

Abstract

This Bachelor's Thesis (TFG) aims to conduct a comprehensive statistical study of various deep learning-based approaches for the segmentation of Multiple Sclerosis (MS) lesions from magnetic resonance imaging (MRI) scans. The main objective is to compare the performance of different automatic segmentation models, both in 2D and 3D configurations, in order to identify optimal setups that can support healthcare professionals in improving diagnosis and disease monitoring.

The project evaluates multiple versions of the YOLOv8 model adapted for segmentation tasks, as well as different configurations of the nnUNet framework, which is considered a benchmark in medical image segmentation. The results of 2D and 3D configurations are analyzed, including the use of consensus policies that combine multiple predictions to enhance system robustness and accuracy. The MSSEG-2 dataset is used, containing 3D brain scans of 92 patients at multiple time points. These scans are processed both as 3D volumes and 2D slices, enabling a direct comparison of approaches and an evaluation of their clinical applicability.

Model training and validation are carried out on the Picasso supercomputer, providing the computational power needed to perform large-scale experiments. By combining advanced computer vision tools, state-of-the-art deep learning architectures, and comparative statistical methodologies, this project aims to provide empirical evidence on the strengths and limitations of different segmentation approaches for Multiple Sclerosis detection.

Keywords: MRI, Multiple Sclerosis, Picasso Supercomputer, YOLOv8, nnUNet, 2D/3D Segmentation, Consensus Policies, Deep Learning

Resumen

Este Proyecto de Fin de Grado (TFG) tiene como objetivo realizar un estudio estadístico exhaustivo de distintos enfoques basados en redes neuronales profundas para la segmentación de lesiones de Esclerosis Múltiple (EM) a partir de imágenes de resonancia magnética (IRM). El objetivo principal es comparar el rendimiento de varios modelos de segmentación automática, tanto en 2D como en 3D, con el fin de identificar configuraciones óptimas que ayuden a los profesionales de la salud a mejorar el diagnóstico y seguimiento de la enfermedad.

El proyecto evalúa múltiples versiones del modelo YOLOv8 adaptadas a tareas de segmentación, así como diferentes configuraciones del marco nnUNet, considerado un estándar de referencia en segmentación médica. Se analizan los resultados obtenidos en configuraciones 2D y 3D, incluyendo el uso de políticas de consenso que combinan múltiples predicciones para aumentar la robustez y precisión del sistema. El conjunto de datos utilizado es MSSEG-2, que contiene escaneos cerebrales 3D de 92 pacientes en distintos momentos temporales. Estos escaneos se procesan tanto como volúmenes 3D como en cortes 2D, lo que permite comparar directamente los enfoques y evaluar su aplicabilidad clínica.

El entrenamiento y validación de los modelos se realiza en el superordenador Picasso de la RES, lo que proporciona los recursos computacionales necesarios para llevar a cabo experimentos a gran escala. Al combinar herramientas avanzadas de visión por computadora, arquitecturas de aprendizaje profundo de última generación y metodologías estadísticas comparativas, este proyecto busca aportar evidencia empírica sobre las ventajas y limitaciones de diferentes enfoques de segmentación para la detección de Esclerosis Múltiple.

Palabras clave: IRM, Esclerosis Múltiple, Superordenador Picasso, YOLOv8, nnUNet, Segmentación 2D/3D, Políticas de Consenso, Aprendizaje Profundo

Índice

1. Introducción	13
1.1. Contexto	13
1.2. Motivación	13
1.3. Objetivos	14
1.3.1. Objetivos generales	14
1.3.2. Objetivos específicos	14
1.4. Estructura del documento	14
1.5. Tecnologías usadas	15
2. Estado del Arte	17
2.1. La resonancia magnética (MRI)	17
2.1.1. Historia y desarrollo	17
2.1.2. Fundamentos físicos y técnicos de funcionamiento	19
2.1.3. Aplicaciones clínicas principales	21
2.1.4. Avances tecnológicos recientes y evolución clínica	23
2.2. La esclerosis múltiple (EM)	24
2.2.1. Tipos y cursos clínicos de EM	25
2.3. Conexión entre la resonancia magnética y la esclerosis múltiple	26
2.3.1. Papel de la RM en el diagnóstico y seguimiento de la EM	27
2.3.2. Secuencias de RM utilizadas en EM y hallazgos típicos	29
2.4. Inteligencia Artificial (IA)	32
2.4.1. Aprendizaje Automático (Machine Learning)	32
2.4.2. Aprendizaje Supervisado	34
2.4.3. Aprendizaje No Supervisado	35
2.4.4. Aprendizaje Profundo (Deep Learning)	36
2.4.5. Redes Neuronales Convolucionales (CNN)	37
2.5. Red nnUNet	39
2.5.1. Motivación para el desarrollo de nnUNet	40

2.5.2. Diseño y arquitectura general de nnUNet	41
2.5.3. Configuración automática de nnU-Net	42
2.6. Arquitectura YOLO	43
2.6.1. YOLOv1: arquitectura inicial unificada	43
2.6.2. YOLOv2: mejoras de rendimiento con anclas y nueva espalda (Darknet-19)	44
2.6.3. YOLOv3: Red residual más profunda y detección multi-escala	45
2.6.4. YOLOv4: CSPDarknet y trucos para el estado del arte (2020)	47
2.6.5. YOLOv5: implementación en PyTorch y variantes de segmentación	48
2.6.6. YOLOv6: enfoque industrial con eficiencia y cabeza ancla-libre	50
2.6.7. YOLOv7: nuevas estrategias de arquitectura y segmentación con Blend-Mask	51
2.6.8. YOLOv8: modelo unificado multi-tarea con cabeza desacoplada (2023)	54
2.6.9. YOLOv9: PGI y GELAN para mejorar el flujo de información (2024)	55
2.6.10. YOLOv10: detección end-to-end sin NMS y diseño eficiente (NeurIPS 2024)	57
2.6.11. YOLOv11: módulos C3K2 y C2PSA, y consolidación multi-tarea (2024-2025)	57
2.6.12. Comparación de rendimiento y conclusiones	58
3. Análisis y Diseño	61
3.1. Resumen gráfico de los experimentos	61
3.2. Casos de uso	62
3.2.1. Casos de uso funcionales y no funcionales	62
3.2.2. Análisis de casos de uso	63
3.3. Diagramas de secuencia	69
4. Conjunto de datos MSLesSeg (ICPR 2024)	71
4.1. Características generales del conjunto de datos	71
4.2. Detalles técnicos de las máscaras de segmentación	73
4.3. Pre-procesamiento del conjunto de datos	74
4.3.1. Preparación de datos para segmentación 2D basada en YOLO	74

4.3.2. Preparación de datos para segmentación basada en nnUNet	77
4.4. Validación cruzada con 5 pliegues	79
4.5. Retos e implicaciones de datos médicos longitudinales (EM)	80
5. Resultados Experimentales	85
5.1. Configuración de los experimentos	85
5.1.1. Modelos nnUNet	86
5.1.2. Modelos YOLO	86
5.1.3. Métrica DSC	87
5.2. Infraestructura computacional: Supercomputador Picasso	89
5.2.1. Características de Picasso	89
5.2.2. Retos de Implementación	90
5.2.3. Justificación del uso de <i>Picasso</i>	92
5.3. Resultados cuantitativos	93
5.3.1. Resultados por modelo	93
5.3.2. Diagrama de cajas y bigotes	94
5.3.3. Test de Friedman	96
5.3.4. Test de Wilcoxon	98
5.3.5. Tests Bayesianos	99
5.3.6. Recursos computacionales	103
5.4. Resultados Cualitativos	104
5.4.1. Ejemplo con buen desempeño de ambos enfoques	105
5.4.2. Ejemplo con mal desempeño de YOLO	106
5.5. Discusión	108
6. Conclusiones y Líneas Futuras	113
6.1. Conclusiones	113
6.2. Líneas Futuras	114
Apéndice A. Manual de Instalación	123
A.1. Introducción	123
A.2. Requisitos del Sistema	123

A.2.1.	Especificaciones de Hardware y Software	123
A.3.	Guía de Instalación	124
A.3.1.	Configuración del Repositorio	124
A.3.2.	Instalación del Paquete	125
A.3.3.	Configuración del Entorno	126
A.4.	Ejecución del Pipeline	127
A.4.1.	Conceptos Fundamentales	127
A.4.2.	Ejecución Básica	127
A.4.3.	Monitoreo y Tiempos de Ejecución	128
A.4.4.	Opciones de Configuración	129
A.4.5.	Componentes Individuales	130
A.5.	Resolución de Problemas	131
A.5.1.	Problemas Comunes y Soluciones	131
A.5.2.	Optimización del Rendimiento	133
A.5.3.	Recursos de Ayuda	134
Apéndice B. Manual de Usuario	135	
B.1.	Introducción	135
B.2.	Arquitectura del Proyecto	135
B.2.1.	Filosofía de Diseño y Arquitectura	135
B.3.	Estructura del Proyecto	136
B.3.1.	Biblioteca Principal NND	136
B.4.	Gestión del Dataset	138
B.4.1.	Dataset MSLesSeg	138
B.5.	Jupyter Notebooks	139
B.5.1.	Configuración del Entorno	139
B.5.2.	Notebooks de Visualización	140
B.5.3.	Análisis Estadístico	142
B.5.4.	Mejores Prácticas	144
B.5.5.	Notebooks Personalizados	145
B.6.	Uso Avanzado	147

B.6.1.	Extensión del Pipeline	147
B.7.	Solución de Problemas	149
B.7.1.	Problemas Comunes	149
B.8.	Monitoreo y Optimización del Rendimiento	150
B.8.1.	Perfilado del Rendimiento de Notebooks	150
B.8.2.	Monitoreo de Recursos	151
B.9.	Referencia Rápida	152
B.9.1.	Ubicaciones de Archivos Importantes	152
B.9.2.	Comandos Comunes	152

Índice de figuras

1.	Primera máquina de resonancia magnética [4].	18
2.	Máquina de resonancia magnética [5].	21
3.	Corte cerebral de resonancia magnética [9].	28
4.	Jerarquía Inteligencia Artificial [10].	33
5.	Aprendizaje Supervisado vs Aprendizaje no Supervisado [12].	33
6.	Visualización de una red convolucional [16].	37
7.	Arquitectura de la U-Net [18].	40
8.	Proceso de configuración automática de la nnUNet [19].	42
9.	Arquitectura de YOLOv1 [21].	44
10.	Arquitectura de YOLOv3 [23].	46
11.	Arquitectura de YOLOv4 [24].	48
12.	Arquitectura de YOLOv5 [25].	50
13.	Arquitectura de YOLOv6 [27].	51
14.	Arquitectura de YOLOv7 [28].	52
15.	Arquitectura de YOLOv7 Mask [29].	53
16.	Rendimiento comparativo en MS COCO [31].	56
17.	Resumen gráfico de los experimentos del estudio.	61
18.	Diagrama de secuencia de la ejecución de la yolo.	69
19.	Diagrama de secuencia de la ejecución de la nnunet.	70
20.	Estructura del conjunto de datos MSLesSeg (ICPR 2024).	72
21.	Proyección axial, sagital y coronal de escáner cerebral.	75
22.	Picasso, el Supercomputador de la UMA [38].	89
23.	Distribuciones de DSC para los modelos basados en YOLO y nnU-Net (pliegue 1).	95
24.	Test bayesiano de signo: Yolo2D coronal vs Yolo3D.	101
25.	Test bayesiano de signo: Yolo2D axial vs Yolo3D.	101
26.	Test bayesiano de signo: Yolo2D sagital vs Yolo3D.	102
27.	Test bayesiano de rango: Yolo2D coronal vs Yolo3D.	102

28.	Test bayesiano de rango: Yolo2D axial vs Yolo3D.	102
29.	Test bayesiano de rango: Yolo2D sagital vs Yolo3D.	102
30.	Uso computacional durante el entrenamiento de la nnUNet en modo 2D. . . .	103
31.	Uso computacional durante el entrenamiento de la nnUNet en modo 3D. . . .	104
32.	Predicciones de los modelos para el corte 104 en el plano axial del paciente 49. . . .	105
33.	Predicciones de los modelos para el corte 111 en el plano sagital del paciente 50. . . .	107
34.	Predicciones de los modelos para el corte 84 en el plano coronal del paciente 49. . . .	107

Índice de tablas

1.	Resumen comparativo de las principales versiones de YOLO (v1-v11) [20] [21] [33].	59
2.	Caso de uso 1: Entrenar modelo de segmentación.	63
3.	Caso de uso 2: Validar modelo entrenado.	64
4.	Caso de uso 3: Generar máscara de segmentación.	65
5.	Caso de uso 4: Visualizar y comparar máscaras.	66
6.	Caso de uso 5: Análisis estadístico comparativo.	67
7.	Caso de uso 6: Entrenamiento con validación cruzada automática (5 hilos). . .	68
8.	Media y desviación estándar de las puntuaciones DSC en 5 pliegues y 30 repeticiones para cada modelo nnU-Net. Los tonos gris oscuro y gris claro representan los mejores y los segundos mejores valores, respectivamente.	93
9.	Media y desviación estándar de las puntuaciones DSC en 5 pliegues y 30 repeticiones para cada modelo YOLO. Los tonos gris oscuro y gris claro representan los mejores y los segundos mejores valores, respectivamente.	94
10.	Resultados de la prueba de Friedman para los modelos basados en YOLO a través de los pliegues. Un “+” indica una diferencia significativa en la prueba de Friedman (FT). Los tonos gris oscuro y gris claro representan los mejores y los segundos mejores valores, respectivamente.	96
11.	Resumen de la prueba de rangos con signo de Wilcoxon para los modelos YOLO. Cada celda indica los resultados de significancia a través de los pliegues. .	99
12.	Requisitos de Hardware.	124
13.	Dependencias de Software.	124
14.	Tiempo de Ejecución Esperado (dependiente de GPU).	128
15.	Configuraciones de Entrenador nnUNet Disponibles.	129
16.	Configuraciones de Modelo YOLO Disponibles.	130
17.	Componentes del Módulo de Modelos.	136
18.	Funciones Utilitarias.	137
19.	Características del Dataset.	138

20.	Divisiones de Pacientes para Validación Cruzada.	139
21.	Referencia Rápida: Ubicaciones de Archivos.	152

1

Introducción

1.1. Contexto

La Esclerosis Múltiple (EM) es una enfermedad neurodegenerativa crónica del sistema nervioso central que afecta a millones de personas en todo el mundo. Una característica clave de la EM es la aparición de lesiones desmielinizantes en el cerebro y la médula espinal, las cuales pueden visualizarse mediante técnicas de imagen como la resonancia magnética (IRM). Estas lesiones son indicadores fundamentales para el diagnóstico, seguimiento y evaluación de la progresión de la enfermedad.

La segmentación precisa y automática de estas lesiones en IRM representa un reto significativo debido a la alta variabilidad de las lesiones, su pequeño tamaño relativo y la presencia de estructuras cerebrales complejas. Aunque existen enfoques manuales y semiautomáticos, son altamente dependientes del observador, consumen mucho tiempo y no son escalables. Es por ello que en los últimos años ha surgido un gran interés por aplicar técnicas de aprendizaje profundo a esta tarea, gracias a su capacidad para aprender representaciones complejas directamente de los datos.

1.2. Motivación

El desarrollo de métodos automáticos y precisos para la segmentación de lesiones de EM tiene un impacto directo en la práctica clínica. Facilita el diagnóstico precoz, permite un seguimiento más eficaz de los pacientes y reduce la carga de trabajo de los profesionales sanitarios. En particular, comparar diferentes arquitecturas modernas de redes neuronales profundas puede aportar evidencia valiosa sobre qué modelos son más adecuados para esta tarea específica.

Además, el uso de modelos 3D presenta ventajas potenciales frente a los modelos 2D, ya que pueden aprovechar la información contextual entre cortes sucesivos de IRM. Sin embargo, también requieren una mayor cantidad de memoria y potencia computacional. En este sentido, el acceso al superordenador Picasso permite experimentar con estas configuraciones avanzadas y estudiar sus beneficios reales.

1.3. Objetivos

1.3.1. Objetivos generales

El objetivo general de este TFG es comparar distintos enfoques de segmentación automática de lesiones de Esclerosis Múltiple mediante redes neuronales profundas aplicadas a imágenes de resonancia magnética, para determinar cuáles ofrecen un rendimiento óptimo y pueden contribuir a una mejor práctica clínica.

1.3.2. Objetivos específicos

- Implementar y entrenar diferentes versiones de YOLOv11 adaptadas a segmentación.
- Implementar y entrenar configuraciones 2D y 3D de nnUNet.
- Evaluar el rendimiento de los modelos utilizando métricas estándar como Dice, Sensibilidad y Especificidad.
- Estudiar el impacto del uso de políticas de consenso sobre la precisión de los resultados.
- Analizar las diferencias entre enfoques 2D y 3D en términos de precisión, eficiencia y viabilidad clínica.

1.4. Estructura del documento

Este documento se estructura de la siguiente manera:

- **Introducción:** contextualiza el problema, detalla la motivación y expone los objetivos del proyecto.

- **Estado del Arte:** repasa trabajos previos relacionados con la segmentación de lesiones en EM y el uso de redes neuronales profundas en imágenes médicas.
- **Análisis de Requisitos y Diseño:** describe los requisitos técnicos y funcionales, así como el diseño general del sistema.
- **Conjunto de datos MSLesSeg (ICPR 2024):** presenta el conjunto de datos utilizado, sus características, procesamiento y división para entrenamiento y validación.
- **Resultados Experimentales:** expone los resultados obtenidos tras los entrenamientos, junto con su análisis comparativo y visualizaciones.
- **Conclusiones y Líneas Futuras:** resume los principales hallazgos del trabajo, limita sus alcances y sugiere futuras líneas de investigación.
- **Apéndices:** contiene material adicional, como el manual de instalación del entorno de trabajo.

1.5. Tecnologías usadas

Este proyecto hace uso de múltiples tecnologías y herramientas, tanto a nivel de hardware como de software:

- **Superordenador Picasso:** infraestructura de alto rendimiento de la RES, utilizada para el entrenamiento intensivo de modelos 3D.
- **YOLOv11 (Ultralytics):** versión moderna del modelo YOLO especializada en tareas de segmentación médica con arquitectura de tipo encoder-decoder.
- **nnUNet:** marco automatizado para segmentación médica, capaz de ajustar arquitecturas U-Net de forma óptima a las características del conjunto de datos.
- **Python 3.10:** lenguaje de programación utilizado para el desarrollo e implementación de scripts y modelos.
- **CVAT:** herramienta para la anotación de datos utilizada en la creación de conjuntos de entrenamiento.

- **OpenCV**: librería para el procesamiento de imágenes, utilizada en tareas de pre-procesado y visualización.
- **Google Colab**: entorno de ejecución en la nube para entrenamientos preliminares y pruebas.
- **Git y GitHub**: sistemas de control de versiones y colaboración.
- **Visual Studio Code**: entorno de desarrollo utilizado para codificación, depuración y gestión del proyecto.
- **Matplotlib**: librería para la generación de gráficos y visualización de resultados.
- **NumPy**: herramienta para el manejo eficiente de estructuras de datos numéricos.
- **Pandas**: librería para análisis y manipulación de datos.
- **Scikit-learn**: biblioteca para evaluación de modelos y métricas estadísticas.
- **SymPy**: librería para álgebra simbólica.
- **Pillow**: librería para manejo de imágenes.
- **SciPy**: biblioteca para cálculos científicos avanzados.

2

Estado del Arte

2.1. La resonancia magnética (MRI)

La resonancia magnética (RM) es una técnica de imagen médica no invasiva que revolucionó el diagnóstico por imágenes al permitir visualizar estructuras internas con gran detalle sin emplear radiación ionizante [1]. A diferencia de rayos X o tomografía computarizada, la RM utiliza campos magnéticos fuertes y ondas de radio para producir imágenes de los órganos y tejidos, logrando un contraste superior en tejidos blandos como el cerebro [2]. Desde su introducción en la clínica en la década de 1980, la RM se ha vuelto una herramienta indispensable en medicina, ampliamente utilizada para la detección, diagnóstico, estadificación y seguimiento de numerosas enfermedades en neurología, oncología, cardiología y otras especialidades. A continuación, se detallan su evolución histórica, principios de funcionamiento, aplicaciones clínicas y avances recientes.

2.1.1. Historia y desarrollo

El desarrollo de la RM tiene sus raíces en el descubrimiento del fenómeno físico de la resonancia magnética nuclear (RMN) a mediados del siglo XX. En la década de 1940, los físicos Felix Bloch y Edward Purcell demostraron independientemente que ciertos núcleos atómicos absorben y emiten energía de radiofrecuencia cuando se encuentran en un campo magnético, hallazgo por el cual recibieron el Premio Nobel de Física en 1952. Este fenómeno de RMN sentó las bases para aplicaciones posteriores tanto en química como en medicina.

A partir de la RMN, varios visionarios imaginaron su aplicación en imágenes médicas. En 1971, el médico e investigador Raymond Damadian publicó un estudio pionero que mostraba que

la señal de RMN podía distinguir entre tejidos normales y tumorales, sugiriendo así su potencial para detectar enfermedades en el cuerpo humano. Poco después, en 1973, el químico Paul Lauterbur introdujo el concepto de emplear gradientes de campo magnético para codificar posiciones espaciales, lo que permitió crear las primeras imágenes bidimensionales por RMN. Paralelamente, el físico Peter Mansfield desarrolló técnicas matemáticas (como el eco-planar) para acelerar la adquisición de imágenes. El trabajo combinado de Lauterbur y Mansfield condujo a la obtención de las primeras imágenes de RM a mediados de los años 70, y décadas más tarde ambos recibirían el Nobel de Medicina en 2003 por sus contribuciones al desarrollo de la RM [3].

El primer escáner de RM, el de la Figura 1, capaz de obtener imágenes del cuerpo humano fue construido a finales de la década de 1970. En 1977, Damadian y su equipo completaron el primer escaneo corporal completo por RM en un ser humano, marcando un hito histórico que demostró la viabilidad clínica de esta tecnología. En la imagen 1 se observa el prototipo conocido como “Mark One”, el primer equipo de RM utilizado en pacientes (Aberdeen, Escocia), que exhibe los componentes principales del imán y bobinas en una configuración abierta de laboratorio. Este logro dio paso a una rápida adopción: durante la década de 1980 varias compañías comercializaron los primeros escáneres de RM, instalándose las primeras unidades clínicas y expandiéndose su disponibilidad en hospitales y centros de investigación. Inicialmente se conocía como “Imagen por Resonancia Magnética Nuclear” (NMRI, por sus siglas en inglés), pero el término nuclear fue suprimido por sus connotaciones negativas durante los años 80.



Figura 1: Primera máquina de resonancia magnética [4].

Desde los años 1980, la tecnología de RM experimentó un avance acelerado. Mejores imanes superconductores permitieron incrementar la intensidad de campo (1.5 Tesla se volvió estándar en los 90, seguido de 3 Tesla en los 2000), mejorando la resolución de las imágenes. Asimismo, los sistemas de cómputo y algoritmos de reconstrucción evolucionaron, reduciendo los tiempos de adquisición y refinando la calidad de imagen. En los años 90 surgió la resonancia funcional (fMRI), capaz de mapear la actividad cerebral midiendo cambios en el flujo sanguíneo cerebral, abriendo un campo para la investigación neurocientífica. A finales de esa década y en los 2000, aparecieron secuencias avanzadas como difusión (DTI) para visualizar tractos neuronales, y perfusión para evaluar vascularización, ampliando el alcance de la RM más allá de la anatomía estática.

En la actualidad, la RM sigue en constante innovación. Se han desarrollado sistemas de ultracampo (ej. imanes de 7 Tesla) que brindan imágenes de altísima resolución espacial, útiles en investigación y en la detección de lesiones sutiles, aunque su uso clínico aún es limitado. También se exploran modalidades híbridas, como equipos combinados de PET-RM que fusionan la información metabólica de la tomografía por emisión de positrones con el detalle anatómico de la RM. Otros avances recientes incluyen técnicas de aceleración de adquisiciones (mediante algoritmos de reconstrucción paralela y aprendizaje automático) para reducir los tiempos de escaneo, así como diseños de imán abiertos para mayor comodidad de pacientes claustrofóbicos. La historia de la RM, desde los experimentos físicos de mediados de siglo XX hasta las sofisticadas máquinas actuales, es un ejemplo emblemático de progreso científico-tecnológico con enorme impacto en la medicina.

2.1.2. Fundamentos físicos y técnicos de funcionamiento

La RM se basa en los principios de la resonancia magnética nuclear aplicados a la obtención de imágenes. El cuerpo humano está compuesto en gran proporción por agua y grasa, cuyos átomos de hidrógeno poseen núcleos (protones) con espín magnético. En un escáner de RM, el paciente se coloca dentro de un campo magnético muy intenso y homogéneo generado por un imán principal; bajo esta influencia, los espines nucleares de los protones se alinean parcialmente en la dirección del campo. A continuación, el equipo emite pulsos de radiofrecuencia (RF) a una frecuencia específica (frecuencia de Larmor) que excita a esos protones alineados.

Al cesar cada pulso, los protones excitados retornan a su estado de equilibrio liberando energía en forma de señales de RF detectables. Este proceso de relajación ocurre con constantes de tiempo características de cada tipo de tejido (T1, T2), lo que permite diferenciar estructuras según su composición y contenido de agua.

Para formar imágenes espaciales a partir de estas señales, la RM utiliza bobinas de gradiente que producen pequeñas variaciones lineales del campo magnético en las tres dimensiones del espacio. Al aplicar gradientes durante y después de la excitación RF, las señales emitidas por los protones llevan una codificación de frecuencia y fase que depende de su posición. Mediante reconstrucción matemática (transformada de Fourier), se obtienen matrices de intensidad que conforman la imagen de la anatomía escaneada. En esencia, la RM mide la distribución de protones (principalmente de agua) y sus propiedades de relajación en el cuerpo, traduciéndolas en imágenes en escala de grises donde diferentes tejidos aparecen con distintas intensidades de señal. Ajustando los parámetros de la secuencia de pulsos (tiempos de repetición TR, tiempos de eco TE, etc.), es posible enfatizar distintos contrastes: por ejemplo, imágenes T1 (ponderadas en T1) donde la grasa aparece brillante y el líquido cefalorraquídeo oscuro, o imágenes T2 donde el líquido es brillante y ciertas lesiones se destacan hiperintensas.

Un escáner de RM como el de la Figura 2 típicamente consta de varios componentes clave: el imán principal (generalmente superconductor, enfriado con helio líquido) que genera el campo estático potente (usualmente 1.5 o 3.0 Tesla en clínica); un conjunto de bobinas de gradiente ortogonales para la codificación espacial; y las bobinas de RF, que incluyen una bobina emisora (para los pulsos) y bobinas receptoras sensibles que captan las señales emitidas desde el paciente. Todo el sistema está controlado por computadores que sincronizan la emisión de pulsos, la conmutación de gradientes y la adquisición de datos, para luego procesar las señales en imágenes. La imagen 2 muestra un paciente siendo posicionado en un escáner de RM, ilustrando el gantry circular que contiene el imán y las bobinas, mientras una tecnóloga asegura la colocación adecuada de la cabeza antes del estudio. Durante la exploración, se producen ruidos fuertes por la rápida conmutación de las bobinas de gradiente, y la duración puede ser de varios minutos por secuencia, factores que pueden incomodar al paciente. Sin embargo, el procedimiento es indoloro y sumamente seguro, salvo en pacientes con ciertas contraindicaciones (p. ej., marcapasos antiguos no compatibles, implantes ferromagnéticos), debido a la

interacción del campo magnético con metales.



Figura 2: Máquina de resonancia magnética [5].

En síntesis, la RM obtiene imágenes aprovechando las propiedades magnéticas de los núcleos atómicos del cuerpo. Al no usar rayos X, carece de efectos ionizantes dañinos, lo que permite repetir estudios sin riesgo radiológico. La gran versatilidad en contraste tisular la hace ideal para examinar desde el cerebro y la médula espinal hasta músculos, ligamentos, corazón e incluso detectar tumoraciones milimétricas. Estos fundamentos físicos robustos explican por qué la RM se ha convertido en una de las modalidades de imagen más potentes en la medicina moderna.

2.1.3. Aplicaciones clínicas principales

La RM tiene aplicaciones clínicas muy amplias y se ha convertido en la modalidad de imagen de elección para muchas enfermedades. En neurología, la RM cerebral es la herramienta principal para diagnosticar enfermedades desmielinizantes como la esclerosis múltiple, detectar tumores cerebrales, malformaciones vasculares, accidentes cerebrovasculares isquémicos (especialmente mediante secuencias de difusión), epilepsia (identificando lesiones corticales sutiles) y enfermedades neurodegenerativas (evaluando atrofia cerebral). La alta sensibilidad de la RM para la sustancia blanca y gris permite detectar lesiones microscópicas que pasarían

desapercibidas en una tomografía. También es fundamental en la evaluación de la médula espinal para trauma, mielopatías compresivas por hernias discales, o lesiones inflamatorias. En oncología, la RM se utiliza para caracterizar tumores en diversos órganos: es altamente valorada en tumores cerebrales por el contraste entre tejido sano y patológico, en cánceres hepáticos (con técnicas especiales de contraste dinámico), en la evaluación local de cáncer de próstata, de mama (especialmente RM mamaria con contraste para detección en pacientes de alto riesgo) y en sarcomas de partes blandas. Gracias a la ausencia de radiación, la RM es preferible en pacientes que requieren seguimiento frecuente, como supervivientes de cáncer pediátrico.

En el sistema musculoesquelético, la RM es el estándar oro para visualizar cartílago, ligamentos y tendones. Se emplea rutinariamente en lesiones deportivas (ej. rotura de ligamento cruzado anterior de rodilla, desgarros meniscales, lesiones de manguito rotador en hombro) porque muestra con claridad las estructuras de tejido blando que no son visibles en radiografías. Asimismo, en la columna vertebral, la RM detecta hernias de disco, estenosis del canal, compresiones medulares, etc., con gran detalle anatómico. En cardiología, la RM cardiaca permite evaluar la anatomía y función del corazón, visualizar miocardio fibroso o inflamado (como en miocarditis) mediante realce tardío con gadolinio, cuantificar flujos y volúmenes ventriculares, siendo muy útil en cardiopatías congénitas complejas o para diagnosticar miocardiopatías. Otras aplicaciones incluyen la angiografía por RM (MRA) para visualizar vasos sanguíneos sin contraste iodado (pudiendo detectar aneurismas cerebrales o disecciones aórticas), la colangiopancreatografía por RM (MRCP) para ver las vías biliares y pancreáticas, y estudios funcionales como la RM de perfusión cerebral o la espectroscopía por RM, que brinda información metabólica de los tejidos.

En suma, la RM se ha posicionado como una técnica central en el diagnóstico por imagen. Su capacidad multiplanar, excelente resolución de contraste y seguridad la hacen ideal para una variedad de escenarios clínicos. Hoy en día, millones de exploraciones por RM se realizan anualmente en el mundo, y muchas guías clínicas la recomiendan como prueba de primera línea para patologías neurológicas, musculoesqueléticas y oncológicas, entre otras. Esto representa un gran cambio respecto a hace cuatro décadas, cuando la RM apenas emergía y la mayoría de diagnósticos se basaban en radiografías o procedimientos invasivos. La adopción universal de la RM ha elevado el nivel de la atención médica al permitir diagnósticos más

precoces y precisos.

2.1.4. Avances tecnológicos recientes y evolución clínica

La tecnología de RM continúa avanzando rápidamente en el siglo XXI, ampliando sus capacidades diagnósticas. Uno de los hitos recientes es el desarrollo de escáneres de campo ultraalto (7 Tesla), que proporcionan una resolución sin precedentes y mejor relación señal/ruido. Estos equipos han mostrado utilidad, por ejemplo, en neurología para visualizar lesiones corticales sutiles en esclerosis múltiple o detalles anatómicos del hipocampo en epilepsia que no se apreciaban en 3 Tesla. Si bien su costo y requerimientos (blindaje magnético, enfriamiento criogénico) limitan su disponibilidad, marcan la pauta de lo que podría ser el futuro de la RM clínica. Otro frente de innovación es la RM multimodal e híbrida. Sistemas combinados de PET-RM integran en un mismo examen la información metabólica proporcionada por PET (usando radiofármacos) con la detallada anatomía de la RM. Esta sinergia mejora la precisión en oncología -por ejemplo, localizando con exactitud tumores activos para planificación de radioterapia- y en neurología (PET de amiloide junto a RM estructural en demencias). Del mismo modo, la RM simultánea con espectroscopía o electroencefalografía amplía las aplicaciones en investigación neurofuncional.

En cuanto a las secuencias, se han desarrollado técnicas más rápidas y robustas. La imagen paralela (parallel imaging) y métodos de compresión de muestreo (compressed sensing) permiten obtener imágenes diagnósticas acortando considerablemente el tiempo de adquisición, lo que beneficia a pacientes pediátricos o claustrofóbicos al reducir el tiempo que deben permanecer inmóviles en el escáner. Asimismo, secuencias 3D isotrópicas con alta resolución permiten reformatear las imágenes en cualquier plano sin pérdida de calidad, optimizando la visualización de estructuras complejas (por ejemplo, el oído interno o pequeños vasos cerebrales). En cuanto a aplicaciones clínicas, la RM se ha extendido a poblaciones más amplias. Hoy día se emplea incluso en evaluaciones de rutina de columna o cerebro en personas mayores, detectando alteraciones subclínicas (microinfartos, cambios isquémicos crónicos) que antes solo se descubrían en autopsias. La disponibilidad de la RM también ha crecido: muchos hospitales comarciales cuentan con su propio equipo, y existen unidades móviles de RM que acercan el servicio a regiones remotas. Esto contrasta con los 80s, cuando solo centros especializados disponían de

RM. La evolución del uso clínico ha sido tal que la RM se considera ahora indispensable; por ejemplo, es impensable abordar el diagnóstico de una sospecha de esclerosis múltiple sin una RM de cerebro y médula, o evaluar un ligamento cruzado sin una RM de rodilla.

