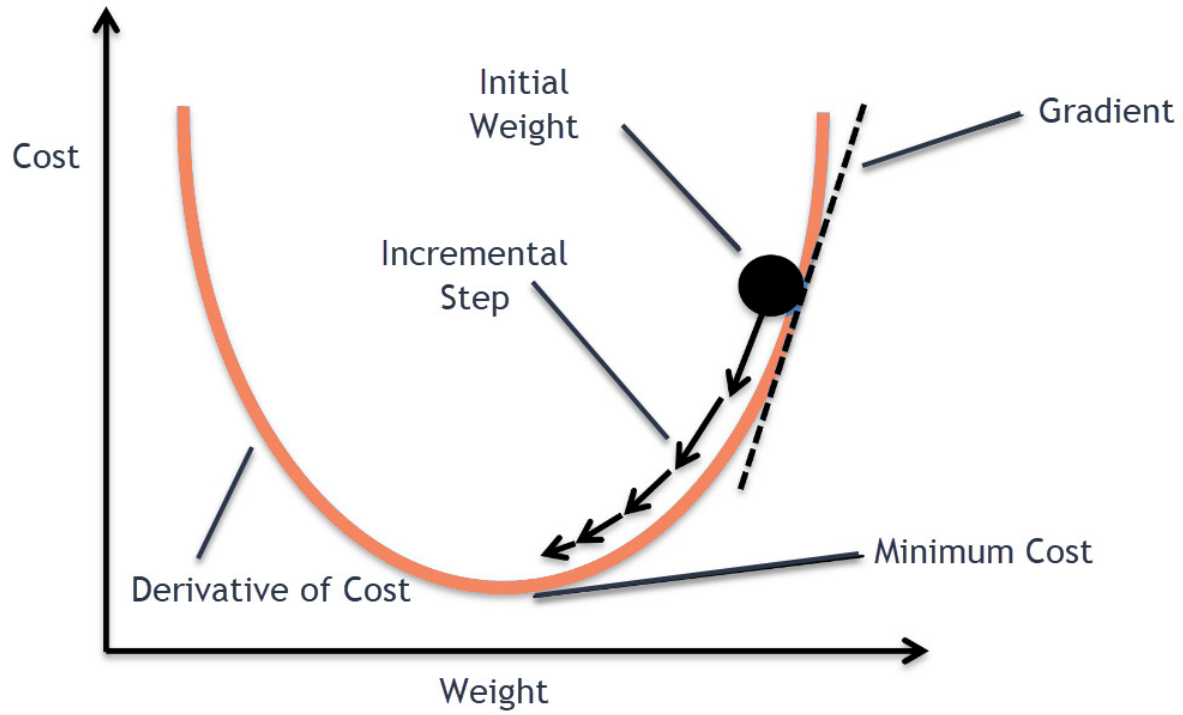


Figura 10. Ilustración del comportamiento del descenso por gradiente

3.7.4. Hiperparametros

Para comenzar, es importante distinguir entre los parámetros y los hiperparámetros de una red neuronal.

Los parámetros son los valores que se aprenden durante el entrenamiento del modelo a partir de los datos. No se definen manualmente, sino que se estiman a través del proceso de entrenamiento. Ejemplos de parámetros incluyen los pesos de las conexiones entre las neuronas en una red neuronal. La estimación de estos parámetros es crucial, ya que son los que permiten al modelo aprender de los datos y realizar predicciones precisas. Los parámetros se estiman comúnmente utilizando un optimizador como el "descenso del gradiente".

Por otro lado, los hiperparámetros son las configuraciones que se utilizan durante el proceso de entrenamiento. A diferencia de los parámetros, los hiperparámetros no se aprenden de los datos y deben ser establecidos manualmente por el desarrollador. El valor óptimo de un hiperparámetro no se conoce de antemano para un problema dado, por lo que se suelen utilizar valores basados en reglas generales, valores que han funcionado bien en problemas similares, o se realiza una búsqueda de la mejor opción mediante prueba y error. La validación cruzada es una técnica comúnmente utilizada para encontrar estos hiperparámetros.

Los hiperparámetros pueden estar relacionados con la estructura y la topología de la red (como el número de capas, el número de neuronas en cada capa, la función de activación

utilizada, etc.) o con los algoritmos de aprendizaje (como la tasa de aprendizaje, el número de épocas de entrenamiento, el tamaño del lote, etc.).

En resumen, al entrenar un modelo de aprendizaje automático, se establecen los valores de los hiperparámetros y luego se utilizan estos para obtener los parámetros.

- El tamaño del lote (batch size) se refiere a la cantidad de datos que se pasan a la red en cada iteración del entrenamiento. Cuando el conjunto de datos es muy grande, es conveniente dividirlo en lotes más pequeños. El tamaño óptimo del lote depende de varios factores, incluyendo la memoria del CPU/GPU que se utilice para realizar los cálculos.

- Las épocas (epochs) se refieren al número de veces que todo el conjunto de datos de entrenamiento pasa por la red neuronal durante el proceso de aprendizaje. Como el proceso de minimización de la función de error es iterativo, se necesitan varias épocas para entrenar la red. El número adecuado de épocas se suele determinar incrementando el número de épocas hasta que el rendimiento en los datos de validación comienza a disminuir.

- La tasa de aprendizaje (learning rate) es un hiperparámetro que controla la velocidad a la que se avanza en la optimización de la función de error durante el entrenamiento. El algoritmo de "descenso del gradiente" utiliza la tasa de aprendizaje para determinar el tamaño del paso en cada iteración del entrenamiento. Si la tasa de aprendizaje es demasiado alta, el algoritmo puede oscilar alrededor del mínimo de la función de error y tener dificultades para converger. Si la tasa de aprendizaje es demasiado baja, el aprendizaje puede ser muy lento o el algoritmo puede quedar atrapado en un mínimo local de la función y nunca converger al mínimo global. Existen técnicas que permiten reducir la tasa de aprendizaje a medida que el algoritmo se acerca al mínimo de la función.

3.7.5. Optimizadores

Aunque el método de "descenso del gradiente" es el optimizador más tradicional, existen otros optimizadores en el campo del aprendizaje profundo que pueden ofrecer ciertas ventajas, aunque a menudo a expensas de un mayor costo computacional. A continuación, se describen algunos de estos optimizadores.

- ➢ *Descenso Estocástico del Gradiente (SGD)*: Este es una variante del método de descenso del gradiente que aplica el algoritmo a un solo ejemplo en cada iteración, es decir, utiliza lotes de datos de tamaño 1. Al comenzar la siguiente época de entrenamiento, los datos de entrenamiento se mezclan de manera aleatoria.

- ➢ *Momentum*: Esta es una modificación del descenso del gradiente que introduce un factor de inercia o momentum. Recuerda el cambio aplicado a los pesos en cada iteración y determina la siguiente actualización basándose en una combinación lineal

del gradiente y estos cambios anteriores. De esta manera, suaviza las oscilaciones alrededor del mínimo durante la convergencia del algoritmo de descenso del gradiente.

- *RMSprop*: Este optimizador, al igual que Momentum, busca suavizar las oscilaciones durante la convergencia del método. Lo hace ajustando automáticamente la tasa de aprendizaje. En cada iteración, cuando se actualizan los pesos, se selecciona un valor diferente para la tasa de aprendizaje. Es un método adaptativo, ya que la tasa de aprendizaje se ajusta durante el entrenamiento.
- *Adam*: Este es un optimizador que combina los métodos de Momentum y RMSprop. Es el optimizador utilizado en los modelos propuestos en este proyecto.

3.8. Redes Neuronales Convolucionales

Las redes neuronales convencionales son redes totalmente interconectadas, lo que implica que cada neurona en una capa oculta está vinculada con todas las neuronas de la capa subsiguiente y precedente. En el ámbito del Deep Learning aplicado a la detección de objetos reciclables mediante el procesamiento de imágenes, surge un desafío. Para un ordenador, una imagen se interpreta como una matriz de píxeles, por lo que cada píxel de la imagen de entrada estaría vinculado a cada neurona de la primera capa oculta de la red. El inconveniente radica en que el número de conexiones requeridas es excesivamente alto, lo que hace que el uso de redes totalmente conectadas sea prácticamente inviable, incluso para imágenes de tamaño relativamente pequeño en redes muy profundas. Para abordar este problema, se introducen las redes neuronales convolucionales (CNN).

Las CNN son una variante de las redes neuronales artificiales que procesan capas emulando el córtex visual del cerebro humano para identificar diferentes características en las entradas, lo que en última instancia permite identificar objetos y "ver". Su principal beneficio es que cada segmento de la red se entrena para llevar a cabo una tarea específica, aprendiendo diferentes niveles de abstracción, lo que reduce significativamente el número de conexiones en las capas ocultas y acelera el entrenamiento. Las CNN son extremadamente eficaces para todo lo relacionado con el análisis de imágenes, ya que contienen varias capas ocultas especializadas y jerarquizadas. Esto significa que las primeras capas pueden detectar características básicas como líneas, curvas o bordes y se van especializando hasta llegar a capas más profundas capaces de reconocer formas complejas como un rostro, una silueta o un objeto.

3.8.1. Arquitectura

La arquitectura de una red neuronal convolucional (CNN) se puede separar en dos grandes etapas, por un lado, la etapa de extracción de características que a su vez está compuesta por

una o más capas convolucionales en donde se realizan pasos fundamentales: convolución, activación y pooling, y por otro lado, la etapa de clasificación, también llamada capa completamente conectada (fully connected layer), que comienza con un aplanamiento y continua con una red neuronal tradicional fully connected, en donde se realizará la clasificación final. En la siguiente Figura 11 se muestra un ejemplo completo de la arquitectura de una CNN.

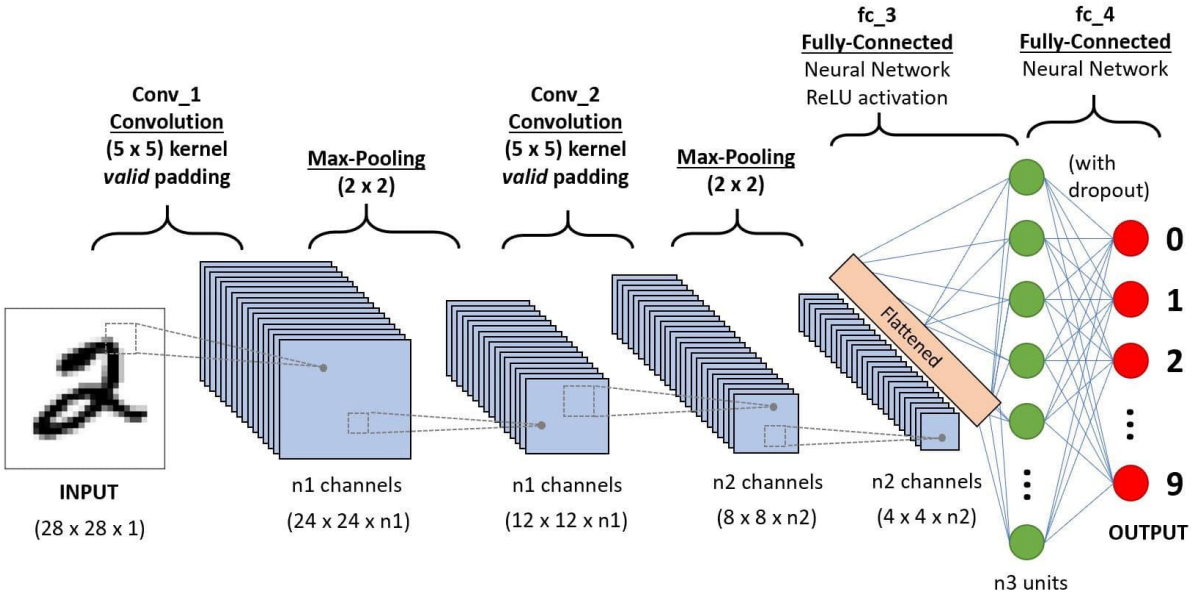


Figura 11. Arquitectura redes neuronales convolucionales

3.8.2. Convolución

La convolución es la base de la estructura de las redes neuronales convolucionales. La distinción clave entre una capa densamente conectada (ANN) y una capa especializada en la operación de convolución, denominada capa de convolución, radica en que la capa densa aprende patrones globales a partir de todo el conjunto de datos de entrada, mientras que las capas de convolución aprenden patrones locales en pequeñas ventanas bidimensionales.

La capa de entrada de la CNN tiene tantas neuronas como píxeles tiene la imagen de entrada (cada píxel se considera una neurona). Durante el proceso de convolución, los datos de entrada se transforman utilizando un producto escalar entre una región de las neuronas (submatriz de la imagen) de la capa de entrada y la matriz de pesos asignados (kernel). Generalmente, el resultado es una nueva imagen convolucionada con dimensiones espaciales menores o iguales a las de la imagen de entrada, y la profundidad de la capa convolucional está determinada por la cantidad de filtros que se apliquen a la imagen de entrada.

La convolución es una operación matemática que describe cómo combinar dos conjuntos de información. Esta operación, también conocida como "detector de características" de una CNN, su resultado es una capa convolucionada, denominada mapa de características.

La Figura 12 muestra cómo el filtro se desplaza a través de los datos de entrada para producir la capa convolucionada. Esta ventana se desliza (de izquierda a derecha y de arriba a abajo) a lo largo de toda la capa de neuronas, realizando el producto escalar entre las matrices. Cada valor obtenido como resultado corresponderá a un elemento de la nueva matriz generada por la convolución (o píxel de la imagen filtrada).

El tamaño de la matriz de salida ($Msal$) se determina por la fórmula:

$$Msal = Ment - K + 1 \qquad \textit{(Ec. 6)}$$

donde K es el tamaño del filtro y $Ment$ es el tamaño de la matriz de entrada.

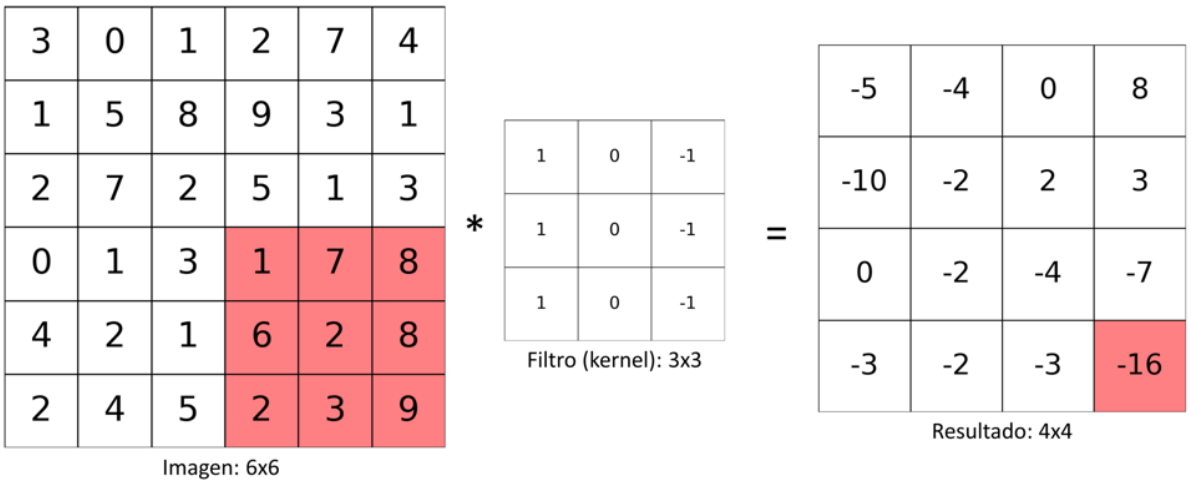


Figura 12. Proceso de convolución

Es importante resaltar que un filtro se define por una matriz K y un sesgo b, y son los pesos (valores) del filtro y el sesgo los que se ajustarán durante el entrenamiento de la red. Esto representa una reducción significativa en la cantidad de parámetros que la red necesita optimizar. Cada filtro solo permite detectar una característica específica en una imagen, por lo tanto, para el reconocimiento de imágenes se utilizan tantos filtros como características se quieran detectar. Por esta razón, cada capa convolucional en una CNN incluye varios filtros, esto se conoce como "apilamiento" o "stacking", como se puede ver en la siguiente Figura 13.

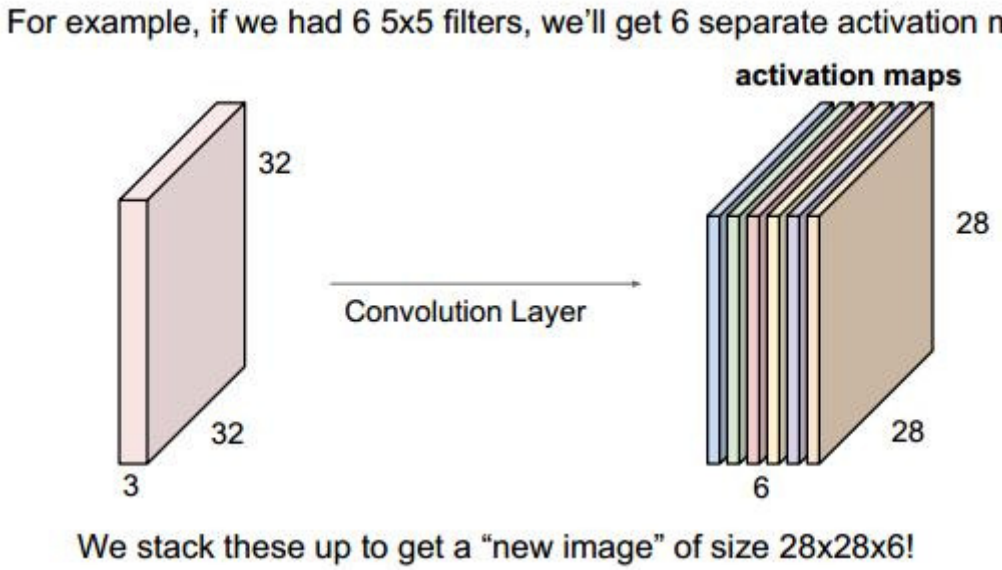


Figura 13. Stacking de filtros

Existen hiperparámetros específicos que deben determinarse en una CNN para establecer la disposición espacial y el tamaño del volumen de salida de una capa convolucional.

- Tamaño del kernel: se refiere a las dimensiones de la matriz de los filtros a utilizar, generalmente este tamaño es mucho menor que el tamaño de la imagen de entrada.
- Stride o paso: este hiperparámetro indica la distancia con la que se mueven los filtros a través de la imagen. Cuanto mayor sea el valor del stride, menor será la salida resultante.
- Zero-Padding: es la cantidad de "anillos" de ceros añadidos alrededor de la imagen de entrada para evitar reducir el tamaño de la salida, es decir, para que la matriz de salida tenga las mismas dimensiones que la matriz de entrada.

3.8.3. Activación

Cada una de las capas convolucionales puede utilizar cualquier función de activación que cumpla con la no linealidad. En particular, para los modelos desarrollados durante el proyecto, se utilizó la función Relu. Esta es una de las más empleadas en los modelos de aprendizaje profundo, ya que ayuda a evitar lo que se conoce como el desvanecimiento del gradiente. Este es un problema común al realizar retropropagación en redes profundas, ya que este desvanecimiento puede hacer que el entrenamiento de la red no produzca el efecto deseado, es decir, que el entrenamiento no mejore de iteración en iteración. La función devuelve 0 si recibe una entrada negativa, pero para cualquier valor positivo de x devuelve ese mismo valor, por lo que se representa de la siguiente manera:

$$f(x) = max(0, x) \qquad \textit{(Ec. 7)}$$

3.8.4. Pooling

Después de realizar la convolución y aplicar la transformación no lineal, el siguiente paso es realizar un agrupamiento o pooling, también conocido como submuestreo. Esta capa tiene como objetivo reducir las dimensiones de la salida de la capa convolucional sin modificar su profundidad, es decir, se reduce el tamaño de las imágenes filtradas, pero se conservan las características más importantes detectadas por cada filtro. De esta manera, se disminuye el número de parámetros y, por lo tanto, el tiempo y la capacidad computacional necesarios para entrenar el modelo; además, ayuda a controlar el sobreajuste. Esta capa también proporciona a la arquitectura una de las características clave que la representan, la capacidad de tener invariabilidad espacial, es decir, que el modelo será capaz de hacer buenas predicciones independientemente de que las imágenes de entrada tengan pequeñas transformaciones como traslaciones y/o rotaciones.

Existen diferentes formas de realizar el pooling, se puede realizar un "average pooling" o un "max pooling". En ambos casos, se aplica un filtro de tamaño m x m (generalmente de 2x2) con un stride generalmente igual a 2, lo que resulta en una matriz de salida de la mitad del tamaño de la matriz original, el recorrido del filtro por la matriz, al igual que en la convolución, se realiza de izquierda a derecha y de arriba a abajo.

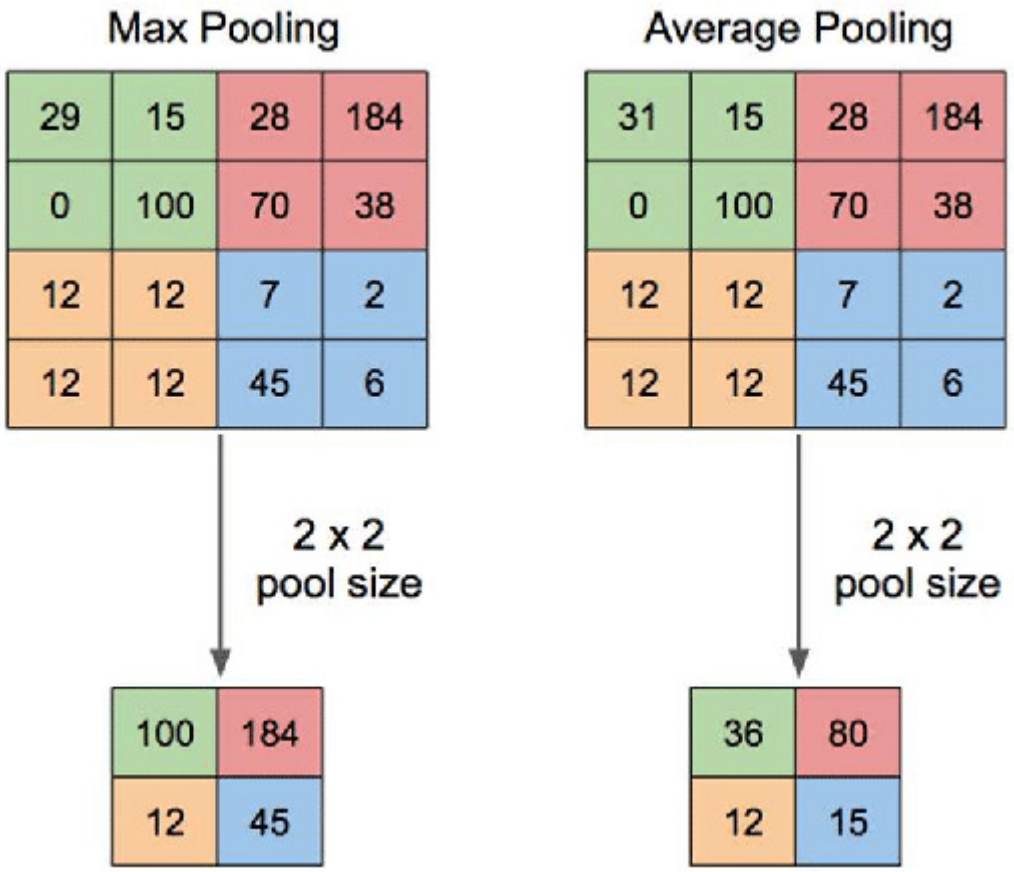


Figura 14. Max Pooling vs Average Pooling

- *Max pooling*: se aplica un filtro de tamaño m x m a través de cada mapa de características y se selecciona el valor más grande de esa porción de la matriz. Este valor será el que se colocará en un píxel de la matriz de salida.
- *Average pooling*: al igual que en el caso anterior, se aplica un filtro de tamaño m x m a través de cada mapa de características, pero esta vez el valor que se colocará en un píxel de la matriz de salida corresponderá al valor medio calculado entre todos los valores de la porción de la matriz donde se encuentra el filtro en ese momento.

3.8.5. Clasificación

La fase de clasificación es la última etapa de una CNN. Esta etapa implica, en primer lugar, realizar un aplanamiento o "flatten" de los mapas de características y luego pasarlos por una red completamente conectada, que se encargará de realizar la clasificación para determinar a qué clase pertenece la imagen de entrada.

Aplanamiento o Flatten

El aplanamiento implica tomar todos los mapas de características obtenidos en la última capa convolucional y pasarlos por una función que transformará estos datos tridimensionales en un vector unidimensional. Este vector se utilizará como entrada para una red completamente conectada.

Red neuronal densa

Después del aplanamiento, los datos se pasan a una red completamente conectada, también conocida como red densa.

La última capa de la red completamente conectada a menudo utiliza la función de activación softmax, que es especialmente útil para la clasificación multiclase. La función softmax toma un vector de números reales y los transforma en probabilidades, asegurando que la suma de todas las probabilidades sea 1. Cada número en el vector de salida de la función softmax representa la probabilidad de que la imagen de entrada pertenezca a una clase específica.

$$Softmax(xi) = \frac{e^{xi}}{\sum_{j=0}^{k} e^{xj}} \; j = 0 \; i = 0, 1, 2, \ldots, k \qquad \textit{(Ec. 8)}$$

Por ejemplo, si estamos clasificando imágenes de perros, gatos y pájaros, la función softmax podría dar como resultado un vector como [0.1, 0.7, 0.2], lo que indica que la red cree que la imagen tiene un 10% de probabilidad de ser un perro, un 70% de probabilidad de ser un gato, y un 20% de probabilidad de ser un pájaro. La clase con la mayor probabilidad se selecciona como la predicción de la red.