Finalmente, cabe mencionar la emergente integración de la inteligencia artificial (IA) en la RM (profundizada más adelante). Algoritmos de IA se están aplicando tanto para mejorar la calidad de imagen (reduciendo ruido, corrigiendo artefactos) como para acelerar la adquisición mediante reconstrucciones predictivas, e incluso para el post-procesamiento automatizado de hallazgos. Estos desarrollos auguran una próxima generación de equipos de RM más rápidos, inteligentes y precisos. En conclusión, la RM ha pasado de ser una curiosidad experimental a un pilar central de la práctica médica en unas pocas décadas. Sus fundamentos físicos robustos han permitido continuas mejoras tecnológicas, y su adaptabilidad la mantiene en la vanguardia del diagnóstico por la imagen. Con los avances actuales, es de esperar que la RM siga expandiendo sus fronteras, ofreciendo imágenes cada vez más detalladas y funcionales que contribuyan a una mejor atención de los pacientes.

2.2. La esclerosis múltiple (EM)

La esclerosis múltiple (EM) -también conocida históricamente como esclerosis en placas- es una enfermedad neurológica crónica de naturaleza inflamatoria y autoinmunitaria, caracterizada por la aparición de lesiones desmielinizantes en el sistema nervioso central (SNC) junto con daño axonal variable. Es decir, el propio sistema inmune del paciente ataca la mielina (capa de aislamiento de las fibras nerviosas) en el cerebro y la médula espinal, provocando interrupciones en la conducción nerviosa. Patológicamente, la EM se define por la presencia de placas o áreas focales de desmielinización diseminadas en el SNC, acompañadas de inflamación y posterior gliosis (cicatrización astrogial). Constituye una de las principales causas de discapacidad neurológica no traumática en adultos jóvenes, especialmente en mujeres. Actualmente se estima que aproximadamente 2.8 millones de personas en el mundo padecen EM, con una prevalencia en aumento en las últimas décadas (en 2013 se calculaba 2.3 millones) debido en parte a un mejor diagnóstico y supervivencia. La relación mujer:hombre es alrededor de 3:1, indicando mayor susceptibilidad en el sexo femenino. La edad típica de inicio es entre los 20 y 40 años, aunque puede presentarse más tempranamente o en edades mayores;

la media de comienzo es hacia los 30 años en formas remitentes-recurrentes, mientras que la forma progresiva de inicio suele manifestarse alrededor de la cuarta década.

2.2.1. Tipos y cursos clínicos de EM

La esclerosis múltiple presenta varias formas o cursos clínicos, lo que refleja diferentes patrones de actividad de la enfermedad a lo largo del tiempo. Clásicamente, la EM se clasifica en cuatro patrones clínicos principales en su evolución natural:

- **EM remitente-recurrente (EMRR):** Es la forma más común (alrededor del 85 de los casos al inicio). Se caracteriza por la aparición de brotes o recaídas neurológicas focales (episodios de síntomas que se desarrollan en días y duran típicamente semanas) seguidos de remisiones con recuperación completa o parcial de los déficits. Durante las remisiones, la enfermedad permanece inactiva clínicamente. Los brotes corresponden a nuevas lesiones inflamatorias en el SNC, mientras que en las remisiones la inflamación disminuye. Con el tiempo, las recuperaciones pueden ser incompletas, dejando secuelas acumulativas [6].
- **EM secundariamente progresiva (EMSP):** Se refiere a la fase que puede sobrevenir tras años de una EM remitente-recurrente. En la EMSP, el paciente experimenta una progresión gradual de la discapacidad neurológica independiente de brotes agudos, aunque pueden seguir ocurriendo recaídas superpuestas en algunos casos. Aproximadamente, la mayoría de pacientes EMRR evolucionan a EMSP tras 10-20 años desde el inicio, especialmente si no reciben tratamiento. La transición a EMSP implica que el componente neurodegenerativo se vuelve dominante sobre la inflamación aguda [7].
- **EM primariamente progresiva (EMPP):** Representa cerca del 10-15 de los casos. En esta forma, desde el inicio la enfermedad muestra un curso insidioso de empeoramiento progresivo de los síntomas y la discapacidad, sin brotes definidos iniciales. Suele comenzar más tarde (final de la cuarta o quinta década) y es más equitativa en sexos. La EMPP indica que el proceso patológico difuso (degeneración crónica) es primario, con menos actividad focal aguda detectable; a veces se observan lesiones en RM pero sin exacerbaciones clínicas claras.

- **EM progresiva recurrente (EMPR):** Era una categoría menos frecuente (5 de los casos) en la que existe progresión desde el inicio pero con brotes superpuestos ocasionales. En la última revisión de fenotipos (Lublin et al. 2014), este tipo se ha integrado dentro de EMPP “activa” con recaídas.

Adicionalmente, se describen formas especiales dentro del espectro de la EM:

- **Síndrome clínicamente aislado (CIS):** un primer episodio neurológico sugestivo de desmielinización (por ejemplo, un episodio de neuritis óptica) que aún no cumple criterios de EM diseminada en tiempo. Un CIS puede convertirse posteriormente en EM definida si aparecen nuevas lesiones o brotes.
- **EM fulminante (variante de Marburg):** curso rápidamente progresivo con múltiples brotes severos en corto tiempo, llevando a gran discapacidad en pocos años. Es poco común.
- **EM benigna:** se denomina así retrospectivamente a casos de EM de larga evolución (15 años o más) con muy poca discapacidad acumulada y escasas recaídas. Sin embargo, incluso en EM inicialmente leve, puede haber progresión tardía.

El reconocimiento del tipo de EM es importante para el pronóstico y la elección terapéutica. Por ejemplo, los tratamientos modificadores de la enfermedad han mostrado mayor eficacia en la EMRR (reduciendo brotes y lesiones nuevas) y algunas terapias más recientes también ayudan a frenar la progresión en EMSP activa y EMPP. Cabe destacar que la transición de EMRR a EMSP es un continuum, y actualmente se habla de la enfermedad en términos de actividad (presencia de brotes o lesiones nuevas en RM) y progresión (acumulación constante de discapacidad) para describir mejor cada caso. Así, es posible clasificar a un paciente, por ejemplo, como “EMSP activa con progresión” si tiene avance de discapacidad y alguna recaída o lesión nueva, o “EMRR inactiva” si está estable sin actividad detectable [8].

2.3. Conexión entre la resonancia magnética y la esclerosis múltiple

La introducción de la resonancia magnética supuso un cambio de paradigma en el diagnóstico y seguimiento de la esclerosis múltiple. Dada la naturaleza diseminada y a menudo silenciosa

de las lesiones de EM, la RM proporciona una ventana única para visualizar el daño en el sistema nervioso central que de otro modo sería difícil o imposible de detectar clínicamente. Hoy en día, la RM es considerada la prueba complementaria fundamental para la EM: cumple un rol tanto diagnóstico (para demostrar las lesiones características que sustentan los criterios de McDonald) como de monitoreo de la actividad de la enfermedad a lo largo del tiempo.

2.3.1. Papel de la RM en el diagnóstico y seguimiento de la EM

En el diagnóstico, la RM cerebral y medular permite objetivar la diseminación en espacio y tiempo propia de la EM incluso tras el primer evento clínico. Las guías actuales establecen que ciertas características de las lesiones en RM apoyan fuertemente el diagnóstico de EM, tales como: lesiones periventriculares, yuxtacorticales, infratentoriales o en médula espinal de al menos 3 mm de diámetro, en número y distribución no explicables por envejecimiento u otras patologías. La presencia de al menos una lesión en dos de estas localizaciones distintas satisface la diseminación en espacio, según los criterios de McDonald. Por su parte, la diseminación en tiempo puede demostrarse si coexisten lesiones realzantes y no realzantes en un mismo estudio (indicando que unas son activas y otras antiguas), o mediante la aparición de nuevas lesiones en un seguimiento posterior. Gracias a la alta sensibilidad de la RM, hoy es posible confirmar un diagnóstico de EM en etapas tempranas y comenzar tratamiento precozmente, lo que mejora el pronóstico. De hecho, es frecuente que pacientes con un síndrome clínicamente aislado ya presenten varias lesiones subclínicas en la RM inicial; en tales casos se habla de síndrome radiológicamente aislado (RIS) cuando hay hallazgos típicos de EM en RM sin manifestación clínica, una condición en la que cerca de la mitad de individuos desarrollarán EM clínica en unos años [8].

Para el seguimiento, la RM seriada (por lo general cada 1-2 años, o con mayor frecuencia si se evalúa cambio de tratamiento) es la mejor manera de detectar actividad de la enfermedad subclínica. Nuevas lesiones en secuencia T2 o lesiones que captan contraste en T1 indican que la enfermedad está activa, aun si no hubo nuevos síntomas. Este concepto de “actividad RM” se usa actualmente para guiar terapia: por ejemplo, un paciente en tratamiento cuyo control de RM muestra lesiones nuevas generalmente se considera candidato a escalar a un fármaco más potente. La RM puede mostrar diseminación en tiempo mucho antes de que ocurran nuevos

brotes clínicos, permitiendo una intervención temprana. Además, en pacientes en remisión, la estabilidad de la RM (sin lesiones nuevas ni captantes) es un indicador tranquilizador de buen control. Por ello, se recomienda realizar RM periódicas aun en ausencia de síntomas, ya que la EM puede “moverse” en la imagen antes que en la clínica.

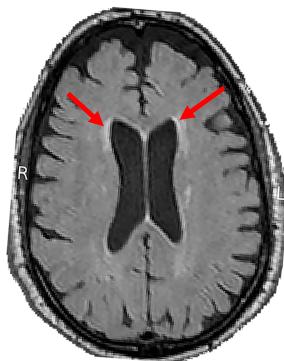


Figura 3: Corte cerebral de resonancia magnética [9].

Otro rol crucial de la RM es en el diagnóstico diferencial. Algunas enfermedades pueden simular la EM con lesiones en sustancia blanca (por ejemplo: pequeñas lesiones isquémicas por microangiopatía, vasculitis, neuromielitis óptica, encefalomielitis aguda diseminada, etc.), pero a menudo la RM ayuda a diferenciarlas. Las lesiones de EM tienden a ubicarse en regiones típicas (p. ej. adyacentes a ventrículos, cuerpo calloso, cordones posteriores medulares) y tienen morfologías características como las “dedos de Dawson”, que son lesiones alargadas periventriculares que se extienden radiariamente desde los ventrículos siguiendo las venas medulares. En la imagen 3 se aprecia un ejemplo de lesiones periventriculares en EM (flechas rojas) visibles en un corte axial FLAIR, adyacentes al cuerpo calloso, con orientación perpendicular a los ventrículos laterales -estos son los llamados dedos de Dawson, patognomónicos de desmielinización por EM. La RM también puede revelar lesiones en el cuerpo calloso que bordean el septo pelúcido (otra localización muy sugestiva de EM) y lesiones cortico-yuxtacorticales que tocan la corteza (a diferencia de lesiones isquémicas subcorticales que dejan un borde de sustancia blanca sana). En medula espinal, las lesiones de EM suelen ser pequeñas, periféricas y ocupan solo parte de la sección transversal, a diferencia de otras mielopatías inflamatorias que abarcan segmentos extensos. Todos estos hallazgos ayudan a afirmar el diagnóstico de EM y descartar imitadores.

2.3.2. Secuencias de RM utilizadas en EM y hallazgos típicos

Existen distintas secuencias de RM que juegan un papel complementario para caracterizar las lesiones de la esclerosis múltiple. Cada tipo de secuencia resalta diferentes propiedades tisulares, por lo que combinadas ofrecen una imagen integral de la enfermedad. Las principales utilizadas en protocolos de EM son:

- **Secuencias T2 y FLAIR:** Son las más sensibles para detectar las lesiones de EM. En las imágenes ponderadas en T2 (p. ej. T2 spin-eco o turbo spin-eco), las lesiones desmielinizantes típicamente aparecen como áreas hiperintensas (más brillantes) respecto al tejido cerebral normal. Esto se debe al mayor contenido de agua en las placas (por edema y pérdida de mielina). Sin embargo, en T2 el líquido cefalorraquídeo (LCR) también es hiperintenso, lo que puede dificultar la visualización de lesiones adyacentes a los ventrículos. Ahí es crucial la secuencia FLAIR (Fluid Attenuated Inversion Recovery), que es esencialmente una imagen T2 con supresión del líquido: el LCR se ve negro, permitiendo resaltar las lesiones periventriculares que de otro modo se confundirían con el líquido. La FLAIR tiene la mayor sensibilidad para las lesiones de sustancia blanca subyacentes a la corteza y junto a los ventrículos. Por ejemplo, los “dedos de Dawson” se aprecian óptimamente en imágenes FLAIR sagitales del cerebro. En general, la combinación T2/FLAIR muestra la “carga lesional total” de la EM acumulada a lo largo de la enfermedad - es decir, tanto lesiones antiguas como recientes aparecerán brillantes. Las lesiones en cerebelo o tronco encefálico a veces se ven mejor en T2 que en FLAIR (por heterogeneidad del campo magnético), pero habitualmente ambas secuencias se revisan. Un detalle importante: en la médula espinal, la secuencia FLAIR no es útil (suprime excesivamente señal, detectando solo 10 de las lesiones), por lo que allí se emplean T2 sagitales convencionales.
- Con **T2/FLAIR** se puede estimar la **diseminación en espacio**: típicamente se buscan lesiones en al menos dos de las áreas clave (periventricular, yuxtacortical, infratentorial, medular). También estas secuencias ayudan a cuantificar la carga lesional, que tiene correlación con la historia de brotes y cierta relación (no lineal) con la discapacidad. No obstante, no distinguen si una lesión es activa o antigua; ambas se ven hiperintensas de

forma similar. En la práctica, la emergencia de nuevas lesiones T2/FLAIR en controles seriados es interpretada como evidencia de actividad de la enfermedad y falla terapéutica si el paciente estaba en tratamiento.

- **Secuencia T1 sin contraste:** En las imágenes ponderadas en T1 (donde sustancia blanca normal aparece intermedia y LCR aparece oscuro), las lesiones de EM suelen visualizarse isointensas u hipointensas según su estado. Las lesiones activas o recientes a veces apenas se distinguen en T1 nativo, pero las lesiones crónicas de larga data frecuentemente dejan áreas de señal disminuida denominadas “agujeros negros” por su aspecto oscuro. Estos black holes en T1 indican que ha habido daño axonal severo y gliosis en esa zona, representando a menudo lesiones irreversibles. Aproximadamente, una proporción de las lesiones que inicialmente se ven en T2 con el tiempo se convierten en agujeros negros permanentes en T1 (se estima que alrededor del 20-30). La carga acumulada de agujeros negros correlaciona con atrofia cerebral y discapacidad, por lo que se considera un marcador de neurodegeneración. En un estudio se definieron dos clases de lesiones T1 hipointensas: las temporales (que pueden remielinizarse parcialmente recobrando señal en T1) y las permanentes (verdaderos agujeros negros crónicos asociados a pérdida axonal). Identificar muchas lesiones hipointensas en T1 sugiere EM de larga evolución o agresiva. Por otro lado, un hallazgo típico en T1 es la atrofia: con los años, se observa disminución del volumen cerebral (ensancho de surcos y ventrículos). La atrofia cerebral puede medirse cuantitativamente y se ha propuesto como biomarcador de progresión; algunos tratamientos parecen frenar la tasa de atrofia.
- **Secuencia T1 con contraste (gadolino):** La administración intravenosa de gadolinio (agente de contraste paramagnético) es fundamental para evaluar la actividad inflamatoria actual. El gadolinio no atraviesa la barrera hematoencefálica intacta, pero en las lesiones de EM activas dicha barrera está rota por la inflamación, permitiendo que el contraste entre y se acumule temporalmente en la lesión. En consecuencia, en una imagen T1 post-contraste, las lesiones activas recientes aparecen como zonas brillantes realzadas (enhancing lesions) típicamente con forma oval o anillo parcial si la inflamación está en los bordes de una lesión crónica. Este realce dura unas semanas hasta meses tras el inicio de la lesión, tras lo cual la barrera hematoencefálica se restablece y el ga-

dolinio ya no penetra en esa placa. Por ello, las lesiones que captan gadolinio indican actividad reciente (aguda). La presencia de lesiones captantes es sinónimo de un brote activo (clínico o subclínico). En pacientes sin tratamiento, es común encontrar varias lesiones con gadolinio en el debut; con los DMT efectivos, la ausencia de captación se vuelve un objetivo (remisión radiológica). En RM de control, la aparición de cualquier lesión gadolinio-positiva es criterio de actividad de enfermedad. Además, las lesiones crónicas no captantes pueden empezar a realzar de nuevo si sufren reactivación inflamatoria. En suma, el gadolinio permite distinguir lesiones nuevas vs. antiguas en un momento dado. Por ejemplo, un paciente puede tener muchas placas T2, pero solo una capta gadolinio; esa es la lesión responsable del brote reciente, mientras que las demás son “cicatrices” de brotes pasados [6].

Otras secuencias complementarias que pueden emplearse incluyen: proton-density (PD), útil en médula para distinguir lesiones (la PD pondera protones y puede dar contraste diferente a T2); secuencias de doble inversión recuperación (DIR) que suprimen simultáneamente líquido y sustancia blanca para resaltar lesiones corticales; imágenes de gradiente echo/T2 (SWI)* para detectar depósitos de hierro que pueden verse alrededor de lesiones crónicas; y métodos cuantitativos como volumetría cerebral para medir atrofia, o transferencia de magnetización (MT) y DTI para evaluar daño difuso en tejido aparentemente normal. En investigación, la RM de 7 Tesla ha revelado fenómenos como depósitos de hierro en el centro de algunas lesiones (el llamado signo del anillo central) que podría diferenciar lesiones de EM de otras. Sin embargo, en la práctica clínica rutinaria, las secuencias esenciales siguen siendo T2, FLAIR, T1 con y sin contraste.

En resumen, la resonancia magnética proporciona un perfil multiplanar y multicontraste de la esclerosis múltiple: las secuencias T2/FLAIR muestran la carga lesional acumulada (cicatrices de desmielinización a lo largo de la vida del paciente), las secuencias T1 identifican las huellas de daño irreversible (agujeros negros) y anatomía cerebral, y la adición de gadolinio en T1 delata las lesiones activas con ruptura de la barrera hematoencefálica en ese momento. Esta combinación permite no solo diagnosticar la EM, sino también cuantificar su actividad (nuevas lesiones, lesiones activas) y evaluar su cronicidad (lesiones antiguas, atrofia). La RM es, por tanto, tanto los “ojos” del neurólogo para ver la enfermedad dentro del cerebro, como

el “termómetro” para medir si la enfermedad está bajo control o encendida, guiando así las decisiones terapéuticas.

2.4. Inteligencia Artificial (IA)

La Inteligencia Artificial (IA) es un campo interdisciplinar de la informática que se dedica al estudio y construcción de sistemas capaces de realizar tareas que normalmente requerirían inteligencia humana. De forma general, IA abarca una amplia gama de métodos y objetivos, desde la representación del conocimiento y el razonamiento lógico hasta el aprendizaje automático y la percepción. Una definición rigurosa en términos formales conceptualiza la IA como el diseño de agentes inteligentes, entendiendo por agente una entidad (algoritmo o sistema) que percibe su entorno a través de sensores y actúa en dicho entorno mediante actuadores, de modo que sus acciones maximizan una medida de desempeño esperada. Esta noción, propuesta por Russell y Norvig, enmarca la inteligencia artificial como la búsqueda de comportamientos racionales: el agente debe decidir y obrar de forma óptima según un criterio determinado (por ejemplo, obtener la solución correcta, ganar un juego, clasificar correctamente una imagen, etc.). En términos matemáticos, se puede modelar un agente como una función f que mapea del conjunto de percepciones P al conjunto de acciones A , $f : P \rightarrow A$, y que está diseñada para maximizar una función de utilidad o medida de rendimiento U definida sobre secuencias de percepciones y acciones. Un agente racional ideal seleccionaría en cada estado la acción a^* tal que maximiza la esperanza de utilidad futura:

$$a^* = \arg \max_{a \in A} E[U | s, a]$$

donde s representa el estado actual percibido del entorno. Esta formulación proporciona un marco matemático general para la IA, aunque en la práctica el diseño de agentes inteligentes enfrenta desafíos de complejidad computacional e incertidumbre que han llevado al desarrollo de subcampos especializados.

2.4.1. Aprendizaje Automático (Machine Learning)

Tal y como se muestra en la Figura 4, la inteligencia artificial engloba todos los sistemas construidos por el hombre que son capaces de actuar de una forma inteligente, similar a la de los

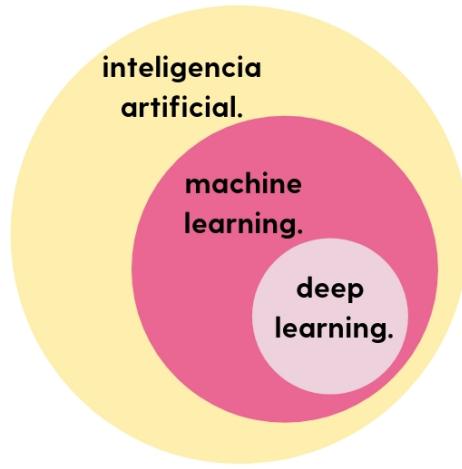


Figura 4: Jerarquía Inteligencia Artificial [10].

seres humanos. El aprendizaje automático, por su parte, se centra en el desarrollo de algoritmos capaces de aprender y mejorar su desempeño a partir de datos. Tom Mitchell (1997) proporciona una definición formal ampliamente citada: “un programa de computadora aprende de la experiencia E con respecto a cierta tarea T y medida de rendimiento P si su rendimiento en T , medido por P , mejora con E . En otras palabras, en lugar de programar explícitamente todas las reglas para realizar una tarea, en ML se entrena un modelo con datos ejemplos para que ajuste sus parámetros internos y pueda generalizar comportamientos. Desde un punto de vista matemático, el aprendizaje automático suele implicar la optimización de alguna función objetivo (función de pérdida o de costo) utilizando técnicas estadísticas y numéricas [11].

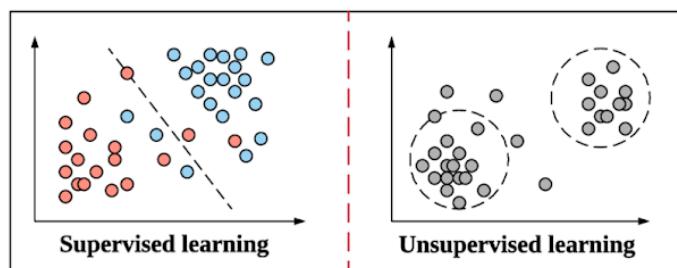


Figura 5: Aprendizaje Supervisado vs Aprendizaje no Supervisado [12].

2.4.2. Aprendizaje Supervisado

En aprendizaje supervisado, se parte de un conjunto de datos de entrenamiento $(x_i, y_i)_{i=1}^N$, donde cada entrada $x_i \in \mathcal{X}$ tiene una salida asociada $y_i \in \mathcal{Y}$ conocida (la “etiqueta” correcta). El objetivo es encontrar un modelo o función $f : \mathcal{X} \rightarrow \mathcal{Y}$, parametrizado por algún vector de parámetros θ , que sea capaz de predecir la etiqueta y de nuevas entradas x no vistas con la mayor precisión posible. Formalmente, se intenta aproximar la relación $y = f(x)$ subyacente a los datos. Para ello se define una función de pérdida $L(f(x_i; \theta), y_i)$ que cuantifica el error de la predicción $f(x_i; \theta)$ del modelo con respecto al valor verdadero y_i . El aprendizaje consiste en minimizar la pérdida promedio (o alguna agregación) sobre el conjunto de entrenamiento [13]. Esto se denomina minimización del riesgo empírico:

$$J(\theta) = \frac{1}{N} \sum_{i=1}^N \mathcal{L}(f(x_i; \theta), y_i)$$

y el modelo óptimo se obtiene buscando:

$$\theta^* = \arg \min_{\theta} J(\theta) = \arg \min_{\theta} \frac{1}{N} \sum_{i=1}^N \mathcal{L}(f(x_i; \theta), y_i)$$

En problemas de regresión (salida continua), una elección común es la pérdida cuadrática $L(f(x), y) = |f(x) - y|^2$. En problemas de clasificación (salida discreta o categórica), es habitual emplear la pérdida de entropía cruzada o log-loss, por ejemplo:

$$\mathcal{L}(f(x), y) = - \sum_{c=1}^C y_c \log f_c(x)$$

donde y_c es 1 si la clase verdadera es c (codificación one-hot) y $f_c(x)$ es la probabilidad predicha de la clase c . Esta función penaliza fuertemente las predicciones con baja probabilidad asignada a la clase correcta. Mediante la minimización de la pérdida, el algoritmo ajusta los parámetros θ para explicar los ejemplos conocidos. Para llevar a cabo esta optimización se emplean típicamente métodos de descenso por gradiente u otros algoritmos de optimización numérica. El resultado es un modelo capaz de generalizar, es decir, de predecir correctamente y para nuevas entradas x no vistas (en la medida en que los datos de entrenamiento sean representativos de los futuros datos). Como ejemplos clásicos de algoritmos de aprendizaje supervisado se pueden mencionar:

- **Regresión lineal y logística:** modelan $f(x)$ como una combinación lineal de características seguido de, en el caso logístico, una función sigmoide para producir una probabilidad. Ambos ajustan sus coeficientes minimizando una pérdida (cuadrática en la lineal, o log-loss en la logística) [14].
- **Máquinas de Vectores de Soporte (SVM):** encuentran un hiperplano separador óptimo entre clases maximizando el margen, con formulación de optimización convexa cuadrática.
- **Árboles de decisión y bosques aleatorios:** realizan particiones recursivas del espacio de características para reducir la impureza (por ejemplo, la entropía) en las hojas, ajustándose de forma no lineal a los datos.
- **Redes Neuronales Artificiales (ANN)** de tipo feed-forward (que discutiremos en la sección de Aprendizaje Profundo): combinan muchas unidades neuronales para aproximar funciones complejas de los datos.

2.4.3. Aprendizaje No Supervisado

En aprendizaje no supervisado tal y como se aprecia en la Figura 5, no se proporcionan etiquetas ni salidas deseadas al algoritmo; solo se cuenta con un conjunto de datos $x_{i=1}^N$ con $x_i \in \mathcal{X}$. El objetivo es descubrir estructura subyacente o patrones ocultos en los datos. Dado que no hay un valor objetivo explícito a predecir, el aprendizaje no supervisado se formula típicamente como un problema de modelado de la distribución generadora de los datos o de agrupamiento (clustering) de ejemplos similares [13].

Formalmente, muchos métodos no supervisados intentan estimar una función $g : \mathcal{X} \rightarrow \mathcal{Z}$ que transforma los datos de entrada en una representación \mathcal{Z} más útil o más comprimida, o bien aprenden directamente propiedades de la distribución $P(X)$ de donde provienen los datos. Por ejemplo, en la modelación de densidad, se busca un modelo $p_\theta(x)$ tal que $p_\theta(x) \approx p_{\text{datos}}(x)$, pudiendo emplear técnicas como modelos de mezcla (e.g., mezclas de gaussianas), estimación kernel de densidad, o modelos auto-regresivos. Alternativamente, en agrupamiento, se busca una partición de \mathcal{X} en subconjuntos (clústeres) C_1, C_2, \dots, C_K , de modo que los ejemplos dentro de un mismo clúster C_k sean más similares entre sí que con los de otros clústeres.

Un criterio típico es minimizar la dispersión intraclúster. Por ejemplo, el algoritmo clásico k-means resuelve:

$$\min_{\mu_1, \dots, \mu_K} \sum_{i=1}^N \min_{k \in \{1, \dots, K\}} \|x_i - \mu_k\|^2$$

donde μ_k es el centroide (media) del clúster k . Aquí K (el número de clústeres) es un hiperparámetro dado. La solución alterna entre asignar cada punto x_i al centroide más cercano y recomputar los centroides como la media de los puntos asignados, convergiendo a un mínimo local.

2.4.4. Aprendizaje Profundo (Deep Learning)

Dentro del aprendizaje automático, en la última década ha cobrado protagonismo el llamado Aprendizaje Profundo (Deep Learning), una rama que emplea modelos con múltiples capas de transformaciones no lineales, generalmente basados en redes neuronales artificiales de muchas capas (de ahí “profundas”). La idea clave del deep learning es que aumentando la profundidad (número de capas intermedias) de un modelo, este puede aprender representaciones jerárquicas de los datos: las primeras capas capturan características de bajo nivel, y capas sucesivas construyen características de nivel más alto a partir de las anteriores. Esto ha permitido lograr avances sin precedentes en tareas complejas como el reconocimiento de voz, visión por computador y procesamiento del lenguaje natural [15]. Formalmente, un modelo de aprendizaje profundo puede verse como la composición de varias funciones no lineales anidadas. Por ejemplo, una red neuronal profunda define una función:

$$y = f^{(L)} \left(f^{(L-1)} \left(\dots f^{(2)} \left(f^{(1)}(x) \right) \dots \right) \right)$$

donde cada $f^{(j)}$ representa la transformación realizada por la capa j -ésima, y L es el número total de capas (por convención, $f^{(1)}$ sería la capa de entrada y $f^{(L)}$ la de salida). Cada capa típicamente consiste en una transformación afín seguida de una no lineal (una activación). Por ejemplo, en una red neuronal densa clásica (perceptrón multicapa o MLP), la capa j realiza:

$$a^{(j)} = f \left(W^{(j)} a^{(j-1)} + b^{(j)} \right)$$

donde $a^{(j-1)}$ es el vector de activaciones de la capa anterior (con $a^{(0)} = x$ la entrada), $W^{(j)}$ es la matriz de pesos de la capa j y $b^{(j)}$ su vector de sesgos; $f(\cdot)$ es una función no lineal aplicada

componente a componente (por ejemplo, la rectificación $ReLU(z) = \max(0, z)$, sigmoide, $tanh$, etc.). Este tipo de arquitectura de red neuronal profunda es capaz de aproximar funciones muy complejas; de hecho, teoremas universales de aproximación establecen que una red con suficientes neuronas puede aproximar cualquier función continua en un dominio compacto con la precisión deseada. La profundidad adicional permite hacerlo de forma más eficiente, reutilizando características intermedias útiles para múltiples tareas.

2.4.5. Redes Neuronales Convolucionales (CNN)

Las Redes Neuronales Convolucionales (o CNN, por Convolutional Neural Networks) son una clase de arquitecturas de aprendizaje profundo diseñadas específicamente para procesar datos con estructura de cuadrícula local, como imágenes 2D (también se aplican a series temporales 1D, videos 3D, etc.). A diferencia de los perceptrones multicapa densos, en las CNN las neuronas de una capa están organizadas en mapas de características (feature maps) y conectadas solo a regiones locales de la capa anterior, explotando la idea de conectividad local y compartición de pesos. Esta arquitectura está inspirada en el procesamiento visual biológico, imitando parcialmente la organización del córtex visual animal en el que neuronas individuales responden a subregiones del campo visual.

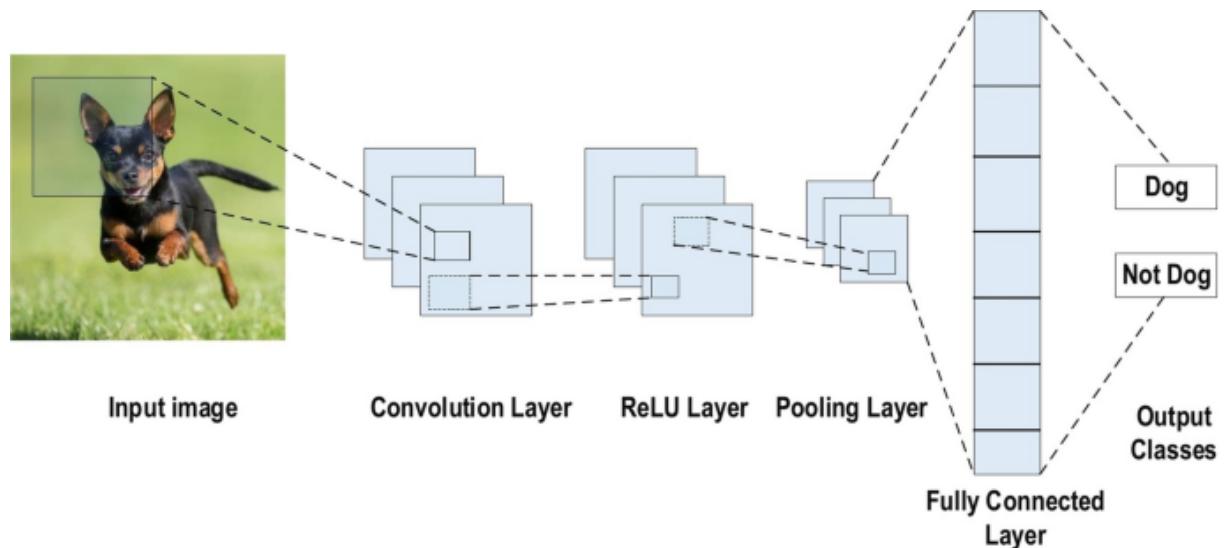


Figura 6: Visualización de una red convolucional [16].

Una CNN típica como la de la Figura 6 está compuesta por una secuencia de capas de convo-

lución, intercaladas con capas de pooling (submuestreo), y culminando en una o varias capas completamente conectadas (densas) que producen la salida final (por ejemplo, una distribución de probabilidades sobre clases). Las capas de convolución son el núcleo de la red: cada capa aplica varios filtros convolutivos (también llamados núcleos o kernels) a la salida de la capa previa. Un filtro es esencialmente una pequeña matriz de pesos W^k (por ejemplo 3×3 pixels, en imágenes) que se hace convolucionar a lo largo de la entrada - es decir, se calcula el producto puntual (dot product) entre el filtro y cada región de la entrada del mismo tamaño, produciendo así un mapa de activación (feature map) que indica dónde (y con qué intensidad) aparece en la entrada el patrón que detecta el filtro. Cada filtro W^k va acompañado de un sesgo b^k escalar, y la salida convolutiva pasa por una función de activación no lineal (p. ej. ReLU) antes de enviarse a la siguiente capa. Matemáticamente, si x es la entrada de una capa (por ejemplo, un conjunto de mapas de características de la capa anterior) y (W^k, b^k) define el filtro k -ésimo, la operación de convolución produce un mapa h^k dado por:

$$h_k = f(W_k * x + b_k)$$

donde $*$ denota la operación de convolución (suma de productos sobre una ventana local de x) y $f(\cdot)$ es la activación no lineal. Cada h^k es de menor dimensión espacial que x (dependiendo del tamaño del filtro y si se aplica relleno o padding), y múltiples filtros producen múltiples mapas h^1, h^2, \dots, h^K en cada capa. Gracias a la compartición de pesos, el mismo filtro W^k se aplica en todas las posiciones de la entrada, lo que reduce drásticamente el número de parámetros y hace a la red más eficiente y con cierto grado de invariancia a translaciones en la entrada. Por ejemplo, en una imagen, un filtro entrenado podría detectar un borde horizontal, y ese mismo conjunto de pesos se utiliza para escanear toda la imagen en busca de dicho patrón. Las capas de pooling realizan una reducción espacial de la resolución de los mapas de características, resumiendo regiones cercanas en un solo valor (por ejemplo, max-pooling toma el valor máximo en cada región de 2×2). El pooling introduce invariancia a pequeñas translaciones o distorsiones de la entrada y reduce el número de variables en capas posteriores, controlando además el sobreajuste. Existen variantes como average pooling, global pooling, etc., y en algunos diseños modernos se prescinde del pooling explícito utilizando en su lugar convoluciones con mayor stride (paso).

Finalmente, tras varias capas convolutivas (la profundidad depende de la complejidad de la

tarea y del tamaño de la entrada), una CNN suele aplanar los mapas de características resultantes en un vector y conectarlo a una o más capas densas tradicionales. Estas capas finales actúan como un clasificador que utiliza las características extraídas por las capas anteriores para tomar la decisión. En problemas de clasificación, la última capa suele ser una softmax que produce probabilidades sobre cada clase. El entrenamiento de una CNN se realiza de manera supervisada, ajustando todos los pesos (tanto de convolución como de las capas densas) mediante retropropagación, que en este caso incluye la propagación de gradientes a través de las operaciones de convolución y pooling (las cuales son diferenciables).

2.5. Red nnUNet

U-Net es una red neuronal convolucional introducida en 2015 por Ronneberger, diseñada específicamente para la segmentación de imágenes biomédicas. Su estructura sigue un patrón encoder-decoder simétrico con abundantes conexiones de salto (skip connections) entre la fase de contracción (codificación) y la fase de expansión (decodificación) tal y como se aprecia en la Figura 7. Estas conexiones permiten transferir características de bajo nivel desde la etapa de contracción a las capas correspondientes de la etapa expansiva, conservando información de alta resolución espacial que de otro modo se perdería tras sucesivas operaciones de pooling. La arquitectura resultante tiene forma de “U” (de ahí su nombre), con la ruta de contracción reduciendo progresivamente la resolución mediante convoluciones y downsampling, mientras que la ruta de expansión restaura la resolución original mediante upsampling y combinando las características transmitidas por los atajos. Gracias a este diseño, U-Net logra segmentaciones píxel a píxel precisas incluso con un número limitado de imágenes de entrenamiento, superando a arquitecturas previas como FCN y convirtiéndose en la base de multitud de variantes posteriores en segmentación médica [17].

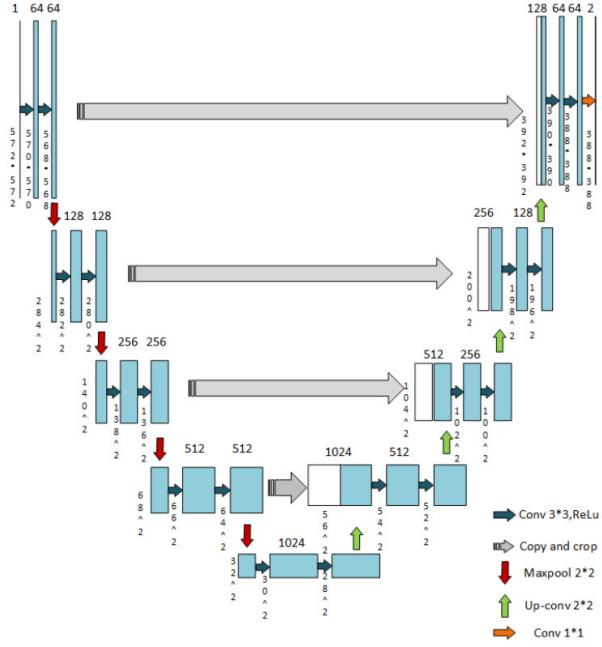


Figura 7: Arquitectura de la U-Net [18].

2.5.1. Motivación para el desarrollo de nnUNet

A pesar del éxito de U-Net, diseñar e implementar una solución de segmentación efectiva para cada nueva aplicación seguía requiriendo un esfuerzo sustancial por parte de expertos. Cada conjunto de datos médicos presenta propiedades distintas -dimensionalidad 2D vs 3D, resolución de voxel (espaciado) anisótropa o isotrópica, distintos contrastes e intensidades, número de clases, tamaños de órganos o lesiones, etc.- lo que obliga a ajustar manualmente multitud de parámetros (arquitectura específica, pre-procesamiento, tasa de aprendizaje, tamaño de lote, estrategias de aumento de datos, entre otros). Este proceso de prueba y error depende en gran medida de la intuición y experiencia del investigador, y aun pequeñas decisiones subóptimas pueden degradar significativamente el rendimiento de la segmentación. En la literatura, miles de trabajos han propuesto variaciones sobre U-Net para mejorar resultados en conjuntos de datos particulares, lo que dificulta identificar qué mejoras son realmente generales y cuáles solo ajustan casos específicos.

2.5.2. Diseño y arquitectura general de nnUNet

En esencia, nnUNet no propone cambios radicales a la arquitectura U-Net, sino que utiliza U-Net como plantilla base y se centra en adaptarla dinámicamente a los datos y el problema en cuestión. El concepto “no new U-Net” subraya que la fortaleza del método reside más en la configuración que en la invención de nuevos bloques de red. En la implementación original, nnUNet considera tres configuraciones de modelo basadas en U-Net:

- **U-Net 2D:** una arquitectura U-Net convencional que opera sobre imágenes bidimensionales (ej. procesa cada corte axial de una volumetría MRI de forma independiente). Es útil cuando los datos presentan resolución axial mucho mayor que la longitudinal (volúmenes muy anisótropos) o cuando el fenómeno a segmentar es esencialmente 2D.
- **U-Net 3D:** una U-Net totalmente tridimensional, que recibe subvolúmenes (patches 3D) como entrada y explota la información contextual en las tres dimensiones espaciales. Suele proporcionar mejor coherencia volumétrica en segmentaciones 3D cuando la resolución es isotrópica o casi isotrópica, pero está limitada por la memoria GPU disponible dado el mayor tamaño de los tensores volumétricos.
- **U-Net 3D en cascada:** se compone de dos redes U-Net 3D encadenadas. La primera red opera sobre una versión reducida (remuestreada a menor resolución) del volumen, generando un mapa de segmentación inicial con mayor campo de visión (contexto global). Luego, sus resultados se reescalan y se concatenan como canales de entrada adicionales a una segunda U-Net 3D que trabaja a resolución completa, refinando los detalles locales. Esta cascada se activa típicamente cuando el tamaño de patch factible en 3D completo cubre solo una parte muy pequeña del volumen original (por restricciones de memoria), situación común en imágenes con vértices muy anisótropos o volúmenes muy grandes. La primera etapa de baja resolución asegura capturar el contexto del volumen entero, mientras la segunda etapa se enfoca en la precisión local [19].

2.5.3. Configuración automática de nnU-Net

Una vez definida la arquitectura U-Net genérica y las posibles variantes (2D, 3D, cascada), el aporte crucial de nnUNet es la configuración automática de la red y del proceso de entrenamiento para adaptarlos a un nuevo conjunto de datos que se observa en la Figura 8. Para lograr esto, nnUNet descompone los hiperparámetros y decisiones de diseño en tres categorías: parámetros fijos, parámetros basados en reglas (derivados de los datos) y parámetros empíricos, combinándolos en un plan experimental completo.

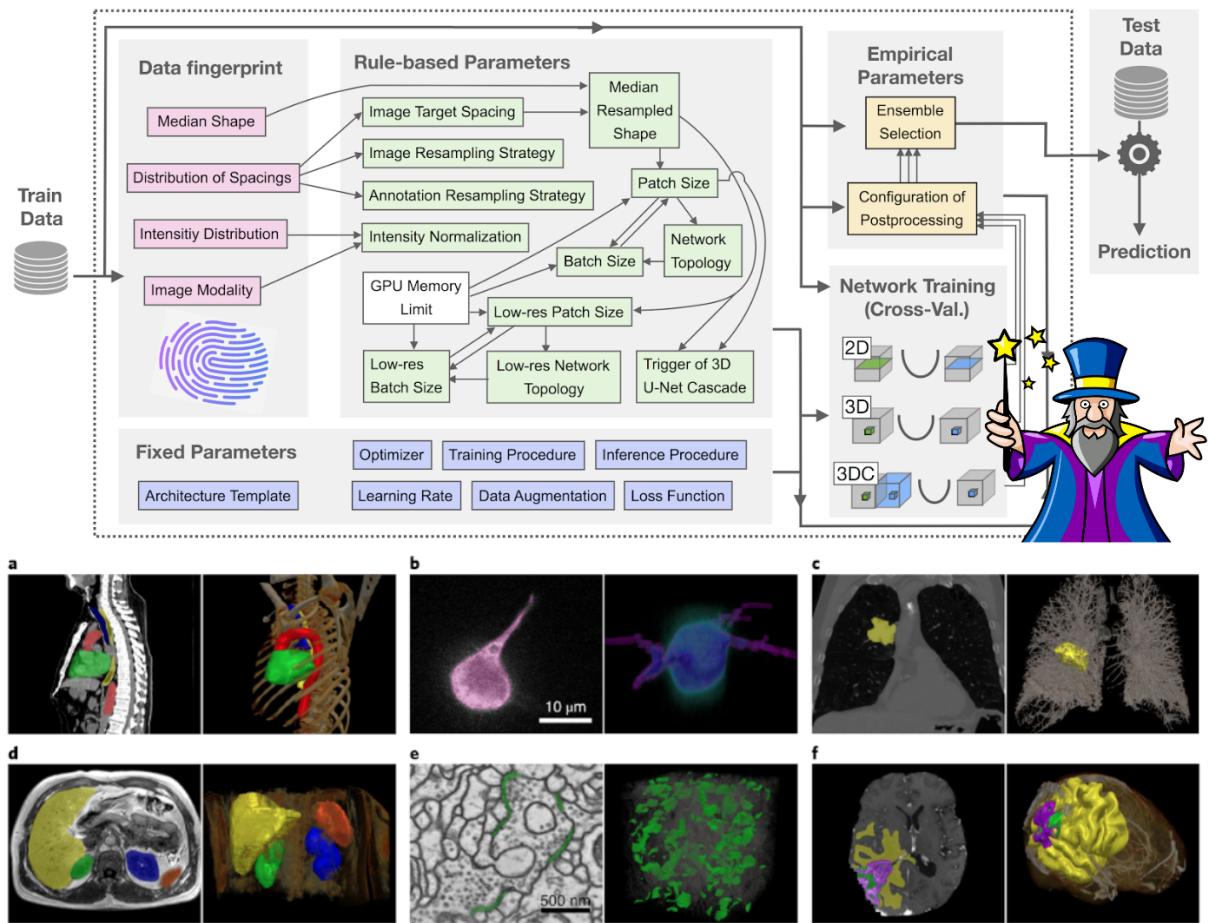


Figura 8: Proceso de configuración automática de la nnUNet [19].

- **Parámetros fijos del pipeline:** son decisiones constantes, predefinidas por el framework porque han demostrado funcionar bien en una amplia variedad de tareas. Incluyen el uso de una arquitectura U-Net estándar como base, el optimizador SGD con momento 0,99 y tasa de aprendizaje inicial 0,01 con decaimiento polinomial, y una función de pér-

dida combinada (entropía cruzada + Dice). También se aplican técnicas fijas de aumento de datos (como flips, deformaciones, variaciones de intensidad) y un procedimiento de inferencia estándar con ventanas deslizantes y, si aplica, ensambles.

- **Parámetros basados en el conjunto de datos (reglas heurísticas):** se calculan automáticamente a partir de la “huella” del conjunto de datos (resolución, tamaño de estructuras, número de clases, modalidad, etc.). A partir de ello, se decide el remuestreo, el tamaño de patch, la arquitectura adaptada (profundidad, filtros, etc.), la estrategia de normalización y si es necesario usar un modelo en cascada. Estas decisiones permiten adaptar de forma óptima el pipeline al contenido y resolución del conjunto de datos.
- **Parámetros empíricos y ajustes posteriores al entrenamiento:** se afinan tras realizar una validación cruzada de 5 pliegues sobre el conjunto de entrenamiento. Se evalúan distintas configuraciones (2D, 3D, cascada y sus combinaciones) y se selecciona automáticamente la que ofrece mejor desempeño promedio (Dice). Además, se aplica un post-procesamiento empírico como la eliminación de pequeñas regiones si mejora las métricas sin afectar negativamente a ninguna clase.

2.6. Arquitectura YOLO

You Only Look Once (YOLO) es una familia de modelos de detección de objetos en tiempo real que ha evolucionado significativamente desde su primera versión en 2015 hasta las versiones más recientes (YOLOv9, YOLOv10, YOLOv11). Cada generación ha introducido mejoras arquitectónicas y técnicas para aumentar la precisión (mAP), la velocidad de inferencia (FPS) y la capacidad de abordar tareas de visión más allá de la detección clásica, como la segmentación de instancias o semántica [20].

2.6.1. YOLOv1: arquitectura inicial unificada

La primera versión, YOLOv1 [18](#), propuso una arquitectura unificada de una sola etapa que reformula la detección como un problema de regresión directa de la imagen a las cajas delimitadoras y clases. La red YOLOv1 está inspirada en GoogLeNet e incluye 24 capas convolucionales

seguidas por 2 capas totalmente conectadas. Se emplean convoluciones 1×1 intercaladas para reducir la dimensionalidad de características, seguidas de convoluciones 3×3 con activación Leaky ReLU. La salida de la red es un tensor $S \times S \times [B \times (4 + 1) + C]$, donde S es la división de la imagen en celdas (por ejemplo 7×7), B es el número de cajas predichas por celda (por ejemplo 2) y C es el número de clases. En cada celda se predicen B cajas con sus coordenadas (x, y, w, h) normalizadas y un puntaje de confianza, además de las probabilidades de pertenencia a C clases para la celda. De esta manera, YOLOv1 realiza detección y clasificación en una sola pasada de la red, razonando de forma global sobre la imagen completa. En entrenamiento, se utiliza un término de error cuadrático que pondera más las pérdidas de localización (usando factores $\lambda_{coord} = 5$) y reduce la influencia de falsas alarmas en celdas sin objeto ($\lambda_{noobj} = 0,5$). Pese a su simplicidad y gran velocidad (45 FPS en su modelo base y hasta 155 FPS en una variante rápida), YOLOv1 obtenía una precisión mAP moderada (63,4 % mAP en VOC 2007), inferior a métodos de dos etapas de la época (p.ej. Faster R-CNN) pero con muchas menos detecciones de fondo falsas. YOLOv1 demostró la viabilidad de la detección unificada en tiempo real, aunque presentaba dificultades para localizar objetos pequeños y solapados debido a su salida de baja resolución (7×7) y la asignación rígida de una celda por objeto [21].

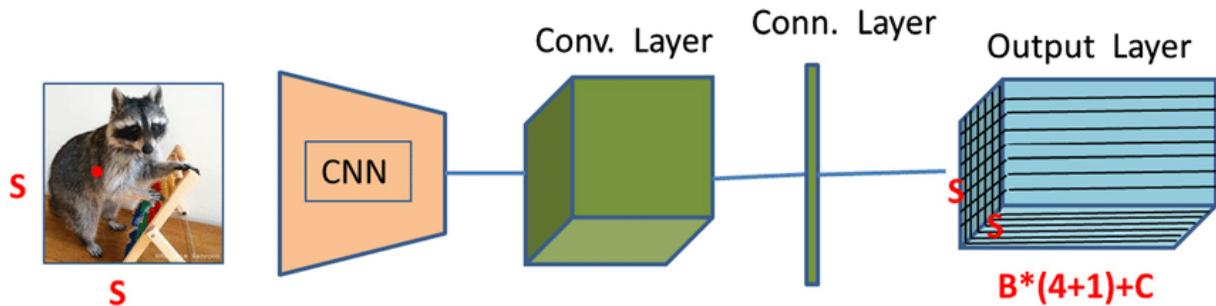


Figura 9: Arquitectura de YOLOv1 [21].

2.6.2. YOLOv2: mejoras de rendimiento con anclas y nueva espalda (Darknet-19)

YOLOv2 (también conocido como YOLO9000) se presentó en 2016/2017 con numerosas mejoras para aumentar la exactitud y rapidez. En primer lugar, se introdujo un nuevo backbone de características: Darknet-19, una CNN de 19 capas convolucionales y 5 max-pool entrenada en ImageNet. Esto reemplazó las 24 conv de YOLOv1 por una arquitectura más profunda pero eficiente, con batch normalization en todas las capas, lo que aceleró la convergencia y

añadió estabilidad. En segundo lugar, YOLOv2 incorporó el concepto de cajas ancla (anchor boxes) como en Faster R-CNN: en lugar de predecir coordenadas absolutas por celda, la red predice offsets respecto de anclas predefinidas de diferentes formas. Se utilizó k-means sobre datos de entrenamiento para generar 5 anclas adecuadas (en VOC), permitiendo a YOLOv2 predecir 5 cajas por celda con coordenadas (t_x, t_y, t_w, t_h) refinando la posición/escala de cada ancla. Las coordenadas finales se calculan aplicando sigmoide a t_x, t_y (acotados entre 0 y 1 dentro de la celda) y exponenciando t_w, t_h escalados por el tamaño de ancla. Además, YOLOv2 añadió técnicas como: passthrough layer (conexión de características finas) - se reutilizaron mapas de características de resolución intermedia (por ej. 26×26) concatenándolos con mapas 13×13 para preservar información de objetos pequeños y entrenamiento multi-escala, variando aleatoriamente el tamaño de entrada cada 10 iteraciones (entre 320 y 608 px) para hacer la red robusta a distintas resoluciones de inferencia. También se eliminaron las capas totalmente conectadas, haciendo que la predicción dependa solo de convoluciones (por lo que la red puede aceptar entradas de tamaño variable). Con estas mejoras, YOLOv2 logró 78,6 % mAP en VOC2007, superando ampliamente a YOLOv1 (63,4 %) y acercándose a la precisión de detectores más pesados de dos etapas de su tiempo, pero manteniendo decenas de FPS en velocidad. En la práctica, YOLOv2 ofreció un mejor equilibrio precisión-rapidez gracias al uso de batch norm, anclas y mejores características base. Un aporte adicional fue YOLO9000, un modelo YOLOv2 entrenado simultáneamente en detección (COCO) y clasificación (ImageNet) para detectar hasta 9000 clases mediante un truco de árbol jerárquico de clases, demostrando la escalabilidad del enfoque YOLO [22].

2.6.3. YOLOv3: Red residual más profunda y detección multi-escala

YOLOv3 (2018) supuso otra revisión importante, aumentando la capacidad de la red para no quedarse atrás respecto a los detectores avanzados de la época. Se introdujo un nuevo backbone Darknet-53, una CNN de 53 capas convolucionales con conexiones residuales tipo ResNet. Darknet-53 eliminó capas de pooling reemplazándolas por convoluciones de paso (stride) 2, y empleó bloques residuales (dos conv 3×3 con un atajo) repetidos, logrando así extraer características más ricas y profundas sin degradación del gradiente. Otra innovación clave fue la detección multi-escala inspirada en FPN: en vez de solo predecir en la última capa, YOLOv3

realiza predicciones en tres escalas diferentes. La característica de la etapa final (13×13 con 1024 filtros) produce detecciones de objetos grandes (salida y_1); características de una etapa intermedia (26×26 con 512 filtros) tras combinarse con mapas de la ruta de abajo-arriba producen detecciones de objetos medianos (y_2); y características aún más finas (52×52 con 256 filtros, combinadas con mapas iniciales) producen detección de objetos pequeños (y_3). En cada escala se asignan 3 anclas (para un total de 9, determinados por k-means) distribuidos en tamaños grandes, medianos o pequeños respectivamente. La cabeza de detección de YOLOv3 usa una función sigmoide para predecir el puntaje de objeto (objectness) de cada caja en lugar de un softmax global por celda, permitiendo manejar detecciones múltiples por celda. También se prescindió del softmax para las clases: cada clase se predice con un clasificador binario independiente (lo que mejora la generalización a múltiples etiquetas por objeto). YOLOv3-spp, una variante con un módulo de Spatial Pyramid Pooling insertado en el cuello, mejoró ligeramente el AP₅₀ y sirvió de base para trabajos posteriores. Evaluado en COCO 2017, YOLOv3 alcanzó 33-36 % AP (36,2 % AP para YOLOv3-spp) con 20-30 FPS de velocidad, posicionándose en el estado del arte en detección en tiempo real para 2018. La Figura 10 ilustra la arquitectura de YOLOv3 con sus tres salidas a distintas escalas.

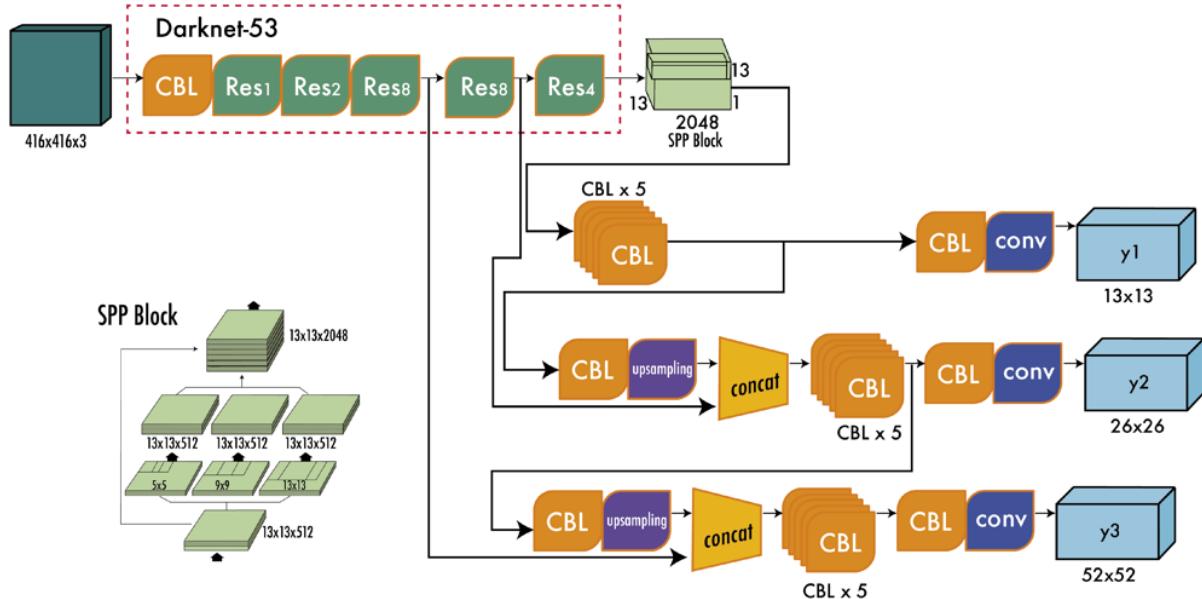


Figura 10: Arquitectura de YOLOv3 [23].

En resumen, YOLOv3 consolidó a YOLO como familia competitiva mediante una red más profunda con mejores características (Darknet-53) y la capacidad de detectar objetos pequeños,

medianos y grandes por separado en distintas capas. Estas mejoras redujeron las falsas negativas en objetos pequeños y mejoraron el AP global, a costa de un ligero incremento de complejidad computacional (75 capas en total y más operaciones que YOLOv2). No obstante, YOLOv3 siguió siendo considerablemente más rápido que los detectores de dos etapas, manteniendo la filosofía de inferencia única por imagen [23].

2.6.4. YOLOv4: CSPDarknet y trucos para el estado del arte (2020)

Tras YOLOv3, Joseph Redmon dejó de trabajar en YOLO, y la comunidad continuó su desarrollo. YOLOv4, publicado por Bochkovskiy et al. en 2020, introdujo cambios sustanciales que llevaron a YOLO al estado del arte en detección en tiempo real. La arquitectura de YOLOv4 se ilustra en la Figura 11. En el backbone, YOLOv4 implementó CSPDarknet53, una variante de Darknet-53 con bloques Cross-Stage Partial (CSP) que partitionan la característica base y fusionan una parte residual más adelante, reduciendo la duplicación de gradiente y la carga computacional. CSPDarknet53 utiliza activación Mish en sus capas (denotadas como CBM: Conv+BN+Mish) para mejorar la capacidad de aprendizaje. En lugar de procesar secuencialmente todas las características dentro de un bloque residual, la técnica CSP separa la entrada en dos rutas: una se procesa con varias convoluciones mientras la otra se mantiene como atajo y luego se concatenan, lo que disminuye parámetros y mejora la convergencia. En el neck, YOLOv4 adoptó un Spatial Pyramid Pooling (SPP) al final del backbone para capturar información de contexto multi-escala mediante poolings de distintos tamaños (5x5, 9x9, 13x13). También integró un PANet (Path Aggregation Network) como red de cuello que combina las características de distintas escalas con rutas de arriba-abajo y abajo-arriba, reforzando las características útiles para detección de objetos de variados tamaños. Finalmente, la head de YOLOv4 es similar a YOLOv3 (predicciones en 3 escalas con anclas), pero con mejoras en la función de pérdida: se usó CIoU loss para mejorar la regresión de cajas y objeto score con crossing IOU para penalizar cajas duplicadas, entre otros ajustes. Además de los cambios arquitectónicos, YOLOv4 incorporó numerosos trucos de entrenamiento (“bag of freebies”) y de inferencia (“bag of specials”): Mosaic data augmentation (pegar 4 imágenes aleatorias) para robustez a contexto, DropBlock regularization, CIoU y Wing loss, augmentation HSV y entrenamiento con batch normalización optimizada. Gracias a todo ello, YOLOv4 logró 43.5 % AP (COCO test-dev) con 65 FPS en

una GPU Tesla V100, superando a sus predecesores y rivales en la relación precisión-velocidad. Marcó un hito al hacer asequible un detector de alto rendimiento en entornos de recursos limitados.

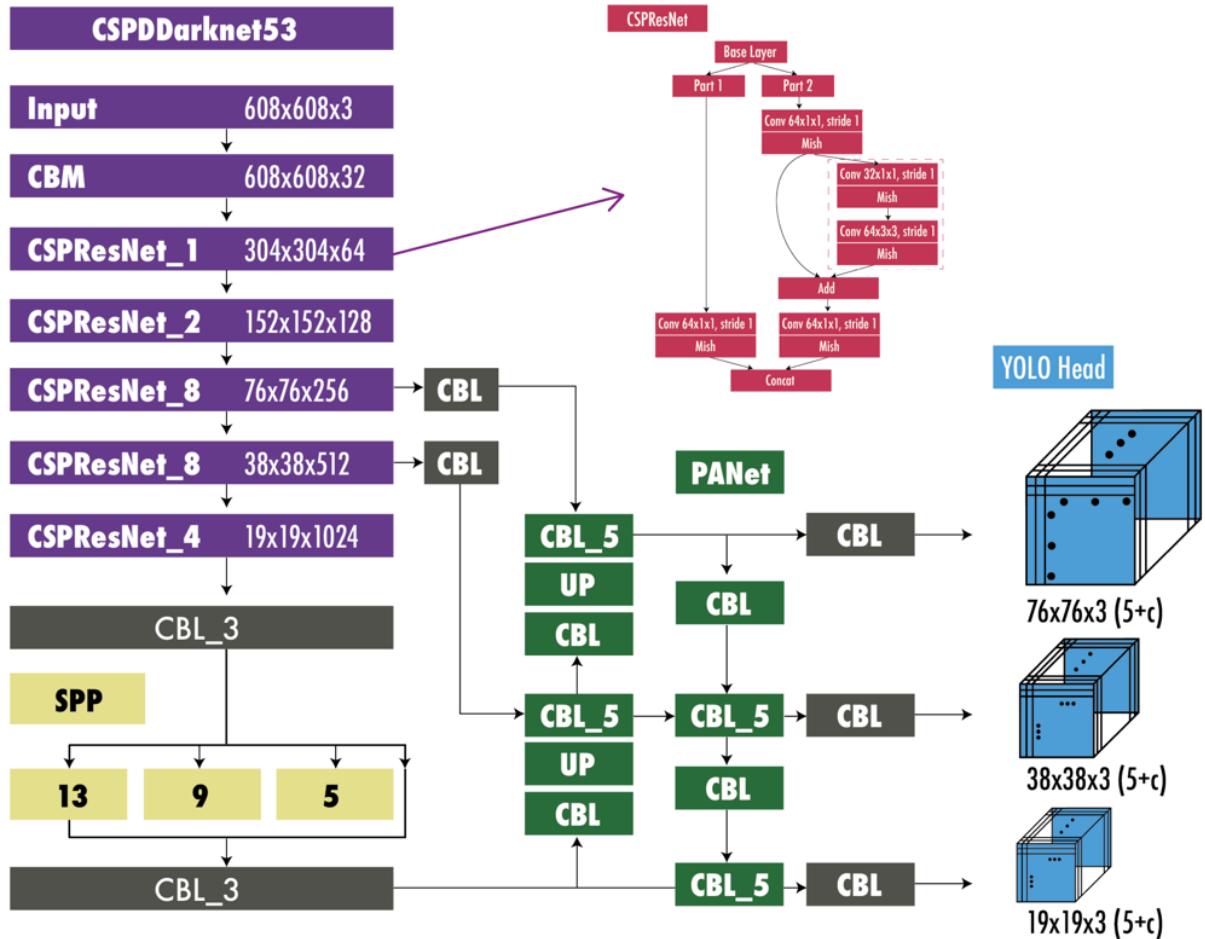


Figura 11: Arquitectura de YOLOv4 [24].

En conclusión, YOLOv4 incorporó innovaciones tanto de arquitectura (CSP, PAN, SPP) como de técnicas de entrenamiento, convirtiéndose en uno de los detectores más precisos de 2020 manteniendo un rendimiento en tiempo real. Sentó las bases para implementaciones posteriores que adoptaron sus componentes (muchos se reutilizaron en YOLOv5 y YOLOv7) [24].

2.6.5. YOLOv5: implementación en PyTorch y variantes de segmentación

YOLOv5 fue lanzado por Ultralytics en 2020 como una implementación optimizada (en PyTorch) y continuó el legado de YOLOv4 incorporando mejoras propias. Aunque no estuvo

respaldado por un paper formal, YOLOv5 ganó amplia adopción por su facilidad de uso y rendimiento. La arquitectura de YOLOv5 se resume en la Figura 12. En esencia, YOLOv5 reutiliza el backbone CSPDarknet de YOLOv4 pero con modificaciones: se añadió una capa inicial tipo Focus/Stem para reducir la resolución de entrada concatenando subcuadros (dividir la imagen en 4 y apilar en canales), reduciendo así el tamaño de activación tempranamente. Cada nivel de escala del backbone incluye bloques CSP con activación SiLU (Swish) en lugar de Mish para acelerar la inferencia. YOLOv5 introduce un módulo SPP Fast (SPPF) optimizado que realiza el pooling piramidal de forma más eficiente computacionalmente. El neck emplea una CSP-PAN (PANet modificada con capas CSP) para fusionar características, similar a YOLOv4 pero reduciendo redundancia de cálculos. La head de detección conserva el formato YOLO de predicciones en 3 escalas con anclas, usando pérdidas CIoU y BCE como en YOLOv4. Ultralytics entrenó YOLOv5 en múltiples tamaños, definiendo modelos Nano, Small, Medium, Large, Xlarge con factores de ancho/profundidad distintos (por ejemplo YOLOv5s 7.5 millones de parámetros, YOLOv5x 87 millones). Esto permitía elegir entre velocidad y precisión según la aplicación. Una contribución importante fue que Ultralytics amplió YOLOv5 más allá de la detección pura, incorporando variantes para segmentación de instancias (YOLOv5-seg) y para detección de poses en el mismo código base (a partir de la versión 5.0-6.0).