Función de pérdida

La función de pérdida utilizada durante el desarrollo de los modelos propuestos en este trabajo es la función de entropía cruzada (cross entropy loss). La entropía cruzada entre dos distribuciones de probabilidad p y q mide el número promedio de bits necesarios para identificar un evento de un conjunto de posibilidades. Si un esquema de codificación utilizado para el conjunto es optimizado para una distribución de probabilidad dada q, en lugar de la distribución auténtica p. la entropía cruzada para dos distribuciones p y q sobre el mismo espacio de probabilidad, siendo p y q variables discretas, se define como sigue:

$$H(p, q) = -\sum x\, p(x)\, log\, q(x) \qquad \textit{(Ec. 9)}$$

3.9. Tareas de la Visión Artificial

Dentro del campo de la visión artificial, es fundamental entender tres conceptos principales: la clasificación, la detección y la segmentación. Cada uno de estos conceptos representa distintas técnicas y enfoques para analizar y comprender imágenes y objetos en una imagen o video. A continuación, explicaremos en detalle cada uno de ellos para comprender su relevancia y aplicaciones en este campo.

3.9.1. Clasificación

La clasificación es una tarea esencial en el campo de la visión artificial, que consiste en asignar una etiqueta o categoría a una imagen u objeto en función de sus características distintivas. Tiene como propósito lograr identificar y categorizar con precisión los objetos presentes en una secuencia de imágenes o videos, lo cual tiene múltiples aplicaciones prácticas, como el reconocimiento de objetos, la recuperación de imágenes y la comprensión de escenas.

Figura 15. Ejemplo de clasificación de un animal

Para realizar la clasificación se pueden utilizar diversas técnicas, entre las que se encuentran el aprendizaje profundo, las máquinas de vectores de soporte y los árboles de decisión.

Particularmente en el aprendizaje profundo, las redes neuronales convolucionales (CNN) se usan comúnmente para realizar tareas de clasificación de imágenes. Estas CNN están entrenadas en grandes conjuntos de datos de imágenes previamente etiquetadas, lo que les permite aprender a reconocer ciertos patrones y características en imágenes asociadas con ciertas categorías o etiquetas.

3.9.2. Detección

La detección de visión artificial se refiere a la capacidad de identificar y ubicar objetos específicos en una imagen o video. En este proceso, se utilizan algoritmos o modelos de aprendizaje automático para analizar y reconocer patrones visuales en los datos de entrada. El descubrimiento tiene dos aspectos principales:

- *Localización*: en este paso, el objetivo es determinar la ubicación exacta de los objetos en la imagen o el video. Esto se logra delimitando o delimitando las áreas en las que se encuentran los objetos de interés detectando sus contornos o creando cuadros delimitadores a su alrededor.
- *Clasificación*: una vez que se encuentra un objeto, se le asigna una etiqueta o categoría específica. Es decir, determina a qué clase o categoría pertenece el objeto detectado, por ejemplo, un gato, un coche, una persona, etc.

Figura 16. Detección de un animal

La detección por visión artificial encuentra aplicación en diversos campos como sistemas de vigilancia, reconocimiento facial, conducción autónoma, detección de objetos en imágenes médicas, entre otros. Al combinar la localización y la clasificación, la detección permite identificar y comprender el contenido visual de una imagen o video, lo cual es importante para muchas aplicaciones prácticas de la visión artificial.

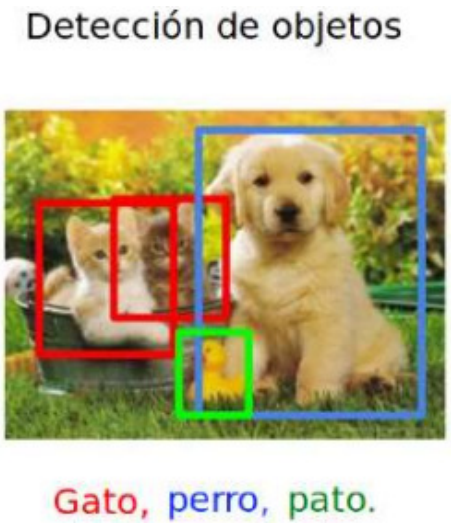


Figura 17. Detección de múltiples objetos

3.9.3. Segmentación

En la visión por computadora, la segmentación es el proceso de dividir una imagen digital en múltiples regiones, o segmentos, con cada segmento agrupando píxeles con características similares. El objetivo principal de la segmentación es localizar áreas significativas dentro de

una imagen. Se utiliza tanto para localizar objetos específicos como para encontrar sus bordes.

Este procedimiento se realiza utilizando técnicas de procesamiento de imágenes y algoritmos de aprendizaje automático. Mediante la aplicación de la segmentación, a cada píxel de la imagen analizada se le asigna una categoría o etiqueta, lo que permite identificar y distinguir diferentes partes o zonas de la imagen.

Figura 18. Segmentación de objetos

Existen diferentes enfoques y métodos para realizar la segmentación en visión artificial, entre los que destacan la segmentación por color, la segmentación por textura y la segmentación basada en superpíxeles. Estos métodos permiten identificar y separar objetos o áreas de interés en una imagen, facilitando su análisis y posterior procesamiento.

3.10. Entrenamiento y Ajuste de Modelos Complejos

3.10.1. Transfer Learning

El aprendizaje de transferencia, o "Transfer Learning" en inglés, es una técnica de aprendizaje automático en la que un modelo desarrollado para una tarea se reutiliza como punto de partida para un modelo en una segunda tarea relacionada. Es una técnica popular en el aprendizaje profundo porque puede entrenar modelos de aprendizaje profundo con un conjunto de datos comparativamente pequeño y lograr un rendimiento significativo.

En un escenario típico de aprendizaje automático, se entrena un algoritmo para una tarea específica en un conjunto de datos específico. Sin embargo, con el aprendizaje de transferencia, se aprovecha el modelo que ya ha sido preentrenado en un conjunto de datos grande y generalmente se ajusta para realizar tareas relacionadas.

Por ejemplo, un modelo que ha sido entrenado para reconocer muchos tipos diferentes de objetos (como la red neuronal convolucional Inception o ResNet entrenada en el conjunto de datos ImageNet) podría ser utilizado como punto de partida para un modelo diseñado para reconocer tipos específicos de aves o para diagnosticar enfermedades a partir de imágenes de resonancia magnética.

El aprendizaje de transferencia se utiliza comúnmente en el aprendizaje profundo por varias razones:

1. Ahorro de tiempo y recursos computacionales: Entrenar una red neuronal profunda desde cero requiere una gran cantidad de datos y una gran cantidad de tiempo de computación. Al utilizar un modelo preentrenado, puedes aprovechar lo que ya ha aprendido y adaptarlo a tu problema específico, lo que puede ahorrar mucho tiempo y recursos.

2. Mejor rendimiento con menos datos: Si tienes un conjunto de datos pequeño, el aprendizaje de transferencia puede ser una forma efectiva de obtener un buen rendimiento. Esto se debe a que el modelo preentrenado ya ha aprendido características útiles de un conjunto de datos más grande, y estas características pueden ser útiles para tu problema específico.

3. Adaptabilidad: El aprendizaje de transferencia es flexible y puede ser utilizado en una amplia gama de tareas. Puedes utilizar un modelo preentrenado y ajustarlo para tu tarea específica, ya sea clasificación de imágenes, análisis de sentimientos, reconocimiento de voz, y más.

En resumen, el aprendizaje de transferencia es una técnica poderosa en el aprendizaje profundo que permite a los modelos aprender de tareas relacionadas y mejorar el rendimiento, especialmente cuando se tiene un conjunto de datos pequeño.

3.10.2. Fine Tuning

El "Fine-Tuning" o ajuste fino es una técnica utilizada en aprendizaje profundo que implica ajustar un modelo preentrenado para adaptarlo a una nueva tarea similar. Es una forma de implementar el aprendizaje de transferencia, donde se toma un modelo que ya ha sido entrenado en un conjunto de datos y se le "afina" para que funcione bien en una tarea relacionada.

El ajuste fino se realiza generalmente de dos maneras:

- Entrenamiento de algunas capas mientras se congelan otras: En este enfoque, las capas iniciales de la red se "congelan", o se mantienen constantes durante el entrenamiento, y sólo las capas más profundas de la red se entrenan. La idea detrás de esto es que las capas iniciales de una red neuronal convolucional, por ejemplo, tienden a aprender características de bajo nivel que son aplicables a muchas tareas

(como bordes, texturas, colores), mientras que las capas más profundas aprenden características de alto nivel más específicas de la tarea original (como la forma de un objeto en una tarea de clasificación de imágenes). Al congelar las capas iniciales, se pueden conservar estas características de bajo nivel y sólo ajustar las características de alto nivel para la nueva tarea.

- Entrenamiento de todas las capas: En este enfoque, todas las capas de la red se ajustan durante el entrenamiento. Esto puede ser útil si la nueva tarea es muy diferente de la tarea original y requiere aprender nuevas características desde cero. Sin embargo, este enfoque puede requerir más datos y tiempo de entrenamiento que el primero.

El ajuste fino es una técnica poderosa que puede permitir a los modelos de aprendizaje profundo adaptarse a nuevas tareas con un rendimiento significativamente mejorado, especialmente cuando se dispone de pocos datos para la nueva tarea. Sin embargo, también requiere un cuidadoso equilibrio para evitar el sobreajuste, que puede ocurrir si el modelo se ajusta demasiado a la nueva tarea y pierde su capacidad para generalizar a nuevos datos.

3.11. Entrenamiento Eficiente

El entrenamiento de redes neuronales implica el ajuste de un gran número de parámetros para que el modelo pueda aprender patrones y características en los datos de entrenamiento. A medida que el tamaño del conjunto de datos y la complejidad del modelo aumentan, el tiempo y los recursos computacionales necesarios para entrenar la red también se incrementan significativamente.

En este contexto, las GPU (Unidades de Procesamiento de Gráficos) se presentan como una solución efectiva para acelerar el proceso de entrenamiento. A diferencia de las CPU (Unidades de Procesamiento Central), que están diseñadas para realizar tareas generales, las GPU están optimizadas para realizar cálculos altamente paralelos en grandes conjuntos de datos. Esto significa que una GPU puede realizar múltiples operaciones matemáticas en paralelo y en menor tiempo que una CPU. Como resultado, el entrenamiento de redes neuronales con GPU permite una mayor velocidad de procesamiento y un menor tiempo de entrenamiento en comparación con utilizar solo CPU.

Por lo tanto, entrenar con GPU puede proporcionar varias ventajas para las redes neuronales:

- *Velocidad*: La utilización de GPU permite llevar a cabo cálculos paralelos de manera considerablemente más rápida que con las CPU, lo que se traduce en una notable reducción del tiempo necesario para el entrenamiento de redes neuronales, especialmente en el caso de modelos de aprendizaje profundo que requieren múltiples iteraciones de entrenamiento para alcanzar alta precisión.

- *Eficiencia*: Las GPU están específicamente diseñadas para manejar grandes volúmenes de datos y llevar a cabo numerosos cálculos de manera simultánea, lo que las convierte en una opción más eficiente que las CPU para entrenar redes neuronales. Gracias a su capacidad para procesar tareas masivamente paralelas, las GPU se destacan en la tarea de entrenamiento, optimizando el uso de recursos computacionales.

- *Escalabilidad*: La escalabilidad de las GPU es otra ventaja significativa, ya que es posible agregar más unidades de procesamiento gráfico a un sistema, lo que incrementa aún más la velocidad y eficiencia del entrenamiento de redes neuronales. Esta capacidad de escalamiento facilita la gestión de conjuntos de datos más grandes y complejos, adaptándose a las necesidades cambiantes y crecientes de los proyectos de inteligencia artificial.

- *Rentabilidad*: Desde una perspectiva económica, las GPU suelen resultar más rentables que las CPU para el entrenamiento de redes neuronales. La capacidad de acelerar los tiempos de entrenamiento y reducir el consumo energético se traduce en una mayor eficiencia y productividad, lo que conlleva un ahorro de recursos a largo plazo.

- *Precisión mejorada*: La utilización de GPU para el entrenamiento de redes neuronales no solo acelera el proceso, sino que también contribuye a mejorar la precisión de los modelos. Al permitir más iteraciones de entrenamiento y trabajar con conjuntos de datos más amplios, las GPU ayudan a obtener modelos más precisos y confiables, lo que es esencial para diversas aplicaciones en visión por computadora, procesamiento de imágenes y otras tareas de inteligencia artificial.

3.11.1. Múltiples GPUs

Cuando el entrenamiento en una sola GPU resulta demasiado lento o los pesos del modelo exceden la capacidad de memoria de una sola GPU, se recurre a una configuración de múltiples GPU para acelerar el proceso. En este escenario, se requiere implementar algún tipo de paralelismo para distribuir el trabajo entre las diferentes GPUs. Existen varias técnicas para lograr esta distribución paralela, entre ellas el paralelismo de datos, el tensor o el de canalización. Sin embargo, no existe una solución única que funcione para todos los casos, ya que la configuración óptima dependerá del hardware específico en el que se esté ejecutando el entrenamiento.

El paralelismo de datos implica dividir el conjunto de datos en partes más pequeñas y distribuirlo entre las GPUs, de manera que cada GPU pueda procesar una porción de datos independientemente. Por otro lado, el paralelismo de tensor se enfoca en dividir el modelo en diferentes partes y asignar cada parte a una GPU para que trabajen de forma simultánea. En cuanto al paralelismo de canalización, se trata de dividir el flujo de datos y operaciones de la red neuronal entre las GPUs para acelerar el entrenamiento.

3.11.2. Paralelismo de datos

El paralelismo de datos o Data Parallel (DP) permite distribuir automáticamente los datos y el modelo a lo largo de las GPUs disponibles, dividiendo el lote de entrenamiento entre ellas y realizando el cálculo de manera paralela. Esto facilita la aceleración del proceso de entrenamiento y, en muchos casos, proporciona una mejora significativa en la velocidad.

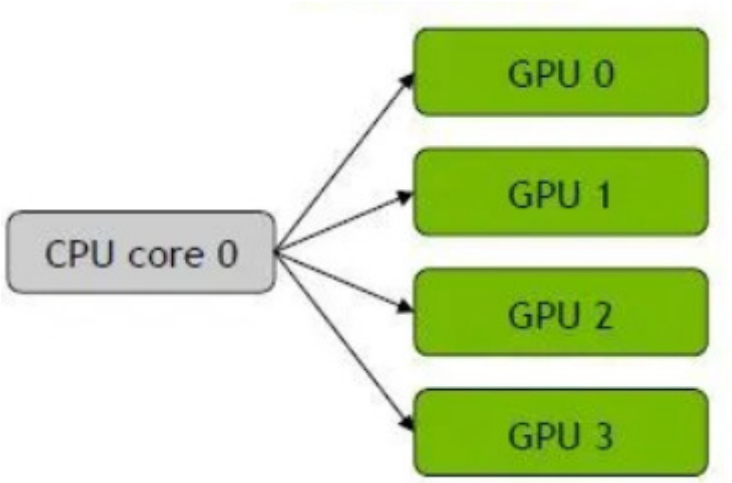


Figura 19. Paralelismo de datos en GPUs

La red se replica en cada GPU, es decir, se crea una copia idéntica de la red en cada uno de los dispositivos. Luego, cada GPU procesa diferentes lotes de datos simultáneamente. Durante el entrenamiento, los datos se dividen en lotes y cada lote se distribuye a una GPU diferente. Cada GPU calcula las predicciones y las pérdidas para su lote específico. Una vez que todas las GPUs han terminado su trabajo, los gradientes se combinan y promedian a través de las GPUs antes de realizar un paso de optimización.

3.12. YOLO (You Only Look Once)

3.12.1. Breve Historia

YOLO (You Only Look Once), es un popular modelo de detección de objetos y segmentación de imágenes, fue desarrollado por Joseph Redmon y Ali Farhadi en la Universidad de Washington. Lanzado en 2015, YOLO rápidamente ganó popularidad por su alta velocidad y precisión.

Figura 20. Tareas de visión artificial que puede hacer YOLO. Referido de https://docs.ultralytics.com/tasks/

En la evolución de la arquitectura YOLO, se han presentado diversas mejoras y versiones a lo largo de los años. YOLOv2, lanzado en 2016, incorporó técnicas como la normalización por lotes, cuadros de anclaje y grupos de dimensiones, lo que contribuyó a mejorar el rendimiento del modelo original.

Posteriormente, YOLOv3, lanzado en 2018, llevó las mejoras aún más lejos mediante la utilización de una red troncal más eficiente, anclajes múltiples y agrupación de pirámide espacial, logrando un mayor avance en el desempeño del modelo.

En 2020, se presentó YOLOv4, que introdujo innovaciones como el aumento de datos Mosaic, un nuevo cabezal de detección sin anclaje y una función de pérdida mejorada, consolidando aún más el rendimiento y la precisión de las detecciones.

Luego, YOLOv5, una versión posterior desarrollada por Ultralytics, continuó elevando el rendimiento del modelo al agregar características como la optimización de hiperparámetros, el seguimiento integrado de experimentos y la exportación automática a formatos populares de exportación.

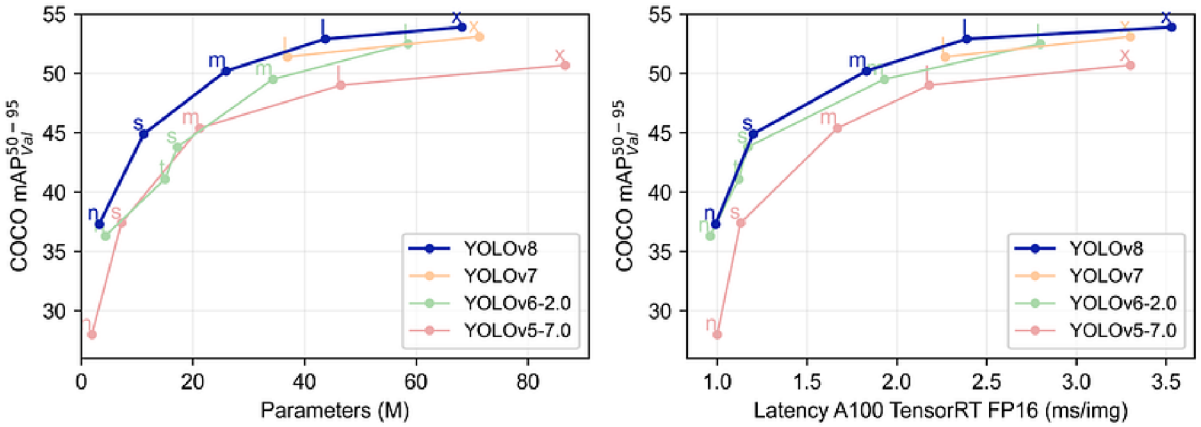


Figura 21. Comparación de las versiones de YOLO hasta la fecha

En 2022, Meituan presentó YOLOv6, utilizado en muchos de los robots de entrega autónomos de la empresa, mientras que YOLOv7 amplió su capacidad al abordar tareas adicionales, como la estimación de poses en el conjunto de datos de puntos clave de COCO.

La última iteración de la arquitectura es YOLOv8, desarrollado por Ultralytics, posicionado como el modelo de última generación (SOTA). Esta versión se basa en el éxito de las iteraciones anteriores y presenta nuevas características y mejoras para mejorar el rendimiento, la flexibilidad y la eficiencia. YOLOv8 ofrece soporte para una amplia variedad de tareas de visión por computadora, que incluyen detección, segmentación, estimación de pose, seguimiento y clasificación. Gracias a esta versatilidad, YOLOv8 se ha convertido en una herramienta poderosa y altamente adaptable para diversas aplicaciones y dominios en el campo de la inteligencia artificial.

3.12.2. Arquitectura

El sistema YOLO divide la imagen de entrada en una cuadrícula y cada celda de la cuadrícula predice un número fijo de bounding boxes junto con las probabilidades de clase para cada caja. Cada bounding box contiene cinco predicciones: x, y, w, h y la confianza. Las coordenadas (x, y) representan el centro de la caja en relación con los límites de la celda de la cuadrícula. La anchura y la altura se predicen en relación con toda la imagen. La predicción de confianza representa la intersección sobre la unión (IOU) entre la caja predicha y cualquier caja de verdad fundamental.

La arquitectura de YOLO está inspirada en el modelo GoogLeNet para la clasificación de imágenes. La red tiene 24 capas convolucionales seguidas de 2 capas completamente conectadas. En lugar de los módulos de inception utilizados por GoogLeNet, YOLO utiliza capas de reducción de 1x1 seguidas de capas convolucionales de 3x3.

YOLO se entrena en imágenes completas y optimiza directamente el rendimiento de la detección. Esto significa que YOLO ve toda la imagen durante el entrenamiento y el tiempo de prueba, por lo que codifica implícitamente información contextual sobre las clases y su apariencia.

En términos de rendimiento, el modelo base de YOLO puede procesar imágenes en tiempo real a 45 fotogramas por segundo. Una versión más pequeña de la red, Fast YOLO, puede procesar hasta 155 fotogramas por segundo, logrando el doble de la precisión media (mAP) de otros detectores en tiempo real.

Aunque YOLO es extremadamente rápido y eficaz para la detección de objetos en tiempo real, tiene algunas limitaciones. Por ejemplo, tiende a tener más errores de localización en comparación con otros sistemas de detección y puede tener dificultades para localizar con precisión algunos objetos, especialmente los más pequeños. Además, YOLO puede tener dificultades para generalizar a objetos en nuevas o inusuales proporciones o configuraciones.

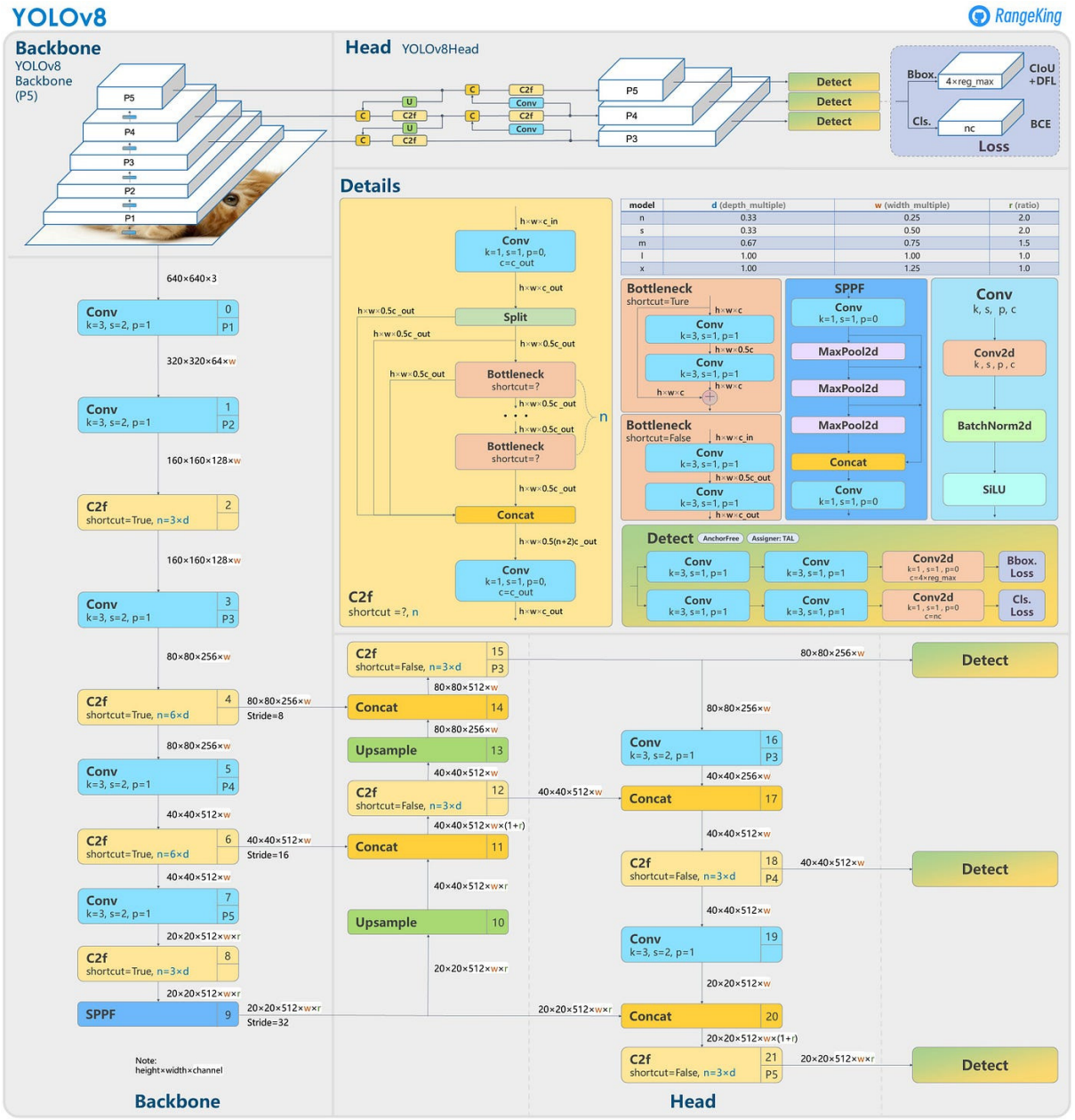


Figura 22. Diagrama de estructura del modelo basado en el código oficial de YOLOv8. Referido de https://github.com/open-mmlab/mmyolo/blob/dev/configs/yolov8/README.md/

3.12.3. Ultralitycs y Roboflow

Ultralytics es una reconocida empresa especializada en inteligencia artificial de código abierto con un enfoque particular en visión artificial. Como creadores de YOLOv5, un modelo de detección de objetos altamente popular y ampliamente utilizado, Ultralytics se esfuerza por hacer que la inteligencia artificial sea accesible y efectiva para aplicaciones del mundo real. La simplicidad y precisión que ofrece YOLOv5 lo convierten en una opción atractiva para diversos escenarios de implementación.