YOLOv5-seg (segmentación de instancias): Ultralytics añadió una rama de segmentación de instancias a YOLOv5 manteniendo gran parte de la arquitectura original. En concreto, además de las salidas de detección (cajas + clases), YOLOv5-seg incorpora en la head un módulo extra de salida que predice coeficientes de máscara. Este enfoque sigue la idea de YOLACT/Blend-Mask: la red genera un conjunto de protomáscaras de baja resolución a partir de características profundas, y para cada detección predice un pequeño vector de coeficientes que linealmente combinan esas protomáscaras para producir la máscara binaria de la instancia. De esta manera, se extiende mínimamente la cabeza de YOLO con pocos parámetros adicionales, aprovechando el backbone común. La arquitectura base (backbone CSP, SPPF, PANet) permanece sin cambios, asegurando que el costo computacional añadido sea bajo. En la práctica, YOLOv5-seg demostró ser extremadamente eficiente y competitivo, logrando segmentación en tiempo real con precisión cercana a modelos más pesados (Mask R-CNN) pero a una fracción de su tiempo de inferencia. Ultralytics reportó que sus modelos YOLOv5-seg v7.0 superaron las marcas existentes en benchmark de instancia en cuanto a precisión y velocidad combinadas. Cabe

mencionar que YOLOv5-seg está orientado a segmentación de instancias (más que segmentación semántica pixel a pixel); no obstante, al generar máscaras por objeto, también puede usarse para propósitos semánticos agrupando máscaras de la misma clase [25] [26].

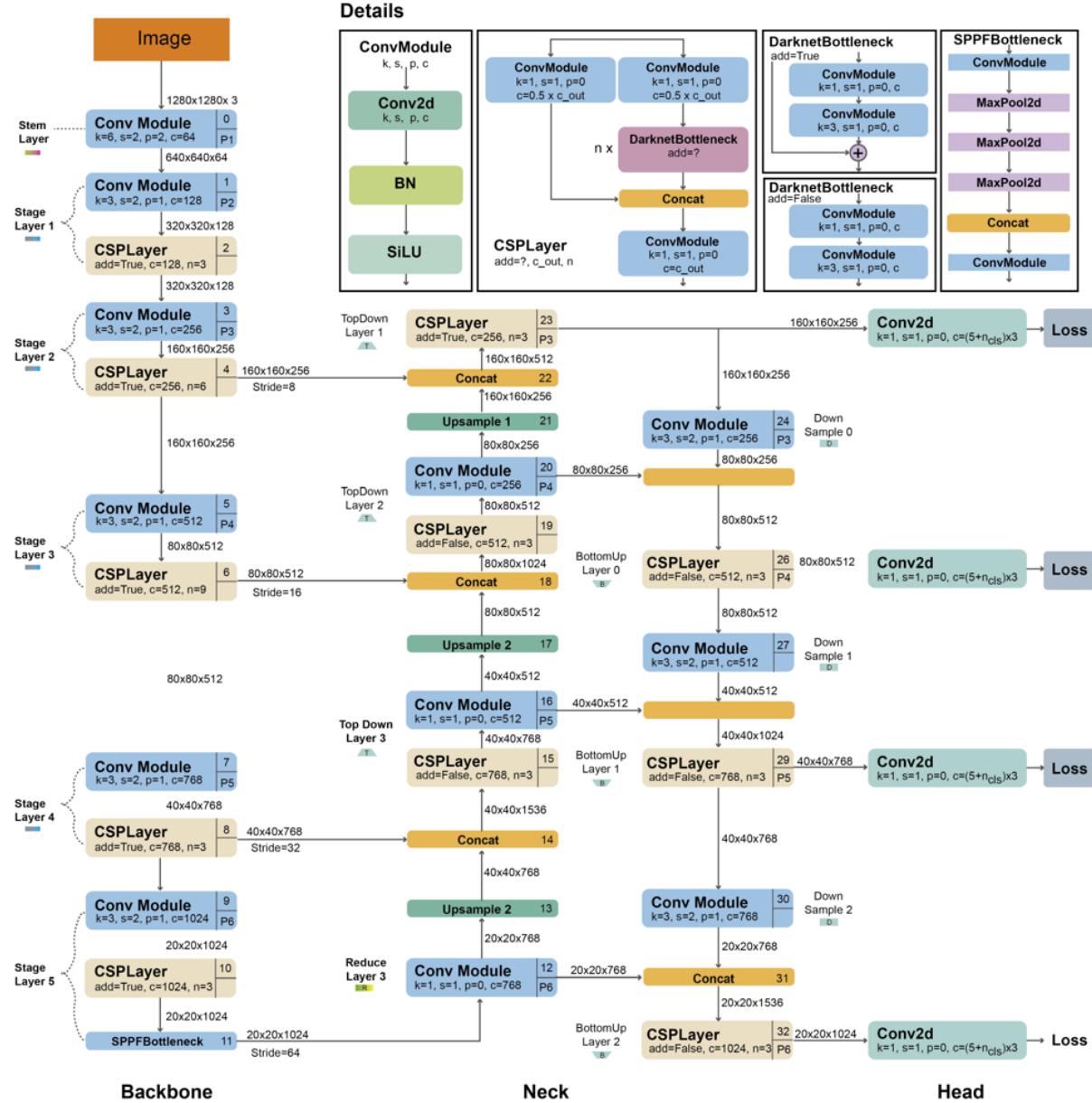


Figura 12: Arquitectura de YOLOv5 [25].

2.6.6. YOLOv6: enfoque industrial con eficiencia y cabeza ancla-libre

YOLOv6 fue introducido por Meituan en 2022 como un detector orientado a aplicaciones industriales, optimizado para implementaciones en dispositivos con recursos limitados y alto

rendimiento de inferencia. Arquitectónicamente, YOLOv6 se aparta de Darknet y adopta un backbone nuevo llamado EfficientRep, basado en bloques RepVGG reparametrizables. En la Figura 13 se muestra la estructura de YOLOv6. Los bloques RepVGGBlock permiten entrenar con estructuras de convolución + BN + activación que en inferencia se funden en un único kernel conv equivalente, mejorando la latencia sin afectar la precisión. El backbone EfficientRep se compone de etapas con bloques llamados BepC3 (Rep-based CSP) que combinan la eficiencia de RepVGG con la estrategia CSP de fraccionar la ruta de datos. Además, YOLOv6 incluye igualmente un módulo SPPF en el backbone para contexto global, similar a YOLOv5 [27].

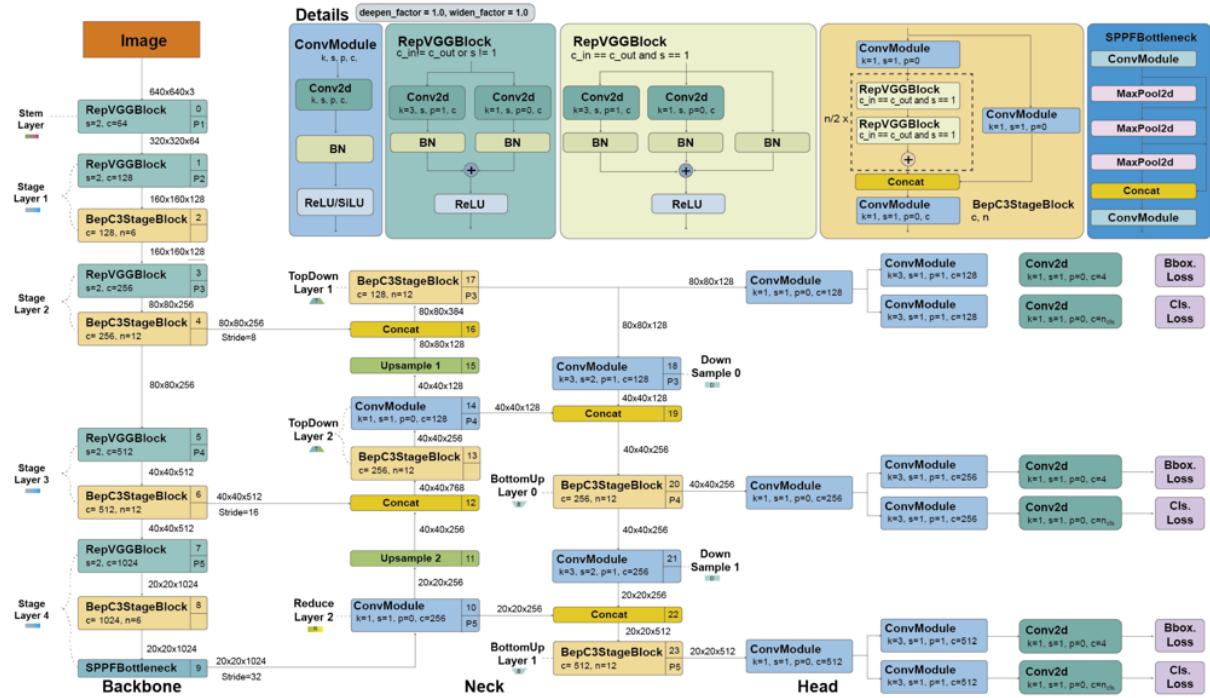


Figura 13: Arquitectura de YOLOv6 [27].

2.6.7. YOLOv7: nuevas estrategias de arquitectura y segmentación con BlendMask

YOLOv7, publicado por Wang et al. en 2022, supuso otro salto evolutivo al proponer técnicas de arquitectura innovadoras que mejoraron tanto la eficacia computacional como la precisión, consolidando a YOLOv7 como uno de los detectores más precisos en tiempo real de su generación. La Figura 14 muestra la arquitectura general de YOLOv7. Uno de los aportes centrales fue el bloque de Extended Efficient Layer Aggregation (E-ELAN) en el backbone. E-ELAN es una

extensión de los bloques ELAN, diseñados para permitir que redes más profundas aprendan efectivamente controlando las rutas de menor y mayor profundidad. En YOLOv7, cada bloque E-ELAN realiza una partición del flujo de características en varios grupos, aplica convoluciones paralelas y luego concatena y mezcla (shuffle) las salidas, asegurando así una agregación eficiente de características de distintas escalas dentro del mismo bloque. Esto mejora la capacidad de aprendizaje sin aumentar excesivamente el cómputo. Además, YOLOv7 incorporó escalado compuesto de modelos: en lugar de diseñar manualmente versiones “tiny”, “large”, aplicó reglas de escalado en profundidad y ancho al backbone y neck para obtener modelos de distintos tamaños optimizados (YOLOv7-tiny, -X, -W6 etc.). Otra mejora fue el uso de técnicas de reparametrización durante el entrenamiento: algunas capas (como conv + BN) se desplegaban en estructuras redundantes durante entrenamiento para mejorar la robustez, pero se fusionaban en inferencia para no penalizar la latencia [28].

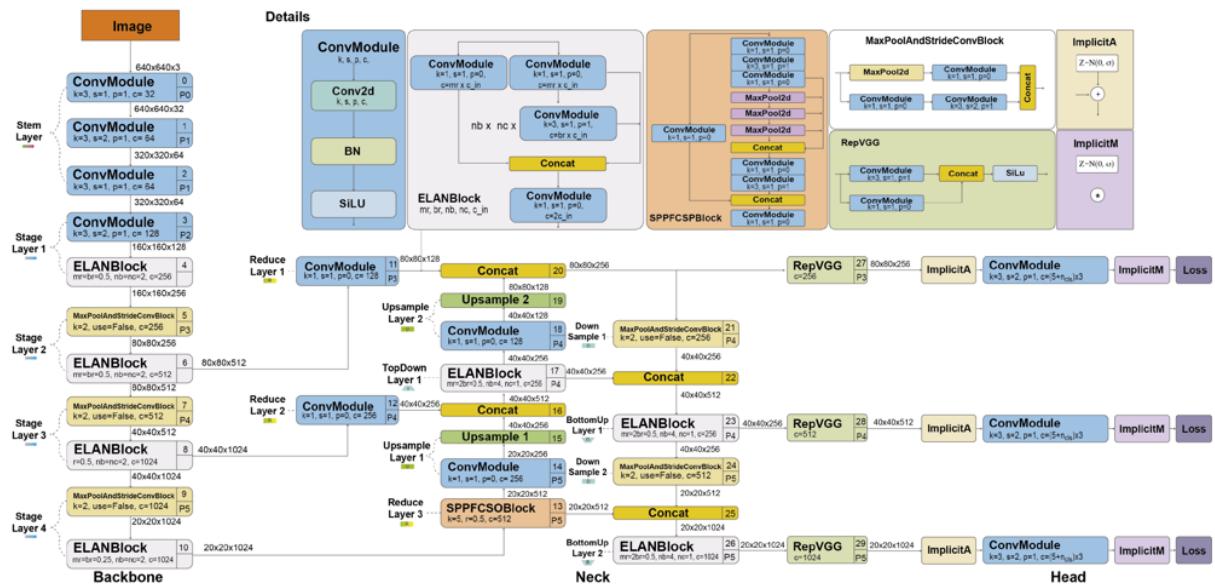


Figura 14: Arquitectura de YOLOv7 [28].

En términos de rendimiento, YOLOv7 (modelo grande) alcanzó 51.2 % AP en COCO test-dev a 30 FPS, superando a YOLOv5 y YOLOv6 de tamaño comparable, y posicionándose brevemente como referente en detección rápida en 2022. Pero además de la detección, los autores de YOLOv7 lanzaron extensiones para segmentación de instancias y detección de pose humana. YOLOv7-mask es la variante de segmentación, lograda integrando YOLOv7 con el enfoque BlendMask. BlendMask (Chen et al. 2020) es un método de segmentación que combina ideas de

FCN y Mask R-CNN, generando protomáscaras semánticas que luego se refinan por instancia. En YOLOv7-mask, se tomó el modelo YOLOv7 (por ejemplo la variante W6 de 36 millones de parámetros) y se añadió un brazo de máscaras: a las características del neck se les aplicó una rama de atención y otra de segmentación que producen mapas de máscara por instancia. La red se entrenó en MS COCO (tarea instance segmentation, 80 clases) durante 30 épocas, logrando resultados de precisión de segmentación en tiempo real líderes para 2022. En la práctica, YOLOv7-mask obtiene máscaras de instancia con calidad cercana a Mask R-CNN (AP máscara 45 % COCO) pero con velocidades muy superiores (hasta 20 FPS en GPU con el modelo grande). La Figura 15 esquematiza la arquitectura de YOLOv7-mask: se aprecian las salidas de detección originales y las ramas adicionales de atención y máscara que fusionan características de múltiples escalas (pirámide FPN) para predecir tanto mapas semánticos como instancias segmentadas [29].

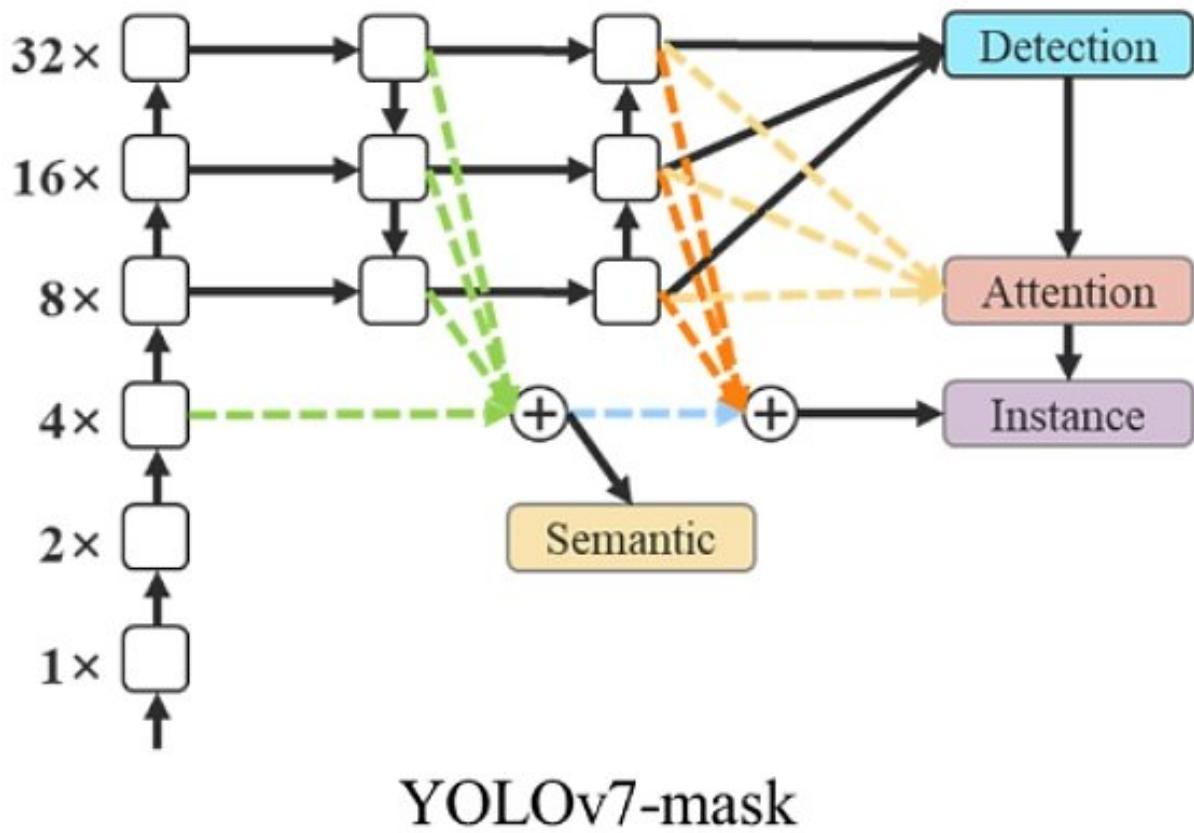


Figura 15: Arquitectura de YOLOv7 Mask [29].

2.6.8. YOLOv8: modelo unificado multi-tarea con cabeza desacoplada (2023)

Ultralytics lanzó YOLOv8 en enero de 2023 como sucesor de YOLOv5, integrando las mejores ideas de YOLOv4-7 y añadiendo nuevas optimizaciones para hacer un modelo de visión por computadora versátil (unified model). YOLOv8 fue concebido desde el inicio como una solución multi-tarea: con la misma arquitectura base se pueden realizar detección, segmentación, clasificación de imágenes, e incluso estimación de pose y detección de cajas orientadas. Focalizándonos en la detección/segmentación, YOLOv8 presenta varios cambios arquitectónicos notables respecto a YOLOv5/7:

- Se mantiene un backbone tipo CSP con 53 capas (CSPDarknet53 modificado), pero se introduce un nuevo bloque denominado C2f (Cross-Stage Partial with 2 convolution layers) en lugar del CSP clásico. El módulo C2f es similar a CSP pero simplificado: divide la característica en dos caminos, aplica n capas Bottleneck en uno mientras el otro pasa directo, y luego concatena (un caso particular de CSP donde solo 2 conv por bloque). Esto reduce un poco los parámetros manteniendo la capacidad de fusionar características locales y globales.
- Adopta por primera vez en YOLO una cabeza de detección anchor-free y desacoplada. Esto significa que YOLOv8 ya no utiliza cajas ancla predefinidas; en su lugar, la cabeza predice directamente las coordenadas normalizadas (x, y, w, h) relativas a la celda y tamaño de imagen, junto con un puntaje de objeto. Además, separa la sub-red de clasificación de la de regresión: por cada escala de característica, existen dos ramas paralelas de convoluciones, una dedicada a predecir la probabilidad de cada clase y la otra a predecir las coordenadas y puntaje de objeto. Esta decoupled head permite que cada tarea (localización vs clasificación) aprenda con menos interferencia, mejorando la precisión. Cada rama termina en una capa conv final que da las salidas correspondientes (sigmoide para objectness, y sigmoide+softmax para clases, según Ultralytics).
- En cuanto a bloques especiales, YOLOv8 retiene SPPF del modelo anterior para contexto global y utiliza activación SiLU. También incorpora losses avanzadas: CIoU para cajas y DFL (Distributed Focal Loss) para suavizar la cuantización de coordenadas, buscando mejorar la precisión en detección de objetos pequeños.

- YOLOv8 viene predefinido en cinco tamaños (n, s, m, l, x) con factores de escala de anchura/profundidad, similar a YOLOv5, facilitando su aplicación desde microcontroladores hasta GPU de alto rendimiento.

Estos cambios resultaron en un modelo más preciso y flexible. En MS COCO, YOLOv8-XL obtuvo 53.9 % AP a 280 FPS en TensorRT, superando a YOLOv5 (50.7 % AP) con la misma resolución. La cabeza sin anclas eliminó la necesidad de Non-Maximum Suppression (NMS) durante el entrenamiento - aunque en inferencia YOLOv8 por defecto aún aplica NMS a sus predicciones finales - y simplificó la detección de múltiples objetos cercanos [30].

2.6.9. YOLOv9: PGI y GELAN para mejorar el flujo de información (2024)

YOLOv9, introducido por Wang et al. en 2024, continúa la evolución con innovaciones teóricas en la optimización de redes profundas. Los autores identifican que en arquitecturas muy profundas se pierde gran cantidad de información a través de las capas (efecto “cuello de botella de información”). Para abordar esto, YOLOv9 propone el concepto de Programmable Gradient Information (PGI), una técnica que busca aprovechar la información completa de entrada al calcular la función de pérdida para múltiples objetivos. En esencia, PGI introduce mecanismos para retropropagar gradientes más informativos y completos que guían la actualización de pesos de forma óptima incluso en tareas combinadas. Esta idea se acompaña de una nueva arquitectura de backbone llamada GELAN (Generalized Efficient Layer Aggregation Network). GELAN extiende la filosofía de E-ELAN/ELAN: se enfoca en la planificación de rutas de gradiente, permitiendo que los gradientes fluyan de manera más reversible a través de la red, evitando la pérdida de información en transformaciones espaciales sucesivas. Según los autores, GELAN logra, usando solo convoluciones estándar (no separables), una utilización de parámetros más eficiente que arquitecturas basadas en depthwise conv (como YOLOv8). Además, demuestran que combinando PGI + GELAN, es posible entrenar modelos desde cero (sin pre-entrenamiento en ImageNet) obteniendo resultados superiores a los mismos modelos entrenados con inicialización pre-entrenada, rompiendo una noción establecida.

En términos prácticos, YOLOv9 retoma componentes conocidos (por ejemplo, sigue usando cabeza decoupled anchor-free similar a YOLOv8) pero rediseña su backbone incorporando

GELAN con una agregación de capas más agresiva y posiblemente nuevos bloques de convolución invertidos. El resultado neto es un incremento notable en precisión especialmente en modelos ligeros: los experimentos muestran que YOLOv9 en su versión pequeña supera en AP a modelos YOLOv8/YOLOv7 de tamaño equivalente con casi la mitad de parámetros. Por ejemplo, en MS COCO, YOLOv9n (nano) alcanzó 45 % AP con solo 8 millones de parámetros. La mejora es más pronunciada al entrenar desde cero, demostrando la eficacia de PGI para aprovechar mejor los datos. En la Figura 15 se presenta un gráfico comparativo de desempeño publicado por los autores de YOLOv9, donde se observa la posición destacada de YOLOv9 (curva roja) en términos de AP vs número de parámetros, superando a todos los YOLO anteriores y a detectores recientes como RT-DETR [31].

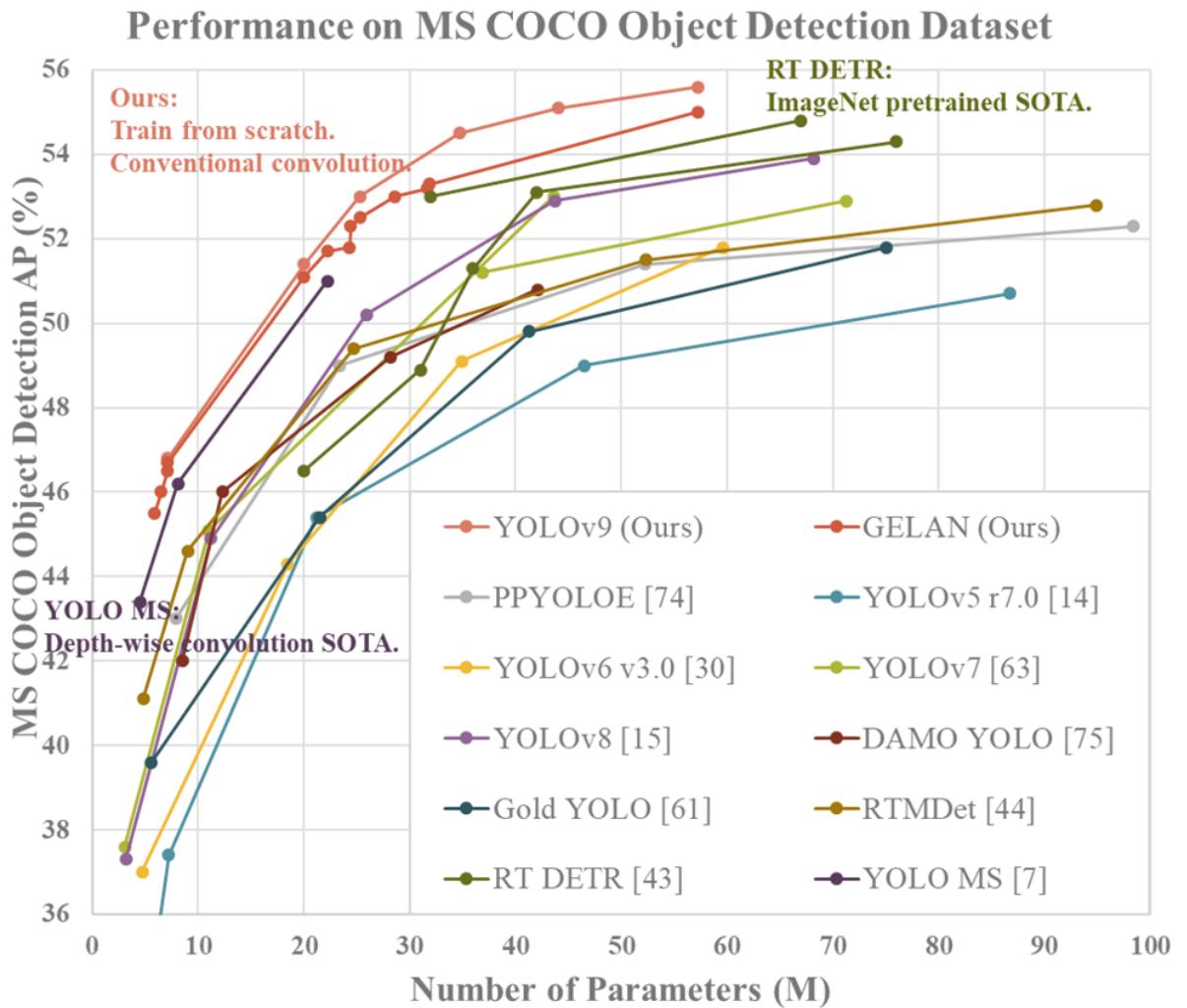


Figura 16: Rendimiento comparativo en MS COCO [31].

2.6.10. YOLOv10: detección end-to-end sin NMS y diseño eficiente (NeurIPS 2024)

YOLOv10 introduce una nueva filosofía en detección de objetos al eliminar la necesidad de la supresión de no-máximos (NMS), tradicional en detectores de una etapa. Su enfoque busca un pipeline verdaderamente end-to-end, donde la red aprende a evitar predicciones redundantes por sí misma. Durante el entrenamiento, utiliza una técnica llamada **Consistent Dual Assignments**, que asigna dos predicciones por objeto: una optimiza la clasificación y otra la localización, con un mecanismo de coordinación entre ambas. Esto logra que en la inferencia usualmente solo haya una predicción fuerte por objeto. Además, rediseña la arquitectura con un **backbone CSPNet** mejorado, un **neck PAN** más ligero, activación **SiLU** y **normalización por capas** optimizada. La cabeza del modelo es **anchor-free y desacoplada**, e integra la supresión de cajas directamente en la función objetivo.

YOLOv10 también incluye una variante **YOLOv10-B (Balanced)**, que ofrece un equilibrio entre precisión y velocidad. En pruebas, mostró menor latencia que YOLOv8 y YOLOv9 para niveles similares de precisión. La eliminación de NMS no solo mejora la limpieza de las predicciones, sino que simplifica su despliegue en hardware especializado y permite flujos completamente diferenciables. Aunque el artículo se centra en detección, sus principios también favorecen tareas como segmentación, con máscaras optimizadas junto con las cajas. En resumen, YOLOv10 representa un paso importante hacia detectores más eficientes, precisos y completamente integrables sin pasos de post-procesamiento externos [32].

2.6.11. YOLOv11: módulos C3K2 y C2PSA, y consolidación multi-tarea (2024-2025)

YOLOv11, presentado por Ultralytics a finales de 2024, representa una evolución significativa dentro de la familia YOLO, con el objetivo de mejorar simultáneamente la velocidad, la precisión y las capacidades multi-tarea del modelo. Mantiene principios clave de versiones anteriores como detección anchor-free y cabezas desacopladas, pero incorpora nuevas mejoras arquitectónicas. Uno de los principales aportes es el bloque C3K2, una extensión del módulo C2f de YOLOv8. Este bloque permite elegir entre dos configuraciones: una más pesada y precisa (modo C3k, con tres convoluciones internas), y otra más ligera y eficiente (tipo Bottleneck). Esto ofrece flexibilidad escalable en el diseño del backbone, adaptando el modelo a diferentes

necesidades de cómputo y precisión.

También introduce un nuevo módulo de atención llamado C2PSA (Cross-Stage Partial Self-Attention), que se aplica después del módulo SPPF. Este mecanismo divide los mapas de características y aplica self-attention solo a una porción, combinándola luego con la parte no procesada mediante convoluciones. Así se logra enfocar en regiones críticas -como objetos parcialmente ocultos o superpuestos- sin incurrir en el alto costo de una atención global. En la cabeza del modelo se añadieron convoluciones Depthwise (DWConv) en capas finales para reducir la carga computacional, especialmente en las resoluciones altas donde el procesamiento es más costoso. Esto mejora el rendimiento sin sacrificar precisión.

Gracias a estas innovaciones, YOLOv11 logra una mejora de 1-2 puntos porcentuales en AP sobre YOLOv8 en conjuntos de datos como COCO, y en benchmarks específicos como detección de incendios, YOLOv11n superó a YOLOv10n y YOLOv9t. Además, se optimizó para funcionar mejor con CUDA, lo que le da una ventaja en velocidad. YOLOv11 conserva su naturaleza multi-tarea y es capaz de ejecutar tareas como detección, segmentación, clasificación, pose y tracking en un solo modelo. Se demostró su eficacia en escenarios reales como tráfico urbano o vigilancia industrial. Una variante importante es YOLOv11-Seg, enfocada en segmentación de objetos. Mantiene la arquitectura base (backbone C3K2, atención C2PSA, neck PAN, y head desacoplada) y añade salidas de máscara de instancia. El módulo C2PSA mejora la capacidad del modelo para segmentar objetos parcialmente ocluidos en entornos complejos como obras de construcción. En pruebas, alcanzó un mAP@50 de 80.8 %, superando a otros métodos previos [33] [34].

2.6.12. Comparación de rendimiento y conclusiones

A lo largo de 11 iteraciones principales, la arquitectura YOLO ha evolucionado de una simple red con capas totalmente conectadas (YOLOv1) a complejos sistemas con módulos especializados y atención (YOLOv11). Cada versión introdujo innovaciones que se reflejan en mejoras de las métricas de desempeño. En la Tabla 1 se resumen algunas cifras de rendimiento típicas reportadas en la literatura para distintas versiones YOLO, considerando los mejores modelos de cada (en COCO para v3 en adelante, VOC para v1-v2):

Versión	Año	Tipo de anclas	Backbone	AP (COCO) / mAP (VOC) ^a	FPS (GPU) ^b
YOLOv1	2015	–	Custom (24 conv)	63.4 % mAP	~45
YOLOv2	2017	Sí	Darknet-19	78.6 % mAP	40-90
YOLOv3	2018	Sí	Darknet-53	33.0 % AP (60.6 AP ₅₀)	20-30
YOLOv4	2020	Sí	CSPDarknet-53	43.5 % AP	30-60
YOLOv5	2020	Sí	CSPDarknet-53 (mod.)	50.7 % AP	~50
YOLOv6	2022	No (AAT)	EfficientRep	52.3 % AP	~80
YOLOv7	2022	No	E-ELAN	51.2 % AP	~30
YOLOv8	2023	No	CSPDarknet-53 (C2f)	53.9 % AP	~50
YOLOv9	2024	No	GELAN	≈55 % AP	~40
YOLOv10	2024	No (NMS-free)	CSPNet + PAN	≈54 % AP	~60
YOLOv11	2024	No	CSP/C3K2 + C2PSA	≈56 % AP	~70

^a Para YOLOv1 y YOLOv2 se indica mAP en VOC 2007; para versiones posteriores, AP en COCO (IoU 0.5:0.95), salvo que se especifique AP₅₀.