En busca de una colaboración estratégica para optimizar el proceso de preparación, etiquetado y entrenamiento de modelos de visión por computadora en conjuntos de datos personalizados, Ultralytics se ha asociado con Roboflow, una plataforma de desarrollo de visión artificial de extremo a extremo. La integración entre Roboflow y YOLOv5 busca brindar una experiencia de implementación y capacitación sin inconvenientes para los usuarios, simplificando aún más el desarrollo y despliegue de soluciones basadas en YOLOv5.

Esta colaboración entre Ultralytics y Roboflow se estableció en 2021, y desde el lanzamiento de YOLOv5 en junio de 2020, Roboflow ha estado comprometido en proporcionar contenido educativo relevante sobre modelos de visión por computadora, incluidos recursos para entrenar detectores YOLOv5 personalizados. La alianza entre estas dos entidades líderes en el campo de la visión artificial promete impulsar el desarrollo y la adopción de soluciones innovadoras y eficientes en diversos ámbitos de aplicación.

3.13. U-Net

3.13.1. Breve Historia

U-NET es un modelo de red neuronal dedicado a tareas de visión artificial, más específicamente a problemas de segmentación semántica. Fue desarrollado originalmente por Olaf Ronneberger, Phillip Fischer y Thomas Brox en 2015 para la segmentación de imágenes médicas. Su desarrollo se llevó a cabo con el objetivo de lograr segmentaciones más precisas a partir de un menor número de imágenes de entrenamiento.

Esta red neuronal es de tipo totalmente convolucional, y su arquitectura ha sido modificada y ampliada para abordar problemas de segmentación con gran eficacia. U-Net realiza una clasificación en cada píxel, lo que implica que tanto la entrada como la salida de la red comparten el mismo tamaño. Gracias a esta característica, la segmentación de imágenes de tamaño 512 × 512 se puede realizar en menos de un segundo cuando se utiliza una GPU moderna.

Debido a su eficacia y capacidad para generar segmentaciones precisas en imágenes médicas, la arquitectura U-Net se ha convertido en una elección popular y se ha implementado en la biblioteca Keras, facilitando su uso y adaptación en diversos proyectos de segmentación biomédica y otras aplicaciones relacionadas con la visión por computadora.

3.13.2. Arquitectura

La arquitectura de U-NET consta de dos partes: la de contracción (también llamada codificador) y la de expansión (también llamada decodificador). La parte de contracción se utiliza para captar el contexto de una imagen. Consiste en un conjunto de capas de convolución y de capas de "max pooling" que permiten crear un mapa de características de

una imagen y reducir su tamaño para disminuir el número de parámetros de la red. La parte de expansión permite una localización precisa mediante la convolución transpuesta.

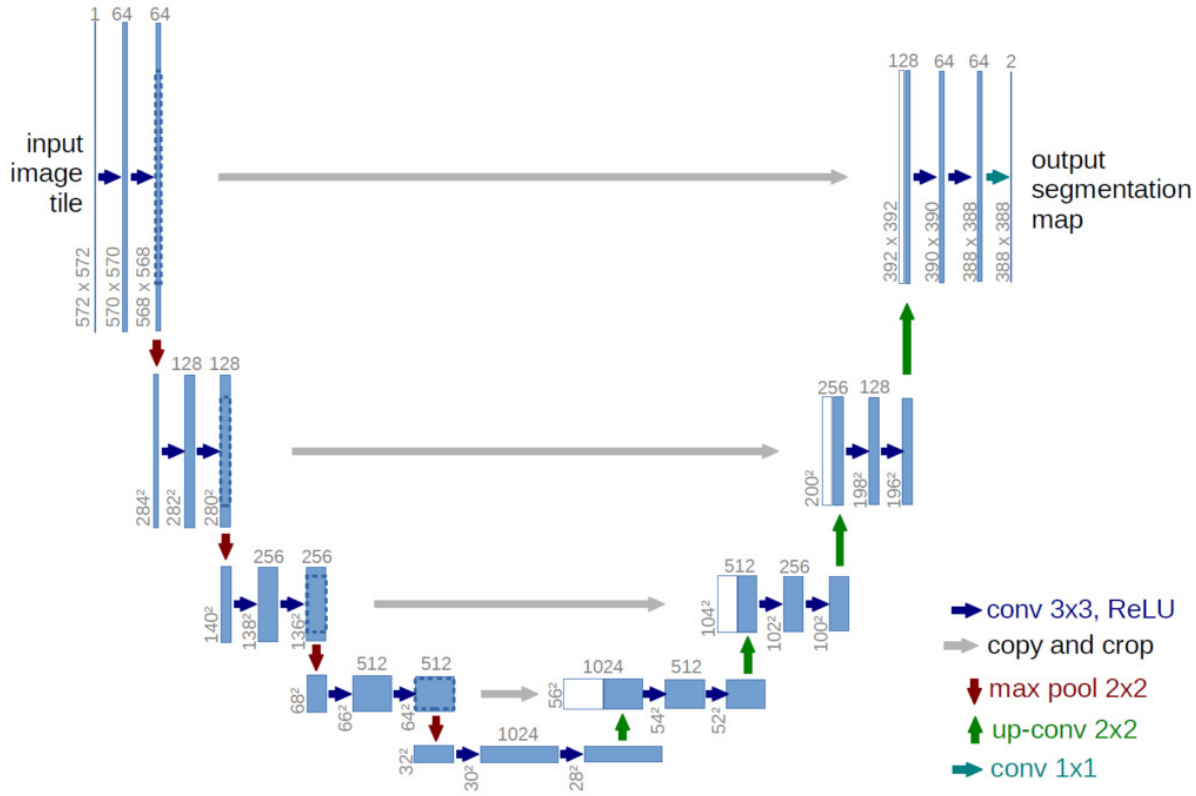


Figura 23. Arquitectura U-NET

U-NET es particularmente útil en aplicaciones donde la precisión de la segmentación es clave. Algunos ejemplos de estas aplicaciones incluyen:

1. *Vehículos autónomos*: U-NET puede ayudar a detectar los espacios libres en los carriles, la señalización de la carretera y las señales de tráfico.

2. *Diagnóstico médico*: U-NET puede apoyar los análisis realizados por los radiólogos, reduciendo el tiempo necesario para realizar los diagnósticos.

3. *Cartografía por satélite*: U-NET puede distinguir los diferentes tipos de terreno de forma automatizada, lo cual es útil para controlar las zonas de deforestación o para la urbanización.

4. *Agricultura de precisión*: U-NET puede distinguir las plantaciones de las malas hierbas, permitiendo la eliminación automatizada de la maleza con menos herbicida.

Una de las principales ventajas de U-NET es su eficacia incluso con un conjunto de datos limitado. También ofrece una mayor precisión que los modelos convencionales. Su arquitectura en forma de U permite evitar el problema del cuello de botella que se produce con una arquitectura de autocodificador y, por tanto, evita la pérdida de características.

CAPÍTULO IV

Implementación

4.1. Herramientas

En esta sección se encuentran todas las herramientas utilizadas para el proyecto y una breve descripción de cada una.

4.1.1. Entornos

Google Colab (Entorno cloud)

Es un entorno de desarrollo basado en la nube que permite a los usuarios ejecutar y colaborar en notebooks de Jupyter. Proporciona acceso gratuito a GPU y TPU (Unidad de Procesamiento Tensorial) para acelerar el entrenamiento y la ejecución de modelos de aprendizaje profundo.

Ubuntu

Es un sistema operativo de código abierto basado en Linux que ofrece una plataforma estable y versátil para el desarrollo de software y análisis de datos. Es ampliamente utilizado en tareas de aprendizaje automático debido a su flexibilidad y soporte para diversas bibliotecas. (*Versión utilizada: Ubuntu 22.04.2 LTS).*

4.1.2. Software

Python

Es un lenguaje de programación ampliamente utilizado en la comunidad de ciencia de datos y aprendizaje automático debido a su sintaxis sencilla y amplia variedad de bibliotecas y frameworks disponibles.

Conda

Conda es un sistema de gestión de paquetes y entornos de código abierto que facilita la instalación y administración de bibliotecas y dependencias.

CUDA

CUDA es una plataforma de computación paralela desarrollada por Nvidia que permite aprovechar la potencia de las GPU para acelerar cálculos intensivos en aplicaciones de aprendizaje automático.

CuDNN

CuDNN (Nvidia CUDA Deep Neural Network Library) es una biblioteca de aceleración para redes neuronales profundas desarrollada por Nvidia, que optimiza el rendimiento de las operaciones de aprendizaje profundo en las GPU Nvidia.

Librerías

Tabla 2. Librerías utilizadas en el proyecto

Paquete	Versión
albumentations	1.3.1
matplotlib	3.7.1
numpy	1.23.5
opencv-python	4.7.0.72
pandas	2.0.2
Pillow	9.0.1
pip	22.0.2
pydicom	2.4.1
scikit-learn	1.3.0
seaborn	0.12.2
tensorboard	2.13.0
torch	2.0.1
torchvision	0.15.2
torchaudio	2.0.0
tqdm	4.65.0
ultralytics	8.0.137
zipp	1.0.0

4.1.3. Hardware

Se utilizaron las siguientes unidades de procesamiento gráfico (GPU) de la marca Nvidia:

Tabla 3. Unidades de procesamiento gráfico utilizado

Cantidad	Marca	Serie	Memoria
1	NVIDIA	GeForce RTX 2060 SUPER	7982 MiB
2	NVIDIA	GeForce RTX 3060 Ti	7982 MiB

Información detallada sobre las tarjetas gráficas (GPUs) de Nvidia instaladas en el sistema:

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 520.61.05    Driver Version: 520.61.05    CUDA Version: 11.8     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  NVIDIA GeForce ...  On   | 00000000:01:00.0  On |                  N/A |
|  0%   41C    P5    18W / 200W |    135MiB /  8192MiB |     28%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   1  NVIDIA GeForce ...  On   | 00000000:07:00.0 Off |                  N/A |
| 29%   27C    P8     9W / 184W |      5MiB /  8192MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   2  NVIDIA GeForce ...  On   | 00000000:08:00.0 Off |                  N/A |
|  0%   38C    P8    18W / 200W |      5MiB /  8192MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A      2180      G   /usr/lib/xorg/Xorg                103MiB |
|    0   N/A  N/A     50100      G   /usr/bin/gnome-shell               30MiB |
|    1   N/A  N/A      2180      G   /usr/lib/xorg/Xorg                 4MiB |
|    2   N/A  N/A      2180      G   /usr/lib/xorg/Xorg                 4MiB |
+-----------------------------------------------------------------------------+
```

Figura 24. Ejecución del comando 'nvidia-smi' en el sistema utilizado

4.2. Datasets

A continuación, se describen los dos datasets empleados para el análisis, extraídos de la plataforma web pública *Kaggle*. Como se mencionó anteriormente, ambos se componen de imágenes radiográficas de la zona torácica, haciendo foco en el estudio de los pulmones.

4.2.1. Pulmonary Chest X-Ray Abnormalities

El dataset "Pulmonary Chest X-Ray Abnormalities" es una colección exhaustiva de imágenes de radiografías de tórax destinada a la segmentación de pulmones. Este conjunto de datos fue creado específicamente para la tarea de identificar y segmentar la región pulmonar en imágenes médicas de rayos X.

Figura 25. Lista de imágenes de rayos X

Este conjunto de datos contiene más de 500 exploraciones de rayos X con etiquetas clínicas recopiladas por radiólogos. Estas imágenes son proporcionadas con sus correspondientes máscaras de segmentación, que indican la ubicación precisa de los pulmones en cada radiografía.

Figura 26. Lista de máscaras binarias correspondientes a las imágenes de rayos X

4.2.2. RSNA Pneumonia Detection Challenge

El dataset "RSNA Pneumonia Detection Challenge" es una colección de imágenes de radiografías de tórax enfocada en la detección y clasificación de neumonía en pulmones. Este conjunto de datos fue creado con el propósito de abordar el desafío de detectar y diagnosticar de manera precisa la neumonía a partir de imágenes médicas de rayos X.

En esta competencia, que fue hecha en el año 2018, el desafío consiste en realizar un algoritmo para detectar una señal visual de neumonía en imágenes médicas. Específicamente, su algoritmo necesita ubicar automáticamente las opacidades pulmonares en las radiografías de tórax.

Figura 27. Desafío de detección de neumonía RSNA. Fuente: Recuperado de RSNA Pneumonia Detection Challenge.

En este desafío, los competidores predicen si existe neumonía en una imagen determinada. Lo hacen mediante la predicción de cuadros delimitadores alrededor de las áreas del pulmón. Las muestras sin cuadros delimitadores son negativas y no contienen evidencia definitiva de neumonía. Las muestras con cuadros delimitadores indican evidencia de neumonía.

4.3. Segmentación de Pulmones

Para llevar a cabo la tarea de segmentación de imágenes, comenzamos con el primer conjunto de datos “Pulmonary Chest X-Ray Abnormalities” (descrito en la sección 4.2.1) Este conjunto tiene la particularidad que las imágenes se encuentran en un formato legible (.png) y no es necesario sustraerse como en el formato DICOM. La segmentación de pulmones es una etapa esencial en la detección y clasificación de enfermedades pulmonares, ya que ayuda a aislar la región de interés y a reducir la cantidad de información innecesaria en el análisis posterior. Para esta primera tarea, se escogió el entorno de Google Colab ya que el dataset no cuenta con grandes cantidades de muestras.

4.3.1. Exploración y Visualización de Datos

El dataset se compone de un total de 1600 imágenes en formato PNG distribuido de la siguiente manera:

- 800 imágenes originales.
- 704 imágenes de máscaras binarias (correspondientes a las muestras originales).
- 96 imágenes para test.
- Las dimensiones de todas las imágenes son de 3000x2919 pixeles.

La Figura 28 muestra un ejemplo de visualización de imagen original en conjunto con la máscara binaria correspondiente.

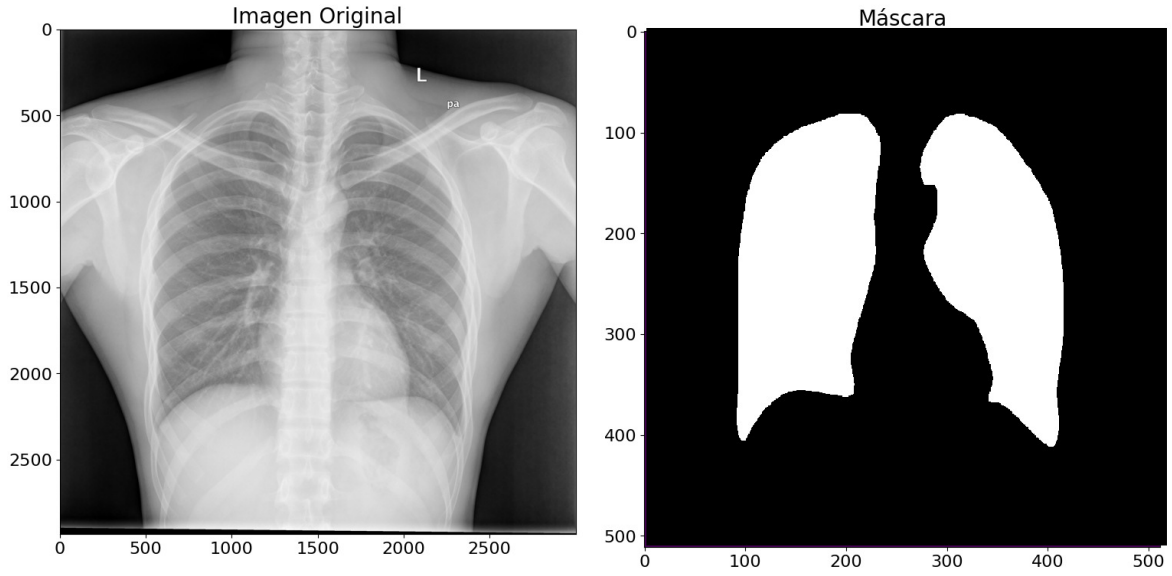


Figura 28. Imagen original y su máscara binaria correspondiente

La Figura 29 muestra otro ejemplo de imágenes originales con su máscara binaria superpuesta.

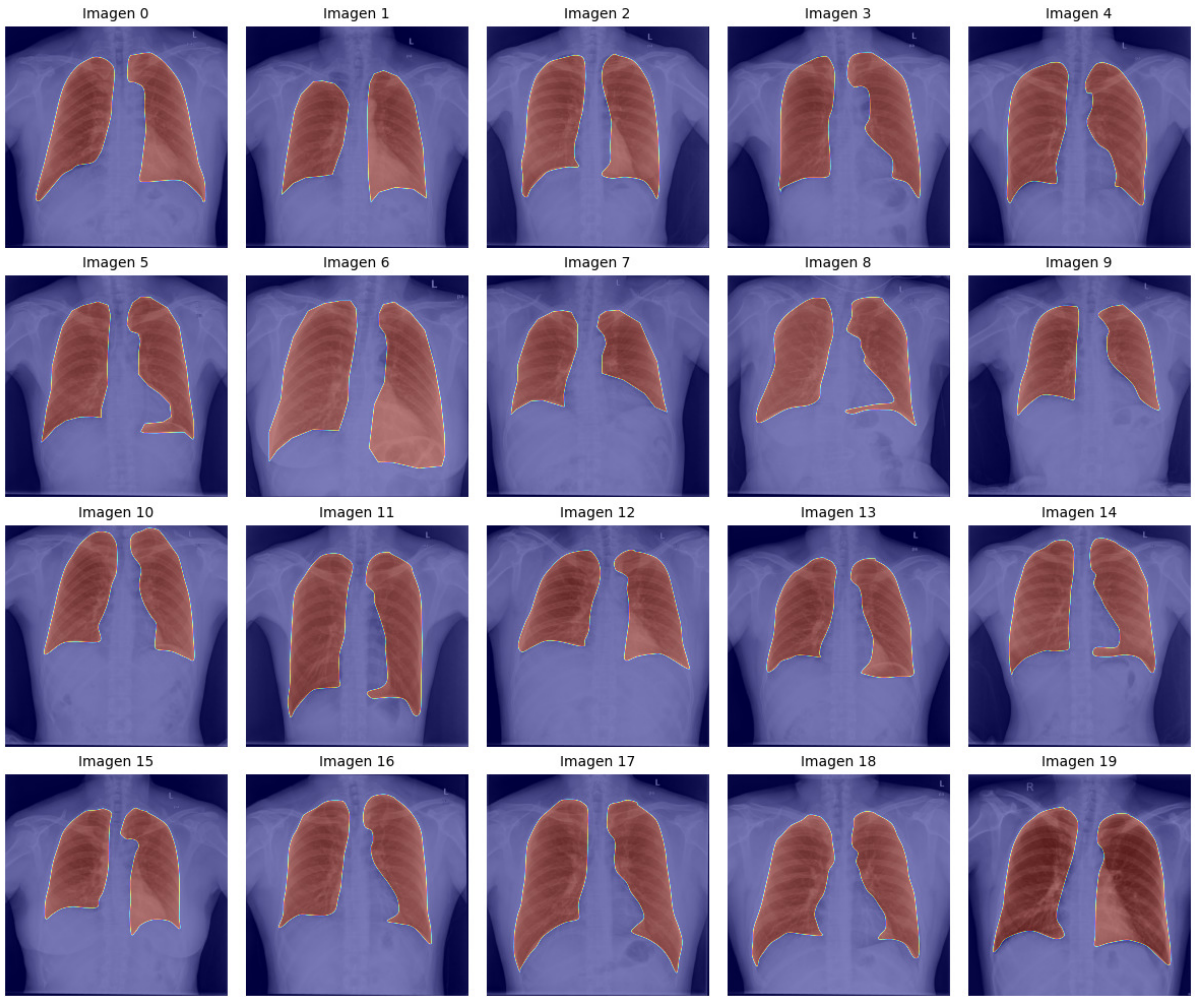


Figura 29. Imágenes originales y máscaras superpuestas

4.3.2. Preprocesamiento de Datos

La realización de esta tarea consistió en la práctica de dos arquitecturas anteriormente comentadas, YOLO y U-Net, en las secciones 3.8 y 3.9, respectivamente.

Preprocesamiento para U-Net

Para el modelo U-Net, el preprocesamiento de las imágenes implicó dos etapas fundamentales:

1. *Conversión y reescalado de imágenes*: Las imágenes originales se convirtieron en matrices de tipo NumPy para facilitar su manipulación en el entorno de Python. Luego, se aplicó un reescalado para homogeneizar las dimensiones y ajustarlas a un formato uniforme de 512x512 píxeles. La escala 512x512 se eligió para adaptarse a la arquitectura U-Net y permitir un procesamiento eficiente de las imágenes.

2. *Preparación de máscaras*: Para el uso de U-Net, las máscaras de segmentación originales se adaptaron para que coincidieran con el formato de entrada de la red. Esto implica la modificación de las dimensiones de las máscaras a 512x512 y, en este caso, se amplió la dimensión de las máscaras a 3 canales para facilitar la coincidencia con la salida de U-Net.

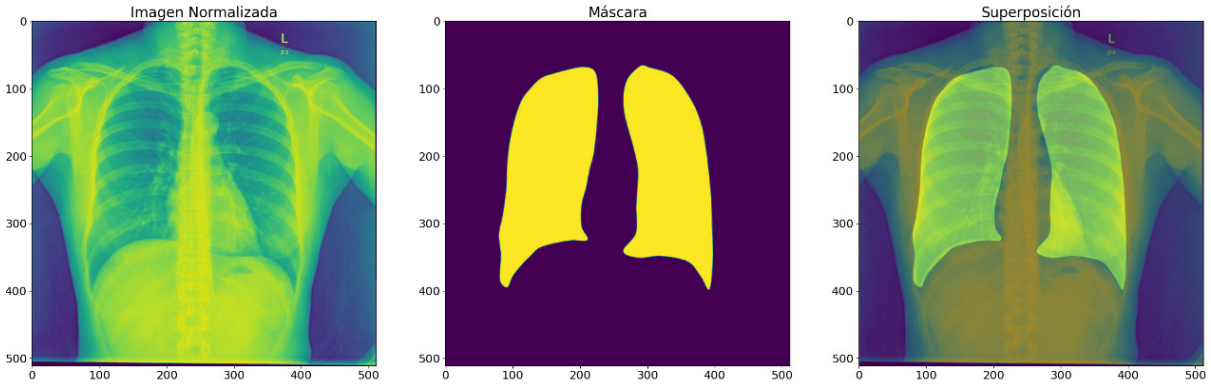


Figura 30. Normalización de imagen y máscara a numpy array

Preprocesamiento para YOLOv8

El modelo YOLOv8 requiere un preprocesamiento diferente debido a que, para segmentar las imágenes, requiere que las máscaras se encuentren en formato de etiquetas con la descripción de las coordenadas del área de los pulmones. Por lo tanto, el formato del conjunto de datos utilizado para entrenar los modelos de segmentación de YOLO es el siguiente:

1. *Reescalado de imágenes:* En una primera instancia, se reescalaron las imágenes a 640x640 pixeles para adaptarlas de mejor manera al modelo de segmentación de YOLOv8.

2. *Obtención de coordenadas desde máscaras*: Para utilizar YOLO, se requiere un conjunto de coordenadas que define las regiones de interés (ROI) en la imagen. Estas coordenadas indican la ubicación de los objetos de interés, en este caso, las regiones pulmonares en las radiografías de tórax. Estas coordenadas se obtuvieron a partir de las máscaras de segmentación, identificando los puntos relevantes que describen la ubicación de las regiones pulmonares.

3. *Guardado en archivos de texto (.txt)*: Las coordenadas obtenidas se guardaron en archivos .txt, que posteriormente se utilizaron como etiquetas para entrenar el modelo YOLO. Estos archivos contienen información sobre las coordenadas y las clases de objetos presentes en la imagen, lo que permite entrenar YOLO para segmentar las regiones pulmonares en las radiografías. El formato de una sola fila en el archivo del conjunto de datos de segmentación es el siguiente:

```
<índice-de-clase> <x1> <y1> <x2> <y2> ... <xn> <yn>
```

- Índice de clase de objeto: es un número entero que representa la clase del objeto (p. ej., 0 para pulmón).
- Coordenadas delimitadoras del objeto: las coordenadas delimitadoras alrededor del área de la máscara, normalizadas entre 0 y 1.

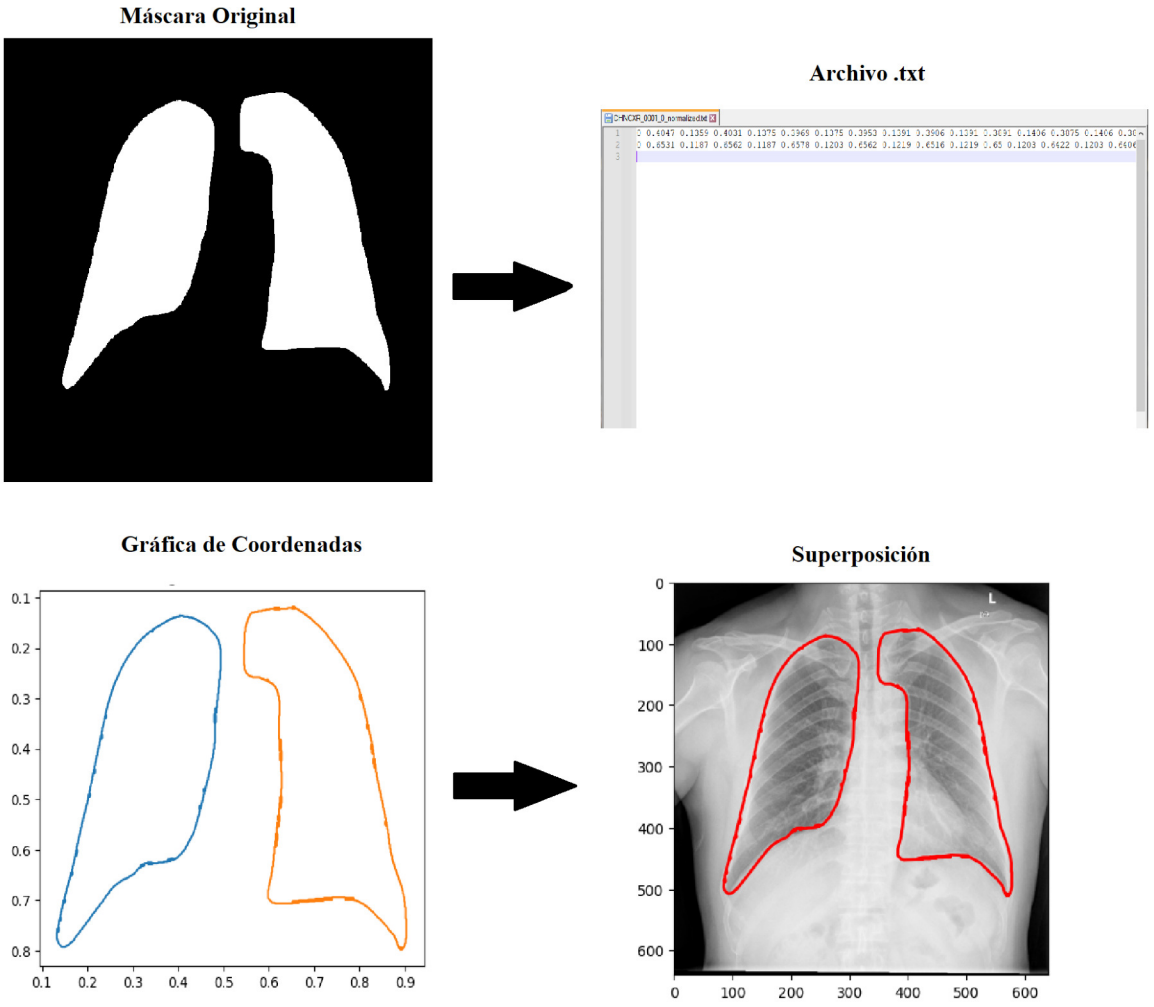


Figura 31. Conversión de máscara original a coordenadas en formato YOLO.

4. *Estructura de directorios:* Se debe construir una estructura de carpetas, tal que, dentro del "dataset" directorio, se encuentren dos directorios más de entrenamiento (train) y validación (val), que a su vez, contengan las imágenes originales de pulmones (images) y las etiquetas (labels) en formatos de archivos de texto anteriormente creados. La estructura debe verse de la siguiente manera:

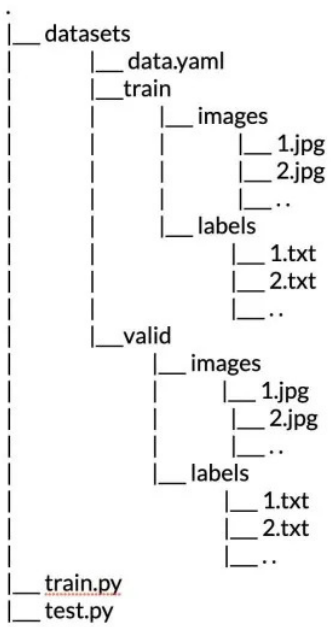


Figura 32. Estructura de carpetas para el formato YOLO

5. *Archivo YAML (.yaml)*: Ultralytics utiliza un formato de archivo YAML para definir el conjunto de datos y la configuración del modelo para entrenar modelos de detección.