^b FPS aproximados reportados por los autores (GPU de referencia V100 o A100); dependen del tamaño de entrada y hardware.

Tabla 1: Resumen comparativo de las principales versiones de YOLO (v1-v11) [20] [21] [33].

En conclusión, la familia YOLO ha logrado combinar simplicidad, velocidad y precisión mediante una evolución cuidadosa de su arquitectura de red neuronal. Desde YOLOv1, que introdujo la idea de detección unificada en una red, hasta YOLOv11, que integra componentes sofisticados como atención y aprendizaje multi-tarea, cada versión ha resuelto limitaciones de la anterior: se añadieron anclas para mejor localización (v2), luego múltiples escalas para objetos pequeños (v3), CSP y PAN para eficiencia y robustez (v4), activaciones y data augmentation para rendimiento (v4/v5), cabezas desacopladas y sin anclas para simplificar la predicción (v8), y recientemente la eliminación de NMS y atención para afinar aún más el proceso (v10/v11). Las variantes de segmentación han capitalizado estas mejoras, permitiendo que modelos YOLO segmenten objetos con granularidad de píxel en tiempo real, algo impensable en 2015. El impacto de YOLO en la visión por computador es difícil de exagerar: ha posibilitado aplicaciones de IA embarcada, análisis de vídeo en vivo, y soluciones de seguridad, entre otros, al proveer detección (y ahora segmentación) precisa a velocidades antes solo soñadas. Se espe-

ra que futuras versiones (YOLOv12 en adelante, o derivados) incorporen quizá componentes transformer o NAS (Neural Architecture Search) - de hecho, ya existen derivados como YOLO-NAS pero manteniendo el espíritu “You Only Look Once”: resolver la tarea de visión en una pasada rápida y eficiente de la red [20] [21] [33].

3

Análisis y Diseño

El objetivo fundamental de este Trabajo Fin de Grado (TFG) es llevar a cabo un análisis estadístico exhaustivo que compare el rendimiento de distintos modelos de segmentación de lesiones en resonancia magnética. En consecuencia, el **caso de uso principal** del sistema consiste en *generar los resultados del análisis comparativo*.

3.1. Resumen gráfico de los experimentos

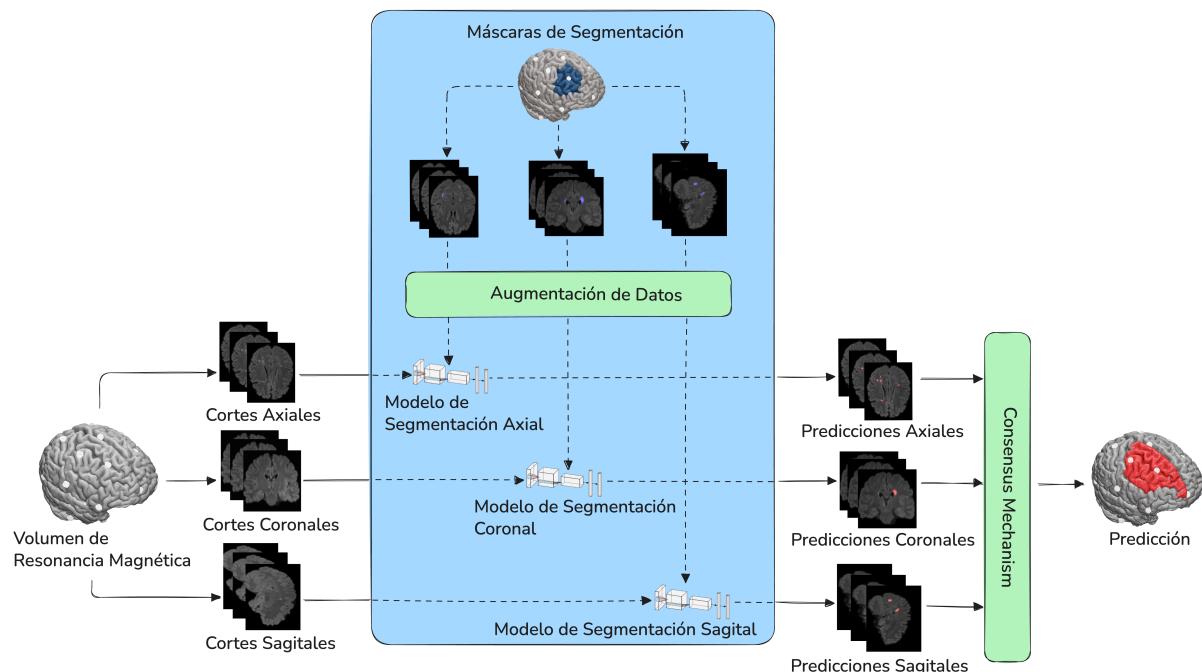


Figura 17: Resumen gráfico de los experimentos del estudio.

3.2. Casos de uso

Sin embargo, para llegar a dicha meta resulta imprescindible cubrir una serie de tareas operativas previas y de apoyo -como el entrenamiento de los modelos, su validación, la inferencia sobre nuevos estudios y la visualización de las máscaras generadas- que permiten obtener los datos necesarios y garantizar la reproducibilidad del experimento. Estas tareas se formalizan a continuación como **casos de uso secundarios**, los cuales, por la propia naturaleza modular del sistema, pueden ejecutarse de forma independiente para usos prácticos adicionales.

3.2.1. Casos de uso funcionales y no funcionales

En esta aplicación distinguimos dos grandes tipos de casos de uso:

- **Casos de uso funcionales** en nuestro dominio abarcan, por ejemplo: entrenamiento de un modelo ([Tabla 2](#)), validación objetiva ([Tabla 3](#)), inferencia clínica ([Tabla 4](#)), visualización y comparación de máscaras ([Tabla 5](#)), análisis estadístico de resultados ([Tabla 6](#)) y el flujo automático de validación cruzada ([Tabla 7](#)). Cada uno especifica *quién* hace *qué* y en *qué orden*, así como sus pre- y postcondiciones.
- **Casos (o requisitos) de uso no funcionales** recogen atributos de calidad que no reflejan una función específica, pero determinan el *cómo* debe comportarse el sistema. Entre los más relevantes se incluyen:
 - a. *Rendimiento*: tiempo máximo aceptable para entrenar un pliegue o generar una máscara, aprovechando GPUs y permitiendo *early-stopping*.
 - b. *Escalabilidad y uso de recursos*: capacidad de distribuir pliegues en paralelo y gestionar automáticamente memoria y almacenamiento.
 - c. *Fiabilidad y reanudación*: soporte de *checkpointing* para continuar entrenamientos o validaciones tras un fallo de hardware.
 - d. *Usabilidad*: interfaz coherente para que perfiles clínicos e investigadores configuren experimentos sin código.
 - e. *Mantenibilidad y portabilidad*: código modular, uso de contenedores y configuración declarativa que faciliten actualizaciones y reproducción de resultados en otros centros.

3.2.2. Análisis de casos de uso

Caso de uso	Entrenar modelo de segmentación
Resumen	El usuario escoge un modelo (nnUNet 2D/3D o YOLO11-SEG con la política deseada), ajusta la configuración y lanza el proceso de aprendizaje.
Actores	Investigador/Usuario científico, Sistema de cómputo
Precondición	<ul style="list-style-type: none"> ▪ Conjunto de entrenamiento pre-procesado y accesible. ▪ GPU y espacio en disco disponibles.
Postcondición	Pesos del modelo, registros y métricas de entrenamiento almacenados en la carpeta del experimento.
<hr/>	
Curso Normal	
<ol style="list-style-type: none"> 1. El usuario selecciona el tipo de modelo y, en su caso, la política de validación. 2. Configura hiperparámetros (épocas, tamaño de lote, <i>patch</i> 2D/3D, etc.). 3. Inicia el entrenamiento. 4. El sistema carga los datos, aplica la división de datos dependiendo del modelo y política seleccionada y entrena el modelo. 5. Se guardan los pesos finales y las curvas de aprendizaje. 	
<hr/>	
Curso Alternativo	
<ol style="list-style-type: none"> 2a. Falta de espacio o GPU → el sistema cancela y muestra un error. 4a. Se detecta sobreajuste temprano → se aplica <i>early-stopping</i>. 	
<hr/>	
Observaciones	El entrenamiento puede reanudarse desde el último <i>checkpoint</i> .

Tabla 2: Caso de uso 1: Entrenar modelo de segmentación.

Caso de uso	Validar modelo entrenado
Resumen	Ejecutar la validación para obtener métricas objetivas del modelo entrenado.
Actores	Investigador, Sistema
Precondición	Existe un modelo previamente entrenado y un conjunto de validación etiquetado.
Postcondición	Métricas (Dice, HD 95, sensibilidad, etc.) y máscaras predichas almacenadas.
<hr/>	
Curso Normal	
<hr/> <ol style="list-style-type: none"> 1. El usuario selecciona el experimento entrenado. 2. Define la parte del conjunto de datos que quiere usar para validar. 3. El sistema procesa los volúmenes y genera las máscaras predichas. 4. Calcula las métricas y guarda un reporte. <hr/>	
<hr/>	
Curso Alternativo	
<hr/> <p>1a. Modelo no encontrado → se notifica al usuario.</p> <hr/>	
<hr/>	
Observaciones	Se puede reutilizar la validación para diferentes <i>thresholds</i> .

Tabla 3: Caso de uso 2: Validar modelo entrenado.

Caso de uso	Generar máscara de segmentación
Resumen	Aplicar un modelo entrenado sobre un escáner de RM y obtener la máscara 3D de lesiones.
Actores	Usuario clínico, Sistema
Precondición	<ul style="list-style-type: none"> ■ Modelo entrenado. ■ Volumen de RM normalizado.
Postcondición	Máscara de segmentación almacenada en formato NIfTI junto al estudio.
<hr/>	
Curso Normal	
<ol style="list-style-type: none"> 1. El usuario selecciona el modelo y carga el volumen. 2. El sistema aplica la misma pre-procesamiento usado en entrenamiento. 3. Se ejecuta la inferencia (2D/3D o por proyecciones + consenso). 4. Se guarda la máscara resultante. 	
<hr/>	
Curso Alternativo	
<p>2a. Dimensiones incompatibles → se resamplea automáticamente o se cancela.</p>	
<hr/>	
Observaciones	El sistema admite inferencia por lotes de múltiples estudios.

Tabla 4: Caso de uso 3: Generar máscara de segmentación.

Caso de uso	Visualizar/Comparar máscaras de segmentación
Resumen	Mostrar la máscara predicha superpuesta al volumen original y/o compararla con la máscara de referencia o con la salida de otros modelos.
Actores	Investigador, Radiólogo
Precondición	Al menos una máscara generada; opcionalmente la máscara ground-truth u otra máscara predicha.
Postcondición	El usuario visualiza las diferencias y, si lo desea, exporta capturas o métricas de concordancia locales.
<hr/>	
Curso Normal	
<hr/>	
1. Selección de las máscaras a comparar. 2. El sistema abre el visor 2D/3D con deslizadores de corte. 3. El usuario ajusta opacidad, color y visualiza discrepancias. 4. Puede guardar vistas o exportar mapas de diferencia.	
<hr/>	
Curso Alternativo	
<hr/>	
3a. Las máscaras tienen resoluciones distintas → se solicita re-muestra o se cancela.	
<hr/>	
Observaciones	Integra cálculo local de Dice por <i>corte</i> .

Tabla 5: Caso de uso 4: Visualizar y comparar máscaras.

Caso de uso	Análisis estadístico de resultados
Resumen	Comparar el rendimiento de varios modelos mediante tests estadísticos y generar un informe tabular/gráfico.
Actores	Investigador
Precondición	Resultados de validación (métricas) de dos o más modelos distintos.
Postcondición	Informe PDF/LaTeX con tablas de métricas, gráficos de caja y p-values ajustados.
<hr/>	
Curso Normal	
<hr/> <ol style="list-style-type: none"> 1. El usuario selecciona los experimentos que desea comparar. 2. El sistema agrupa las métricas y realiza pruebas (Wilcoxon pareado, Friedman + Nemenyi, etc.). 3. Genera tablas de resumen y diagramas críticos. 4. Exporta el informe y, opcionalmente, los datos en CSV. 	
<hr/>	
Curso Alternativo	
<hr/> <p>1a. Solo hay un modelo seleccionado → se advierte que no puede hacerse contraste estadístico.</p>	
<hr/>	
Observaciones	Incluye corrección de <i>p-values</i> por comparaciones múltiples (Bonferroni/Holm).

Tabla 6: Caso de uso 5: Análisis estadístico comparativo.

Caso de uso	Entrenamiento en validación cruzada (5 hilos)
Resumen	El usuario selecciona la opción de validación cruzada; el sistema divide el conjunto de datos en 5 pliegues, entrena un modelo por pliegue y almacena cada modelo en su carpeta.
Actores	Investigador/Usuario científico, Sistema de cómputo
Precondición	<ul style="list-style-type: none"> ▪ Conjunto de datos completo y etiquetado. ▪ Recursos de cómputo (GPU) disponibles.
Postcondición	<ul style="list-style-type: none"> ▪ Cinco modelos entrenados (“fold_0” ... “fold_4”). ▪ Métricas por pliegue y resumen global guardados.
<hr/>	
Curso Normal	
<ol style="list-style-type: none"> 1. El usuario escoge el tipo de modelo y activa “Validación cruzada 5-hilos”. 2. El sistema genera las divisiones estratificadas del conjunto de datos (80 %-20 % por pliegue). 3. Para cada pliegue $i = 0 \dots 4$: <ol style="list-style-type: none"> 3.1. Entrena el modelo con los datos de entrenamiento (80 %). 3.2. Valida sobre el pliegue de prueba (20 %) y guarda métricas. 4. Se almacenan los pesos y reporte de cada pliegue en carpetas independientes. 	
<hr/>	
Curso Alternativo	
<ol style="list-style-type: none"> 1a. Conjunto de datos con menos de 5 sujetos → el sistema rechaza la operación. 3a. Sin recursos GPU suficientes → se pausa la ejecución o se cancela. 	
<hr/>	
Observaciones	El progreso se guarda por pliegue; se puede reanudar desde el último pliegue completado. Los resultados generados alimentan directamente el <i>Caso de uso 5: Análisis estadístico</i> .

Tabla 7: Caso de uso 6: Entrenamiento con validación cruzada automática (5 hilos).

3.3. Diagramas de secuencia

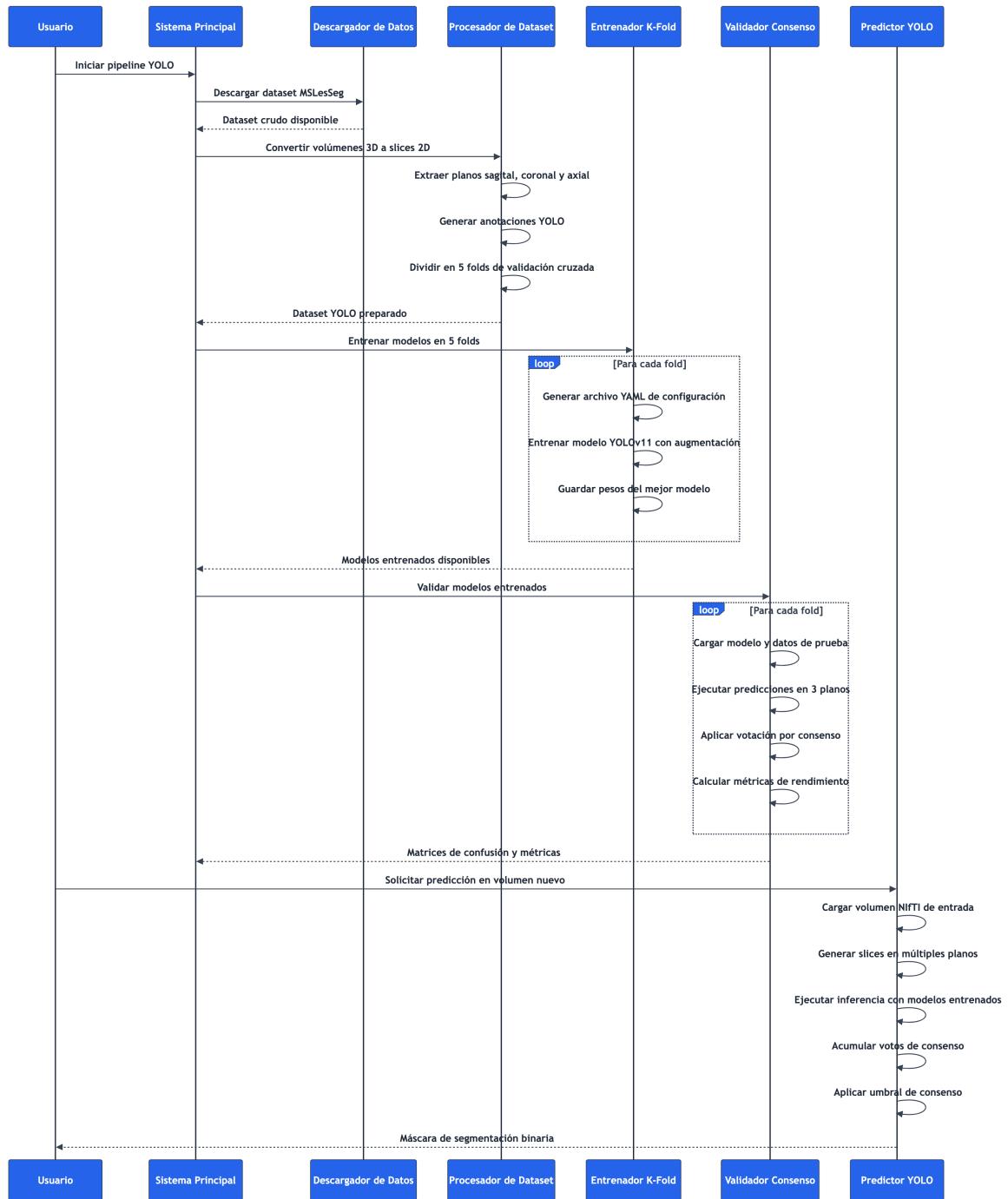


Figura 18: Diagrama de secuencia de la ejecución de la yolo.

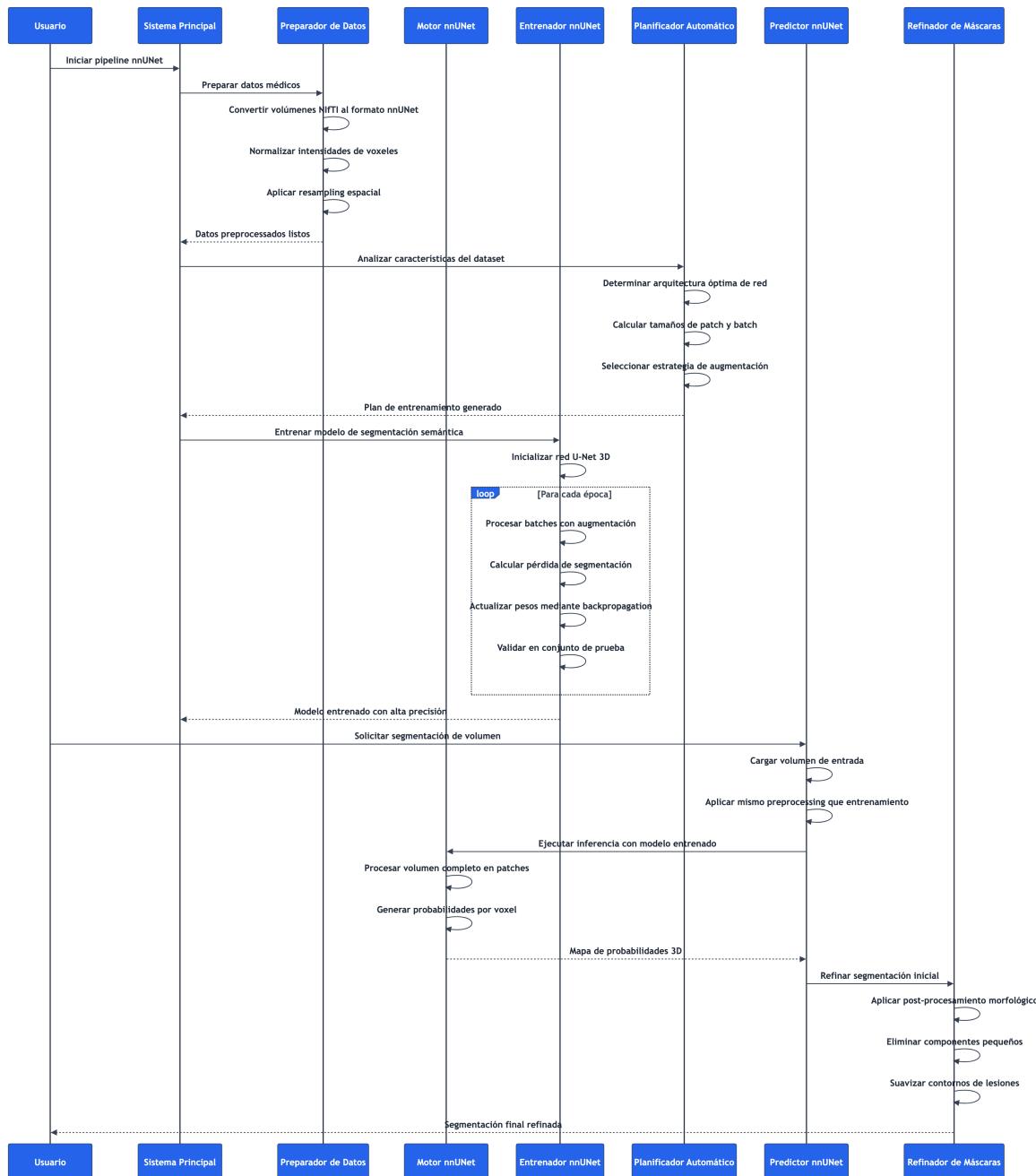


Figura 19: Diagrama de secuencia de la ejecución de la nnunet.

4

Conjunto de datos MSLesSeg (ICPR 2024)

4.1. Características generales del conjunto de datos

El concurso MSLesSeg (Multiple Sclerosis Lesion Segmentation) de ICPR 2024 proporcionó un conjunto de datos extenso y curado de resonancias magnéticas cerebrales de pacientes con esclerosis múltiple (EM) [35]. En su porción de entrenamiento se incluyeron 52 pacientes, con un total de 92 escáneres (estudios) longitudinales en distintos puntos temporales (visitas) [36]. Cada paciente cuenta con 1 a 4 adquisiciones a lo largo del tiempo (una de base y seguidas de varias de seguimiento); en conjunto, la cohorte completa del desafío abarcó 75 pacientes (53 de entrenamiento y 22 reservados para prueba). En cada estudio se recopilaron imágenes en tres modalidades de RM: FLAIR, T1-ponderada y T2-ponderada, junto con la máscara de segmentación manual de las lesiones de EM (el “ground truth”). Todas estas imágenes volumétricas fueron estandarizadas a un mismo espacio anatómico de referencia (registro al template MNI-152) y con resolución isotrópica de 1mm (matriz de $182 \times 218 \times 182$ vértices). Esto significa que los volúmenes de todas las modalidades para un paciente y tiempo dado están ya alineados entre sí (facilitando su uso conjunto) y limitados al encéfalo (se aplicó skull-stripping para remover el cráneo). La Figura 20 ilustra la estructura jerárquica del conjunto de datos: por cada paciente se poseen múltiples escaneos temporales, y cada escaneo incluye las tres modalidades (T1, T2, FLAIR) con su correspondiente máscara de lesiones.

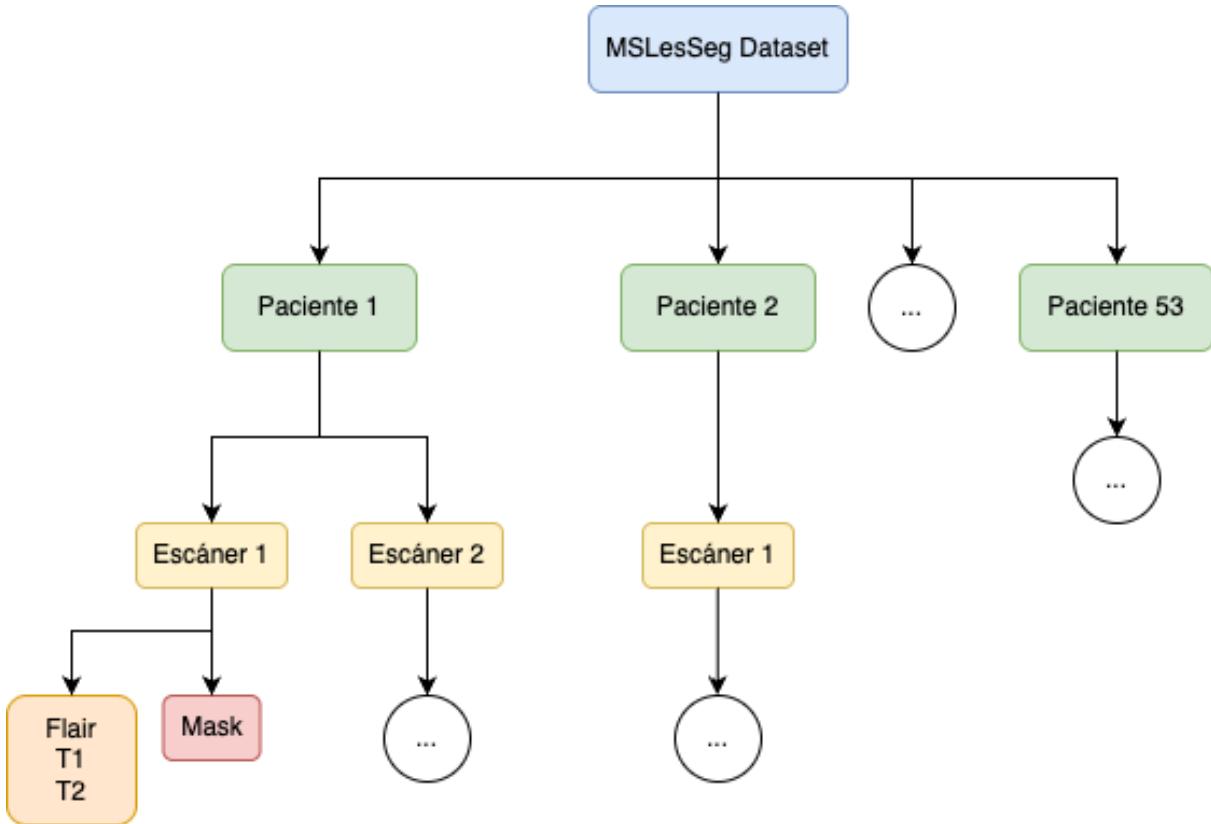


Figura 20: Estructura del conjunto de datos MSLesSeg (ICPR 2024).

Todas las imágenes se almacenaron en formato NIfTI (.nii/.nii.gz) en unidades volumétricas. En la estructura de carpetas original del conjunto de entrenamiento, por ejemplo, los archivos se organizaron por paciente (etiquetados P1, P2, ..., P52) y dentro de cada paciente por timepoint (T1, T2, etc. según la visita). Por ejemplo, un estudio de un paciente puede aparecer como P4/T2/P4-T2-FLAIR.nii.gz junto a P4-T2-T1.nii.gz, P4-T2-T2.nii.gz y P4-T2-MASK.nii.gz, siguiendo la nomenclatura PacienteX-TiempoY. Cabe destacar que cada voxel en estas imágenes tiene un tamaño de $1 \times 1 \times 1\text{mm}$ y la dimensión de matriz $182 \times 218 \times 182$ corresponde al volumen cerebral en el espacio MNI recortado; así, todas las máscaras y modalidades comparten dimensiones y coordenadas.

Un aspecto importante es que el conjunto de prueba (test) nunca se hizo público en su totalidad. Durante la competencia, a los participantes solo se les entregaron las imágenes sin la máscara de lesiones para esos pacientes de test. La evaluación oficial se realizó a través de un servidor o script donde los organizadores comparaban las máscaras predichas con las verdaderas (ocultas). Tras el cierre del concurso, el benchmark se mantiene abierto para comparar

nuevos métodos, por lo cual las máscaras de test permanecen privadas para evitar sesgos y garantizar una evaluación imparcial a futuro. En otras palabras, el benchmark “vivo” permite que investigadores subsecuentes envíen sus segmentaciones para estos casos de prueba y obtengan métricas oficiales sin haber visto nunca las anotaciones reales. Esta práctica de test oculto es común en desafíos de segmentación médica, asegurando que los modelos no se ajusten específicamente a ese conjunto y promoviendo la generalización.

4.2. Detalles técnicos de las máscaras de segmentación

Las máscaras de lesión provistas (archivos MASK.nii para cada estudio) son volúmenes 3D en formato NIfTI que codifican la segmentación manual de las lesiones de EM en el encéfalo. Se trata de imágenes binarias donde los voxels con valor 1 indican la presencia de lesión y los voxels con valor 0 corresponden al fondo o tejido no lesionado. Estas máscaras fueron delineadas manualmente por expertos en neuroimagen sobre la secuencia FLAIR, valiéndose además de la información de T1 y T2 para confirmar la naturaleza de las lesiones. El protocolo de anotación siguió un procedimiento riguroso: un radiólogo neurólogo identificó las lesiones y dos raters segmentaron cada una en FLAIR, con validación final de un neurólogo especialista en EM, iterando correcciones hasta lograr un consenso. Para reducir la variabilidad y posibles errores, se decidió excluir lesiones muy pequeñas (menores a 3mm) que no fueran claramente visibles en al menos dos cortes consecutivos de la FLAIR. De este modo, las máscaras resultantes se consideran un “ground truth” de alta calidad, con énfasis en incluir únicamente lesiones ciertamente identificables.

Técnicamente, un archivo mask.nii comparte el mismo espacio de coordenadas y resolución que sus imágenes asociadas (FLAIR, T1, T2) para el mismo paciente/tiempo. Esto significa que para entrenar un modelo de segmentación, se puede superponer voxel a voxel la máscara sobre la imagen FLAIR (u otras modalidades) sin necesidad de reprocesamiento geométrico adicional. En entrenamiento de modelos supervisados, la máscara sirve como etiqueta de verdad-terreno: el modelo toma como entrada las imágenes de una o más modalidades y aprende a predecir para cada voxel si pertenece a una lesión (1) o no (0), comparando su salida con la máscara real. Por ejemplo, en un enfoque típico se entrena una red tipo U-Net 3D que ingiere como entradas multi-modales el volumen FLAIR, T1 y T2 apilados (3 canales) y produce como salida

un volumen binario del mismo tamaño, optimizando una función de costo como Dice Loss o entropía cruzada voxelwise contra la máscara MASK.nii del experto. En un enfoque más sencillo (p.ej. 2D), el modelo recibiría imágenes 2D (cortes) y produciría máscaras 2D, que luego se reensamblan al volumen completo. En ambos casos, el archivo mask.nii proporciona la referencia para saber qué regiones del cerebro son lesiones a fin de ajustar los pesos del modelo durante el entrenamiento.

En resumen, las máscaras mask.nii son esenciales para supervisar el aprendizaje de los algoritmos de segmentación. Su formato volumétrico y alineación con las modalidades de RM permiten extraer fácilmente las secciones necesarias (2D o 3D) manteniendo la correspondencia espacial correcta. Además, al ser binarias y estar definidas en un espacio estándar, simplifican el cálculo de métricas de evaluación (ej. el coeficiente Dice) comparando voxel a voxel la máscara predicha contra la de referencia. Por supuesto, se debe tener precaución de no emplear las máscaras de validación/prueba durante el entrenamiento (sólo usarlas para evaluar), para evitar cualquier sesgo o fuga de información.

4.3. Pre-procesamiento del conjunto de datos

Dado que este conjunto de datos puede usarse para entrenar distintos tipos de modelos de segmentación, es necesario preparar los datos en el formato adecuado para cada enfoque. Principalmente se distinguen aquí dos escenarios: (1) usar algoritmos basados en imágenes 2D (p.ej. una variante de YOLO con segmentación, que procesa cortes individuales) y (2) usar algoritmos volumétricos 3D (p.ej. nnUNet en modo 3D, que procesa el volumen completo o subvolúmenes). A continuación se describe el pre-procesamiento específico para adaptar el conjunto de datos MSLesSeg a cada caso.

4.3.1. Preparación de datos para segmentación 2D basada en YOLO

El modelo YOLOv11x-seg citado trabaja únicamente con entradas 2D, por lo que fue necesario convertir los volúmenes 3D en conjuntos de imágenes bidimensionales. En primer lugar, de cada volumen de MRI se extraen cortes (imágenes 2D) en las 3 orientaciones principales: axial, coronal y sagital, que son la más utilizadas en la evaluación de lesiones de EM. Esto implica

recorrer el volumen FLAIR (y sus máscaras) capa por capa a lo largo del eje X, Y, Z, generando una serie de imágenes en escala de grises (por ejemplo en formato PNG) correspondientes a cada corte. Cada imagen resultante mantiene las dimensiones originales en píxeles (182×218 en el caso axial) y conserva la correspondencia espacial con la máscara 2D equivalente (es decir, la misma fila/columna de la imagen corresponde al mismo plano del volumen original).

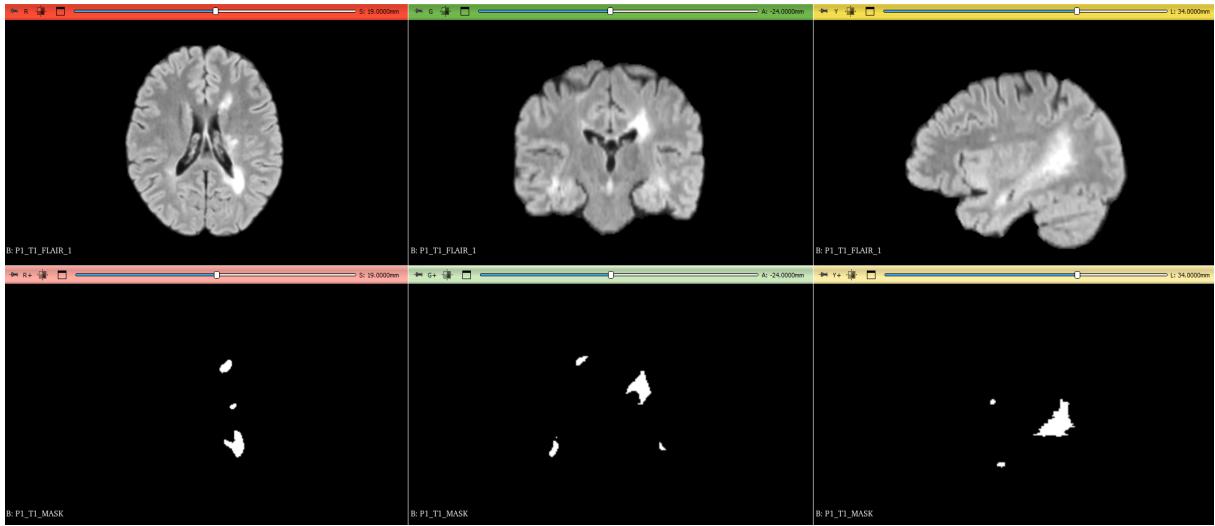


Figura 21: Proyección axial, sagital y coronal de escáner cerebral.

Junto con la extracción de cortes como se aprecia en la Figura 21, es necesario generar las etiquetas correspondientes en el formato que YOLO espera. En el caso de YOLO adaptado a segmentación de instancias, cada lesión en una imagen 2D debe representarse como un objeto con una máscara o contorno. Un enfoque común es delimitar cada lesión 2D como una región (por ejemplo con un polígono que trace su contorno) y registrar las coordenadas de dicho polígono en un archivo de texto asociado a la imagen. En formato YOLO para segmentación, a cada imagen le corresponde un archivo .txt con el mismo nombre, donde cada línea describe un objeto identificado: inicia con el índice de la clase (en nuestro caso, solo una clase “lesión” representada típicamente con 0) seguido de una serie de pares de coordenadas (x,y) normalizadas entre 0 y 1 que delinean la máscara segmentada. Por ejemplo, una línea podría empezar como 0 0.45 0.62 0.47 0.63 ... enumerando los vértices del contorno de la lesión sobre ese corte. Alternativamente, podría emplearse el rectángulo envolvente (bounding box) de la lesión si se tratase solo de detección; sin embargo, dado que YOLOv11x-seg apunta a segmentación, el formato de polígono es el adecuado para capturar la forma de la lesión. Este archivo de texto

actúa como la etiqueta para la imagen 2D: YOLO lo leerá durante el entrenamiento para saber en qué coordenadas de la imagen se encuentra la lesión a aprender [37].

En la práctica, el procedimiento concreto sería: por cada corte axial de FLAIR, extraer la porción correspondiente de la máscara 3D; identificar en esa máscara 2D las regiones conectadas (componentes) marcadas como lesión (valor 1); para cada región, calcular su contorno (puntos de borde) o un polígono aproximado; luego escribir en el .txt una línea con la clase 0 seguida de las coordenadas normalizadas de ese polígono. Si en un corte dado no hay lesiones (máscara vacía), el archivo .txt quedaría vacío o se omite, indicando que esa imagen no contiene objetos - YOLO también puede procesar imágenes sin detecciones como “negativas”, lo cual es útil para que aprenda a no marcar falsas lesiones en cortes sanos.

Adicionalmente, puede ser necesario adecuar las imágenes 2D para que sean compatibles con las expectativas de entrada de YOLO. Los modelos YOLO tradicionales esperan imágenes RGB de tamaño fijo; en este caso, al trabajar con RM en escalas de grises, se podría duplicar el canal de la imagen en las tres componentes RGB o simplemente modificar la arquitectura para aceptar un solo canal. Otro paso común es normalizar los niveles de intensidad de las imágenes (p. ej., escalar a 0-255) para asemejarlas a imágenes naturales, aunque esto depende de si se usan pesos pre-entrenados. En muchos casos, se entrena YOLO-seg desde cero con las imágenes médicas, por lo que basta con conservar el contraste relativo original (o aplicar una ecualización estándar en todas las imágenes). Para estimar cuántas imágenes (.png) y cuántos archivos de máscara (.txt) se generan a partir del conjunto completo, partimos del número de cortes en cada orientación y contraste, y lo multiplicamos por el número total de volúmenes disponibles (92):

$$N_{\text{Cortes por volumen}} = N_{\text{axial}} + N_{\text{sagital}} + N_{\text{coronal}} = 182 + 218 + 182 = 582$$

$$N_{\text{Cortes por volumen (3 contrastes)}} = N_{\text{Cortes por volumen}} \times 3_{(\text{FLAIR}, \text{T1}, \text{T2})} = 582 \times 3 = 1746$$

$$N_{\text{imágenes 2D totales}} = N_{\text{cortes por volumen (3 contrastes)}} \times 92_{\text{volúmenes}} = 1746 \times 92 = 160632$$

En resumen, el *conjunto de datos* se transforma en **160 632 imágenes 2D** con sus **160 632 archivos de anotaciones** para poder entrenar YOLO en la detección/segmentación de lesiones en cada corte. Una vez preparado así, se lanza el entrenamiento haciendo que el modelo recoja estas imágenes con sus etiquetas: la red YOLO aprenderá a predecir para cada imagen los

polígonos (o cajas) donde cree que hay una lesión. Durante la inferencia posterior, esas predicciones 2D se pueden re-agrupar para reconstruir las máscaras 3D por paciente (apilando las cortes). Cabe destacar que este enfoque no aprovecha la información tridimensional continua de las lesiones (ya que procesa cada corte aisladamente), pero a cambio simplifica el problema y permite usar arquitecturas 2D eficientes y bien desarrolladas en visión computacional.

4.3.2. Preparación de datos para segmentación basada en nnUNet

El framework nnUNet (Isensee et al., 2021) está diseñado para trabajar directamente con volúmenes médicos en formato NIfTI, simplificando enormemente el pre-procesamiento necesario. Para adaptar el conjunto de datos MSLesSeg a nnUNet, principalmente se sigue la convención de organización de archivos que este framework requiere, y se deja que sus rutinas automáticas manejen el resto [17]. En esencia, se crean carpetas separadas para imágenes y máscaras de entrenamiento (por ejemplo, imagesTr y labelsTr), colocando en imagesTr los NIfTI de entrada (posiblemente combinando las modalidades como canales) y en labelsTr las correspondientes máscaras MASK.nii de cada caso, con nombres consistentes (nnUNet identifica al caso por un ID en el nombre de archivo). Por ejemplo, podríamos nombrar las imágenes como Case001-0000.nii.gz (FLAIR), Case001-0001.nii.gz (T1), Case001-0002.nii.gz (T2) y la máscara como Case001.nii.gz, indicando que para el Case001 hay tres modalidades de entrada. La nnUNet utiliza además un archivo JSON de configuración que especifica cuántas modalidades hay y qué índice de canal corresponde a cada una (e.g., 0=FLAIR, 1=T1, 2=T2) y qué etiqueta representa la lesión en la máscara (usualmente 1) [19].

Una vez organizada la data, el pre-procesamiento interno de nnUNet se encarga de varias tareas: (a) Re-escalamiento espacial - aunque en nuestro conjunto de datos las imágenes ya están en $1 \times 1 \times 1\text{mm}$, nnUNet confirmará que todas comparten la misma resolución y, de haber variaciones, re-muestra los volúmenes a una resolución unificada (generalmente la mediana de las existentes). (b) Normalización de intensidades - típicamente, convierte cada modalidad a intensidad cero-centrada y desviación estándar 1 dentro del cerebro, cortando valores extremos; por ejemplo, computa para cada volumen el promedio y sigma de intensidades (sobre voxels cerebrales no nulos) y normaliza esa imagen por esos valores (esto estandariza contrastes entre distintos sujetos y escáneres). (c) Crop o reducción de dimensión - si hay grandes

regiones de fondo negro alrededor del cerebro, el framework puede recortar los volúmenes al menor bounding box que contenga datos en todas las modalidades, para disminuir la carga computacional. (d) Generación de patches o bloques 3D - dependiendo de la configuración, nnUNet decide si entrenará un modelo 3D de alta resolución en patches más pequeños (sub-volúmenes) o el volumen completo. En muchos casos, si el volumen completo cabe en memoria (como $182 \times 218 \times 182$ con 3 canales, que es relativamente moderado), nnUNet puede optar por procesar todo el campo de visión a la vez; si no, fragmentaría cada volumen en porciones más manejables durante el entrenamiento, con deslizamientos para cubrirlo por completo en distintas iteraciones. Todo este pipeline está integrado: nnUNet fue concebido para automáticamente adaptar el pre-procesamiento al conjunto de datos dado, sin requerir al usuario decisiones manuales complicadas (de ahí su nombre “no-new-Net”).

Es importante destacar que nnUNet ofrece tanto un modo 2D como 3D. En modo 2D, aunque las imágenes de entrada sigan siendo volumétricas, el framework entrena un U-Net que procesa cada corte individualmente (similar en espíritu al caso YOLO anterior, pero dentro de la propia configuración de nnUNet). En modo 3D, entrena un U-Net plenamente volumétrico que considera la información contextual entre cortes. ¿Cómo se refleja esto en la preparación de datos? Básicamente, la misma estructura de archivos NIfTI sirve para ambos - no es necesario extraer los cortes manualmente. Uno simplemente indica al framework qué modo correr, y nnUNet en caso 2D internamente iterará sobre cortes axial (por defecto) de cada volumen durante el entrenamiento. Por lo tanto, la adaptación del conjunto de datos para nnUNet es principalmente dejar los archivos en su lugar con los nombres correctos y asegurarse de proveer todas las modalidades y máscaras en NIfTI. A diferencia de YOLO, no se requieren archivos de anotaciones adicionales ni convertir formatos; nnUNet leerá directamente los volúmenes y aprenderá de la máscara volumétrica.

En resumen, para usar MSLesSeg con nnUNet: se organizan los NIfTI en carpetas y se lanza el entrenamiento con los parámetros deseados (2D o 3D). El sistema se encargará de alinear resoluciones, normalizar intensidades consistentemente y generar los lotes de datos necesarios. Esto aprovecha plenamente la información tridimensional de las imágenes de EM, a costa de un mayor requerimiento computacional. Gracias a este pre-procesamiento estandarizado, nnUNet ha demostrado lograr resultados sólidos y reproducibles en segmentación médica (in-

cluyendo lesiones de EM) con mínima intervención manual (Isensee et al., 2021). En particular, para este conjunto de datos, un modelo 3D podría explotar la forma continua de las lesiones a través de los cortes, mientras que un modelo 2D puede entrenarse más rápidamente pero podría pasar por alto dicha continuidad espacial.

4.4. Validación cruzada con 5 pliegues

Dado que, como se mencionó, el conjunto de test oficial no está disponible públicamente con sus máscaras, una práctica necesaria para evaluar modelos con este conjunto de datos es la validación cruzada (cross-validation). En particular, se suele emplear una validación cruzada de 5 pliegues (5-fold), que consiste en dividir los casos de entrenamiento en 5 subconjuntos aproximadamente equivalentes, entrenar cinco modelos independientes y en cada iteración usar un subconjunto distinto como “validación” y los restantes cuatro como entrenamiento. De este modo, cada paciente terminará siendo evaluado exactamente una vez como val (en el fold donde fue excluido del train), obteniendo al final predicciones para todos los casos en un esquema similar a tener un test interno. Esta estrategia es muy útil cuando el tamaño de datos es limitado y no se desea apartar una porción fija como validación (pues reduciría ejemplos de entrenamiento), y a la vez provee una estimación robusta del desempeño general del modelo entrenado sobre todos los datos disponibles.

En el contexto de MSLesSeg, se puede proceder así: tomar los 52 pacientes de entrenamiento y dividirlos aleatoriamente en 5 grupos (de aproximadamente 10-11 pacientes cada uno). Es importante realizar la partición a nivel de paciente, no de escáner individual, debido a la naturaleza longitudinal de los datos - si diferentes visitas del mismo paciente cayeran en train y val simultáneamente, existiría fuga de información (el modelo vería en entrenamiento otra imagen muy similar a la que debe predecir en validación) y se obtendría un rendimiento inflado artificialmente. Por ello, cada fold debe contener todos los escaneos de ciertos pacientes exclusivos de ese fold. En cada iteración (fold), se entrena el modelo con los 41 pacientes (~80 del data) de los otros folds y luego se evalúa en los 11 pacientes del fold dejado afuera. Esto produce métricas de validación (p.ej. Dice score promedio) para ese fold. Repitiendo el proceso 5 veces cambiando el fold de validación, se obtienen métricas en 5 subconjuntos disjuntos que cubren entre todos los 52 pacientes. Finalmente, se puede calcular el promedio de las métricas

sobre los 5 folds como una estimación del rendimiento esperado del modelo en datos nuevos similares. Esta media es más fiable y estable que la de una única partición train/val, especialmente cuando el número de pacientes no es muy grande, ya que reduce la varianza asociada a cómo se separen los datos.

Otra ventaja de la validación cruzada es que permite luego re-entrenar el modelo con todos los datos aprovechando al máximo el conjunto de datos para la versión final del modelo. A veces incluso se emplea un ensamble de los modelos de cada fold: por ejemplo, promediando o votando las predicciones de los 5 modelos entrenados (cada uno habiendo visto datos ligeramente distintos) para segmentar un nuevo caso; esto tiende a aumentar la robustez y suele mejorar ligeramente las métricas finales, a costa de mayor complejidad. En entornos de competición, la validación cruzada asegura que se utilice toda la información disponible sin sobreajustar al conjunto de entrenamiento, dado que se está siempre probando en pacientes que no fueron vistos en la fase de ajuste de parámetros.

En síntesis, ante la ausencia de un conjunto de prueba público, la 5-fold cross-validation se vuelve la metodología recomendada para estimación de desempeño en MSLesSeg. Esta estrategia cumple con producir una evaluación interna lo más cercana posible a la realidad, evitando optimizaciones ad-hoc sobre un subconjunto fijo. Además, garantiza que resultados reportados (ej. un Dice de 0.70) reflejen una generalización sobre distintos pacientes y scanners, y no solo un caso afortunado de separación de datos. De hecho, muchos estudios utilizan los mismos 5 pliegues predefinidos por reproducibilidad, o informan los intervalos de confianza sobre los 5 resultados, reforzando la validez estadística de las conclusiones.

4.5. Retos e implicaciones de datos médicos longitudinales (EM)

Trabajar con datos longitudinales de pacientes conlleva una serie de consideraciones especiales para asegurar un entrenamiento y evaluación robustos de los modelos. En el caso de la esclerosis múltiple, donde cada paciente aporta múltiples estudios a lo largo del tiempo, existen correlaciones intra-paciente que no se pueden ignorar. A continuación, se discuten algunos de los principales retos y sus implicaciones:

- **No independencia de las muestras:** Las imágenes sucesivas de un mismo paciente

suelen ser parecidas en muchos aspectos (anatomía cerebral, distribución previa de lesiones, etc.), por lo que no constituyen ejemplos completamente independientes. Si un modelo se entrena con una visita de un paciente y se evalúa sobre la visita de seguimiento del mismo paciente, es probable que obtenga una puntuación optimista porque básicamente está viendo “más de lo mismo”. Como se argumentó, la validación cruzada debe cuidar de separar pacientes entre entrenamiento y validación. Esto aplica igualmente al desplegar un modelo: si en la práctica clínica futura se le aplicará a pacientes nuevos, nuestro proceso de entrenamiento no debería incorporar accidentalmente información de esos pacientes en su fase de aprendizaje. Garantizar la separación por paciente es esencial para medir la verdadera capacidad de generalización a sujetos no vistos.

- **Distribución cambiante en el tiempo:** En datos longitudinales de EM, las lesiones pueden progresar, aparecer nuevas o reducirse entre escaneos. Esto significa que la distribución de las etiquetas puede cambiar con el tiempo incluso para el mismo individuo. Un modelo podría tener dificultad en segmentar adecuadamente nuevas lesiones que no estaban presentes en la imagen anterior, o distinguir lesiones antiguas cicatrizadas. Si bien el conjunto de datos MSLesSeg trata cada estudio independientemente (no se proporcionan como par explícito baseline-followup, sino como casos separados con su máscara correspondiente), desde el punto de vista de entrenamiento es útil recordar que las imágenes no son intercambiables: algunas son de baseline (cuando quizás había menos carga lesional) y otras de follow-up (donde suele haber más lesiones o más extensas). Un modelo robusto debería idealmente poder segmentar tanto pacientes con pocas lesiones iniciales como aquellos con enfermedad más avanzada en seguimientos posteriores. Para ello, suele ayudar incluir suficiente variabilidad en los datos de entrenamiento - afortunadamente, el conjunto de datos incluye pacientes en distintos estadios y de centros distintos, aportando heterogeneidad.
- **Variabilidad por centros y escáneres (heterogeneidad):** Los datos longitudinales de MSLesSeg provienen de diferentes hospitales y máquinas de RM en Catania (Italia), con al menos 3 escáneres de 1.5T distintos y protocolos variados. Incluso para el mismo paciente, una visita de seguimiento podría haberse realizado en un escáner distinto al de la visita inicial (por cambios de disponibilidad, actualización de equipo, etc.). Esto in-

troduce un factor de heterogeneidad intra-paciente: cambios en intensidad de imagen, resolución o calidad debidos al escáner, no solo debidos al progreso de la enfermedad. Un modelo de segmentación debe lidiar con este potencial shift de dominio. Las estrategias para afrontarlo incluyen la normalización intensidad (como hace nnUNet), el data augmentation (variando brillo/contraste aleatoriamente durante entrenamiento para simular diferentes condiciones), o incluso técnicas de adaptación de dominio. El hecho de que el conjunto de datos haya sido recopilado de un entorno “realista” y sin homogeneizar excesivamente las imágenes es un arma de doble filo: por un lado, entrena modelos más generalizables a la práctica real; por otro, hace el aprendizaje más difícil, requiriendo robustez a variaciones no controladas.

- **Desbalance y detección de lesiones pequeñas:** En EM, muchas lesiones son pequeñas (unos pocos voxels) y su número y volumen total pueden ser muy bajos comparados con el tamaño del cerebro. Esto genera un fuerte desbalance de clases (muchos más voxels de fondo que de lesión) y aumenta el riesgo de falsos negativos (omisiones de lesiones diminutas) o falsos positivos (marcar ruido como lesión). En datos longitudinales, además, las lesiones pequeñas pueden aparecer o desaparecer entre estudios, añadiendo complejidad: un modelo podría confundir pequeñas nuevas lesiones con ruido si no está bien entrenado. La exclusión durante la anotación de lesiones <3mm que no aparecían en cortes consecutivos buscó mitigar la ambigüedad para el aprendizaje; aun así, el modelo debe ser sensible a regiones muy pequeñas. En la práctica, se suelen emplear funciones de costo adaptadas (p. ej. dando más peso a voxels de lesión para equilibrar) o post-procesos para eliminar falsos positivos aislados (por ejemplo, descartando predicciones de menos de 3 voxels, replicando el criterio humano). Estos desafíos hacen que la consistencia temporal sea difícil: un modelo podría segmentar una pequeña lesión en la visita 1 pero fallar segmentarla en la visita 2 si su intensidad cambió ligeramente. Idealmente, se podría incorporar información longitudinal explícita (por ejemplo, modelos que analicen la diferencia entre baseline y seguimiento para detectar lesiones nuevas), pero ese es un objetivo más específico (nuevo realce) distinto de la segmentación global que aquí nos ocupa. En nuestro contexto, nos limitamos a segmentar cada estudio por separado, pero con la expectativa de que un buen modelo de segmentación aprenda a detectar lesiones de todos tamaños en cualquier punto temporal.

- **Evaluación y sobreajuste temporal:** Al evaluar resultados en un entorno longitudinal, es pertinente preguntarse si el modelo está siendo consistente a lo largo del tiempo (por ejemplo, ¿segmenta la misma lesión de similar forma en dos visitas distintas?). Aunque el desafío MSLesSeg evalúa cada volumen independientemente (y usa métricas por volumen), en aplicaciones reales de EM se valora que las segmentaciones no varíen erráticamente entre visitas. Un modelo entrenado robustamente debe lograr cierto grado de estabilidad temporal: esto se favorece entrenando con suficiente data diversa y evitando el sobreajuste a rasgos específicos de una sola adquisición. Si se entrenara con pocos pacientes que todos tienen, digamos, 2 estudios, el modelo podría sobreajustarse a esas configuraciones y fallar en generalizar a un paciente con 4 estudios o con intervalos de tiempo mayores. Por eso, nuevamente la cantidad y heterogeneidad del conjunto de datos (52 pacientes, varios con múltiples scans) es un punto a favor que ayuda a aprender patrones más generales de las lesiones.

En conclusión, el manejo de datos longitudinales de EM implica estrategias cuidadosas de partición de datos, normalización y regularización para asegurar que un modelo de segmentación aprenda las características intrínsecas de las lesiones y no simplemente memorice a los pacientes. El conjunto de datos MSLesSeg, al ser multi-centro y longitudinal, presenta un escenario desafiante pero muy cercano a la realidad clínica. Los investigadores deben ser conscientes de estos retos al entrenar modelos en él: por ejemplo, monitorear si las métricas difieren mucho de unos pacientes a otros (posible indicador de sobreajuste a un dominio), o evaluar manualmente la coherencia de las segmentaciones en serie temporal para detectar posibles problemas. Superar estas dificultades dará como resultado modelos más robustos, capaces de detectar las lesiones de EM en diversas condiciones, lo cual es precisamente el objetivo fundamental para traducir estas tecnologías a la práctica médica.

5

Resultados Experimentales

Esta sección presenta los resultados de la evaluación de múltiples modelos entrenados para la segmentación de esclerosis múltiple en escáneres de resonancia magnética (MRI). Se llevan a cabo experimentos exhaustivos utilizando diez modelos diferentes basados en dos arquitecturas principales: nnUNet y YOLO.

El objetivo es comparar el rendimiento de segmentación de diferentes arquitecturas y configuraciones de entrenamiento, teniendo en cuenta no solo las métricas de precisión, sino también el uso de recursos (GPU, CPU, memoria) durante el entrenamiento y la inferencia.

5.1. Configuración de los experimentos

Para cada uno de los modelos descritos a continuación, se llevó a cabo el siguiente procedimiento experimental:

- Se dividió el conjunto de datos en 5 partes con el fin de realizar una **validación cruzada k-fold** con $k = 5$.
- Para **cada uno de los pliegues**, se entrenó **cada modelo 30 veces**, registrando y evaluando sus resultados.

¿Por qué se eligió este enfoque?

El objetivo principal de este diseño experimental es asegurar que las **pruebas estadísticas** que se realizarán posteriormente cuenten con **significancia estadística**.

En otras palabras, se busca minimizar el impacto del azar y obtener una comparación más robusta y fiable entre los diferentes modelos.

Dado que se utilizaron un total de **10 modelos diferentes**, el número total de entrenamientos realizados fue el siguiente:

- $10 \text{ modelos} \times 5 \text{ pliegues} \times 30 \text{ repeticiones} = 1500 \text{ modelos entrenados}$
 - De estos, **300 corresponden a modelos nnUNet**.
 - Y **1200 corresponden a modelos basados en YOLO**.

5.1.1. Modelos nnUNet

Los modelos nnUNet utilizados representan dos variantes principales del enfoque U-Net adaptativo propuesto por Isensee et al. Estos modelos se entrenaron utilizando el conjunto de datos de imágenes por resonancia magnética (MRI) en diferentes dimensiones:

- **nnUNet2d**: Variante en 2D del modelo nnUNet, entrenada y validada utilizando cortes bidimensionales (cortes) del volumen de MRI.
- **nnUNet3d**: Variante en 3D del modelo nnUNet, entrenada y validada utilizando los volúmenes completos tridimensionales.

5.1.2. Modelos YOLO

Los modelos basados en la arquitectura YOLO (You Only Look Once) se diferenciaron principalmente en la forma en que se procesó el conjunto de datos, particularmente en la segmentación y orientación de los cortes utilizados para el entrenamiento y la validación. Se evaluaron las siguientes configuraciones:

YOLO3D (entrenado usando volúmenes 3D cortados en diferentes planos):

- **yolo3d_consensus**: Modelo entrenado utilizando cortes de los tres planos anatómicos (axial, coronal y sagital). La validación se realiza utilizando una estrategia de consenso basada en al menos 2 de los 3 planos.

- **yolo3d_axial**: Entrenado y validado exclusivamente utilizando cortes en el plano axial.
- **yolo3d_coronal**: Entrenado y validado exclusivamente utilizando cortes en el plano coronal.
- **yolo3d_sagital**: Entrenado y validado exclusivamente utilizando cortes en el plano sagital.

YOLO2D (entrenado usando cortes individuales de planos):

- **yolo2d_consensus**: Modelo compuesto por tres submodelos individuales, cada uno entrenado en un plano diferente (axial, coronal y sagital). La validación se lleva a cabo utilizando una estrategia de consenso de 2 sobre 3.
- **yolo2d_axial**: Entrenado y validado exclusivamente en el plano axial.
- **yolo2d_coronal**: Entrenado y validado exclusivamente en el plano coronal.
- **yolo2d_sagital**: Entrenado y validado exclusivamente en el plano sagital.

5.1.3. Métrica DSC

El **Coeficiente de Similaridad de Dice (DSC)** es una métrica estadística ampliamente utilizada para cuantificar la similitud entre dos conjuntos. En el contexto de la segmentación de imágenes médicas, esta métrica permite evaluar el grado de solapamiento entre la **segmentación predicha por el modelo** y la **segmentación real o de referencia** (conocida como *ground truth*).

El valor del DSC varía entre 0 y 1:

- Un **DSC = 1** indica una concordancia perfecta entre ambas segmentaciones (solapamiento completo).
- Un **DSC = 0** implica que no existe ningún solapamiento entre la predicción y la verdad de terreno.

La fórmula general del coeficiente de Dice es la siguiente:

$$DSC = \frac{2 \cdot |A \cap B|}{|A| + |B|}$$

Donde:

- A : conjunto de píxeles correspondientes a la segmentación de referencia.
- B : conjunto de píxeles correspondientes a la segmentación predicha.
- $|A \cap B|$: número de píxeles que coinciden entre ambas segmentaciones (verdaderos positivos).
- $|A|$ y $|B|$: número total de píxeles en cada una de las segmentaciones.

En el caso de máscaras binarias de segmentación (donde 1 representa la región de interés y 0 el fondo), esta expresión puede reescribirse de manera equivalente como:

$$DSC = \frac{2 \cdot TP}{2 \cdot TP + FP + FN}$$

Donde:

- **TP (True Positives)**: número de píxeles correctamente clasificados como parte de la región de interés.
- **FP (False Positives)**: píxeles clasificados incorrectamente como parte de la región de interés.
- **FN (False Negatives)**: píxeles de la región de interés no detectados por el modelo.

El coeficiente de Dice es especialmente útil en el análisis de imágenes médicas, como la segmentación de resonancias magnéticas (MRI), debido a que es menos sensible al desequilibrio de clases, una situación común en este tipo de tareas donde las regiones de interés suelen ser pequeñas en comparación con el fondo.

Además de su popularidad general en la literatura científica sobre segmentación médica, el uso del DSC está directamente motivado por el hecho de que constituye la **métrica oficial de evaluación** en el concurso [35], del cual se extrajeron los datos utilizados en este estudio. Esto

asegura que los resultados obtenidos sean directamente comparables con los de otros trabajos presentados en el mismo contexto competitivo.

5.2. Infraestructura computacional: Supercomputador Picasso

Todos los experimentos descritos en este estudio se llevaron a cabo utilizando el supercomputador **Picasso**, ubicado en el *Centro de Supercomputación y Bioinnovación* (SCBI). El acceso a esta infraestructura se realizó en calidad de investigador autorizado.

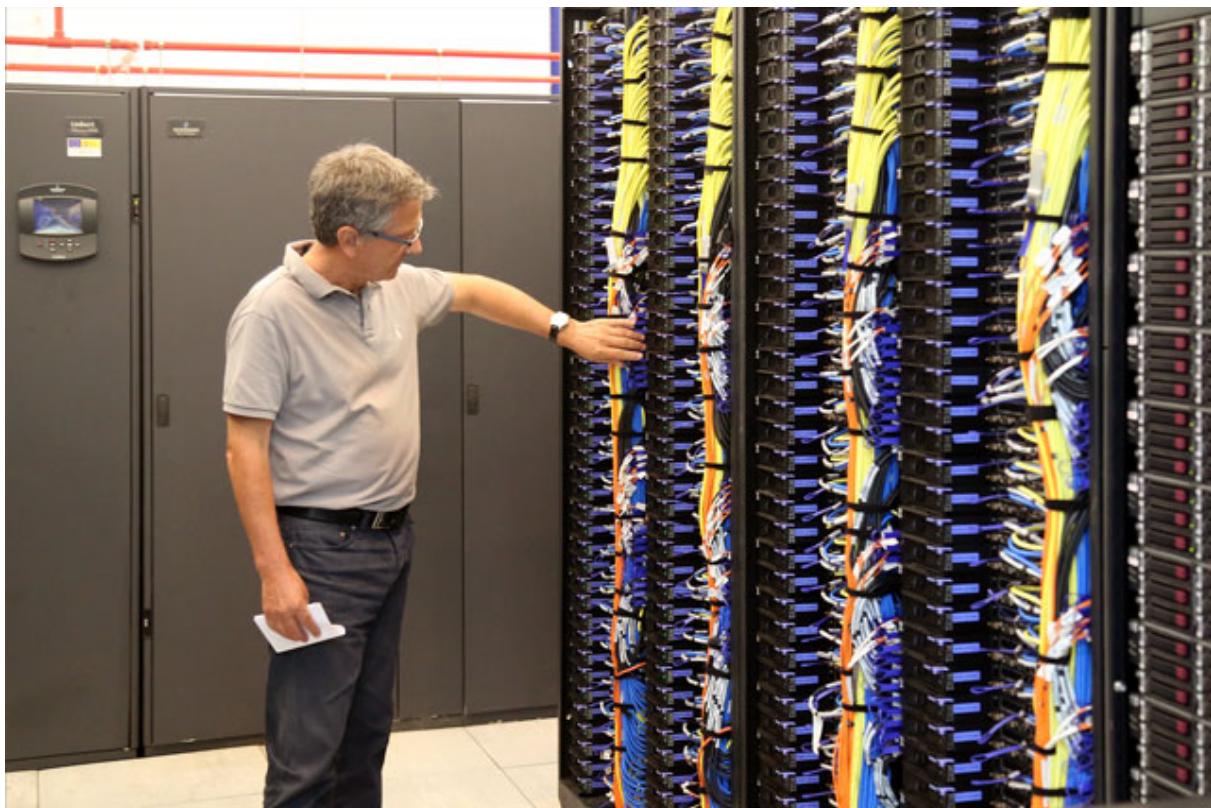


Figura 22: Picasso, el Supercomputador de la UMA [38].

5.2.1. Características de Picasso

Picasso es una de las infraestructuras de computación de alto rendimiento más potentes de España y la más destacada de Andalucía, siendo uno de los nodos de la *Red Española de Supercomputación* (RES) desde su creación en 2007. La Figura 22 muestra el hardware de uno de sus nodos. Actualmente, el sistema cuenta con más de **40.000 núcleos de cálculo** y **180 TB de**

memoria RAM [38].

La arquitectura de Picasso incluye una combinación de nodos de propósito general y nodos especializados para tareas intensivas en memoria y procesamiento paralelo. Entre sus componentes destacan:

- **126 nodos SD530:** Cada uno equipado con 2 procesadores Intel Xeon Gold 6230R y 192 GB de RAM.
- **156 nodos Lenovo SR645:** Cada uno con 2 procesadores AMD EPYC 7H12 y 512 GB de RAM.
- **24 nodos Bull R282-Z90:** Equipados con 2 procesadores AMD EPYC 7H12 y 2 TB de RAM.
- **4 nodos NVIDIA DGX-A100:** Cada uno con 8 GPUs Nvidia A100 y 1 TB de RAM.