```
import yaml
data = {
    'path': '/content/data',
    'train': '/content/data/train/images',
    'val': '/content/data/val/images',
    'nc': 1,
    'names': ['pulmon']
}
with open('custom_data.yaml', 'w') as file:
    yaml.dump(data, file)
```

Los campos `'train'` y `'val'` especifican las rutas a los directorios que contienen las imágenes de entrenamiento y validación, respectivamente. `'names'` es un diccionario de nombres de clases (en nuestro caso solo tenemos una clase, `'pulmon'`). El orden de los nombres debe coincidir con el orden de los índices de clase de objeto en los archivos del conjunto de datos de YOLO.

4.3.3. Selección de Modelos

Selección de modelo U-Net

En primer lugar, se utilizó el modelo U-Net detallado en la sección 3.8.4. La estructura de la red "encoder-decoder" permite una fusión de información a diferentes escalas espaciales, ayudando a mejorar la precisión de la segmentación.

Selección de modelo YOLOv8

Para la selección del modelo YOLOv8, utilizamos un modelo pre-entrenado llamado *YOLOv8n-seg* que facilitará la tarea de segmentar los pulmones. Este modelo fue entrenado con el conjunto de datos COCO para la segmentación de objetos.

4.3.4. Entrenamiento de Modelos

Entrenamiento U-Net

Para llevar a cabo el entrenamiento de U-Net se estableció una función "fit" utilizando *BCEWithLogitsLoss* como función de pérdida y *Adam* como optimizador. También se calcula el *IoU* (*Intersection over Union*) para evaluar el rendimiento del modelo durante el entrenamiento y la evaluación.

El ciclo de entrenamiento se repite durante 30 épocas y se realiza basándose en los datos proporcionados a través del dataloader. El dataloader proporciona los datos de entrenamiento (602) y prueba (102) en lotes. Los DataLoader en PyTorch son útiles para dividir los datos en mini lotes (batches) para el entrenamiento y la evaluación. En este caso, se establece un "batch_size" de 8. Esto significa que durante cada época de entrenamiento, la red neuronal procesará 8 imágenes (y sus respectivas máscaras) a la vez antes de realizar una actualización de los pesos basada en el error de ese lote.

Tabla 4. Hiperparámetros utilizados para entrenar a los modelos U-Net

Hiperparámetro	Descripción	U-Net
epoch	Cantidad de iteraciones	30
pretrained	Si usar un modelo preentrenado	False
lr	Tasa de aprendizaje	0.001
optimizer	Optimizador	Adam
batch	Tamaño de lote	8
imgsz	Tamaño de imagen	512

device	Dispositivo para entrenar	GPU

En cada época, se lleva a cabo el ciclo de entrenamiento y el ciclo de evaluación. Durante el ciclo de entrenamiento, los gradientes se calculan y se actualizan los pesos del modelo mediante el optimizador para minimizar la función de pérdida. Durante el ciclo de evaluación, el modelo no se entrena y simplemente se utilizan los pesos actuales para calcular la función de pérdida y el IoU en el conjunto de prueba.

Figura 33. Entrenamiento de U-Net ejecutado en el entorno Google Colab en un tiempo de 11 minutos y 23 segundos

Al final de cada época, se calcula el promedio de la pérdida y el IoU tanto en el conjunto de entrenamiento como en el conjunto de prueba, y se almacenan en un diccionario. Al final de todas las épocas, se devuelve este diccionario, que contiene información sobre la pérdida y el IoU en el conjunto de entrenamiento y prueba a lo largo del tiempo.

Entrenamiento YOLOv8

Se debe indicar a YOLO los hiperparametros que considerará la red para el entrenamiento. En nuestro caso, indicamos el número de épocas establecido en 30 y el tamaño de imagen (640x640). Existen otros hiperparámetros de configuración para entrenar al modelo pero en esta prueba se establecieron los valores por defecto. La Tabla 5 muestra algunos de los hiperparametros utilizados.

Tabla 5. Hiperparámetros utilizados para entrenar el modelo YOLOv8n-seg

Hiperparámetro	Descripción	YOLOv8n-seg
epoch	Cantidad de iteraciones	30
pretrained	Si usar un modelo preentrenado	True
lr	Tasa de aprendizaje	0.002
optimizer	Optimizador	Adamw
batch	Tamaño de lote	16
imgsz	Tamaño de imagen	640
device	Dispositivo para entrenar	GPU

La Figura 34 muestra información relacionada con la ejecución del modelo en el proceso de entrenamiento y evaluación. “Epoch” representa el número de época o iteración del proceso de entrenamiento, “GPU_mem” muestra la cantidad de memoria utilizada en la GPU durante este proceso, “box_loss” indica la pérdida o error en la regresión de los bounding boxes (cuadros delimitadores), “seg_loss” representa la pérdida asociada con la segmentación semántica, “cls_loss” es la pérdida relacionada con la clasificación y “size” muestra el tamaño de las imágenes utilizadas.

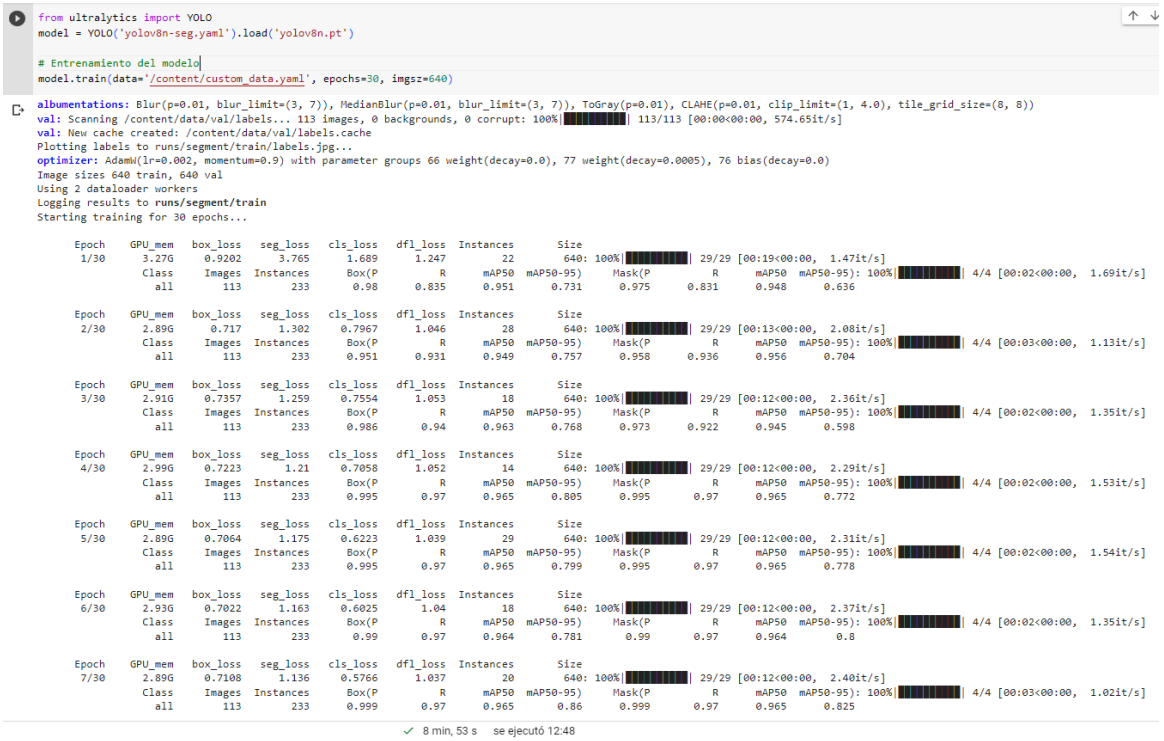


Figura 34. Entrenamiento de YOLO ejecutado en el entorno Google Colab en un tiempo de 8 minutos y 53 segundos

4.3.5. Evaluación de modelos

Análisis y Resultados de U-Net

Para analizar el rendimiento del modelo utilizaremos, por un lado su comportamiento durante el entrenamiento a partir de la función de pérdida (loss), el coeficiente de IoU (iou), la pérdida en el conjunto de pruebas (test_loss) y el IoU en el conjunto de pruebas (test_iou); y por otro lado, el comportamiento a nivel de métricas de calidad, como la precisión (Precision), la sensibilidad (Recall), la exactitud (Accuracy) y el coeficiente de Dice (Dice Coefficient).

Algunos puntos importantes a considerar al observar la Figura 35 son:

- *Loss*: La función de pérdida (loss) se ha reducido significativamente durante el entrenamiento, pasando de 0.10578 al inicio a 0.03694 al final de las 30 épocas. Una pérdida más baja indica que el modelo está aprendiendo a hacer predicciones más precisas y ajustadas a los datos reales.

- *IoU (Intersection over Union)*: El coeficiente de IoU ha aumentado a lo largo del entrenamiento desde 0.87135 hasta 0.94285. Un IoU más alto indica una mejor superposición y ajuste entre las áreas segmentadas por el modelo y las áreas reales de las máscaras.

- *Test Loss*: La pérdida en el conjunto de pruebas ha seguido una tendencia similar a la pérdida de entrenamiento, disminuyendo desde 0.08034 al inicio hasta 0.03669 al final. Esto muestra que el modelo está generalizando bien y manteniendo su rendimiento en datos no vistos durante el entrenamiento.
- *Test IoU*: El IoU en el conjunto de pruebas ha aumentado de 0.89831 a 0.94757 a lo largo del entrenamiento. Esto indica que el modelo está siendo capaz de generalizar bien y realizar predicciones precisas en datos de prueba.

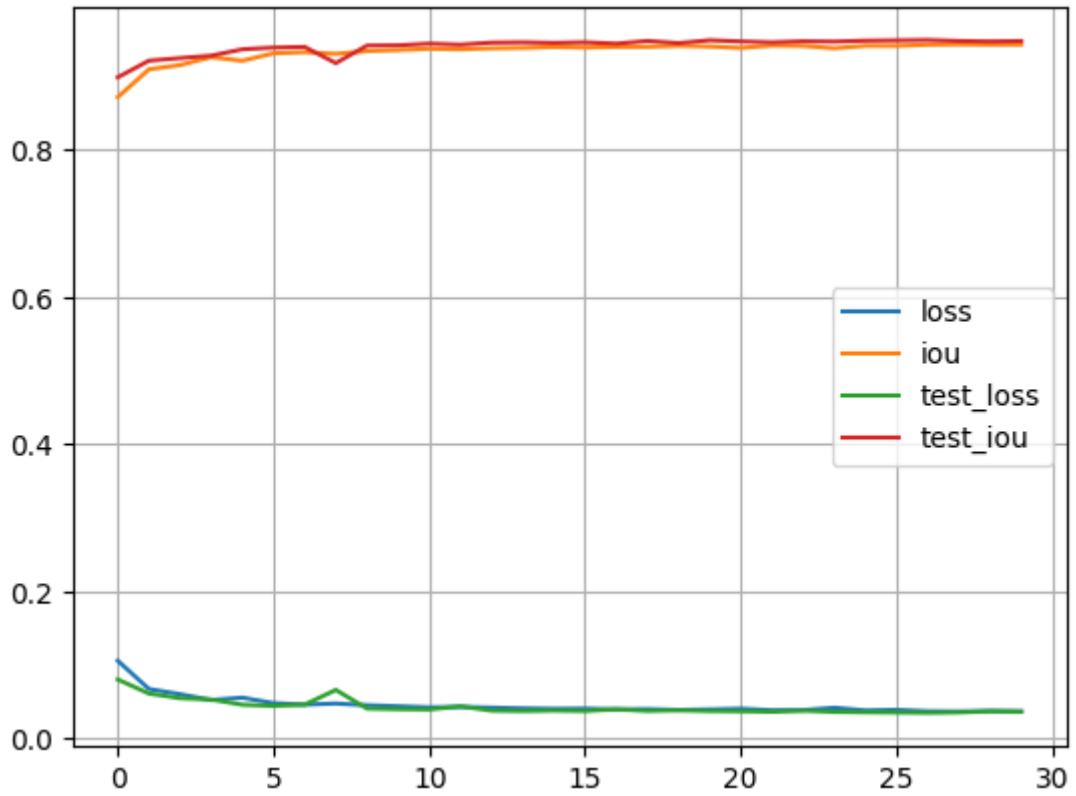


Figura 35. Comportamiento del modelo durante el entrenamiento

Así mismo, evaluamos el modelo con las métricas expuestas en la Tabla 6, de la cuál podemos destacar lo siguiente:

- *Precision*: En este caso, una precisión de 0.9836 significa que el modelo tiene un alto porcentaje de verdaderos positivos en relación con los falsos positivos, lo que indica que las predicciones positivas del modelo son muy precisas.
- *Recall*: Con un recall de 0.9652, el modelo tiene un alto porcentaje de verdaderos positivos en relación con los falsos negativos, lo que indica que el modelo es capaz de detectar la mayoría de las instancias positivas en el conjunto de datos.

- *Accuracy*: Con una exactitud de 0.9834, el modelo está teniendo un rendimiento muy alto en términos de predicciones precisas.
- *Dice Coefficient*: Un valor de 0.9743 indica que la segmentación del modelo se ajusta bien a la máscara de la verdad terreno.

Tabla 6. Evaluación de Métricas del modelo U-Net

Métrica	**Valor**
Precision	0.9836
Recall	0.9652
Accuracy	0.9834
Dice Coefficient	0.9743

Predicciones de U-Net

Luego del entrenamiento del modelo, se pone a prueba en términos de predicción, analizando muestras que el modelo jamás haya visto. En este caso se seleccionaron 5 muestras al azar y se visualizaron las predicciones de la siguiente manera:

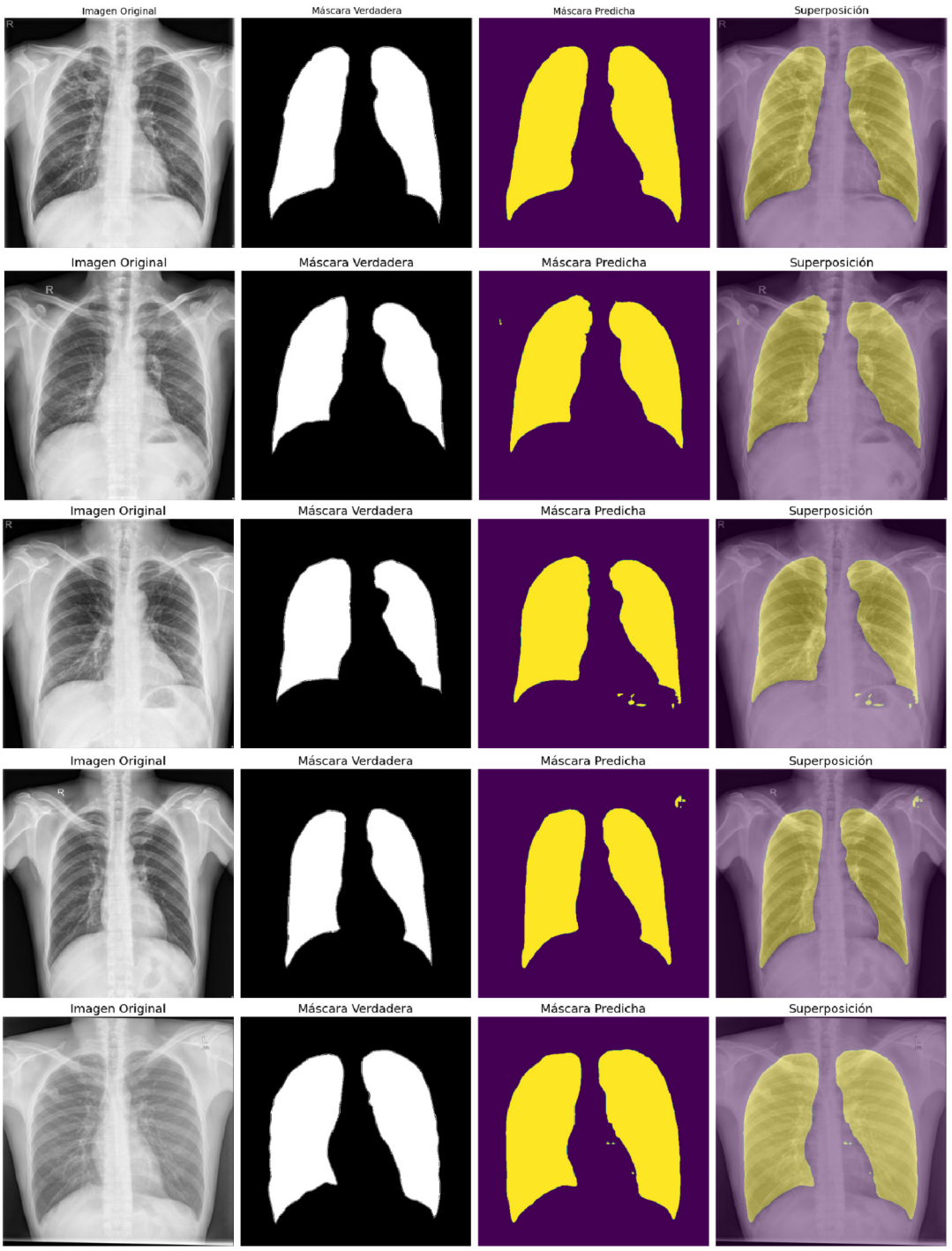


Figura 36. Predicciones del modelo U-Net

Análisis y Resultados de YOLO

En este caso, nos apoyamos de la herramienta Ultralityc (sección 3.9.3.) que genera automáticamente resultados sobre el modelo entrenado.

Como se indica en la Figura 37, a medida que avanzan las épocas, podemos observar cómo las métricas evolucionan. El objetivo es que las pérdidas disminuyan y las métricas de precisión y recuperación aumenten, lo que indica que el modelo está aprendiendo y mejorando.

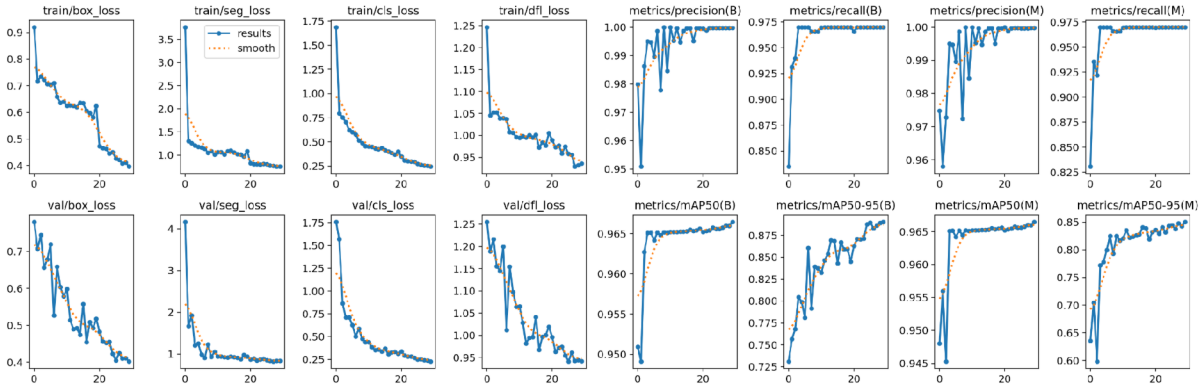


Figura 37. Gráficos de pérdida y mAp después de entrenar el modelo YOLOv8n para la segmentación

A su vez, podemos observar las curvas de precisión y recuperación de la Figura 38. Estas curvas son útiles para evaluar el rendimiento del modelo en términos de detección del pulmón y segmentación de máscaras en diferentes niveles de confianza o umbrales. Un rendimiento óptimo se logra cuando las curvas indican altas puntuaciones de F1 y áreas bajo la curva (AUC) para la detección y segmentación.

- *Box F1 Curve*: Representa la puntuación F1 para la detección de cajas delimitadoras (bounding boxes) a diferentes niveles de confianza o umbral.
- *Box PR Curve*: Es la curva de precisión-recuperación para la detección de cajas delimitadoras. La precisión es la proporción de verdaderos positivos entre todas las detecciones positivas, y la recuperación es la proporción de verdaderos positivos entre todos los ejemplos positivos.
- *Box P Curve*: Muestra cómo la precisión varía a medida que se ajusta el umbral de confianza para considerar una detección como positiva.
- *Box R Curve*: Representa la recuperación (recall) de la detección de cajas delimitadoras en función del umbral de confianza.

- *Mask F1 Curve*: Es similar a la Box F1 Curve, pero se aplica a la segmentación de máscaras (segmentación de instancias).
- *Mask PR Curve*: Es la curva de precisión-recuperación para la segmentación de máscaras.
- *Mask P Curve*: Representa la precisión de la segmentación de máscaras en función del umbral de confianza.
- *Mask R Curve*: Representa la recuperación (recall) de la segmentación de máscaras en función del umbral de confianza.

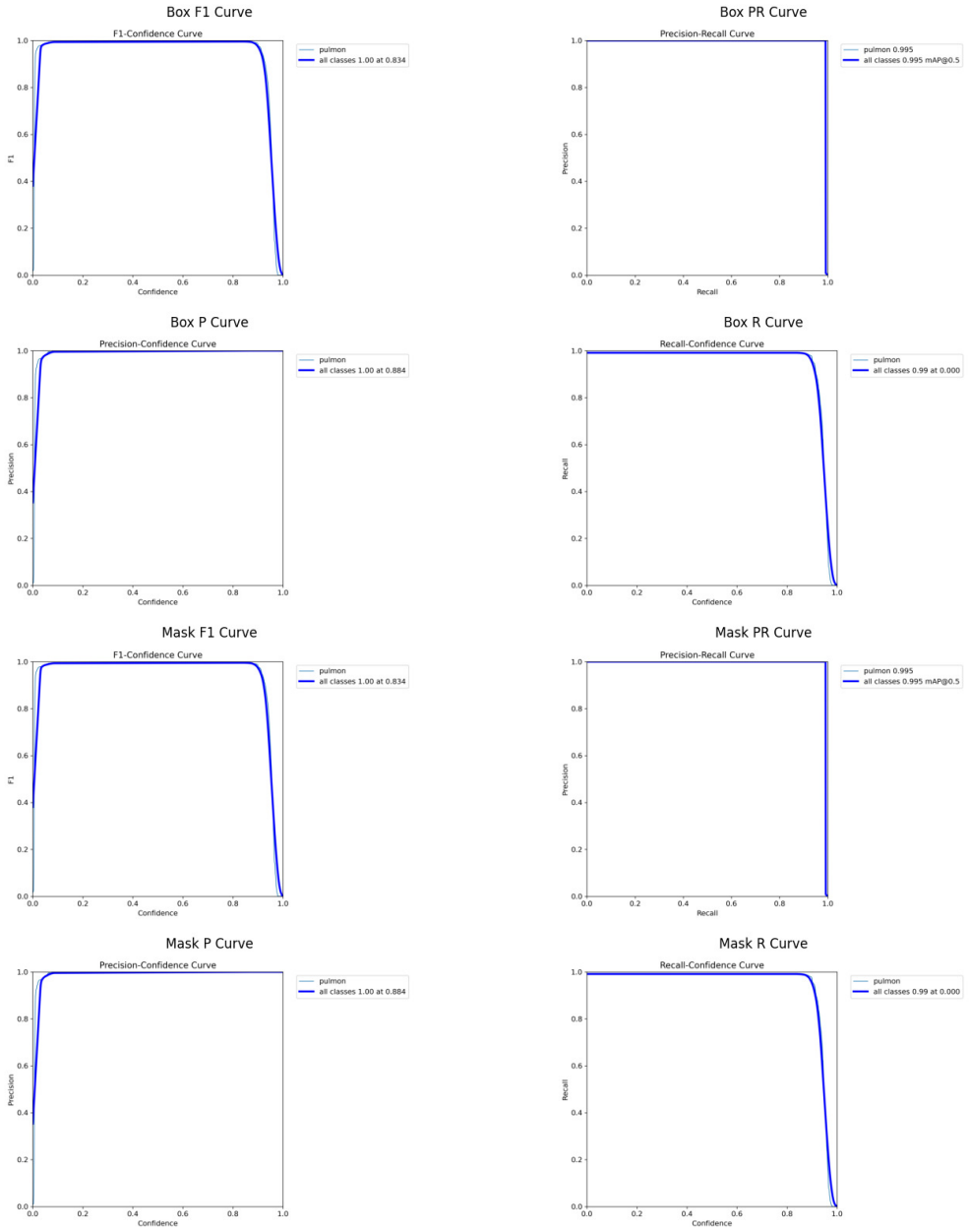


Figura 38. Curvas de precisión y recuperación del modelo YOLOv8n-seg

Otra métrica importante que nos provee la herramienta Ultralitycs es la matriz de confusión, recordando que, no solo se aplica la tarea de segmentación y detección, sino que, también se está clasificando los pulmones.

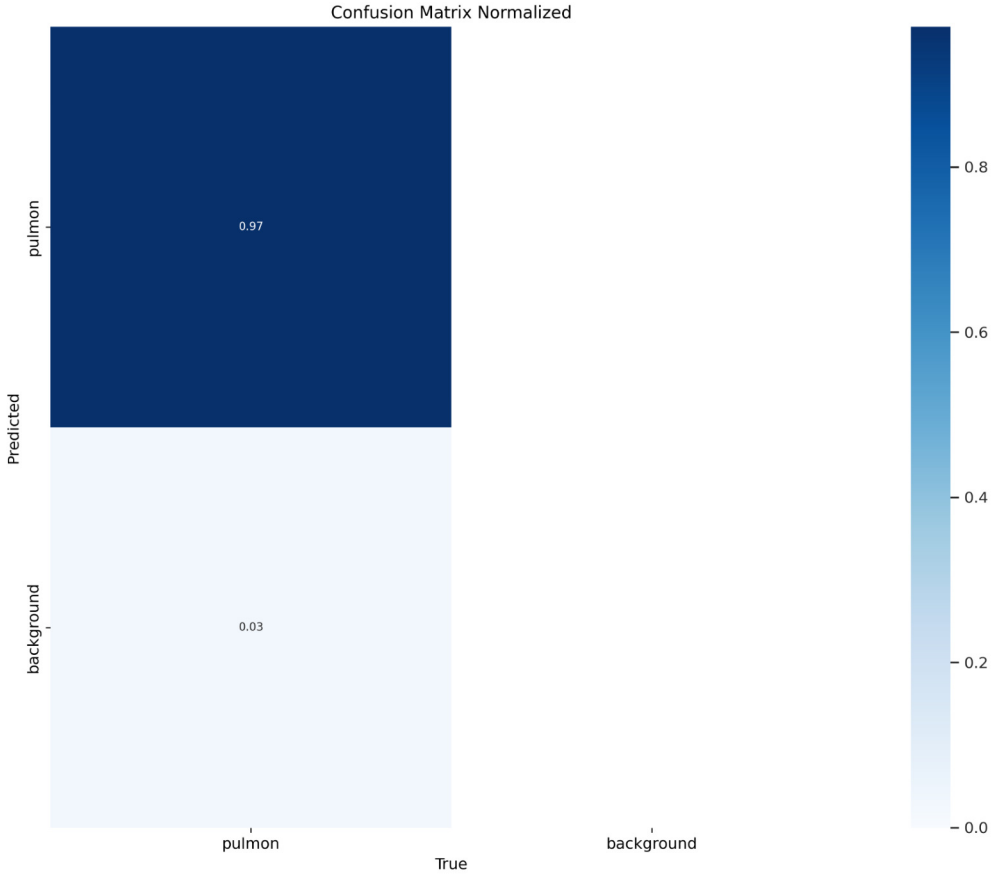


Figura 39. Matriz de confusión del modelo YOLOv8n-seg

Predicciones de YOLO

Por otra parte, el entrenamiento de yolo se desarrolló de manera óptima, alcanzando grandes valores de precisión. En la Figura 40 se visualiza la predicción de algunas muestras segmentando la ROI del pulmón e incluso generando un cuadro delimitador con el porcentaje de acierto.

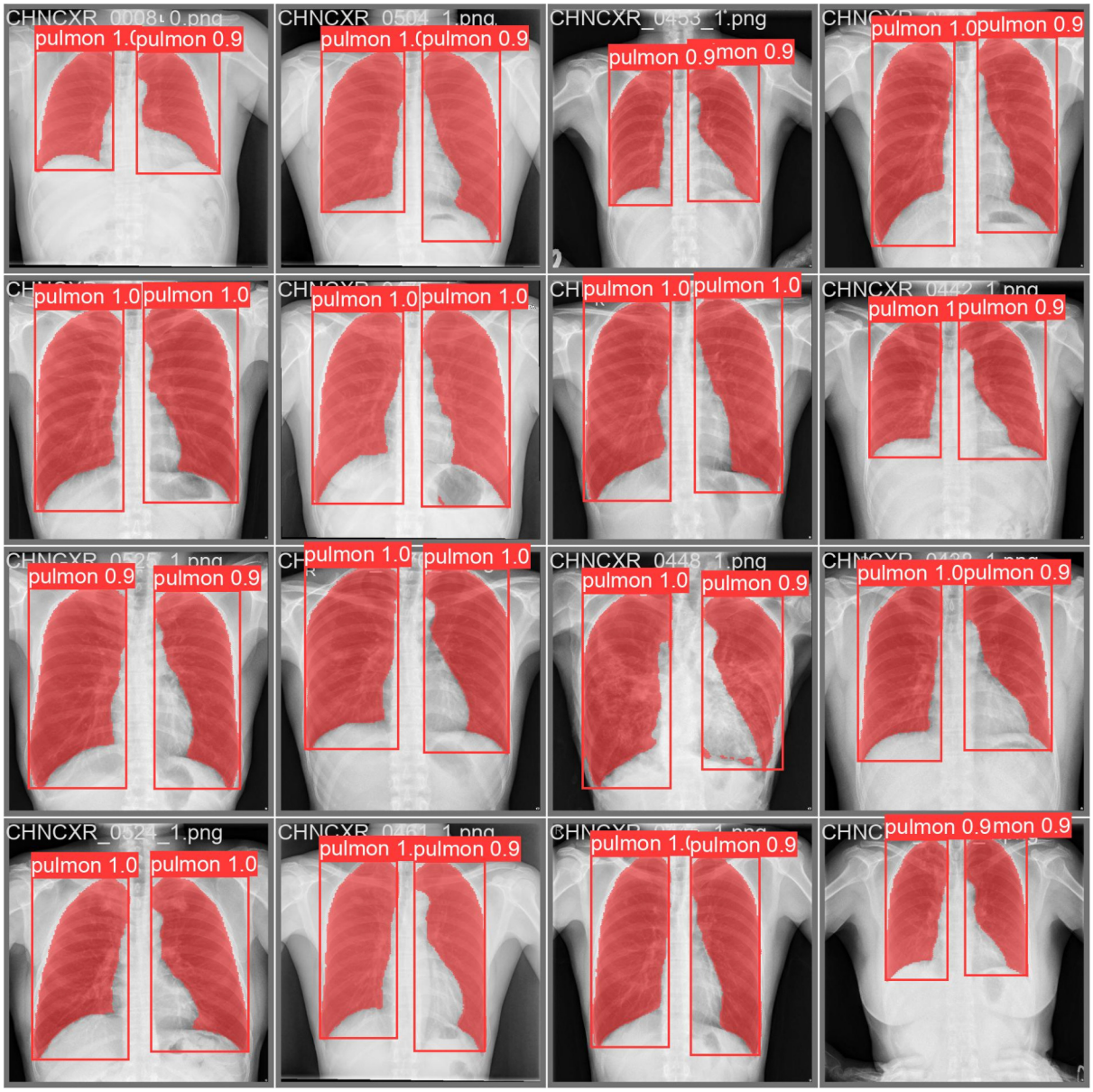


Figura 40. Detección y segmentación de pulmones

4.4. Clasificación y Detección de Neumonía

En este apartado ponemos en marcha las tareas de clasificación y detección de neumonía a partir de imágenes en formato DICOM. El propósito de desarrollar estas tareas es avanzar a algo más complejo como lo es el descubrimiento de una enfermedad difícil de detectar a simple vista. A diferencia de la segmentación, se escogió emplear estas tareas a un mismo dataset "RSNA Pneumonia Detection Challenge" (descrito en la sección 4.2.2) ya que, contiene la información necesaria para llevarlas a cabo. La diferencia radica en que la segmentación depende de áreas previamente señaladas (como las máscaras binarias). A su vez.

Por otra parte, se ejecutaron dichas tareas localmente, haciendo uso de 3 unidades de procesamiento gráfico (GPU) expuesto en la sección 4.1.3. En YOLOv8, la biblioteca de PyTorch, "torch.nn.DataParallel", se utiliza para distribuir el modelo en múltiples GPUs. Con DataParallel (ver sección 3.9.2.), el modelo se replica en cada GPU y cada GPU procesa diferentes lotes de datos en paralelo. Esto nos permitió acelerar el entrenamiento al dividir la carga de trabajo en las GPUs utilizadas.

4.4.1. Exploración y visualización de Datos

El conjunto de datos “RSNA Pneumonia Detection Challenge” contiene un total de 26.684 estudios médicos bajo el estándar DICOM. A su vez, contiene un archivo llamado “stage_2_train_labels.csv” donde se encuentra la información de cada estudio como se muestra en la Figura 41.

	patientId	x	y	width	height	Target
0	0004cfab-14fd-4e49-80ba-63a80b6bddd6	NaN	NaN	NaN	NaN	0
1	00313ee0-9eaa-42f4-b0ab-c148ed3241cd	NaN	NaN	NaN	NaN	0
2	00322d4d-1c29-4943-afc9-b6754be640eb	NaN	NaN	NaN	NaN	0
3	003d8fa0-6bf1-40ed-b54c-ac657f8495c5	NaN	NaN	NaN	NaN	0
4	00436515-870c-4b36-a041-de91049b9ab4	264.0	152.0	213.0	379.0	1

Figura 41. Etiquetas para las imágenes de entrenamiento.

Esta información cuenta con las coordenadas de los cuatro delimitadores para los casos que sí tienen neumonía. Asimismo, cuenta con una columna llamada “Target” en donde se especifica la clase a la cuál un estudio pertenece, siendo 0 para identificar que no tiene neumonía y 1 para afirmar que es un caso con neumonía.

Por otra parte, podemos observar los estudios DICOM con más detalle con la ayuda de la biblioteca “pydicom”. La Figura 42 muestra un ejemplo del primer paciente de la lista que contiene información sobre la imagen radiográfica del tórax, incluyendo detalles del paciente y metadatos de la imagen. Utilizando esta información, se puede acceder a la imagen y realizar tareas de procesamiento de imágenes, como la detección de neumonía.