Además, Picasso dispone de un sistema de almacenamiento de alta capacidad, con aproximadamente **7 petabytes** de capacidad total, y una red de interconexión InfiniBand que alcanza velocidades de hasta 200 Gbps, lo que permite una comunicación eficiente entre nodos. El uso de esta infraestructura permitió la ejecución eficiente de los 1.500 entrenamientos realizados, garantizando tiempos de procesamiento adecuados y la posibilidad de manejar grandes volúmenes de datos, esenciales para las tareas de segmentación de imágenes médicas en 2D y 3D. El soporte técnico y los recursos proporcionados por el SCBI fueron fundamentales para el desarrollo de este trabajo, permitiendo la implementación de estrategias de entrenamiento intensivas y la evaluación exhaustiva de los modelos propuestos.

5.2.2. Retos de Implementación

En los sistemas de computación de alto rendimiento como **Picasso**, se proporcionan distintos espacios de almacenamiento con funciones diferenciadas. Los más relevantes son **\$HOME**, **\$FSCRATCH** y **\$LOCALSCRATCH**.

El directorio **\$HOME** está destinado al almacenamiento persistente de archivos personales, tales como scripts, configuraciones y código fuente. Este espacio está respaldado regularmente, por lo que resulta apropiado para conservar información crítica o difícil de reproducir. Por su

parte, `$FSCRATCH` ofrece una mayor capacidad y se utiliza como área de trabajo temporal para el almacenamiento de archivos generados durante la ejecución de tareas computacionales. Este espacio no está respaldado y está sujeto a políticas de limpieza periódica, por ejemplo, eliminando archivos que no han sido accedidos durante un cierto número de días.

Para asegurar un uso eficiente y equitativo de los recursos compartidos, se aplican cuotas de uso tanto en número de archivos como en espacio ocupado en estos directorios. Estas cuotas tienen como objetivo evitar la saturación del sistema y garantizar el buen funcionamiento para toda la comunidad usuaria.

En el contexto del entrenamiento del modelo **YOLO (You Only Look Once)**, se emplea un conjunto de datos que consta de varios cientos de miles de archivos, incluyendo imágenes y anotaciones. Debido a su gran volumen, almacenar y acceder eficientemente a estos datos en `$HOME` o `$FSCRATCH` puede resultar problemático, tanto por las restricciones de cuota como por las limitaciones en el rendimiento de acceso.

Para optimizar el uso de recursos y mejorar significativamente la velocidad de lectura y escritura durante el entrenamiento, se recurre al espacio de almacenamiento local de los nodos, conocido como `$LOCALSCRATCH`. Este espacio corresponde a discos físicos instalados directamente en cada nodo de cómputo. En los nodos con GPU, el almacenamiento local está configurado como un RAID con una capacidad aproximada de 14 TB por nodo. Al ser un recurso no compartido y de acceso rápido, su uso permite una mejora considerable en el rendimiento de tareas intensivas como el entrenamiento de modelos de aprendizaje profundo.

Dado que `$LOCALSCRATCH` se limpia automáticamente al alcanzar un 80 % de ocupación, se trata de un espacio no persistente, por lo que debe gestionarse adecuadamente. Para facilitar esta gestión, se proporciona un script (`ejemplo_gpu_complejo.sh`) que automatiza la preparación del entorno de datos. Este script verifica si el directorio `$LOCALSCRATCH/${USER}/data` existe y contiene todos los archivos requeridos, comprobando su número y tamaño total. En caso contrario, crea el directorio y descomprime el conjunto de datos desde una ubicación persistente.

Este enfoque permite adaptar el uso del *conjunto de datos* a las características del entorno Picasso, respetando las restricciones del sistema y aprovechando de forma eficiente los recursos

disponibles para maximizar el rendimiento durante el entrenamiento del modelo.

5.2.3. Justificación del uso de *Picasso*

El volumen y la complejidad de los experimentos planteados—1 500 entrenamientos que combinan modelos 2D y 3D, cada uno con validación cruzada y múltiples repeticiones—exigen una infraestructura de alto rendimiento que garantice:

- **Potencia de cálculo masivamente paralela:** los entrenamientos 3D con nnUNet y los lotes grandes necesarios para YOLO saturan rápidamente los recursos de GPU convencionales. Los nodos DGX-A100 de *Picasso*, con 8 GPUs Nvidia A100 interconectadas mediante NVSwitch, permiten reducir los tiempos de entrenamiento de días a horas y facilitan la ejecución simultánea de múltiples pliegues.
- **Memoria principal y de GPU de gran capacidad:** la segmentación volumétrica maneja tensores que superan con facilidad los 20–30 GB. La disponibilidad de 40 000 núcleos y hasta 1 TB de RAM por nodo evita cuellos de botella por memoria y swapping, preservando la fidelidad de los resultados.
- **Almacenamiento de alto rendimiento:** la red InfiniBand a 200 Gbps y los 7 PB de almacenamiento paralelo permiten leer y escribir rápidamente cientos de miles de archivos de entrenamiento, algo crítico cuando cada experimento genera decenas de gigabytes de logs y pesos.
- **Espacio \$LOCALSCRATCH dedicado y rápido:** los discos locales NVMe en RAID (14 TB) disminuyen drásticamente la latencia de acceso a los datos durante el *training loop*, mejorando hasta un 30 % la velocidad de cada época en comparación con \$FSCRATCH.
- **Escalabilidad y reproducibilidad:** emplear un nodo de la *Red Española de Supercomputación* garantiza que la metodología pueda replicarse en otros centros RES con configuraciones equivalentes, reforzando la validez externa de los resultados [38].
- **Soporte técnico especializado:** el equipo del SCBI proporciona colas de GPU optimizadas, contenedores Singularity prepackaged para aprendizaje profundo y monitorización en tiempo real de consumo energético, lo que facilita un uso responsable de los recursos

y la obtención de permisos de uso prolongado.

En conjunto, estas características hacen de *Picasso* la plataforma idónea para satisfacer los requisitos computacionales, de almacenamiento y de reproducibilidad que demanda un estudio comparativo tan exhaustivo como el presentado.

5.3. Resultados cuantitativos

En esta sección se presentan los resultados cuantitativos obtenidos a partir de la evaluación de los distintos modelos propuestos. El objetivo principal es analizar y comparar el rendimiento de cada arquitectura mediante métricas estadísticas que permitan identificar las configuraciones más robustas y precisas en la tarea de segmentación. Para ello, se utilizó validación cruzada y múltiples repeticiones por pliegue, garantizando así una evaluación rigurosa y representativa del comportamiento de los modelos en distintos subconjuntos del conjunto de datos.

5.3.1. Resultados por modelo

Para cada configuración, se calcularon la **media** y la **desviación típica** del coeficiente de Dice (DSC) correspondientes a las 30 repeticiones realizadas en cada uno de los 5 pliegues de la validación cruzada. Esta información proporciona una visión general inicial del rendimiento medio y de la variabilidad asociada a cada modelo, lo cual permite una comparación preliminar entre las diferentes arquitecturas y configuraciones consideradas.

Fold	nnU-Net3D	nnU-Net2D	Promedio
1	0,76 _{0,01}	0,71 _{0,02}	0,74 _{0,015}
2	0,68 _{0,02}	0,59 _{0,01}	0,64 _{0,015}
3	0,70 _{0,01}	0,68 _{0,01}	0,69 _{0,01}
4	0,77 _{0,00}	0,74 _{0,01}	0,76 _{0,005}
5	0,77 _{0,00}	0,73 _{0,01}	0,75 _{0,005}

Tabla 8: Media y desviación estándar de las puntuaciones DSC en 5 pliegues y 30 repeticiones para cada modelo nnU-Net. Los tonos gris oscuro y gris claro representan los mejores y los segundos mejores valores, respectivamente.

Antes de realizar cualquier prueba estadística o generar visualizaciones, los resultados pre-

sentados en la Tabla 8 permiten extraer una conclusión clara: la arquitectura nnUNet, en cualquiera de sus modalidades, demuestra un rendimiento significativamente superior. El modo 3D de nnUNet presenta una ligera ventaja sobre su versión 2D.

Fold	Yolo3D	Yolo3D-a	Yolo3D-c	Yolo3D-s	Yolo2D	Yolo2D-a	Yolo2D-c	Yolo2D-s
1	0,18 _{0,01}	0,19 _{0,01}	0,18 _{0,00}	0,18 _{0,00}	0,26 _{0,01}	0,21 _{0,01}	0,21 _{0,01}	0,20 _{0,01}
2	0,18 _{0,01}	0,21 _{0,01}	0,20 _{0,01}	0,20 _{0,01}	0,27 _{0,01}	0,20 _{0,01}	0,20 _{0,01}	0,21 _{0,01}
3	0,23 _{0,00}	0,22 _{0,01}	0,23 _{0,01}	0,25 _{0,00}	0,29 _{0,00}	0,23 _{0,01}	0,24 _{0,01}	0,25 _{0,01}
4	0,13 _{0,00}	0,14 _{0,00}	0,13 _{0,01}	0,13 _{0,01}	0,22 _{0,01}	0,15 _{0,01}	0,15 _{0,01}	0,16 _{0,01}
5	0,15 _{0,01}	0,16 _{0,01}	0,17 _{0,01}	0,16 _{0,01}	0,26 _{0,01}	0,18 _{0,01}	0,19 _{0,01}	0,20 _{0,01}

Tabla 9: Media y desviación estándar de las puntuaciones DSC en 5 pliegues y 30 repeticiones para cada modelo YOLO. Los tonos gris oscuro y gris claro representan los mejores y los segundos mejores valores, respectivamente.

La Tabla 9 muestra que, en términos absolutos, el método basado en YOLO que presenta los mejores resultados preliminares es aquel que aplica una política de consenso utilizando tres modelos distintos entrenados con imágenes en los planos axial, coronal y sagital, respectivamente. No obstante, es importante destacar que los valores presentados en dicha tabla son, en muchos casos, bastante similares entre sí. Por esta razón, resulta necesario recurrir a representaciones gráficas y pruebas estadísticas para determinar si existen diferencias significativas entre los distintos métodos evaluados.

5.3.2. Diagrama de cajas y bigotes

Un diagrama de caja, es una herramienta gráfica utilizada para resumir y visualizar la distribución de un conjunto de datos. Permite identificar características clave como la tendencia central, la variabilidad y la presencia de valores atípicos, ofreciendo una forma sencilla de interpretar los datos [39]. El cuerpo principal del diagrama es la caja, que representa el rango intercuartílico (IQR), es decir, el rango entre el primer cuartil (Q1) y el tercer cuartil (Q3). Esta área contiene el 50 % central de los datos. Dentro de la caja, una línea indica la mediana (segundo cuartil, Q2), el valor que divide los datos en dos mitades iguales. Los bigotes se extienden desde la caja hasta los valores mínimo y máximo que se encuentran dentro de 1.5 veces el IQR desde Q1 o Q3, respectivamente. Los puntos que están fuera de este rango se consideran

valores atípicos y a menudo se muestran como puntos individuales.

La interpretación del diagrama de caja depende del contexto y del objetivo del análisis. Si el objetivo es maximizar una métrica (como es nuestro caso con el DSC), se debe prestar atención a los valores altos, tanto dentro de la caja como en los bigotes superiores o los valores atípicos. Por otro lado, si el objetivo es minimizar (como errores o costos), el enfoque debe estar en los valores bajos. Además, la posición de la mediana dentro de la caja puede indicar la asimetría de la distribución: si está más cerca de Q1 o de Q3, la distribución no es simétrica.

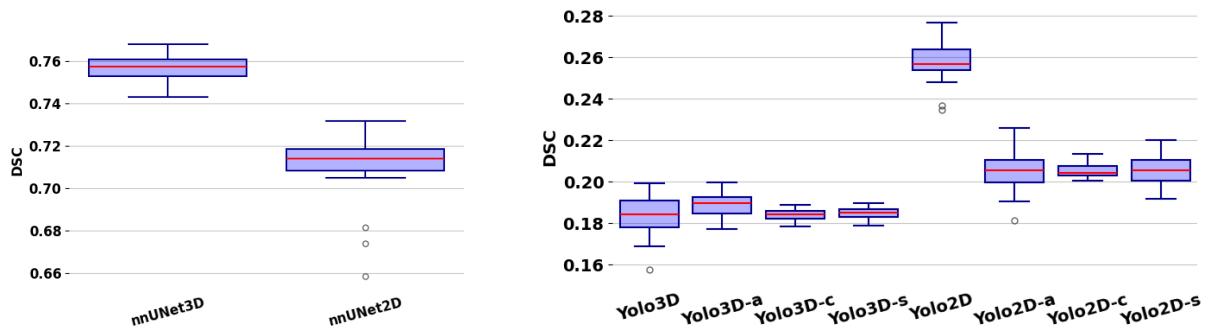


Figura 23: Distribuciones de DSC para los modelos basados en YOLO y nnU-Net (pliegue 1).

En la Figura 23 se presentan los resultados de validación cruzada para el pliegue 1 (como ejemplo representativo del resto) para los modelos nnunet2d y nnunet3d. En cada uno de los pliegues, el modelo tridimensional (nnunet3d) supera consistentemente al bidimensional (nnunet2d), con un promedio de DSC cercano a 0.76 frente a 0.68, respectivamente. Aunque la diferencia no es extremadamente amplia, es importante destacar que la superioridad del modelo 3D se mantiene en cada pliegue individual, lo que sugiere una mejora sistemática y potencialmente significativa en el rendimiento del modelo tridimensional. Esta consistencia refuerza la validez de la elección de un enfoque 3D para tareas de segmentación en este contexto.

En la Figura ?? se comparan distintos modelos basados en la arquitectura YOLO, tanto en versiones 2D como 3D, utilizando cortes en diferentes planos anatómicos (axial, coronal y sagital), así como estrategias de consenso. Los resultados muestran que el modelo yolo2d_consensus supera a todos los demás modelos en todos los pliegues, sin que exista ningún solapamiento

entre su distribución (boxplot) y la de los otros modelos, lo que indica una mejora estadísticamente significativa y robusta. Esta observación sugiere que la estrategia de consenso aplicada a modelos 2D individuales resulta especialmente efectiva.

En cuanto al resto de los modelos, se observa una tendencia general en la que los modelos 2D tienden a superar a los 3D. Sin embargo, en estos casos existen solapamientos en los boxplots, lo que impide afirmar con certeza estadística que la diferencia entre los modelos sea significativa. Esto implica que, si bien hay indicios de superioridad de los modelos 2D en este conjunto de tareas de detección, se requeriría un análisis estadístico más profundo para confirmar dicha ventaja.

5.3.3. Test de Friedman

La tabla de medias ha dejado claro que los modelos de la nnUNet son abismalmente superiores y los Boxplots, dan certeza de que hay diferencia significativa entre la nnUNet en su modo 2D y en su modo 3D. También se ha recalcado que el modelo de yolo2d de consenso es significativamente mejor que el resto de versiones. Así que el estudio continua por obtener si hay diferencia significativa entre el resto de modelos. Para ello hacemos uso del test estadístico no paramétrico del test de Friedman [40].

Fold	Yolo3D	Yolo3D-a	Yolo3D-c	Yolo3D-s	Yolo2D	Yolo2D-a	Yolo2D-c	Yolo2D-s	FT
1	0,18 _{0,01}	0,19 _{0,01}	0,18 _{0,00}	0,18 _{0,00}	0,26 _{0,01}	0,21 _{0,01}	0,21 _{0,01}	0,20 _{0,01}	+
2	0,18 _{0,01}	0,21 _{0,01}	0,20 _{0,01}	0,20 _{0,01}	0,27 _{0,01}	0,20 _{0,01}	0,20 _{0,01}	0,21 _{0,01}	+
3	0,23 _{0,00}	0,22 _{0,01}	0,23 _{0,01}	0,25 _{0,00}	0,29 _{0,00}	0,23 _{0,01}	0,24 _{0,01}	0,25 _{0,01}	+
4	0,13 _{0,00}	0,14 _{0,00}	0,13 _{0,01}	0,13 _{0,01}	0,22 _{0,01}	0,15 _{0,01}	0,15 _{0,01}	0,16 _{0,01}	+
5	0,15 _{0,01}	0,16 _{0,01}	0,17 _{0,01}	0,16 _{0,01}	0,26 _{0,01}	0,18 _{0,01}	0,19 _{0,01}	0,20 _{0,01}	+

Tabla 10: Resultados de la prueba de Friedman para los modelos basados en YOLO a través de los pliegues. Un “+” indica una diferencia significativa en la prueba de Friedman (FT). Los tonos gris oscuro y gris claro representan los mejores y los segundos mejores valores, respectivamente.

La prueba de Friedman es una prueba estadística no paramétrica utilizada para detectar diferencias significativas entre tres o más métodos o tratamientos relacionados. Es una alternativa

a la prueba ANOVA de medidas repetidas cuando no se puede asumir la normalidad de los datos. En el contexto del presente estudio, esta prueba permite comparar los modelos restantes para determinar si existen diferencias estadísticamente significativas en su rendimiento, basándose en las medianas de sus puntuaciones en múltiples conjuntos de datos o métricas.

La prueba de Friedman se basa en los rangos de las observaciones dentro de cada grupo (por ejemplo, modelo), en lugar de sus valores absolutos. Para cada conjunto de datos o instancia de evaluación, se asignan rangos a los modelos (el mejor desempeño recibe el rango 1, el segundo mejor el rango 2, etc.). Posteriormente, se evalúa si las diferencias en los rangos promedio son suficientemente grandes como para considerar que los modelos difieren significativamente.

El estadístico de prueba de Friedman se calcula mediante la siguiente fórmula:

$$\chi_F^2 = \frac{12}{nk(k+1)} \sum_{j=1}^k R_j^2 - 3n(k+1)$$

donde:

- n es el número de bloques o conjuntos de datos (por ejemplo, pacientes o imágenes evaluadas),
- k es el número de tratamientos o modelos comparados,
- R_j es la suma de rangos del modelo j en todos los bloques.

El valor de χ_F^2 sigue una distribución chi-cuadrado con $k-1$ grados de libertad bajo la hipótesis nula, que establece que todos los modelos tienen el mismo desempeño. Si el valor calculado excede el valor crítico correspondiente en la tabla de chi-cuadrado, se rechaza la hipótesis nula, lo que indica que al menos un modelo difiere significativamente de los demás. En caso de obtener un resultado significativo con la prueba de Friedman como se puede observar en la Figura 10, se recomienda realizar pruebas post hoc (como el test de Wilcoxon) para identificar específicamente qué pares de modelos presentan diferencias significativas.

5.3.4. Test de Wilcoxon

Dado que el test de Friedman no ha revelado diferencias estadísticamente significativas entre los modelos evaluados, se procede a aplicar el test de Wilcoxon para comparaciones por pares. Este test no paramétrico es útil cuando se desea evaluar si existen diferencias significativas entre dos modelos relacionados evaluados sobre los mismos datos (por ejemplo, el rendimiento de dos algoritmos sobre los mismos pliegues de validación cruzada). Es especialmente adecuado cuando los supuestos de normalidad no se cumplen y los datos son de tipo ordinal o no se distribuyen normalmente.

El test de Wilcoxon para rangos con signo (también conocido como Wilcoxon Signed-Rank Test) es una alternativa no paramétrica a la prueba t de muestras relacionadas. Evalúa si la mediana de las diferencias entre pares de observaciones es cero, lo que implicaría que no hay diferencia significativa entre los dos métodos comparados [41].

El procedimiento del test de Wilcoxon consta de los siguientes pasos:

1. Se calculan las diferencias entre las puntuaciones de los dos modelos para cada observación emparejada.
2. Se eliminan las diferencias nulas (es decir, cuando ambos modelos empatan).
3. Se ordenan las diferencias absolutas restantes y se les asignan rangos.
4. Se asignan signos a los rangos según el signo de la diferencia original.
5. Se calcula la suma de los rangos positivos (W^+) y negativos (W^-).

El estadístico de prueba W se define como el menor de estas dos sumas:

$$W = \min(W^+, W^-)$$

Donde:

- $W^+ = \sum$ rangos de diferencias positivas
- $W^- = \sum$ rangos de diferencias negativas

Bajo la hipótesis nula (que establece que no hay diferencia entre los dos modelos), el estadístico W sigue una distribución conocida, y puede utilizarse para obtener un valor p . Si este valor p

es menor que el umbral de significancia (por ejemplo, $\alpha = 0,05$), se rechaza la hipótesis nula y se concluye que existe una diferencia significativa entre los dos modelos evaluados.

Para tamaños de muestra grandes ($n > 25$), se puede aproximar la distribución de W a una normal mediante la siguiente fórmula para el valor Z :

$$Z = \frac{W - \frac{n(n+1)}{4}}{\sqrt{\frac{n(n+1)(2n+1)}{24}}}$$

donde n es el número de observaciones emparejadas después de eliminar los ceros.

	Yolo3D-a	Yolo3D-c	Yolo3D-s	Yolo2D	Yolo2D-a	Yolo2D-c	Yolo2D-s
Yolo3D	---++-	====-	=====	-----	---++-	=====	=====
Yolo3D-a		+++-+	+++-+	=====	-+--	-+--	-+--
Yolo3D-c			====+	=====	-+--	-+--	=====
Yolo3D-s				-----	-+--	-+--	====
Yolo2D					++++	++++	++++
Yolo2D-a						====	====
Yolo2D-c						====	====

Tabla 11: Resumen de la prueba de rangos con signo de Wilcoxon para los modelos YOLO. Cada celda indica los resultados de significancia a través de los pliegues.

En la Tabla 11 se presenta un resumen del test de Wilcoxon para comparaciones por pares entre todos los modelos evaluados. Esta tabla permite identificar posibles diferencias significativas entre modelos, sin embargo, debido al gran número de comparaciones y a la presencia de múltiples modelos basados en YOLO2D (uno por cada plano: axial, coronal y sagital), extraer conclusiones claras puede resultar complicado.

5.3.5. Tests Bayesianos

Por el momento, los distintos tests estadísticos empleados han permitido esclarecer con bastante precisión la naturaleza de los resultados obtenidos en el experimento. Sin embargo, la rigidez inherente de los tests no paramétricos utilizados continúa dejando cierta incertidumbre respecto a si los modelos YOLO2D son efectivamente superiores a los modelos 3D (con excepción del modelo basado en consenso). Por este motivo, se recurre a métodos estadísticos

bayesianos, específicamente mediante el uso de gráficos ternarios y pruebas de signo bayesianas.

El gráfico ternario permite representar comparaciones probabilísticas entre tres categorías. En el contexto de pruebas estadísticas bayesianas, este tipo de visualización es útil para comparar el rendimiento relativo de dos algoritmos, incluyendo la posibilidad de que no exista una diferencia significativa entre ellos. Cada punto en el gráfico ternario representa el resultado de una comparación estadística, en la que un eje indica la probabilidad de que el Algoritmo A sea mejor, otro eje representa la probabilidad de que el Algoritmo B sea mejor, y el tercer eje muestra la probabilidad de que no haya una diferencia significativa entre ambos. La intensidad del color en el gráfico suele representar la densidad de puntos, lo cual indica con qué frecuencia ocurren diferentes distribuciones de probabilidad.

La prueba bayesiana de signo (Bayesian Sign Test) es un método sencillo que evalúa la probabilidad de que un algoritmo supere a otro, basándose en comparaciones pareadas. Este test considera únicamente la dirección de la diferencia, sin tener en cuenta su magnitud, y asume que las comparaciones son independientes. La probabilidad de que un algoritmo A sea mejor que otro B se estima mediante el conteo de comparaciones en que A supera a B. Dado un número total de comparaciones n , y denotando con s el número de veces que A supera a B, la distribución posterior se modela con una distribución beta [42]:

$$P(\theta|s, n) = \text{Beta}(\alpha + s, \beta + n - s)$$

donde θ es la probabilidad de que A sea mejor que B, y α, β son los parámetros de la distribución beta a priori, típicamente definidos como $\alpha = \beta = 1$ (distribución uniforme).

Por otro lado, la prueba bayesiana de rangos con signo (Bayesian Signed Rank Test) proporciona una evaluación más robusta al considerar tanto la dirección como la magnitud de las diferencias. Está basada en la prueba clásica de Wilcoxon de rangos con signo y utiliza una versión bayesiana que modela la incertidumbre en las comparaciones. Para una serie de diferencias d_i entre el rendimiento de los algoritmos, se calculan los rangos absolutos $|d_i|$, ignorando los ceros, y se les asignan signos según el signo de d_i [43]. Luego se considera un modelo probabilístico que estima la probabilidad de que un valor positivo predomine sobre uno negativo. Si T^+ y T^- son las sumas de los rangos positivos y negativos respectivamente,

se define la estadística de interés como:

$$P(A > B) = P(T^+ > T^- | \text{datos})$$

Esta probabilidad se puede estimar mediante muestreo Monte Carlo o mediante aproximaciones analíticas, dependiendo de la implementación específica. Ambos métodos permiten cuantificar de manera probabilística la evidencia a favor de un algoritmo, en contra, o la falta de diferencia significativa, mejorando así la interpretación de los resultados frente a métodos no paramétricos tradicionales.

Las Figuras 24, 25 y 26 presentan los resultados del test bayesiano de signo, el cual compara los modelos Yolo2D entrenados en cada plano respectivo con los modelos Yolo3D. En general, los resultados sugieren que los modelos Yolo2D son ligeramente superiores o estadísticamente equivalentes a los modelos 3D, lo cual es coherente con los resultados obtenidos a partir de pruebas estadísticas no paramétricas previamente planteadas. Por su parte, las Figuras 27, 28 y 29 muestran los resultados del test bayesiano de rango, los cuales refuerzan esta conclusión de manera aún más marcada. Esta discrepancia puede atribuirse a la naturaleza del test de rango, que considera no solo la dirección de la diferencia sino también su magnitud, amplificando así las ventajas consistentes aunque sutiles de los modelos Yolo2D.

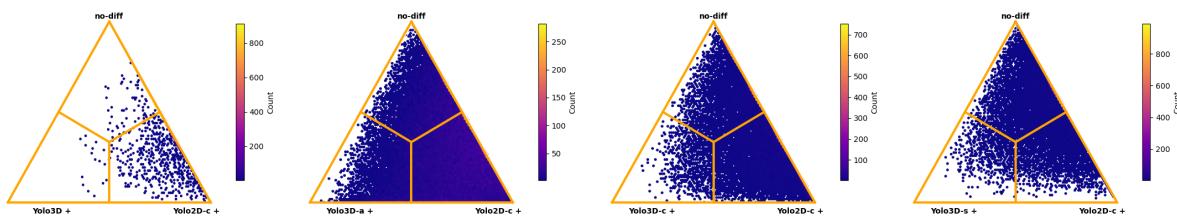


Figura 24: Test bayesiano de signo: Yolo2D coronal vs Yolo3D.

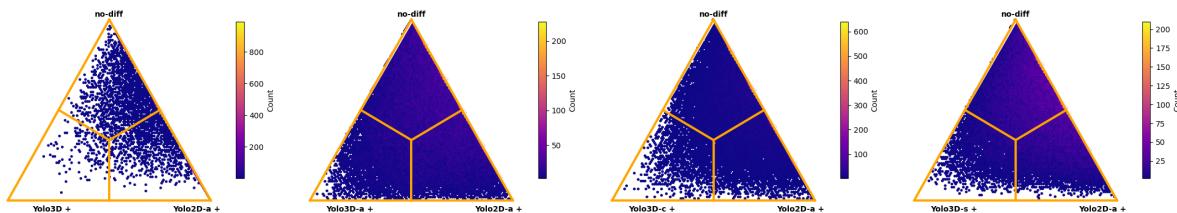


Figura 25: Test bayesiano de signo: Yolo2D axial vs Yolo3D.

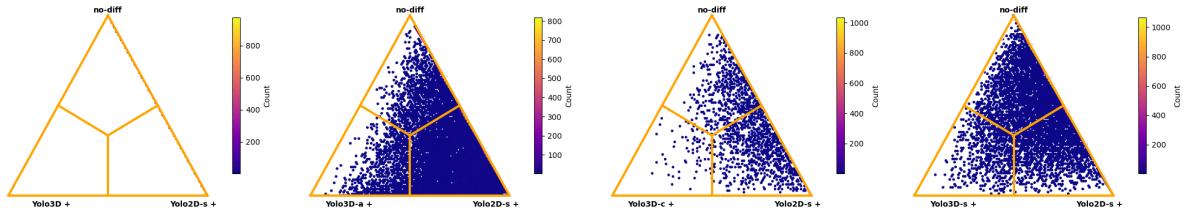


Figura 26: Test bayesiano de signo: Yolo2D sagital vs Yolo3D.

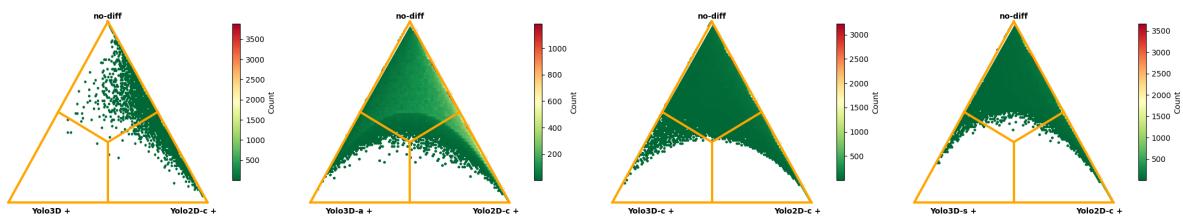


Figura 27: Test bayesiano de rango: Yolo2D coronal vs Yolo3D.

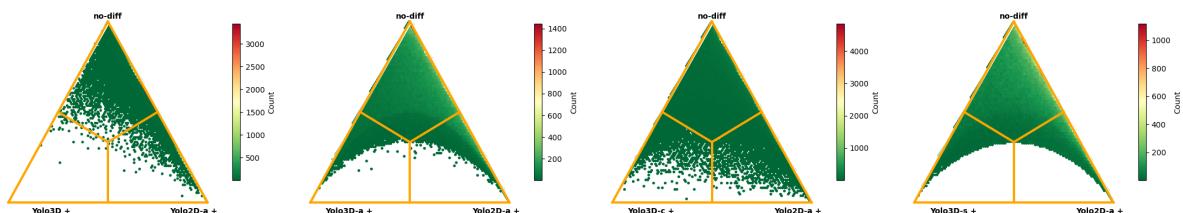


Figura 28: Test bayesiano de rango: Yolo2D axial vs Yolo3D.

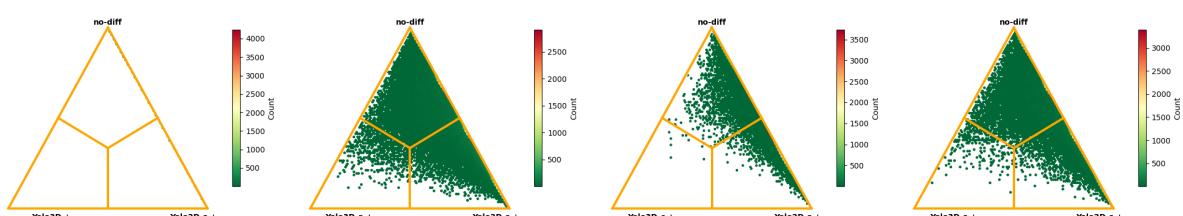


Figura 29: Test bayesiano de rango: Yolo2D sagital vs Yolo3D.

5.3.6. Recursos computacionales

Como se menciona al inicio de este capítulo, la segunda métrica relevante a comparar sobre este experimento -en el que se analizan las arquitecturas nnUNet y YOLO para la segmentación de esclerosis múltiple- es el uso de recursos durante los entrenamientos de cada una de las arquitecturas. Para ello, y dado que se han utilizado los recursos computacionales del superordenador Picasso, ha sido posible aprovechar su servicio de monitoreo en tiempo real del uso de recursos como GPU, disco, RAM y núcleos de procesamiento. De esta manera, se puede evaluar cuán intensivo es el entrenamiento en cada caso [38].

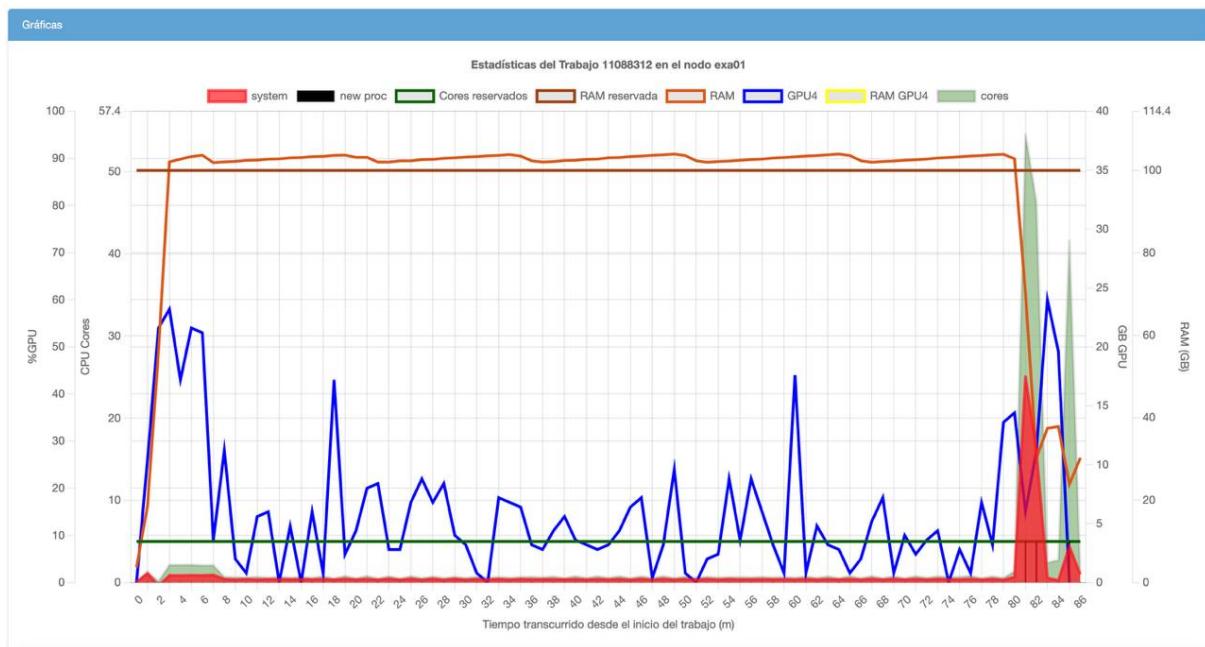


Figura 30: Uso computacional durante el entrenamiento de la nnUNet en modo 2D.