```
patientId = annots['patientId'][0]
dcm_file = os.path.join(train_dcm_dir,"{}.dcm".format(patientId))
dcm_data = pydicom.read_file(dcm_file)

print(dcm_data) #Muestra los metadatos del archivo dcm
```

```
Dataset.file_meta -------------------------------
(0002, 0000) File Meta Information Group Length  UL: 202
(0002, 0001) File Meta Information Version       OB: b'\x00\x01'
(0002, 0002) Media Storage SOP Class UID         UI: Secondary Capture Image Storage
(0002, 0003) Media Storage SOP Instance UID      UI: 1.2.276.0.7230010.3.1.4.8323329.28530.1517874485.775526
(0002, 0010) Transfer Syntax UID                 UI: JPEG Baseline (Process 1)
(0002, 0012) Implementation Class UID            UI: 1.2.276.0.7230010.3.0.3.6.0
(0002, 0013) Implementation Version Name         SH: 'OFFIS_DCMTK_360'
-------------------------------------------------
(0008, 0005) Specific Character Set              CS: 'ISO_IR 100'
(0008, 0016) SOP Class UID                       UI: Secondary Capture Image Storage
(0008, 0018) SOP Instance UID                    UI: 1.2.276.0.7230010.3.1.4.8323329.28530.1517874485.775526
(0008, 0020) Study Date                          DA: '19010101'
(0008, 0030) Study Time                          TM: '000000.00'
(0008, 0050) Accession Number                    SH: ''
(0008, 0060) Modality                            CS: 'CR'
(0008, 0064) Conversion Type                     CS: 'WSD'
(0008, 0090) Referring Physician's Name          PN: ''
(0008, 103e) Series Description                  LO: 'view: PA'
(0010, 0010) Patient's Name                      PN: '0004cfab-14fd-4e49-80ba-63a80b6bddd6'
(0010, 0020) Patient ID                          LO: '0004cfab-14fd-4e49-80ba-63a80b6bddd6'
(0010, 0030) Patient's Birth Date                DA: ''
(0010, 0040) Patient's Sex                       CS: 'F'
(0010, 1010) Patient's Age                       AS: '51'
(0018, 0015) Body Part Examined                  CS: 'CHEST'
(0018, 5101) View Position                       CS: 'PA'
(0020, 000d) Study Instance UID                  UI: 1.2.276.0.7230010.3.1.2.8323329.28530.1517874485.775525
(0020, 000e) Series Instance UID                 UI: 1.2.276.0.7230010.3.1.3.8323329.28530.1517874485.775524
(0020, 0010) Study ID                            SH: ''
(0020, 0011) Series Number                       IS: '1'
(0020, 0013) Instance Number                     IS: '1'
(0020, 0020) Patient Orientation                 CS: ''
(0028, 0002) Samples per Pixel                   US: 1
(0028, 0004) Photometric Interpretation          CS: 'MONOCHROME2'
(0028, 0010) Rows                                US: 1024
(0028, 0011) Columns                             US: 1024
(0028, 0030) Pixel Spacing                       DS: [0.14300000000000002, 0.14300000000000002]
(0028, 0100) Bits Allocated                      US: 8
(0028, 0101) Bits Stored                         US: 8
(0028, 0102) High Bit                            US: 7
(0028, 0103) Pixel Representation                US: 0
(0028, 2110) Lossy Image Compression             CS: '01'
(0028, 2114) Lossy Image Compression Method      CS: 'ISO_10918_1'
(7fe0, 0010) Pixel Data                          OB: Array of 142006 elements
```

Figura 42. Ejemplo de un estudio de un paciente en formato DICOM

Entre la metadata que contiene este estudio, podemos resaltar la siguiente información:

- *Modality*: Indica el tipo de modalidad de imagen utilizada para capturar la imagen médica. En este caso, 'CR' se refiere a "Computed Radiography", técnica de imagen radiográfica digital.
- *Patient's Name, Patient ID, Patient's Age y Patient's Sex*: Son detalles del paciente asociado con la imagen. La información no incluye el nombre del paciente, en su lugar se describe su identificación única. También podemos observar la edad y género.

- *Study Date y Study Time*: Indican la fecha y hora en que se realizó el estudio radiológico.
- *Pixel Spacing, Rows y Columns*: Proporcionan información sobre las dimensiones de la imagen y el espaciado entre píxeles. Para todos los casos, las dimensiones son de 1024x1024 píxeles.
- *Photometric Interpretation y Pixel Representation*: Describen cómo se interpreta y representa cada píxel de la imagen. En este caso, la imagen es monocromática (MONOCHROME2) y utiliza una representación de 8 bits.

De igual modo, podemos ver la cantidad de pacientes diagnosticados con neumonía como se ve en la Figura 43.

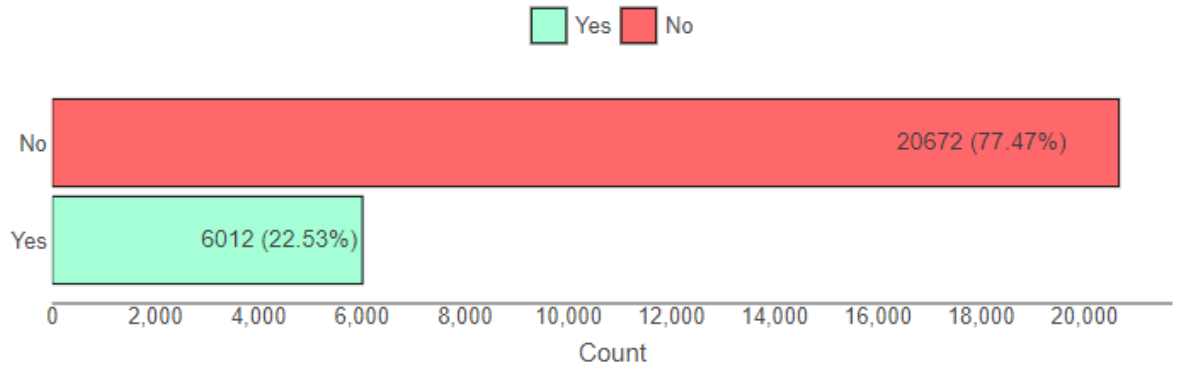


Figura 43. Cantidad de muestras con y sin neumonía

El 77.47% corresponde a estudios sin neumonía detectada, mientras que, el 22,53% restante son muestras con neumonía detectada.

¿Cómo se ve la neumonía?

Para comprender cómo se ve la neumonía en una radiografía, mostraremos dos imágenes de pulmones: una correspondiente a un pulmón sano y otra con opacidades pulmonares, la cual categorizamos (a modo de simplificación) como "NEUMONÍA" (ver Figura 44).

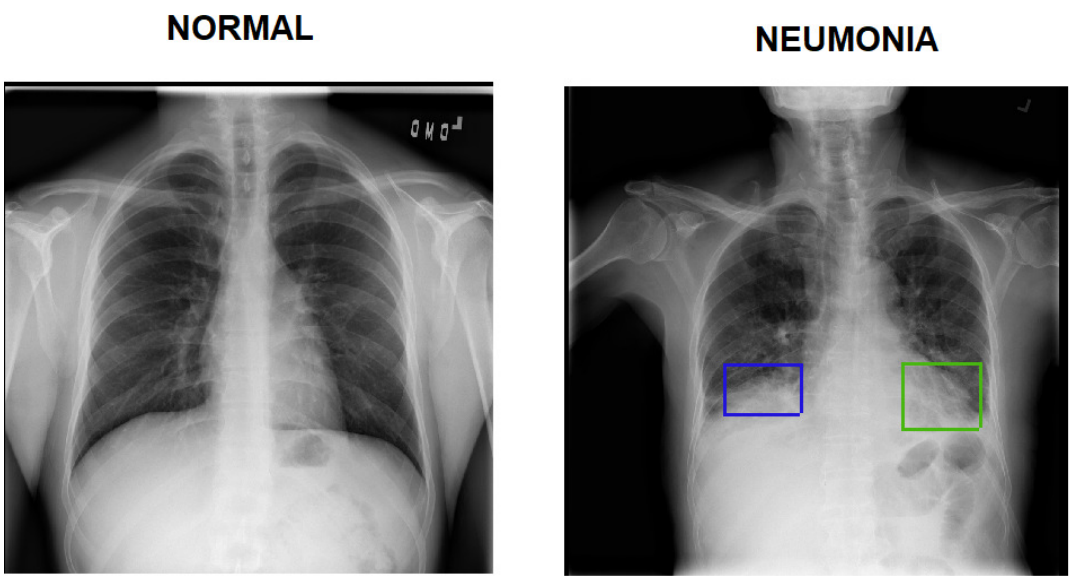


Figura 44. Ejemplo de pulmones normales vs pulmones con opacidades

La imagen de pulmones sanos, clasificada como "NORMAL", representa la apariencia típica y saludable de los pulmones en una radiografía. Los pulmones aparecen claros y transparentes, sin ninguna anomalía o señal de enfermedad. En contraste, observamos una radiografía con opacidades pulmonares, lo cual es indicativo de presencia de neumonía en la imagen.

Las opacidades pulmonares son áreas más densas o sombreadas que pueden aparecer en la radiografía debido a la acumulación de líquido, inflamación o infección en los tejidos pulmonares. Estas opacidades pueden manifestarse como manchas, parches o áreas borrosas en la imagen, y su presencia es un signo característico de la neumonía en una radiografía.

4.5. Clasificación

4.5.1. Preprocesamiento de Datos

Preprocesamiento para la Clasificación

El proceso de sustracción de imágenes se realizó de manera sistemática, seleccionando radiografías de pacientes con diagnósticos confirmados de neumonía y otras radiografías de pacientes sin evidencia de esta enfermedad pulmonar. Por lo tanto, se creó un nuevo conjunto de datos compuesto por 6012 imágenes de radiografías de tórax con casos de neumonía y otras 6012 imágenes de radiografías de tórax sin presencia de neumonía.

Es importante destacar que el equilibrio en la cantidad de imágenes con y sin neumonía fue un criterio relevante para garantizar la adecuada representación de ambas clases establecidas como NORMAL y NEUMONIA.

Considerando que se tiene un total de 12.024 muestras, se optó por utilizar una división del 70% para entrenamiento, 15% para validación y 15% para prueba:

- *Train (70%)* = 8.416 muestras
- *Val (15%)* = 1.804 muestras
- *Test (15%)* = 1.804 muestras

El árbol de directorios del dataset debe estar estructurado de la siguiente manera:

```
. Dataset
├── test
│   ├── NEUMONIA
│   └── NORMAL
├── train
│   ├── NEUMONIA
│   └── NORMAL
└── val
    ├── NEUMONIA
    └── NORMAL
```

4.5.2. Selección de Modelos

Modelo YOLOv8n-cls (Clasificación)

El modelo YOLOv8n-cls fue seleccionado para abordar la tarea de clasificación de las imágenes de las radiografías del tórax en dos clases: NORMAL y NEUMONÍA. Esta arquitectura está diseñada específicamente para la clasificación y se enfoca en asignar una etiqueta a cada imagen según su categoría correspondiente.

4.5.3. Entrenamiento de Modelos

Al igual que en la tarea de segmentación, se debe indicar a YOLO los hiperparametros que considerará la red para el entrenamiento.Para este caso en concreto, indicamos el número de épocas establecido en 30 y el tamaño de imagen (640x640). También el “batch” en 21 para que sea divisible por el número de tarjetas gráficas utilizadas.

Tabla 7. Hiperparámetros utilizados para entrenar el modelo YOLOv8n-cls

Hiperparámetro	Descripción	YOLOv8n-cls
epoch	Cantidad de iteraciones	30
pretrained	Si usar un modelo preentrenado	True
lr	Tasa de aprendizaje	0.000714
optimizer	Optimizador	Adamw
batch	Tamaño de lote	21
imgsz	Tamaño de imagen	640
device	Dispositivo para entrenar	3xGPU

Una vez teniendo los hiperparametros establecidos, se pone a entrenar la red como indica la Figura 45. Primero se establece conexión con la placas gráficas mediante CUDA, luego la red lee los hiperparametros que fueron establecidos y los que no, se toman por defecto. Luego se hace una lectura del modelo de la red y se verifica que el modo DDP (expuesto en la sección 3.9.2) esté disponible.

```
minero@minero-desktop:~/Escritorio/RSNAv2/yolov8$ python3 cl_train.py
New https://pypi.org/project/ultralytics/8.0.145 available 😃 Update with 'pip install -U ultralytics'
Ultralytics YOLOv8.0.137 🚀 Python-3.10.6 torch-2.0.1+cu117 CUDA:0 (NVIDIA GeForce RTX 3060 Ti, 7982MiB)
                                                    CUDA:1 (NVIDIA GeForce RTX 3060 Ti, 7982MiB)
                                                    CUDA:2 (NVIDIA GeForce RTX 2060 SUPER, 7982MiB)
engine/trainer: task=classify, mode=train, model=yolov8n-cls.pt, data=/home/minero/Escritorio/RSNAv2/yolov8/datase
, device=[0, 1, 2], workers=8, project=RSNA_cl, name=yolov8n_cls2, exist_ok=False, pretrained=True, optimizer=auto
lose_mosaic=10, resume=False, amp=True, fraction=1.0, profile=False, overlap_mask=True, mask_ratio=4, dropout=0.0,
, half=False, dnn=False, plots=True, source=None, show=False, save_txt=False, save_conf=False, save_crop=False, sh
alse, agnostic_nms=False, classes=None, retina_masks=False, boxes=True, format=torchscript, keras=False, optimize=
=0.01, lrf=0.01, momentum=0.937, weight_decay=0.0005, warmup_epochs=3.0, warmup_momentum=0.8, warmup_bias_lr=0.1,
 hsv_s=0.7, hsv_v=0.4, degrees=0.0, translate=0.1, scale=0.5, shear=0.0, perspective=0.0, flipud=0.0, fliplr=0.5,
olov8n_cls2
Overriding model.yaml nc=1000 with nc=2

                   from  n    params  module                                       arguments
  0                  -1  1       464  ultralytics.nn.modules.conv.Conv             [3, 16, 3, 2]
  1                  -1  1      4672  ultralytics.nn.modules.conv.Conv             [16, 32, 3, 2]
  2                  -1  1      7360  ultralytics.nn.modules.block.C2f             [32, 32, 1, True]
  3                  -1  1     18560  ultralytics.nn.modules.conv.Conv             [32, 64, 3, 2]
  4                  -1  2     49664  ultralytics.nn.modules.block.C2f             [64, 64, 2, True]
  5                  -1  1     73984  ultralytics.nn.modules.conv.Conv             [64, 128, 3, 2]
  6                  -1  2    197632  ultralytics.nn.modules.block.C2f             [128, 128, 2, True]
  7                  -1  1    295424  ultralytics.nn.modules.conv.Conv             [128, 256, 3, 2]
  8                  -1  1    460288  ultralytics.nn.modules.block.C2f             [256, 256, 1, True]
  9                  -1  1    332802  ultralytics.nn.modules.head.Classify         [256, 2]
[W NNPACK.cpp:64] Could not initialize NNPACK! Reason: Unsupported hardware.
YOLOv8n-cls summary: 99 layers, 1440850 parameters, 1440850 gradients, 3.4 GFLOPs
Transferred 156/158 items from pretrained weights
DDP command: ['/usr/bin/python3', '-m', 'torch.distributed.run', '--nproc_per_node', '3', '--master_port', '46773'
WARNING:__main__:
*****************************************
Setting OMP_NUM_THREADS environment variable for each process to be 1 in default, to avoid your system being overl
eded.
*****************************************
[W NNPACK.cpp:64] Could not initialize NNPACK! Reason: Unsupported hardware.
New https://pypi.org/project/ultralytics/8.0.145 available 😃 Update with 'pip install -U ultralytics'
[W NNPACK.cpp:64] Could not initialize NNPACK! Reason: Unsupported hardware.
Overriding model.yaml nc=1000 with nc=2
[W NNPACK.cpp:64] Could not initialize NNPACK! Reason: Unsupported hardware.
YOLOv8n-cls summary: 99 layers, 1440850 parameters, 1440850 gradients, 3.4 GFLOPs
Transferred 156/158 items from pretrained weights
DDP info: RANK 0, WORLD_SIZE 3, DEVICE cuda:0
TensorBoard: Start with 'tensorboard --logdir RSNA_cl/yolov8n_cls2', view at http://localhost:6006/
AMP: running Automatic Mixed Precision (AMP) checks with YOLOv8n...
AMP: checks passed ✅
```

Figura 45. Establecimiento de parámetros de la red

Una vez hecho esto, la red se pone a entrenar como indica la Figura 46. Durante esta etapa podemos ver la cantidad de memoria que está usando de las GPUs, en este caso es menor a 1Gb.

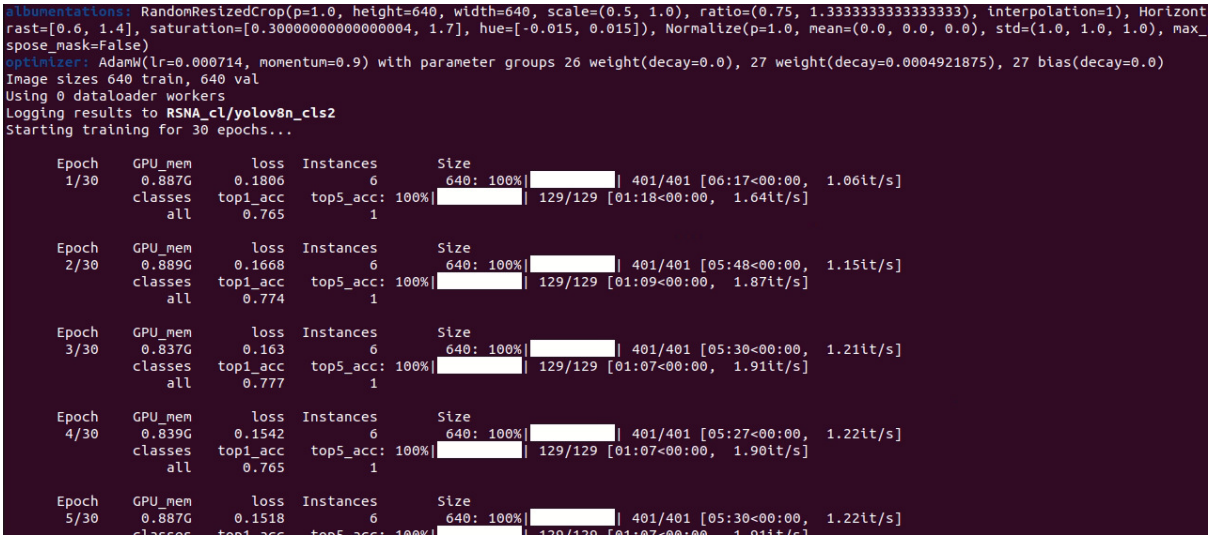


Figura 46. Entrenamiento de YOLOv8n-cls ejecutado en el entorno Local

El entrenamiento de 30 épocas tuvo una duración de aproximadamente 3 horas, 13 minutos y 1 segundo.

4.5.4. Evaluación de Modelos

Análisis y Resultados

Para evaluar el rendimiento del modelo, nos basamos en tres métricas relevantes: La pérdida del conjunto de entrenamiento (train/loss), la pérdida del conjunto de validación (val/loss) y la precisión (accuracy) del conjunto de entrenamiento (Figura 47).

- *train/loss*: La pérdida en el conjunto de entrenamiento disminuye progresivamente a medida que avanzan las épocas. Esto es una señal positiva y sugiere que el modelo está aprendiendo a ajustarse mejor a los datos de entrenamiento.

- *val/loss*: La pérdida en el conjunto de validación también muestra una tendencia general a disminuir, lo cual es positivo, ya que indica que el modelo generaliza adecuadamente para datos no vistos. Sin embargo, la disminución de la pérdida en el conjunto de validación parece fluctuar un poco.

- *metrics/accuracy_top1*: La precisión en el conjunto de entrenamiento también mejora con el tiempo y alcanza un valor de aproximadamente 0.79 en la última época. Esto indica que el modelo está clasificando correctamente alrededor del 79% de las muestras en el conjunto de entrenamiento.

El *accuracy_top5* no es relevante en este caso, ya que se refiere a la precisión entre las 5 clases principales predichas, y en nuestro problema solo tenemos 2 clases.

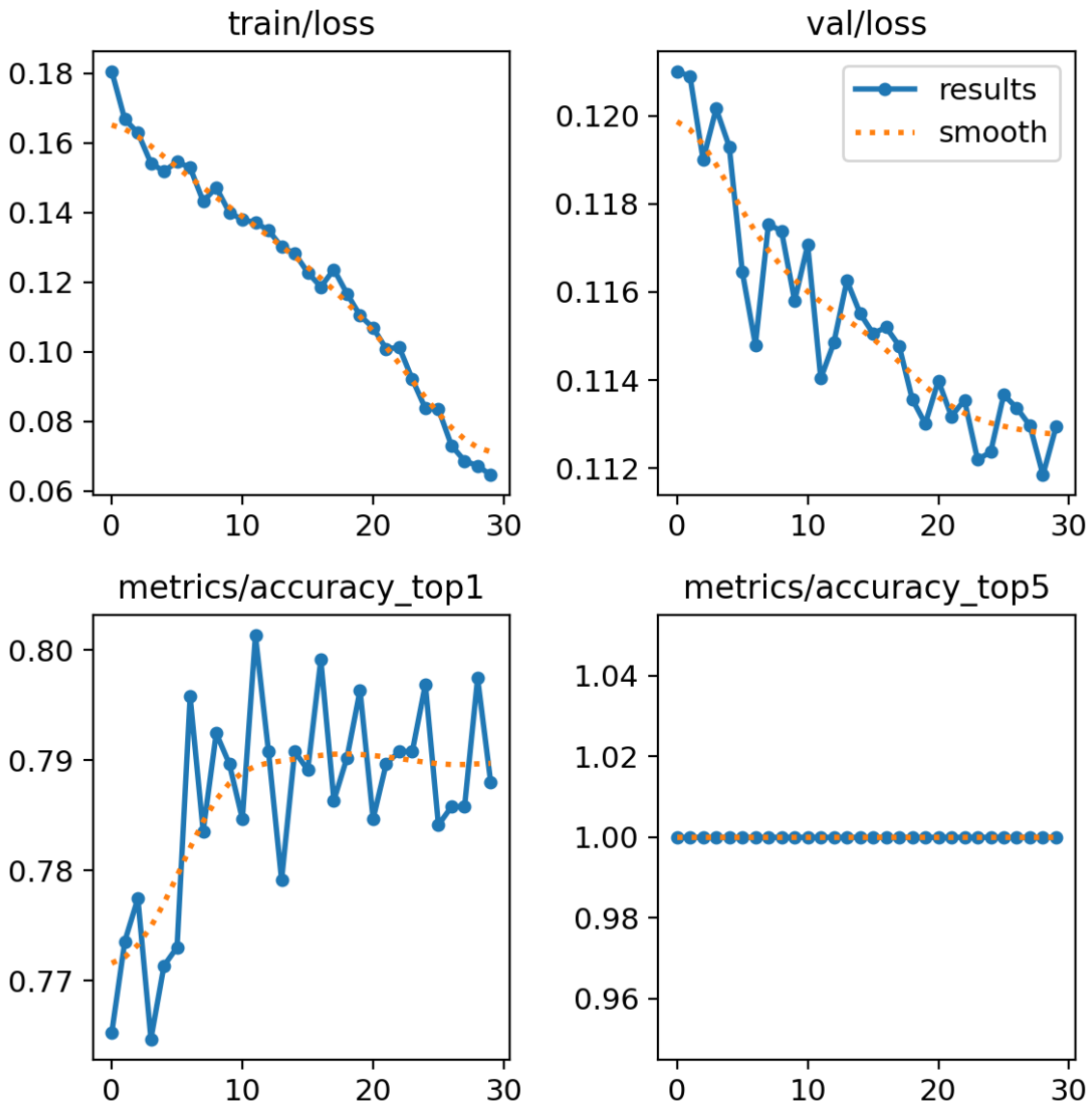


Figura 47. Métricas de precisión del modelo YOLOv8n-cls

La matriz de confusión muestra cómo el modelo ha clasificado las muestras en función de sus clases reales (ver Figura 48).

- Verdaderos Positivos (TP) para la clase "NEUMONIA": 0.80

  Esto significa que el 80% de las muestras que son realmente de la clase "NEUMONIA" fueron clasificadas correctamente como "NEUMONIA" por el modelo.

- Falsos Negativos (FN) para la clase "NEUMONIA": 0.20

  El 20% de las muestras que son realmente de la clase "NEUMONIA" fueron clasificadas incorrectamente como "NORMAL" por el modelo.

- Verdaderos Positivos (TP) para la clase "NORMAL": 0.77

  El 77% de las muestras que son realmente de la clase "NORMAL" fueron clasificadas correctamente como "NORMAL" por el modelo.

- Falsos Negativos (FN) para la clase "NORMAL": 0.23

  El 23% de las muestras que son realmente de la clase "NORMAL" fueron clasificadas incorrectamente como "NEUMONIA" por el modelo.

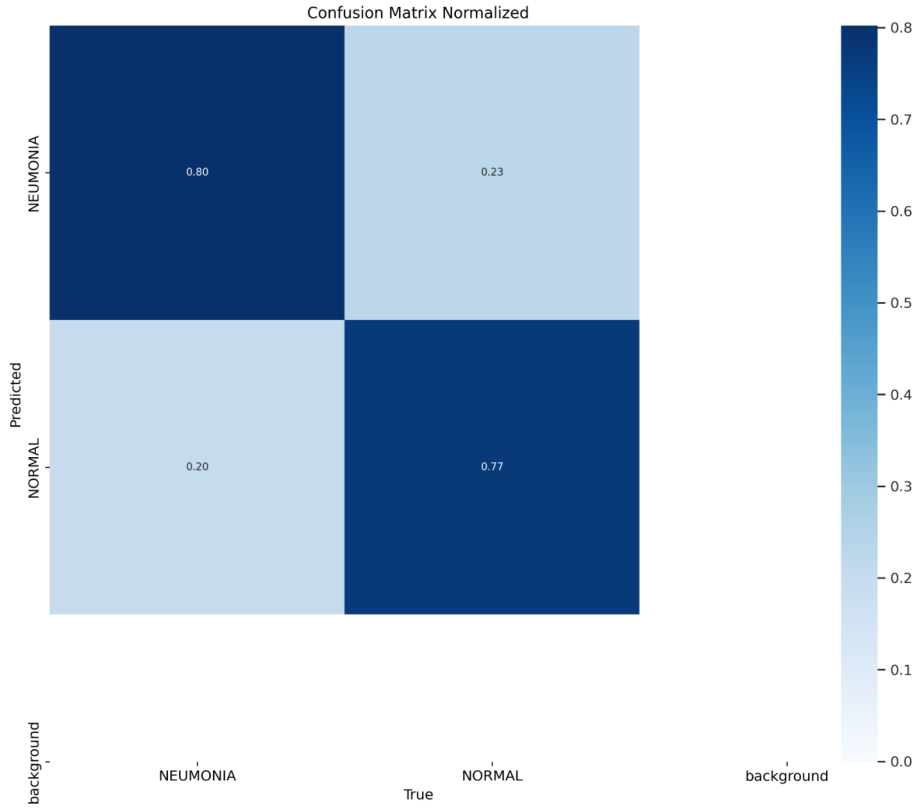


Figura 48. Matriz de confusión del modelo YOLOv8n-cls

Predicciones

Una vez completado el entrenamiento del modelo, ponemos a prueba sus predicciones de clasificación. Observando la Figura 49 , la columna izquierda nos indica las muestras con las etiquetas reales y la columna derecha con las etiquetas predichas. Cabe resaltar que estas predicciones se realizaron sobre el conjunto de test.

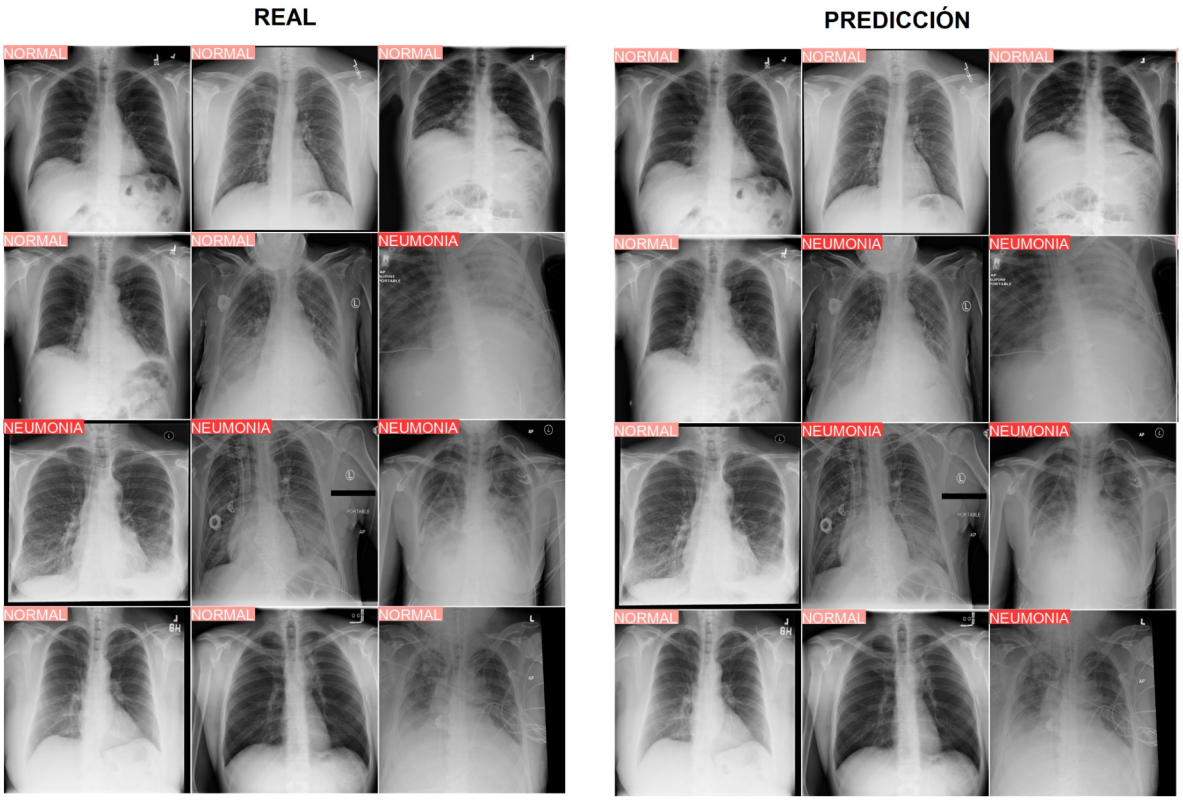


Figura 49. Predicciones del modelo YOLOv8n-cls

4.6. Detección

4.6.1. Preprocesamiento de Datos

Preprocesamiento para la Detección

Como se mencionó en la sección anterior, las imágenes se encuentran bajo el estándar DICOM, por lo tanto, se debe sustraer la imagen contenida del estudio. Para ello se utilizó la librería "pydicom". Al mismo tiempo, se tomaron las coordenadas de los pacientes que sí tenían neumonía (6.012 muestras). La Figura 50 visualiza un ejemplo de imagen con los cuadros delimitadores señalando las ROIs de la enfermedad.

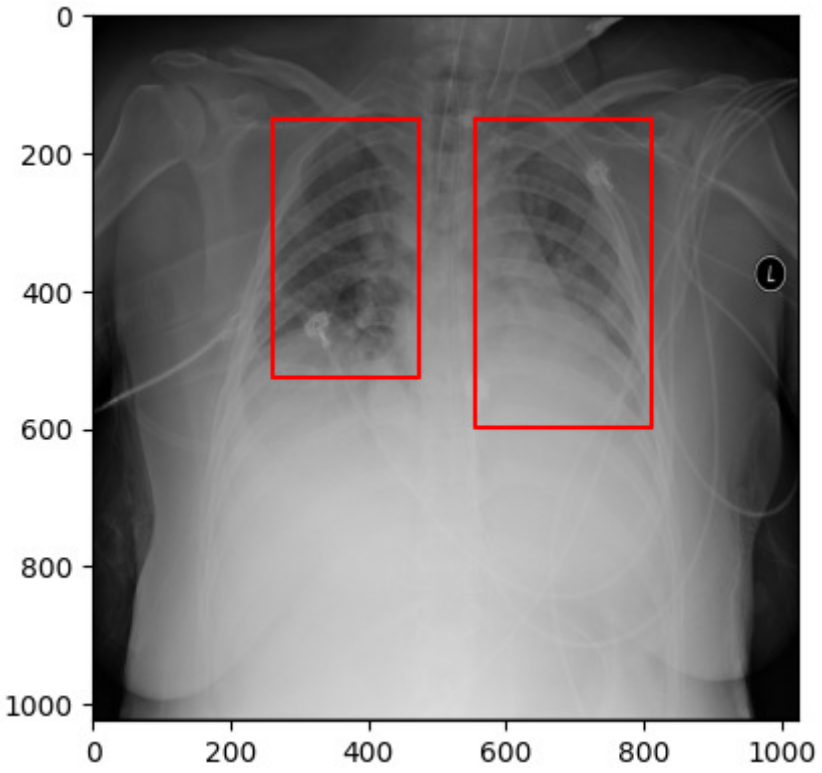


Figura 50. Radiografía con cuadros delimitadores señalando la neumonía.

Se procedió a realizar el preprocesamiento adecuado para ajustar las imágenes al formato requerido por YOLO, incluyendo la obtención y preparación de las etiquetas correspondientes a las coordenadas de las regiones de interés (bounding boxes) asociadas a las imágenes con neumonía.

Dado que solo estamos considerando las muestras con neumonía, nos aseguramos de mantener una división representativa y equilibrada entre los conjuntos:

- *Train (70%)* = 4208 muestras
- *Val (15%)* = 901 muestras
- *Test (15%)* = 901 muestras

El formato YOLO para las etiquetas requiere un archivo de texto por cada imagen en el conjunto de datos. Si una imagen no contiene objetos que necesiten ser detectados, no se requiere un archivo de texto para esa imagen. Cada línea del archivo de texto representa un objeto detectado en la imagen y debe contener la siguiente información:

```
<class_index x_center y_center width height>
```

Donde:

- *class_index*: Es el índice de la clase a la que pertenece el objeto detectado. Los índices comienzan desde 0 y se numeran en orden ascendente para cada clase presente en el conjunto de datos. Como tenemos una clase (neumonia) el índice corresponde a '0'.
- *x_center*: Es la coordenada x del centro del objeto en la imagen, normalizada en un rango de 0 a 1.
- *y_center*: Es la coordenada y del centro del objeto en la imagen, normalizada en un rango de 0 a 1.
- *width*: Es el ancho del objeto, normalizado al rango [0, 1].
- *height*: Es la altura del objeto, normalizada al rango [0, 1].

Al igual que en la segmentación, para poder entrenar el modelo de detección, es necesario indicar a la red un archivo YAML de configuración, en donde se especifique los directorios de: el conjunto de datos, las imágenes de entrenamiento y de validación, el número y el nombre de la clase (1, neumonía). Se tomó la misma configuración que en la segmentación comentada en la sección 4.3.2.

4.6.2. Selección de modelos

Modelo YOLOv8n (Detección)

El modelo YOLOv8n fue seleccionado para abordar la tarea de detección de neumonía en las imágenes de rayos X. A diferencia del modelo de clasificación, YOLOv8n está diseñado para detectar y localizar objetos de interés (en este caso, regiones de neumonía).

4.6.3. Entrenamiento de Modelos

Este modelo se entrenó con los hiperparametros establecidos en la Tabla 8. Se iteraron por un total de 100 épocas, un batch de 21 y se redujo el tamaño de las imágenes a 640x640.

Tabla 8. Hiperparámetros utilizados para entrenar el modelo YOLOv8n

Hiperparámetro	Descripción	YOLOv8n
epoch	Cantidad de iteraciones	100
pretrained	Si usar un modelo preentrenado	True
lr	Tasa de aprendizaje	0.000714

optimizer	Optimizador	Adamw
batch	Tamaño de lote	21
imgsz	Tamaño de imagen	640
device	Dispositivo para entrenar	3xGPU

Observando la Figura 51, la memoria de la GPU utilizada se mantiene relativamente constante a lo largo del entrenamiento, fluctuando entre 1.02 GB y 1.03 GB. También tenemos información acerca de la pérdida promedio en todas las clases (box_loss, cls_loss y dfl_loss), la precisión promedio de las bounding boxes (Box P) y el recall (Box R), el mAP50 y mAP50-95 (mAP a diferentes niveles de IoU).

```
41 DDP info: RANK 0, WORLD_SIZE 3, DEVICE cuda:0
42 TensorBoard: Start with 'tensorboard --logdir runs/detect/train2', view at http://localhost:6006/
43 Overriding model.yaml nc=80 with nc=1
44 [W NNPACK.cpp:64] Could not initialize NNPACK! Reason: Unsupported hardware.
45 [W NNPACK.cpp:64] Could not initialize NNPACK! Reason: Unsupported hardware.
46 [W NNPACK.cpp:64] Could not initialize NNPACK! Reason: Unsupported hardware.
47 Transferred 319/355 items from pretrained weights
48 AMP: running Automatic Mixed Precision (AMP) checks with YOLOv8n...
49 AMP: checks passed ✅
50 train: Scanning /home/minero/Escritorio/RSNAv2/yolov8/train/labels... 4784 images, 0 backgrounds, 0 corrupt: 100%|██████
51 train: New cache created: /home/minero/Escritorio/RSNAv2/yolov8/train/labels.cache
52 val: Scanning /home/minero/Escritorio/RSNAv2/yolov8/val/labels... 1228 images, 0 backgrounds, 0 corrupt: 100%|██████████| 1228/1228 [00:02<00:00, 440.50it/s]
53 val: New cache created: /home/minero/Escritorio/RSNAv2/yolov8/val/labels.cache
54 Plotting labels to runs/detect/train2/labels.jpg...
55 optimizer: AdamW(lr=0.000714, momentum=0.9) with parameter groups 57 weight(decay=0.0), 64 weight(decay=0.00046875), 63 bias(decay=0.0)
56 Image sizes 640 train, 640 val
57 Using 0 dataloader workers
58 Logging results to runs/detect/train2
59 Starting training for 100 epochs...
60
61       Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
62       1/100      1.02G      2.028      2.939      1.985         17        640: 100%|██████████| 319/319 [04:55<00:00,  1.08it/s]
63                  Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 123/123 [01:12<00:00,  1.71it/s]
64                    all       1228       1939      0.438       0.42      0.376       0.12
65
66       Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
67       2/100      1.03G      1.914      2.411      1.828         14        640: 100%|██████████| 319/319 [04:19<00:00,  1.23it/s]
68                  Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 123/123 [01:10<00:00,  1.75it/s]
69                    all       1228       1939       0.41       0.416      0.362       0.12
70
71       Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
72       3/100      1.03G      1.897      2.246        1.8         14        640: 100%|██████████| 319/319 [04:14<00:00,  1.26it/s]
73                  Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 123/123 [01:09<00:00,  1.77it/s]
74                    all       1228       1939      0.352      0.368      0.299      0.108
75
76       Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
77       4/100      1.03G      1.912      2.136       1.82          9        640: 100%|██████████| 319/319 [04:12<00:00,  1.26it/s]
78                  Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 123/123 [01:09<00:00,  1.78it/s]
79                    all       1228       1939      0.324        0.3      0.226      0.072
```

Figura 51. Entrenamiento de YOLOv8n ejecutado en el entorno Local

El entrenamiento de 100 épocas tuvo una duración de aproximadamente 8 horas, 19 minutos y 48 segundos.

Aumento de datos

En el proceso de entrenamiento, se aprovechó la funcionalidad proporcionada por la biblioteca "Albumentations", la cual se encuentra integrada en el repositorio de Ultralytics. Esta biblioteca facilita la aplicación de diversas transformaciones a las imágenes, como rotaciones, traslaciones, ajustes de iluminación, contraste, cambios de tamaño, entre otras. La elección de utilizar esta librería en este modelo específico se fundamenta en la naturaleza de la tarea de detección, que es altamente compleja y requiere alcanzar una precisión óptima en los resultados.

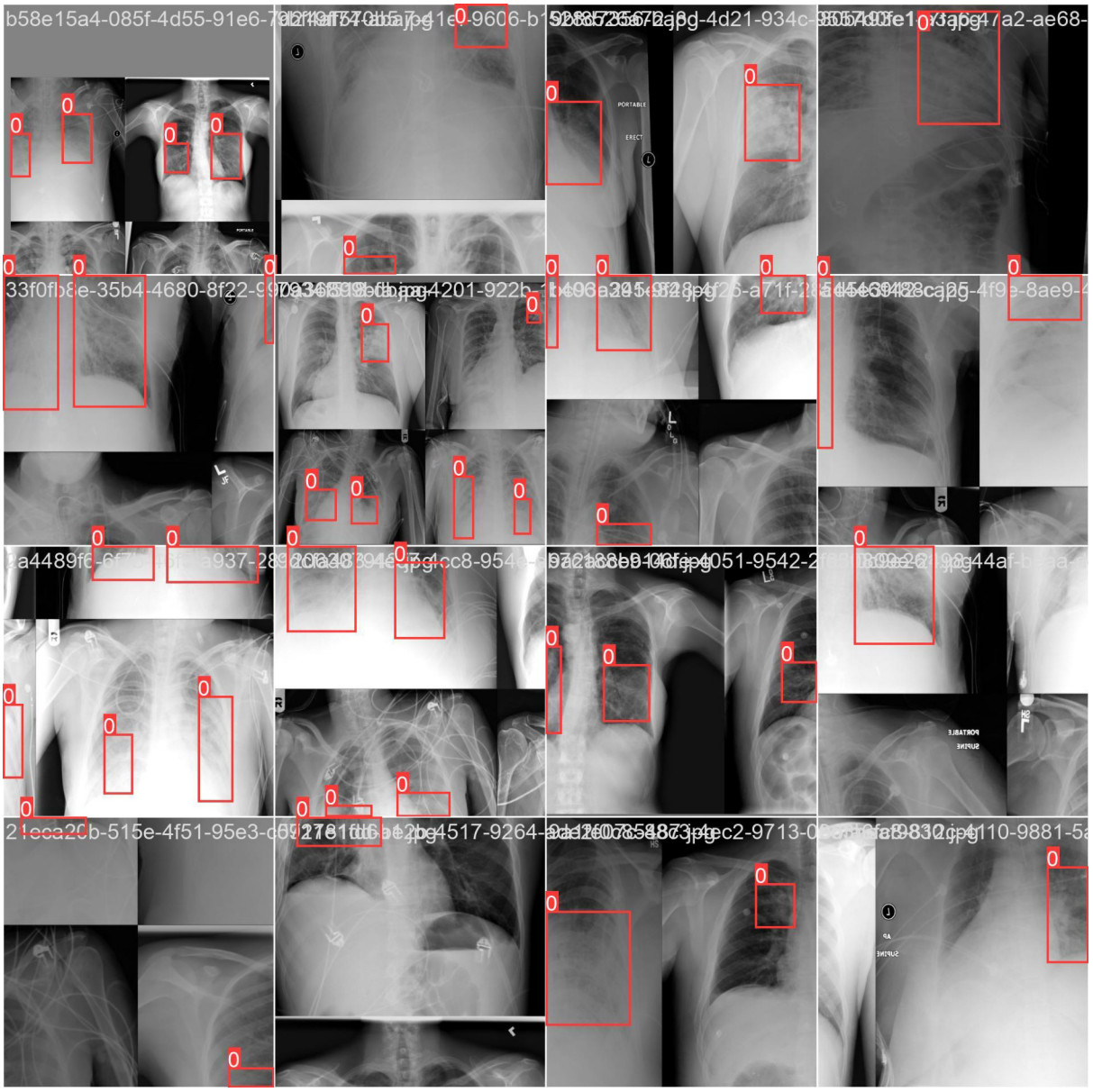


Figura 52. Entrenamiento con aumento de datos

Al incorporar la biblioteca el aumento de datos, se busca que el proceso de entrenamiento se vuelva más versátil y eficiente, ya que las transformaciones se aplican en tiempo real durante el entrenamiento, evitando la necesidad de generar nuevas imágenes transformadas previamente.

4.6.4. Evaluación del Modelo

Análisis y Resultados

Las gráficas relacionadas con las métricas de evaluación del modelo, son las que se encuentran en la Figura 53. De allí podemos destacar lo siguiente:

- *Box_loss, cls_loss y dfl_loss*: Estas pérdidas de entrenamiento están disminuyendo a lo largo de las épocas, lo cual es una señal positiva. Indica que el modelo está aprendiendo a realizar mejores predicciones tanto en las coordenadas de los cuadros delimitadores (box_loss) como en las clasificaciones (cls_loss) y posiblemente en otra tarea adicional (dfl_loss). Un descenso en estas pérdidas indica que el modelo se está ajustando mejor a los datos de entrenamiento. Aunque hay pérdidas que vuelven a elevar los niveles como val/box_loss o val/dfl_loss, como si tuvieran una tendencia a la forma de una parábola, podría deberse al fenómeno del sobre ajuste.

- *Metrics/Precision y Metrics/Recall*: La precisión y el recall para la clase parecen fluctuar en cada época. En general, tener una precisión alta junto con un recall alto es deseable. Sin embargo, si uno de ellos es significativamente más bajo que el otro, podría indicar un desequilibrio. En este caso, los valores alcanzan un máximo de 0.60.

- *Metrics/mAP50 y Metrics/mAP50-95*: Estas métricas representan el promedio de precisión a lo largo de diferentes umbrales de IoU (50% al 95%) para la clase. La mejora en mAP50 indica que el modelo está obteniendo mejores detecciones a medida que aumenta el umbral de superposición requerido para considerar una detección como positiva. No obstante, los valores de mAP50 y mAP50-95 no parecen superar los 0.55 y los 0.2, respectivamente.

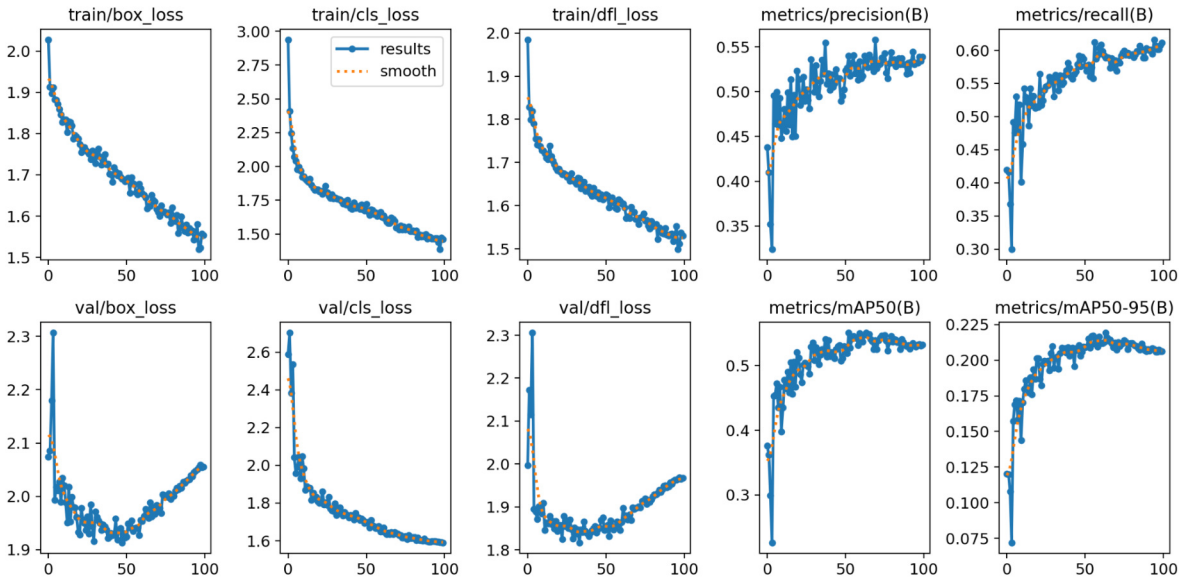


Figura 53. Métricas de evaluación del modelo YOLOv8n

Asimismo, tenemos los resultados de las curvas de precisión y recuperación:

- *Recall-Confidence Curve*: Esta curva muestra cómo cambia el Recall (sensibilidad) del modelo a medida que se ajusta el umbral de confianza. Al disminuir el umbral de confianza, es probable que el modelo clasifique más muestras como positivas, lo que aumentará los verdaderos positivos (TP) y, potencialmente, también los falsos positivos (FP), lo que puede llevar a un aumento en el recall. Sin embargo, también puede resultar en una disminución de la precisión.

- *Precision-Confidence Curve*: Esta curva muestra cómo cambia la Precision del modelo a medida que se varía el umbral de confianza. Al aumentar el umbral de confianza, es probable que el modelo clasifique menos muestras como positivas, lo que disminuirá tanto los verdaderos positivos (TP) como los falsos positivos (FP). Esto puede llevar a un aumento en la precisión, pero también puede disminuir el recall.

- *Precision-Recall Curve*: Esta curva muestra la relación entre la precisión y el recall del modelo a través de diferentes umbrales de confianza. Cada punto en la curva representa una combinación de precisión y recall.

- F1-Confidence Curve: Esta curva muestra cómo cambia el valor F1-score del modelo a medida que se ajusta el umbral de confianza. El valor máximo alcanzado es de 0.57.

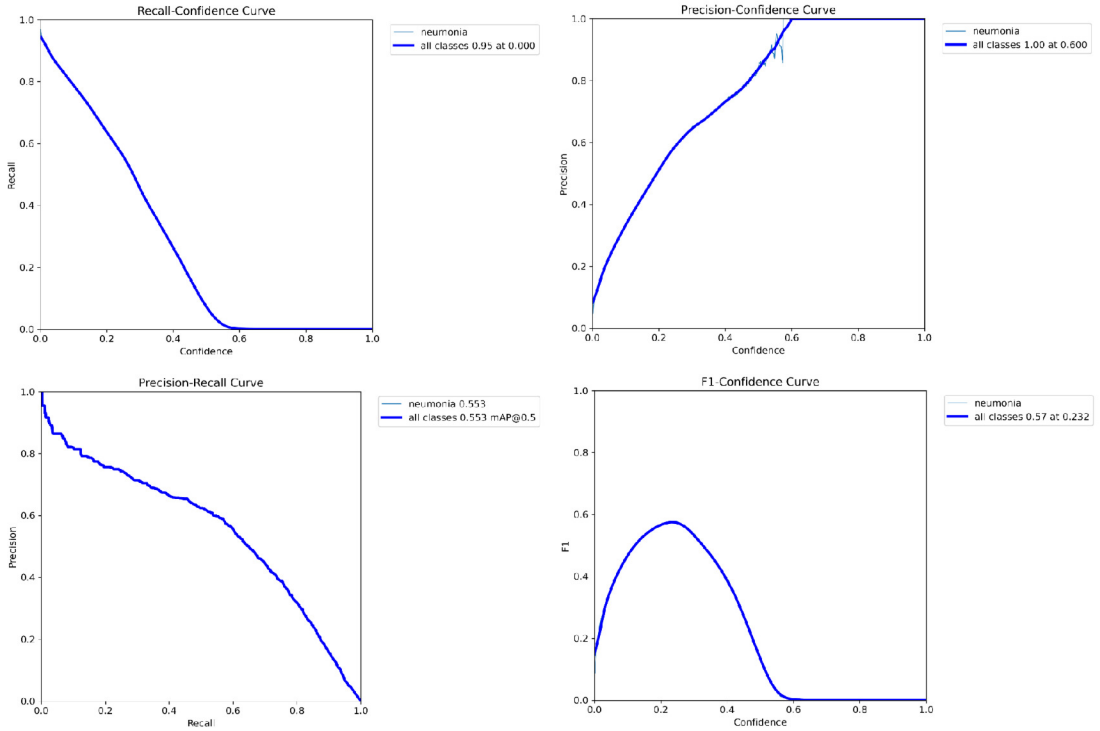


Figura 54. Curvas de precisión y recuperación del modelo YOLOv8n

Continuando con el análisis, la matriz de confusión nos brinda información valiosa sobre el desempeño de la red neuronal en para la detección en las muestras entre las clases "NEUMONIA" y "background" (fondo o no NEUMONIA).

De todas las muestras que realmente pertenecen a la clase que establecimos como "NEUMONIA", la red neuronal clasificó correctamente el 59% de ellas. Esto se refleja en un alto valor de "verdaderos positivos" en la matriz de confusión, lo que indica que la red identificó correctamente una parte significativa de los casos positivos de NEUMONIA. Sin embargo, también observamos que el 41% de las muestras fueron incorrectamente clasificadas por la red como "background". Este fenómeno se refleja en la sección de "falsos negativos" de la matriz de confusión, lo que significa que hubo casos de NEUMONIA que la red no logró detectar adecuadamente. (ver Figura 55).

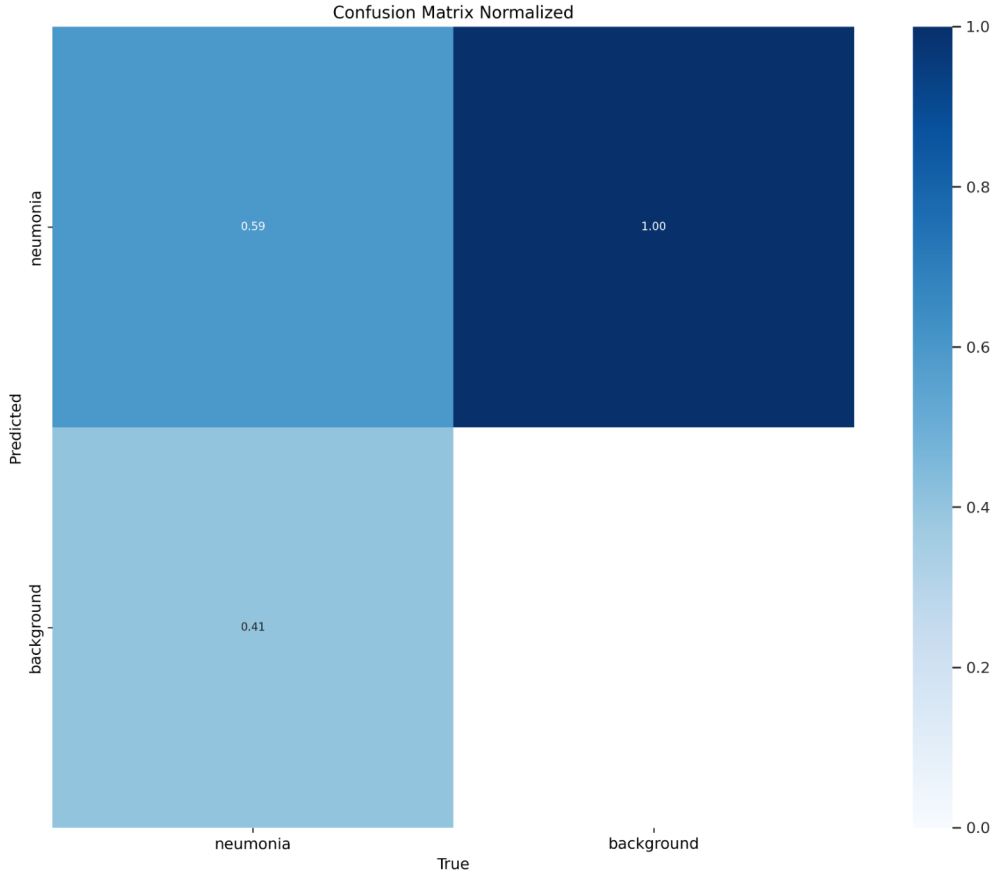


Figura 55. Matriz de confusión del modelo YOLOv8n

Predicción

Finalmente, para evaluar el rendimiento y la capacidad predictiva de nuestro modelo entrenado de detección de neumonía, procedimos a realizar pruebas utilizando un conjunto de muestras que nunca antes habían sido utilizadas durante el proceso de entrenamiento. Estas muestras se seleccionaron de manera aleatoria para garantizar la imparcialidad de la evaluación y evitar cualquier sesgo en los resultados.

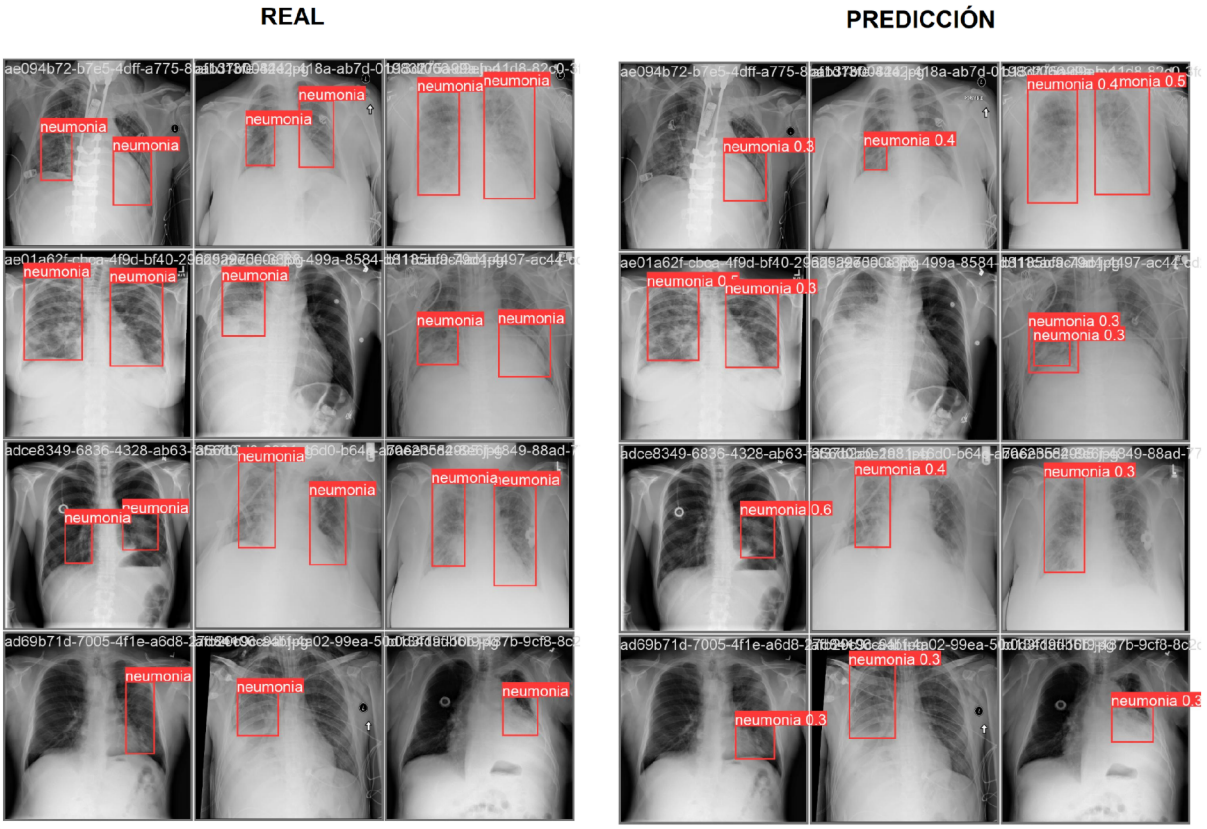


Figura 56. Predicciones del modelo YOLOv8n

4.7. Analisis General

El análisis general de los resultados obtenidos al aplicar las tres tareas de visión artificial (Segmentación, Clasificación y Detección) arroja una visión integral del desempeño de los modelos utilizados.

Primeramente, el análisis comparativo entre las dos arquitecturas, U-Net y YOLO, para la segmentación de pulmones en radiografías del tórax arroja resultados interesantes. U-Net muestra un excelente rendimiento con valores de precisión, recuperación y coeficiente de Dice cercanos a 98%. Estos valores indican que U-Net logra una alta capacidad para identificar correctamente los píxeles que pertenecen al pulmón en las radiografías. En contraparte, YOLO mostró una probabilidad de detección de pulmones del 97%.

En general, ambos modelos han demostrado un buen desempeño en la tarea de segmentación de pulmones, pero con enfoques distintos. U-Net es una red diseñada específicamente para la segmentación semántica, en cambio, YOLO es una arquitectura más versátil que se enfoca en la detección de objetos, lo que le permite localizar los pulmones en la imagen y asignarles una etiqueta. En esta última versión de YOLO los desarrolladores se han enfocado más en el trabajo de segmentar objetos y es por ello que se aproximan a campos más complejos como la medicina.

A su vez, estas dos arquitecturas presentan enfoques diferentes en cuanto a su implementación y facilidad de uso. En el caso de U-Net, es una red neuronal que uno debe programar por su cuenta, lo que implica realizar la implementación de la arquitectura y todas sus funciones desde cero. Esto puede ser un desafío técnico para aquellos que no tienen experiencia en programación avanzada y en la construcción de modelos de aprendizaje profundo. Para utilizar U-Net, se requiere un conocimiento sólido en programación y en el manejo de librerías de Python, lo que podría dificultar su adopción y uso, especialmente para aquellos usuarios que no son expertos en el área. Por su parte, YOLO tiene la ventaja de estar diseñado por Ultralytics, lo que significa que esta arquitectura viene pre-implementada y lista para su uso. En otras palabras, el trabajo de programación y configuración ya está hecho por el equipo de Ultralytics, lo que facilita en gran medida su utilización. Al ser una solución predefinida, YOLO abstrae a los usuarios de la necesidad de generar código adicional para su funcionamiento. Esta ventaja es especialmente valiosa para aquellos que desean utilizar la arquitectura sin la necesidad de sumergirse en los detalles de su programación y simplemente enfocarse en la aplicación y los resultados que proporciona.

Continuando con las tareas de Clasificación y Detección de neumonía, en la tarea de Clasificación, se alcanzó una precisión del 80%, lo que muestra una buena capacidad del modelo para asignar la etiqueta correcta a las imágenes en función de sus características. Por otro lado, en la tarea de Detección, se obtuvo una precisión del 60%, lo que sugiere que aún hay margen para mejorar las predicciones. Para mejorar la precisión en la detección de neumonía, se aplicó la técnica de aumento de datos con la librería "Albumentations", aplicando transformaciones a las muestras durante el entrenamiento, lo que indicó una búsqueda activa de optimización.

Es importante destacar que las pruebas de segmentación se realizaron en un entorno de nube, específicamente en Google Colab, debido a la cantidad limitada de muestras en el dataset (aproximadamente 1.600 muestras). Por otro lado, las tareas de detección y clasificación se llevaron a cabo localmente, aprovechando el poder de 3 GPUs mediante la técnica "DataParallel", y se utilizaron aproximadamente 12.000 muestras para estas tareas. Esta diferencia en la cantidad de muestras utilizadas en cada entorno puede haber influido en los resultados obtenidos, y se sugiere que en futuras investigaciones se pueda contar con un dataset más amplio y diverso para mejorar aún más el rendimiento de los modelos.

En definitiva, las tareas de Segmentación y Clasificación arrojaron resultados satisfactorios, lo que indica que los modelos empleados han aprendido a segmentar y clasificar adecuadamente los pulmones y detectar neumonía en las imágenes. Sin embargo, en

la tarea de Detección, se identificó la necesidad de realizar ajustes para mejorar las predicciones, como aumentar el entrenamiento, probar otros modelos de detección o utilizar un dataset con muestras más precisas para adaptar la red y mejorar su capacidad de detección.

CAPÍTULO V

Conclusiones

El desarrollo del trabajo se organizó por medio de etapas fundamentales: investigación y capacitación, diseño, desarrollo e implementación y, obtención de resultados y análisis. Las conclusiones presentadas a continuación se basan en el progreso y los hallazgos logrados en cada una de estas etapas.

En primer lugar, se llevó a cabo una rigurosa investigación y capacitación sobre los algoritmos de aprendizaje automático y las redes neuronales artificiales. El objetivo principal era adquirir un sólido entendimiento de cómo se implementan estas técnicas, su funcionamiento interno y las diversas variantes que existen. Se descubrió que las redes neuronales son herramientas sumamente poderosas para resolver problemas complejos, lo que permite mejorar significativamente el rendimiento en comparación con las técnicas convencionales. Esta etapa de investigación sentó las bases teóricas para abordar las tareas de visión artificial planteadas en el trabajo.

En segundo lugar, la siguiente etapa se centró en el diseño, desarrollo e implementación de las arquitecturas de redes neuronales adecuadas para las tareas de segmentación, clasificación y detección en imágenes médicas de formato DICOM. Se seleccionaron cuidadosamente las arquitecturas más adecuadas para cada tarea y se procedió a la implementación de los modelos. Durante esta etapa, se enfrentaron desafíos técnicos y se realizaron ajustes para optimizar el rendimiento de los modelos en cada tarea específica. La implementación de estas redes neuronales fue un proceso meticuloso que permitió obtener modelos precisos y efectivos en la localización de regiones de interés y la clasificación de neumonía.

En tercer lugar, se obtuvieron los resultados mediante la evaluación y validación de los modelos desarrollados. Se realizaron pruebas utilizando diferentes conjuntos de datos para medir el rendimiento y la eficacia de los modelos en cada tarea. Los resultados obtenidos fueron alentadores, mostrando un excelente desempeño en la tarea de segmentación de pulmones y una precisión satisfactoria en la clasificación de neumonía. Sin embargo, se identificó que la tarea de detección requiere de mejoras para alcanzar resultados óptimos.

En conclusión, el trabajo ha logrado cumplir satisfactoriamente con los objetivos planteados, demostrando la efectividad de las técnicas de aprendizaje profundo aplicadas en imágenes médicas. La investigación y capacitación en el área de visión artificial permitieron comprender y aplicar adecuadamente los algoritmos y arquitecturas de redes neuronales para abordar problemas médicos complejos. Los resultados obtenidos son alentadores y abren nuevas oportunidades para futuras investigaciones y aplicaciones en el campo de la medicina y el procesamiento automatizado de imágenes médicas

Bibliografía

[1] A. Bhan, S. Kapoor and M. Gulati, "Diagnosing Parkinson's disease in Early Stages using Image Enhancement, ROI Extraction and Deep Learning Algorithms," 2021 2nd International Conference on Intelligent Engineering and Management (ICIEM), London, United Kingdom, 2021, pp. 521-525, doi: 10.1109/ICIEM51511.2021.9445381.

[2] Abdelmaguid, E., Huang, J., Kenchareddy, S., Singla, D., Wilke, L., Nguyen, M.H., & Altintas, I. (2018). Left Ventricle Segmentation and Volume Estimation on Cardiac MRI using Deep Learning. ArXiv, abs/1809.06247.

[3] Abdulkader Helwan, Dilber Uzun Ozsahin (2017). "Sliding Window Based Machine Learning System for the Left Ventricle Localization in MR Cardiac Images". Applied Computational Intelligence and Soft Computing, vol. 2017, Article ID 3048181, 9 páginas. https://doi.org/10.1155/2017/3048181.

[4] Al-Antari, M. A., Al-Masni, M. A., Choi, M. T., Han, S. M., & Kim, T. S. (2018). A fully integrated computer-aided diagnosis system for digital X-ray mammograms via deep learning detection, segmentation, and classification. International journal of medical informatics, 117, 44–54. https://doi.org/10.1016/j.ijmedinf.2018.06.003

[5] Alkadi, R., Taher, F. D., El-Baz, A., & Werghi, N. (2018). "A Deep Learning-Based Approach for the Detection and Localization of Prostate Cancer in T2 Magnetic Resonance Images". Journal of Digital Imaging, 32. https://doi.org/10.1007/s10278-018-0160-1.

[6] B. Li, G. Kang, K. Cheng and N. Zhang, "Attention-Guided Convolutional Neural Network for Detecting Pneumonia on Chest X-Rays," 2019 41st Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC), Berlin, Germany, 2019, pp. 4851-4854, doi: 10.1109/EMBC.2019.8857277.

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

[8] Dev, C., Kumar, K., Palathil, A., Anjali, T., & Panicker, V. (2019). "Machine Learning Based Approach for Detection of Lung Cancer in DICOM CT Image". En: Proceedings of the International Conference on Smart Innovations in Communications and Computational Sciences, 15. https://doi.org/10.1007/978-981-13-5934-7_15.

[9] Du, M., Wu, X., Ye, Y., Fang, S., Zhang, H., & Chen, M. (2022). A Combined Approach for Accurate and Accelerated Teeth Detection on Cone Beam CT Images.

Diagnostics (Basel, Switzerland), 12(7), 1679. https://doi.org/10.3390/diagnostics12071679

[10] G. Hamed, M. Marey, S. E. Amin and M. F. Tolba, "Automated Breast Cancer Detection and Classification in Full Field Digital Mammograms Using Two Full and Cropped Detection Paths Approach," in IEEE Access, vol. 9, pp. 116898-116913, 2021, doi: 10.1109/ACCESS.2021.3105924.

[11] Good Audience. (s. f.). YOLO Object Detection Walkthrough for the RSNA Pneumonia Detection Challenge. Recuperado de https://blog.goodaudience.com/yolo-object-detection-walkthrough-for-the-rsna-pneumonia-detection-challenge-123ec9a9adf2

[12] Google Scholar, from https://scholar.google.com

[13] Guo, W. L., Geng, A. K., Geng, C., Wang, J., & Dai, Y. K. (2022). Combination of UNet++ and ResNeSt to classify chronic inflammation of the choledochal cystic wall in patients with pancreaticobiliary maljunction. The British journal of radiology, 95(1135), 20201189. https://doi.org/10.1259/bjr.20201189

[14] H. R. Roth et al., "Improving Computer-Aided Detection Using Convolutional Neural Networks and Random View Aggregation," in IEEE Transactions on Medical Imaging, vol. 35, no. 5, pp. 1170-1181, May 2016, doi: 10.1109/TMI.2015.2482920.

[15] Hugging Face. (s. f.). Performance en entrenamiento en múltiples GPU. Recuperado de https://huggingface.co/docs/transformers/v4.23.1/en/perf_train_gpu_many

[16] IEEE Xplore, from https://ieeexplore.ieee.org/Xplore/home.jsp

[17] Jaeger, S., Candemir, S., Antani, S., Wáng, Y. X., Lu, P. X., & Thoma, G. (2014). Two public chest X-ray datasets for computer-aided screening of pulmonary diseases. Quantitative imaging in medicine and surgery, 4(6), 475–477. https://doi.org/10.3978/j.issn.2223-4292.2014.11.20

[18] Jocher, G., Chaurasia, A., & Qiu, J. (2023). YOLO by Ultralytics (Version 8.0.0) [Computer software]. https://github.com/ultralytics/ultralytics

[19] Jocher, G. (2020). YOLOv5 by Ultralytics (Version 7.0) [Computer software]. https://doi.org/10.5281/zenodo.3908559

[20] K. Roy et al., "A Comparative study of Lung Cancer detection using supervised neural network," 2019 International Conference on Opto-Electronics and Applied Optics (Optronix), Kolkata, India, 2019, pp. 1-5, doi: 10.1109/OPTRONIX.2019.8862326.

[21] Kumar, V., & Bakariya, B. (2021). Classification of malignant lung cancer using deep learning. Journal of medical engineering & technology, 45(2), 85–93. https://doi.org/10.1080/03091902.2020.1853837

[22] Loey, M., Manogaran, G., Taha, M.H., & Khalifa, N.M. (2020). Fighting against COVID-19: A novel deep learning model based on YOLO-v2 with ResNet-50 for medical face mask detection. Sustainable Cities and Society, 65, 102600 - 102600.

[23] Mushtaq, M., Akram, M.U., Alghamdi, N.S., Fatima, J., & Masood, R.F. (2022). Localization and Edge-Based Segmentation of Lumbar Spine Vertebrae to Identify the Deformities Using Deep Learning Models. Sensors (Basel, Switzerland), 22.

[24] Nemoto, T., Futakami, N., Yagi, M., Kumabe, A., Takeda, A., Kunieda, E., & Shigematsu, N. (2020). Efficacy evaluation of 2D, 3D U-Net semantic segmentation and atlas-based segmentation of normal lungs excluding the trachea and main bronchi. Journal of Radiation Research, 61, 257 - 264.

[25] Palanisamy, Kalavathi & Dhavapandiammal, A. (2016). "Segmentación de Tumor Pulmonar en Imágenes de Tomografía Computarizada utilizando Algoritmos FA-FCM". IOSR Journal of Computer Engineering, 18, 74-79. https://doi.org/10.9790/0661-1805047479.

[26] Pub Med, from https://pubmed.ncbi.nlm.nih.gov

[27] Radiological Society of North America (RSNA). (s. f.). RSNA Pneumonia Detection Challenge 2018. Recuperado de https://www.rsna.org/education/ai-resources-and-training/ai-image-challenge/rsna-pneumonia-detection-challenge-2018

[28] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (n.d.). You Only Look Once: Unified, Real-Time Object Detection. University of Washington, Allen Institute for AI, Facebook AI Research. Recuperado de http://pjreddie.com/yolo/

[29] Ronneberger, O., Fischer, P., & Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. ArXiv, abs/1505.04597.

[30] Ronneberger, O., Fischer, P., & Brox, T. (n.d.). U-Net: Convolutional Networks for Biomedical Image Segmentation. Computer Science Department and BIOSS Centre for Biological Signalling Studies, University of Freiburg, Germany. Recuperado de http://lmb.informatik.uni-freiburg.de/

[31] S. Kim, Y. Ji and K. -B. Lee, "An Effective Sign Language Learning with Object Detection Based ROI Segmentation," 2018 Second IEEE International Conference on Robotic Computing (IRC), Laguna Hills, CA, USA, 2018, pp. 330-333, doi: 10.1109/IRC.2018.00069.

[32] Semantic Scholar, from https://www.semanticscholar.org

[33] Sendra-Balcells, C., Salvador, R., Pedro, J. B., Biagi, M. C., Aubinet, C., Manor, B., Thibaut, A., Laureys, S., Lekadir, K., & Ruffini, G. (2020). "Convolutional neural network MRI segmentation for fast and robust optimization of transcranial electrical current stimulation of the human brain". bioRxiv, 2020.01.29.924985. https://doi.org/10.1101/2020.01.29.924985.

[34] Sengun, K.E., Cetin, Y.T., Güzel, M.S., Can, S., & Bostanci, E. (2021). Automatic Liver Segmentation from CT Images Using Deep Learning Algorithms: A Comparative Study. ArXiv, abs/2101.09987.

[35] Shelatkar, T., Urvashi, D., Shorfuzzaman, M., Alsufyani, A., & Lakshmanna, K. (2022). Diagnosis of Brain Tumor Using Light Weight Deep Learning Model with Fine-Tuning Approach. Computational and Mathematical Methods in Medicine, 2022.

[36] Yao, S., Chen, Y., Tian, X., Jiang, R., & Ma, S. (2020). An Improved Algorithm for Detecting Pneumonia Based on YOLOv3. Applied Sciences.

ANEXO A

Preprocesamiento de Datos

En este anexo "A", se presentarán las líneas de código utilizadas en el preprocesamiento de datos de ambos datasets empleados en los entrenamientos de los modelos. Estos datasets incluyen el conjunto de datos de "Pulmonary Chest X-Ray Abnormalities" y el "RSNA Pneumonia Detection Challenge". El propósito de este anexo es mostrar de manera detallada las etapas de preparación y transformación de los datos antes de ser utilizados en el proceso de entrenamiento de los modelos de segmentación, clasificación y detección.

Librerías

Estas son las librerías que fueron utilizadas para llevar a cabo el preprocesamiento de los datos:

```
import numpy as np
import pandas as pd
import os
import scipy.ndimage
import matplotlib.pyplot as plt
import pathlib
import math
import shutil
import sys
import seaborn as sns
import glob
import cv2
from sklearn.model_selection import train_test_split
from tqdm import tqdm

# Libreria para manipular imagenes DICOM (requiere instalación)
!pip install pydicom
import pydicom
```

Dataset: Pulmonary Chest X-Ray Abnormalities

El siguiente código realiza el preprocesamiento de datos para el dataset "Pulmonary Chest X-Ray Abnormalities". El objetivo es preparar las imágenes y sus respectivas máscaras para ser utilizadas en el entrenamiento de modelos de segmentación. El código recorre los directorios que contienen las imágenes originales y sus máscaras. Luego, realiza las siguientes operaciones:

1. Se crean dos directorios de salida, uno para las imágenes procesadas y otro para las máscaras procesadas, asegurándose de que los directorios existan o los crea en caso contrario.
2. Se obtiene una lista de los nombres de archivo en el directorio de imágenes.
3. Luego, se itera sobre cada nombre de archivo en la lista y se obtiene el nombre base del archivo sin la extensión.
4. Se comprueba si existe una máscara correspondiente para la imagen actual. Si existe, continúa con la carga de la imagen y la máscara utilizando la biblioteca "Pillow" (PIL). Además, convierte las imágenes a escala de grises para simplificar el procesamiento.
5. Se cambia el tamaño de ambas imágenes a 512x512 píxeles utilizando el método de interpolación 'Image.ANTIALIAS' para evitar pérdidas de calidad.
6. Se convierte las imágenes a arrays de numpy y normaliza los valores de los píxeles para que estén en el rango de 0 a 1.
7. Por último, se guardan las matrices numpy resultantes en los directorios de salida con el nombre del archivo original y una extensión ".npy" para indicar que son archivos de numpy.

```
img_dir = "/content/CXR/CXR_png"
mask_dir = "/content/CXR/CXR_mask"
output_img_dir = "/content/imgs"
output_mask_dir = "/content/masks"

# Crear directorios de salida si no existen
os.makedirs(output_img_dir, exist_ok=True)
os.makedirs(output_mask_dir, exist_ok=True)

# Obtener la lista de nombres de archivos en el directorio de imágenes
img_files = os.listdir(img_dir)

# Iterar sobre los nombres de archivos
```

```
for img_file in img_files:
    # Obtener el nombre base sin la extensión
    img_name = os.path.splitext(img_file)[0]

    # Comprobar si existe la máscara correspondiente
    mask_file = img_name + "_mask.png"
    if mask_file in os.listdir(mask_dir):
        # Cargar la imagen y la máscara utilizando Pillow
        img_path = os.path.join(img_dir, img_file)
        mask_path = os.path.join(mask_dir, mask_file)
          img = Image.open(img_path).convert('L')  # Convertir a escala
de grises
         mask = Image.open(mask_path).convert('L')  # Convertir a escala
de grises

         # Cambiar el tamaño a 512x512
        img = img.resize((512, 512), Image.ANTIALIAS)
        mask = mask.resize((512, 512), Image.ANTIALIAS)

        # Convertir la imagen y la máscara en arrays de numpy
        img_array = np.array(img, dtype=np.float32) / 255.0
        mask_array = np.array(mask, dtype=np.float32) / 255.0

        # Guardar la matriz numpy en el directorio de salida
             output_img_path = os.path.join(output_img_dir, img_name +
".npy")
           output_mask_path = os.path.join(output_mask_dir, img_name +
"_mask.npy")
        np.save(output_img_path, img_array)
        np.save(output_mask_path, mask_array)
```

La siguiente función crea una figura con cuatro subfiguras en una cuadrícula horizontal. Cada subfigura muestra diferentes aspectos relacionados con el procesamiento de las imágenes y máscaras.

```
# Crear una figura y una cuadrícula de subfiguras
fig, axs = plt.subplots(1, 4, figsize=(40, 10))

# Cargar y mostrar la imagen original en la primera subfigura
imagen_original = Image.open('/content/CRX/CXR_png/CHNCXR_0003_0.png')
axs[0].imshow(imagen_original)
```

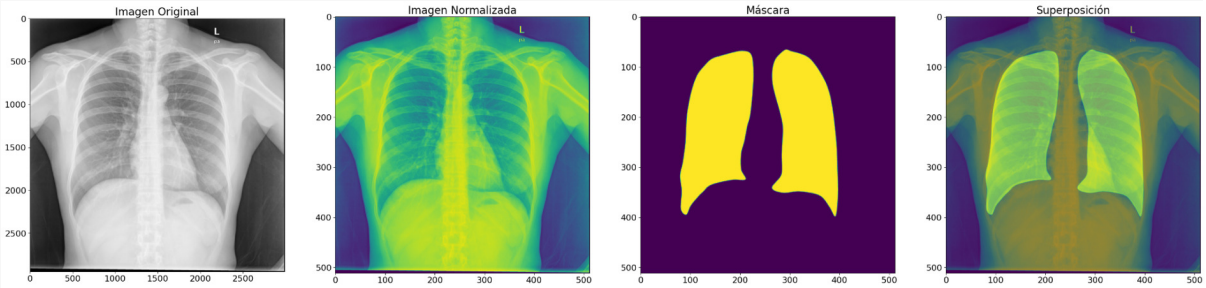
```
axs[0].set_title('Imagen Original', fontsize=20)
axs[0].tick_params(axis='both', labelsize=16)

# Cargar y mostrar la imagen normalizada en la segunda subfigura
img = np.load(imgs[2])
axs[1].imshow(img)
axs[1].set_title('Imagen Normalizada', fontsize=20)
axs[1].tick_params(axis='both', labelsize=16)

# Cargar y mostrar la máscara en la tercera subfigura
mask = np.load(masks[2])
axs[2].imshow(mask)
axs[2].set_title('Máscara', fontsize=20)
axs[2].tick_params(axis='both', labelsize=16)

# Mostrar la superposición de imagen y máscara en la cuarta subfigura
axs[3].imshow(img)
axs[3].imshow(mask, alpha=0.4)
axs[3].set_title('Superposición', fontsize=20)
axs[3].tick_params(axis='both', labelsize=16)

plt.show()
```



A su vez, podemos ver la superposición de múltiples muestras entre las imágenes originales y sus máscaras.

```
# Número de filas y columnas en la matriz
num_rows = 4
num_cols = 5

# Crear una figura con una matriz de subfiguras
```

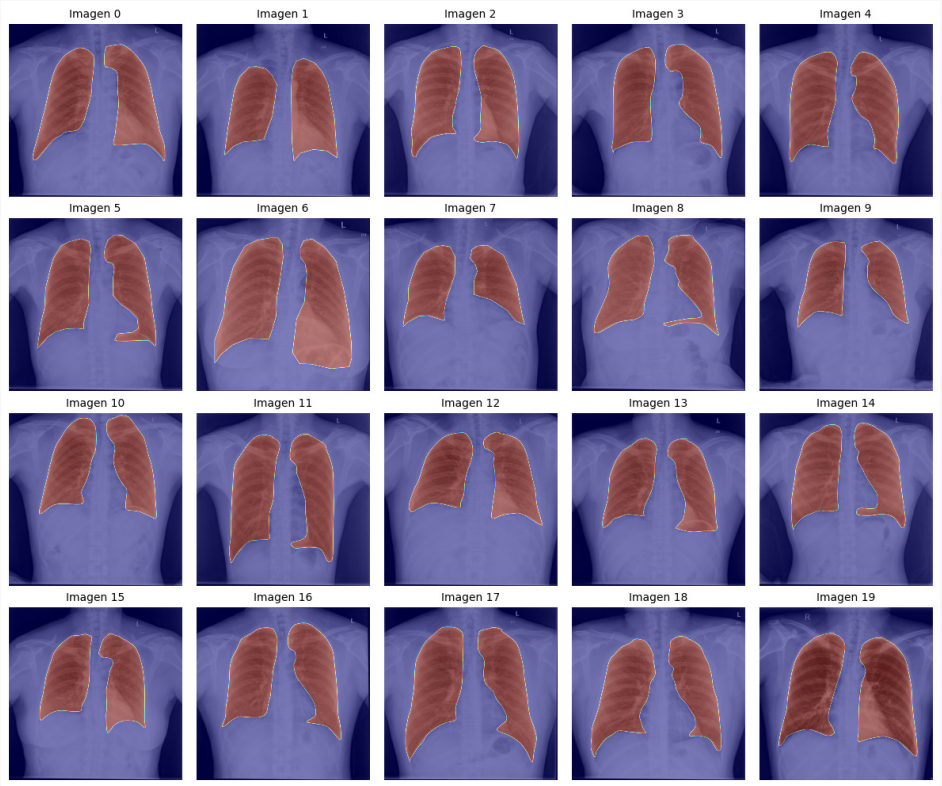
```
fig, axs = plt.subplots(num_rows, num_cols, figsize=(12, 10))

# Recorrer las listas de imágenes y máscaras y mostrarlas en subfiguras
for i in range(num_rows):
    for j in range(num_cols):
        # Índice de la imagen dentro de las listas
        idx = i * num_cols + j

        # Cargar la imagen y la máscara
        img = np.load(imgs[idx])
        mask = np.load(masks[idx])

        # Mostrar la imagen en la subfigura correspondiente
        axs[i, j].imshow(img, cmap='gray')
        axs[i, j].imshow(mask, cmap='jet', alpha=0.5)
        axs[i, j].set_title(f'Imagen {idx}', fontsize=10)
        axs[i, j].axis('off')

# Ajustar el espacio entre subfiguras
plt.tight_layout()
plt.show()
```



Dataset: RSNA Pneumonia Detection Challenge

Para preprocesar los datos de este conjunto, dividimos los mismos para detectar y clasificar.

En primer lugar, se necesita convertir las imágenes DICOM en un formato que YOLO pueda trabajarlas. Para ello nos centramos en la conversión de las imágenes y sus respectivas coordenadas para la detección.

- La función 'save_img_from_dcm' toma la ruta al directorio de imágenes DICOM dcm_dir, la ruta al directorio de imágenes de salida img_dir y el ID del paciente y guarda una imagen JPEG en el directorio img_dir con el nombre del ID del paciente. La imagen JPEG se crea a partir de la imagen DICOM original y se guarda en el formato RGB de 3 canales.
- La función 'save_label_from_dcm' toma la ruta al directorio de etiquetas 'label_dir', el ID del paciente y una fila de datos de anotación RSNA, y guarda una etiqueta de detección de objetos en formato YOLOv8 en un archivo de texto con el nombre del ID del paciente en el directorio 'label_dir'.
- La función 'save_yolov_data_from_rsna' toma la ruta al directorio de imágenes DICOM 'dcm_dir', la ruta al directorio de imágenes de salida 'img_dir', la ruta al directorio de etiquetas de salida 'label_dir' y un objeto 'annots' que contiene las anotaciones RSNA, y genera un conjunto de datos YOLOv8 en los directorios de salida.

```
def save_img_from_dcm(dcm_dir, img_dir, patient_id):
    img_fp = os.path.join(img_dir, "{}.jpg".format(patient_id))
    if os.path.exists(img_fp):
        return
    dcm_fp = os.path.join(dcm_dir, "{}.dcm".format(patient_id))
    img_1ch = pydicom.read_file(dcm_fp).pixel_array
    img_3ch = np.stack([img_1ch]*3, -1)

    img_fp = os.path.join(img_dir, "{}.jpg".format(patient_id))
    cv2.imwrite(img_fp, img_3ch)

def save_label_from_dcm(label_dir, patient_id, row=None):
    # tamaño de las imagenes por defecto
    img_size = 1024
    label_fp = os.path.join(label_dir, "{}.txt".format(patient_id))

    if row is None:
        return
```

```
    top_left_x = row[1]
    top_left_y = row[2]
    w = row[3]
    h = row[4]

    # 'r' significa relativo. 'c' significa centro.
    rx = top_left_x/img_size
    ry = top_left_y/img_size
    rw = w/img_size
    rh = h/img_size
    rcx = rx+rw/2
    rcy = ry+rh/2

    line = "{} {} {} {} {}\n".format(0, rcx, rcy, rw, rh)

    f = open(label_fp, "a")
    f.write(line)
    f.close()

def save_yolo_data_from_rsna(dcm_dir, img_dir, label_dir, annots):
    for row in tqdm(annots.values):
         patient_id = row[0]

         img_fp = os.path.join(img_dir, "{}.jpg".format(patient_id))
         if os.path.exists(img_fp):
             save_label_from_dcm(label_dir, patient_id, row)
             continue
          # No se guardarán aquellas imágenes que no tengan coordenadas
para ser delimitadas
         target = row[5]
         if target == 0:
             continue

         save_label_from_dcm(label_dir, patient_id, row)
         save_img_from_dcm(dcm_dir, img_dir, patient_id)

def          save_yolo_data_normal_from_rsna(dcm_dir,          img_dir,
annots,num_samples=6012):
    count = 0
    for row in tqdm(annots.values):
        if count >= num_samples:
             break  # Detener el bucle si se han procesado suficientes
muestras
```

```
        patient_id = row[0]
        target = row[5]

         # No se guardarán aquellas imágenes que tengan coordenadas para
ser delimitadas
        if target == 1:
            continue

        save_img_from_dcm(dcm_dir, img_dir, patient_id)
          count += 1  # Incrementar el contador después de procesar una
muestra
```

Por último, dividimos los datos para la clasificación. El código realiza lo siguiente:

1. Se crean tres carpetas principales en la ruta especificada por base_dir: 'train' (entrenamiento), 'val' (validación) y 'test' (prueba).
2. Se define una lista de clases de imágenes, en este caso 'NEUMONIA' y 'NORMAL'.
3. Para cada clase:
    a. Obtiene la lista de imágenes en la carpeta correspondiente.
    b. Calcula el número de imágenes que se asignarán a cada conjunto (entrenamiento, validación, prueba) en función de las proporciones proporcionadas (train_ratio, val_ratio, test_ratio).
    c. Divide las imágenes en conjuntos según las proporciones calculadas.
    d. Mueve las imágenes a las carpetas correspondientes en la estructura de carpetas creada anteriormente.

```
def create_split_folders(base_dir, train_ratio, val_ratio, test_ratio):
    # Crear las carpetas de entrenamiento, validación y prueba
    train_dir = os.path.join(base_dir, 'train')
    val_dir = os.path.join(base_dir, 'val')
    test_dir = os.path.join(base_dir, 'test')

    os.makedirs(train_dir, exist_ok=True)
    os.makedirs(val_dir, exist_ok=True)
    os.makedirs(test_dir, exist_ok=True)
```

```
    # Lista de clases (NEUMONIA y NORMAL)
    classes = ['NEUMONIA', 'NORMAL']

    for class_name in classes:
        # Ruta a las imágenes de cada clase
        class_images_dir = os.path.join(base_dir, class_name)

        # Obtener la lista de imágenes en la carpeta
        images_list = os.listdir(class_images_dir)
        total_images = len(images_list)

        # Calcular el número de imágenes para cada split
        num_train = int(train_ratio * total_images)
        num_val = int(val_ratio * total_images)
        num_test = total_images - num_train - num_val

        # Barajar aleatoriamente la lista de imágenes
        random.shuffle(images_list)

        # Dividir las imágenes en cada split
        train_images = images_list[:num_train]
        val_images = images_list[num_train:num_train + num_val]
        test_images = images_list[num_train + num_val:]

        # Mover las imágenes a las carpetas correspondientes
        for img_name in train_images:
            src = os.path.join(class_images_dir, img_name)
            dst = os.path.join(train_dir, class_name, img_name)
            if not os.path.isdir(dst):
              os.makedirs(dst)
            shutil.copy(src, dst)

        for img_name in val_images:
            src = os.path.join(class_images_dir, img_name)
            dst = os.path.join(val_dir, class_name, img_name)
            if not os.path.isdir(dst):
              os.makedirs(dst)
            shutil.copy(src, dst)

        for img_name in test_images:
            src = os.path.join(class_images_dir, img_name)
            dst = os.path.join(test_dir, class_name, img_name)
            if not os.path.isdir(dst):
```

```
                os.makedirs(dst)
            shutil.copy(src, dst)

# Proporciones para el split (70% entrenamiento, 15% validación, 15% prueba)
train_ratio = 0.7
val_ratio = 0.15
test_ratio = 0.15

# Crear la estructura de carpetas con la distribución de imágenes deseada
create_split_folders(DATA_DIR, train_ratio, val_ratio, test_ratio)
```