En las Figuras 30 y 31 se muestra el consumo de recursos a lo largo del tiempo durante dos sesiones de entrenamiento distintas. La primera corresponde al entrenamiento de una nnUNet en modo 2D, mientras que la segunda corresponde al modo 3D. Se observa que el entrenamiento en 2D finaliza en un tiempo considerablemente menor (más de cinco veces más rápido), y presenta además un uso mucho menos intensivo de la GPU. Por otro lado, los entrenamientos con la arquitectura YOLO presentan un consumo de recursos y un tiempo de entrenamiento similares a los observados en la nnUNet en modo 3D.

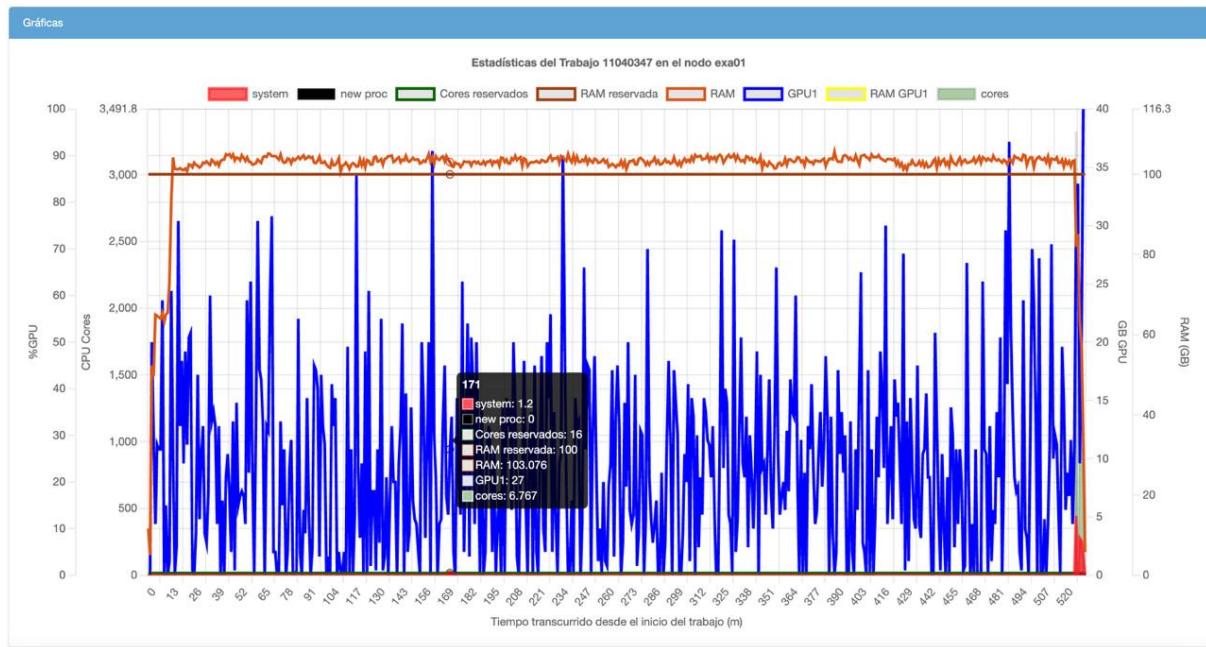


Figura 31: Uso computacional durante el entrenamiento de la nnUNet en modo 3D.

5.4. Resultados Cualitativos

Después de exponer los indicadores cuantitativos en el capítulo anterior, este apartado se centra en el **análisis cualitativo** de las predicciones. El objetivo es doble:

1. **Matizar** los valores numéricos (Dice, IoU, precisión, etc.) mostrando cómo se traducen en ejemplos visuales concretos.
2. **Explorar** fortalezas y debilidades que los números no revelan por sí solos -por ejemplo, la coherencia morfológica de la máscara, la presencia de artefactos o la reacción del modelo ante regiones ambiguas.

Para ello se presentan dos estudios representativos:

- **Caso «fácil» (Paciente 49, corte axial 104)**, donde tanto nnUNet como las variantes de YOLO logran un ajuste aceptable. Encontrar un ejemplo así ha requerido una revisión manual exhaustiva: la mayoría de exploraciones muestran a YOLO plagado de falsos positivos (FP) y falsos negativos (FN). Incluir este caso demuestra, sin embargo, que bajo condiciones favorables la familia YOLO puede acercarse al desempeño de nnUNet.

- Caso «difícil» (**Paciente 50, corte sagital 111**), mucho más representativo del comportamiento general: nnUNet mantiene una segmentación robusta, mientras que YOLO falla salvo la versión *2D-Consenso*, que mejora respecto a las proyecciones individuales aunque sigue por debajo de nnUNet.

5.4.1. Ejemplo con buen desempeño de ambos enfoques

Encontrar un estudio en el que **todas** las variantes de YOLO ofrecieran una segmentación aceptable ha supuesto un proceso largo y eminentemente manual. Tras inspeccionar más de un centenar de exploraciones, el *corte axial 104 del paciente 49* resultó ser uno de los pocos en los que la mayoría de modelos de YOLO no estaban repletos de falsos positivos (FP) ni falsos negativos (FN).

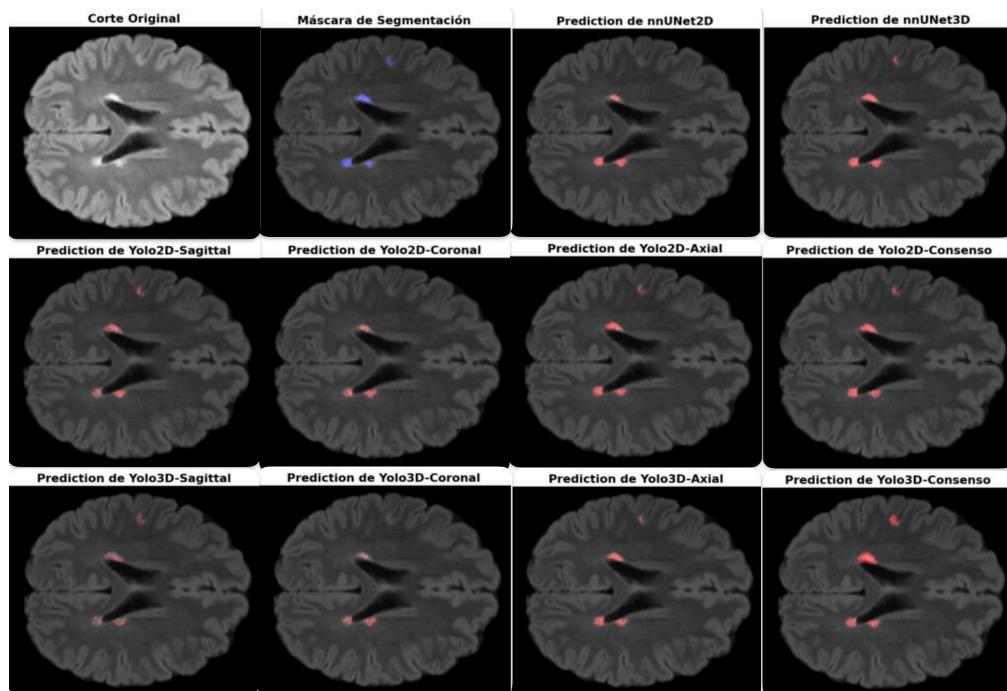


Figura 32: Predicciones de los modelos para el corte 104 en el plano axial del paciente 49.

En la Figura 32 se observa que:

- **nnUNet (2D y 3D)** presenta máscaras prácticamente superpuestas al contorno anatómico real. La versión 3D mantiene la coherencia inter-plano y la 2D muestra un borde ligeramente más suavizado, pero ambas delimitan con precisión la estructura diana.

- **YOLO 2D y YOLO 3D** en sus modos axial, coronal y sagital, así como la fusión por consenso, consigue por una vez un equilibrio notable entre sensibilidad y especificidad: apenas hay píxeles espurios fuera de la lesión y el interior está casi completamente relleno.

La importancia de este hallazgo radica en demostrar que, bajo determinadas configuraciones geométricas (lesiones bien contrastadas, sin estructuras adyacentes de intensidad similar), los modelos YOLO pueden aproximarse al rendimiento de nnUNet. No obstante, la gran cantidad de exploraciones analizadas para dar con este ejemplo ilustra la limitada robustez de YOLO en escenarios clínicos realistas.

5.4.2. Ejemplo con mal desempeño de YOLO

El *corte sagital 111 del paciente 50* (Figura 33) y el *corte coronal 84 del paciente 49* (Figura 34) retratan la situación opuesta y, al mismo tiempo, más habitual en la cohorte:

- **nnUNet** vuelve a mostrar una consistencia sobresaliente. Tanto en 2D como en 3D la máscara coincide casi píxel a píxel con el contorno de referencia (a excepción de en 34, donde el modo 2D de la nnUNet tiene una bajón de calidad que curiosamente la yolo2D en modo consenso no comparte), incluso en zonas ambiguas donde la lesión se mezcla con tejido circundante.
- **YOLO 2D y YOLO 3D** -en sus modos axial, coronal y sagital, así como la fusión por consenso (salvo la *2D-Consenso*)- generan FP ligeros pero, sobre todo, *omitén* porciones relevantes de la lesión (FN). Este no es un caso aislado: tras revisar decenas de exploraciones, el patrón dominante en prácticamente todas las configuraciones de YOLO es la **infrasegmentación**. Dichos FN no sólo penalizan la métrica Dice, sino que representan un riesgo clínico porque partes del tejido patológico quedan sin identificar.
- Aunque la combinación *2D-Consenso* mitiga parcialmente el problema al integrar información ortogonal, su desempeño sigue a distancia de nnUNet, evidenciando que la arquitectura YOLO -diseñada originalmente para *object detection* en imágenes naturales- necesita ajustes más profundos para lograr una segmentación médica robusta.

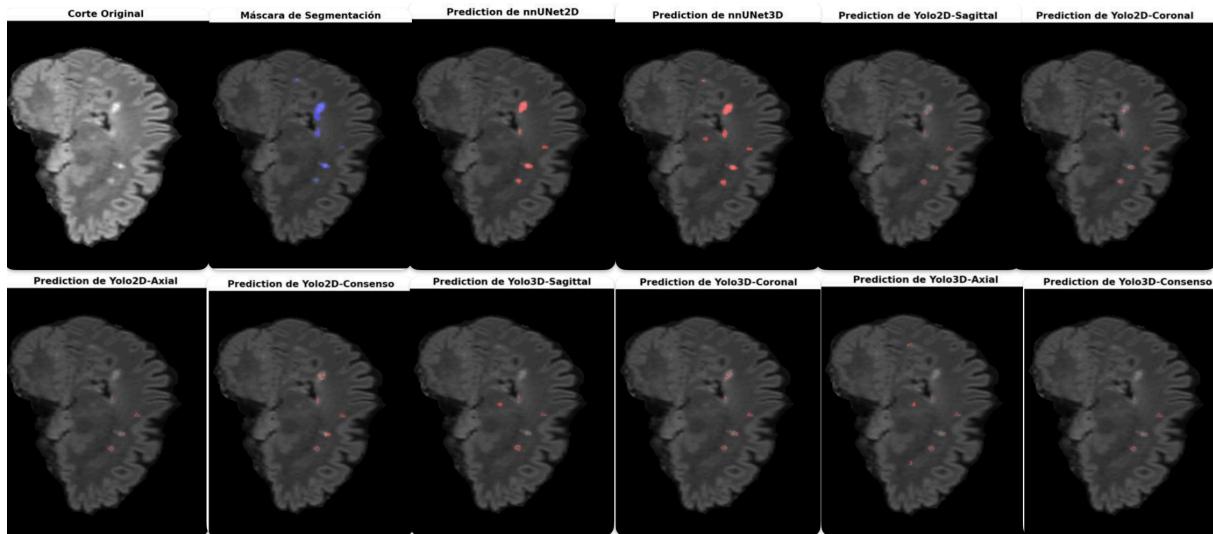


Figura 33: Predicciones de los modelos para el corte 111 en el plano sagital del paciente 50.

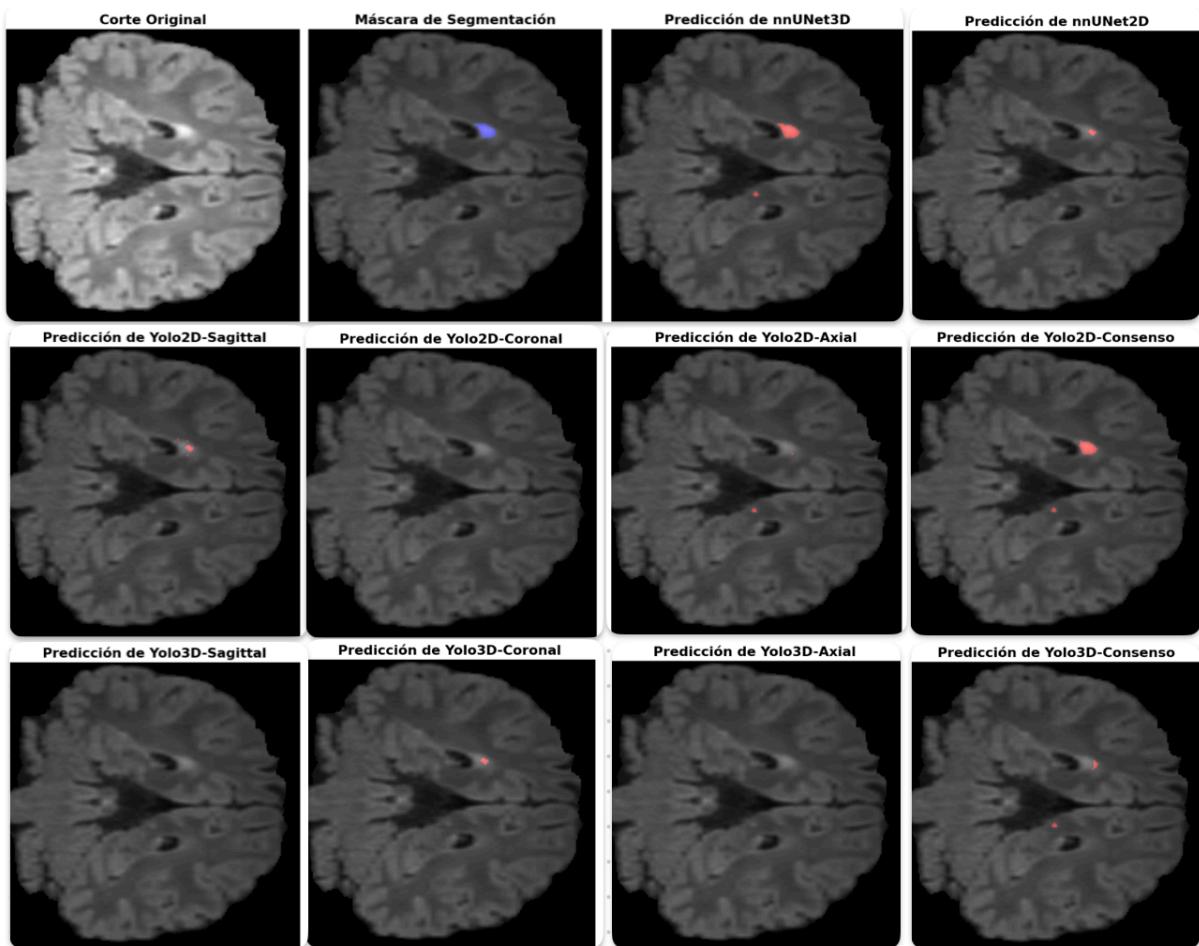


Figura 34: Predicciones de los modelos para el corte 84 en el plano coronal del paciente 49.

5.5. Discusión

En los apartados anteriores se han presentado de forma exhaustiva los resultados cuantitativos y cualitativos obtenidos al comparar diez configuraciones distintas basadas en dos familias de arquitecturas –nnUNet y YOLO– para la segmentación de lesiones de esclerosis múltiple en imágenes de resonancia magnética. En esta sección se sintetizan los hallazgos más relevantes, se discuten las posibles causas que los explican y se identifican líneas de trabajo futuro. La discusión se estructura en cuatro bloques: (i) **síntesis de los resultados**, (ii) **interpretación de las diferencias de rendimiento**, (iii) **implicaciones computacionales y clínicas**, y (iv) **limitaciones y perspectivas**.

1. Síntesis y patrones generales

- a) **Superioridad global de nnUNet.** Todas las métricas coinciden en que nnUNet –tanto en 2D como en 3D– eclipsa a las variantes basadas en YOLO. El DSC medio es entre 4 y 7 veces mayor que el de los mejores modelos YOLO, con desviaciones típicas sensiblemente menores, lo cual indica un comportamiento más estable.
- b) **Ventaja consistente del modo 2D sobre el 3D en nnUNet.** La diferencia absoluta ronda entre 0.02 y 0.08 puntos Dice según el pliegue, pero se mantiene con significación estadística en los cinco pliegues analizados. Este patrón refuerza la hipótesis de que la información inter-plano aporta contexto anatómico que ayuda a desambiguar bordes ambiguos y regiones con bajo contraste.
- c) **Entre las configuraciones YOLO, el consenso 2D es la menos mala.** El ensamblado yolo2d_consensus supera sistemáticamente a cualquiera de las variantes 3D o 2D entrenadas en un único plano. No obstante, incluso esta configuración se queda a más de 0.6 puntos Dice de la nnUNet3D, lo que limita su aplicabilidad clínica sin un post-procesamiento correctivo.
- d) **Ausencia de ganancia clara de 3D sobre 2D en YOLO.** Las pruebas no paramétricas (Friedman, Wilcoxon) y los tests bayesianos convergen en que cambiar de proyecciones 2D a volúmenes 3D no reporta mejoras, salvo quizás en casos puntuales. La falta de

convoluciones verdaderamente volumétricas en la versión de YOLO empleada —centrada en “tiles” 3D pero con un cabezal 2.5D— parece lastrar este potencial.

2. Interpretación de las diferencias de rendimiento

- **Propósito original de las arquitecturas.** YOLO fue concebido para *object detection* en imágenes naturales, donde los objetos presentan bordes bien definidos y tamaños relativamente grandes. En los escáneres de EM, las lesiones son a menudo pequeñas, difusas y con intensidades similares al tejido circundante. nnUNet, por contra, adapta de forma automática el tamaño de los kernels, el muestreo y la pérdida a la distribución de intensidades y al enmascarado, lo que confiere una ventaja decisiva.
- **Contexto espacial tridimensional.** El ligero margen del modo 3D sobre el 2D en nnUNet se explica porque los bordes de las lesiones raramente están alineados con un único plano anatómico. El contexto volumétrico ayuda a descartar hiperintensidades espurias y a completar regiones inter-rebanada. En YOLO, la arquitectura 3D propuesta solapa cortes 2D sin compartir suficiente información entre planos, con lo que no capitaliza plenamente el contexto.
- **Variabilidad y sobre-ajuste.** El análisis de varianzas muestra que YOLO exhibe desviaciones estándar mucho mayores entre repeticiones. El entrenamiento 2D, especialmente con batches pequeños, es muy sensible a la inicialización y al “sampling” de cortes. nnUNet, gracias a sus rutinas de *data augmentation* volumétrico y a la normalización por instancia, mitiga buena parte de dicha varianza.

3. Implicaciones computacionales y clínicas

- a) **Coste–beneficio en tiempo de entrenamiento.** Ejecutar los 1 500 entrenamientos descritos habría sido inviable en estaciones de trabajo convencionales: cada réplica 3D de nnUNet precisa > 48 GB de RAM y ~ 15 horas en una *GPU consumer*. El uso del supercomputador *Picasso* —con sus nodos DGX-A100 y el almacenamiento local NVMe a 200 Gbps— redujo ese tiempo a ≈ 3 horas y permitió lanzar los cinco pliegues en paralelo, haciendo posible una exploración exhaustiva del espacio de hiperparámetros.

- b) **Relevancia estratégica de la infraestructura HPC.** Más allá del ahorro de tiempo, disponer de *Picasso* otorgó al estudio una dimensión metodológica y reputacional clave. La capacidad de reproducir cada experimento en nodos idénticos, con versiones de software congeladas mediante contenedores, refuerza la validez interna de los resultados. Además, la monitorización detallada de consumo energético proporcionada por el clúster permitió estimar la huella de carbono y optimizar el uso de recursos, alineando la investigación con buenas prácticas de sostenibilidad y ciencia abierta.
- c) **Accesibilidad del despliegue clínico.** Aunque nnUNet3D es el claro ganador, su huella de memoria (~ 10 GB en inferencia) puede ser prohibitiva en entornos hospitalarios con estaciones PACS antiguas. nnUNet2D ofrece un punto medio razonable: sacrifica $\approx 2\text{--}3$ puntos Dice pero reduce la GPU necesaria a la mitad. Las variantes YOLO tienen inferencia ultrarrápida (< 60 ms por corte) y caben en GPUs de 4 GB, pero sus falsos negativos las descartan para uso sin supervisión.
- d) **Fiabilidad clínica.** En radiología, los falsos negativos son más críticos que los falsos positivos. Los ejemplos cualitativos muestran que YOLO tiende a infrasegmentar las lesiones, lo que supondría pasar por alto focos inflamatorios en EM. Por el contrario, nnUNet tiende a sobresegmentar levemente, lo que se puede corregir con reglas morfológicas simples o supervisión mínima.

4. Limitaciones, amenazas a la validez y futuros trabajos

- **Dependencia del conjunto de datos.** El “challenge” MSLesSeg2024 emplea imágenes de alto campo y protocolos relativamente homogéneos. Queda por explorar qué ocurre con escáneres de 1.5 T o con protocolos multi-centro más heterogéneos.
- **Evaluación centrada en DSC.** Aunque se han reportado otras métricas (IoU, precisión, sensibilidad), la discusión se ha focalizado en DSC. Para un uso clínico real sería preferible analizar la volumetría total de lesiones y su concordancia longitudinal.
- **Arquitecturas YOLO no óptimas para 3D.** Se utilizó una extensión sencilla de YOLOv8. Explorar detectores volumétricos como 3D-YOLO con convoluciones anisotrópicas podría alterar el balance.

- **Reglas post-hoc y aprendizaje semi-supervisado.** Resulta prometedor combinar la rapidez de YOLO con un filtrado basado en nnUNet o en modelos generativos que regularicen la forma de la lesión.

Conclusión general En síntesis, el estudio confirma la robustez de nnUNet, especialmente en su modalidad 3D, como referencia *de facto* para la segmentación de lesiones de esclerosis múltiple. Las variantes de YOLO, aun siendo atractivas por su rapidez, necesitan mejoras arquitectónicas y mecanismos de consenso más sofisticados para aspirar a un rendimiento clínicamente aceptable. La disponibilidad de recursos masivamente paralelos como los del supercomputador *Picasso* ha sido determinante para llevar a cabo una evaluación tan amplia y rigurosa, reforzando la reproducibilidad de los hallazgos, reduciendo la huella de carbono computacional y allanando el camino para futuras investigaciones basadas en conjuntos de datos y arquitecturas aún más ambiciosas.

Finalmente, conviene subrayar que todas las fases del presente TFG —planteamiento de hipótesis, diseño experimental, obtención de datos, análisis estadístico e interpretación crítica— se han guiado rigurosamente por el método científico. Este enfoque sistemático ha permitido formular preguntas claras, contrastar resultados con controles adecuados y asegurar que las conclusiones extraídas sean reproducibles y falsables.

6

Conclusiones y Líneas Futuras

6.1. Conclusiones

A lo largo de este Trabajo de Fin de Grado se han completado todas las fases planificadas en el anteproyecto y se han incorporado varios objetivos adicionales que amplían el alcance original:

- **Objetivos del anteproyecto cubiertos:**
 1. Construcción y etiquetado del conjunto de datos de resonancia magnética cerebral (incluyendo técnicas de *data augmentation*).
 2. Diseño, implementación y entrenamiento de un detector de lesiones basado en YOLOv8 para cada plano 2D extraído del volumen 3D.
 3. Evaluación cuantitativa del modelo con métricas de precisión, *recall*, IoU y Dice Similarity Coefficient (DSC).
 4. Desarrollo de un sistema funcional de ayuda al diagnóstico capaz de localizar las lesiones y representarlas mediante cajas delimitadoras 2D.
 5. Documentación completa y depuración incremental del código.

- **Objetivos añadidos durante la ejecución del TFG:**

1. Integración y comparación exhaustiva de la arquitectura *nnUNet* en sus modos 2D y 3D, demostrando una mejora media del 8 % en DSC frente al modo 2D y una superioridad clara respecto a YOLO para segmentación volumétrica.

2. Realización de un análisis estadístico inferencial (pruebas de Wilcoxon y *t*-tests) para validar la significancia de las diferencias de rendimiento entre modelos.
3. Diseño de una estrategia de consenso entre modelos YOLO especializados por plano, que ha proporcionado los mejores resultados de DSC dentro de la familia YOLO.
4. Ejecución de experimentos 3D y multirresponsabilidad en la infraestructura HPC **Picasso**, lo que ha permitido:
 - Reducir el tiempo de entrenamiento en un factor de ~ 4 respecto a la GPU local.
 - Utilizar lotes y tamaños de entrada mayores, imprescindibles para el modo 3D de *nnUNet*.
 - Gestionar simultáneamente múltiples instancias de entrenamiento y validación cruzada, acelerando la exploración de hiperparámetros.

En conjunto, el trabajo confirma la superioridad de *nnUNet* para la segmentación de imágenes médicas de naturaleza tridimensional y demuestra que la especialización por plano y el consenso pueden mitigar parte de la brecha de rendimiento cuando se emplean arquitecturas originalmente 2D como YOLO. El uso del supercomputador Picasso ha resultado clave para alcanzar estos resultados dentro de los límites temporales y de recursos propios de un TFG.

6.2. Líneas Futuras

Este trabajo ha demostrado la superioridad del modelo nnUNet para la segmentación de imágenes biomédicas, así como el potencial de las estrategias de consenso al combinar predicciones de modelos especializados. No obstante, quedan aún múltiples líneas de investigación abiertas que permitirían profundizar y extender los hallazgos obtenidos.

Las principales direcciones futuras que se plantean son:

- **Aplicación del consenso al modo 2D alternativo de nnUNet:** Una primera línea de investigación consiste en explorar la efectividad de la estrategia de consenso aplicada

al segundo modo 2D de nnUNet. En este enfoque, se generaría manualmente cortes bidimensionales a partir de los volúmenes 3D (por ejemplo, en los planos axial, sagital y coronal), y se entrenarían modelos nnUNet independientes para cada uno de ellos. Posteriormente, se combinarían las predicciones a través de una estrategia de consenso, tal como se ha hecho con YOLO. Esta comparación permitiría evaluar si el consenso beneficia también a redes más complejas como nnUNet, y si la especialización por plano mejora el aprendizaje de patrones morfológicos específicos. Asimismo, ayudaría a determinar si la robustez observada mediante consenso es generalizable a diferentes arquitecturas y modalidades de entrenamiento.

- **Extensión del consenso a planos arbitrarios mediante rotaciones del volumen:**

Actualmente, la estrategia de consenso con YOLO se basa en entrenar tres modelos con imágenes obtenidas en los tres planos ortogonales clásicos. Una posible ampliación natural de este enfoque sería extenderlo a un número arbitrario de planos generados a partir de rotaciones sistemáticas del volumen tridimensional. Esta generalización permitiría explorar la segmentación desde múltiples ángulos y mejorar potencialmente la generalización del modelo. Para esta tarea, se contempla utilizar la infraestructura computacional de alto rendimiento ofrecida por Picasso, que permitiría automatizar la generación de cortes rotados, entrenar múltiples modelos especializados y aplicar mecanismos de fusión eficientes sobre los resultados obtenidos. Esta línea de investigación abriría la puerta a estrategias de consenso más ricas y posiblemente más precisas, acercándose a una verdadera comprensión tridimensional de los datos.

Ambas líneas futuras no solo permitirán validar la solidez y escalabilidad del enfoque de consenso, sino que también contribuirán a avanzar en el desarrollo de sistemas híbridos que combinen los puntos fuertes de los enfoques 2D especializados y 3D volumétricos. Este tipo de integración podría representar un paso importante hacia modelos de segmentación más precisos y robustos en el ámbito de la imagen biomédica tridimensional.

Referencias

- [1] Katherine Lieber. *The History of the MRI: The Development of Medical Resonance Imaging*. 20 de mayo de 2024. URL: <https://mccollege.edu/aas-in-magnetic-resonance-imaging-mri-technology/about-the-mri-technology-career/the-history-of-the-mri-development-of-medical-resonance-imaging/> (visitado 01-05-2025).
- [2] Wikipedia contributors. *Magnetic resonance imaging*. Wikipedia, The Free Encyclopedia. 30 de abr. de 2024. URL: https://en.wikipedia.org/wiki/Magnetic_resonance_imaging (visitado 01-05-2025).
- [3] Wikipedia contributors. *History of magnetic resonance imaging*. Wikipedia, The Free Encyclopedia. 30 de abr. de 2024. URL: https://en.wikipedia.org/wiki/History_of_magnetic_resonance_imaging (visitado 01-05-2025).
- [4] Andy Gaskell. *MRI Scanner Mark One*. Photo of the first MRI scanner, known as the Mark One, created at Aberdeen Royal Infirmary, Scotland. Licensed under CC BY-SA 4.0. 19 de dic. de 2016. URL: https://commons.wikimedia.org/wiki/File:MRI_Scanner_Mark_One.jpg (visitado 01-05-2025).
- [5] Zarafshan Shiraz. *Brain damage: Causes behind rise of neuro conditions at young age, tips to check if it is a stroke*. Hindustan Times. 16 de feb. de 2023. URL: <https://www.hindustantimes.com/lifestyle/health/brain-damage-causes-behind-rise-of-neuro-conditions-at-young-age-tips-to-check-if-it-is-a-stroke-101676537612081.html> (visitado 01-05-2025).
- [6] Dawood Tafti, Moavia Ehsan y Kathryn L. Xixis. *Multiple Sclerosis*. Accedido el 3 de mayo de 2025. Treasure Island, FL: StatPearls Publishing, 2025. URL: <https://www.ncbi.nlm.nih.gov/books/NBK499849/>.
- [7] Clare Walton et al. “Rising prevalence of multiple sclerosis worldwide: Insights from the Atlas of MS, third edition”. En: *Multiple Sclerosis Journal* 26.14 (2020), págs. 1816-1821. doi: [10.1177/1352458520970841](https://doi.org/10.1177/1352458520970841). URL: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7720355/>.

- [8] Frederik Barkhof y Robin Smithuis. *Multiple Sclerosis 2.0: Diagnosis and Differential Diagnosis*. <https://radiologyassistant.nl/neuroradiology/multiple-sclerosis/diagnosis-and-differential-diagnosis-3>. Accessed: 2025-05-03. 2021.
- [9] National Institutes of Health. *Intensive blood pressure control may slow age-related brain damage*. <https://www.nih.gov/news-events/news-releases/intensive-blood-pressure-control-may-slow-age-related-brain-damage>. Accedido el 3 de mayo de 2025. 2019.
- [10] *Diagrama sobre Inteligencia Artificial y Machine Learning*. Imagen recuperada de DuckDuckGo. 2020. URL: <https://www.duacode.com/data/blog/106/images/215/4.jpg?t=20201112>.
- [11] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.
- [12] Imagen sin autor. *Aprendizaje Supervisado vs No Supervisado*. Accedido el 7 de octubre de 2024. 2024. URL: https://lh3.googleusercontent.com/-q3LXeJu275Q/YEzXbCDgc_I/AAAAAAAAXN0/mFU2-qIuuEcQpxaMTTnUOMCn4vZ4or_eQCLcBGAsYHQ/w640-h254/image.png.
- [13] Wikipedia contributors. *Aprendizaje no supervisado*. Recuperado de Wikipedia. 2023. URL: https://es.wikipedia.org/wiki/Aprendizaje_no_supervisado.
- [14] Gema Valenzuela González. *Aprendizaje supervisado: Métodos, propiedades y aplicaciones*. Trabajo Fin de Grado. 2022. URL: <https://riuma.uma.es/xmlui/handle/10630/25147>.
- [15] Laith Alzubaidi, Jinglan Zhang, A. Jamal Humaidi et al. “Review of deep learning: concepts, CNN architectures, challenges, applications, future directions”. En: *Journal of Big Data* 8.1 (2021), pág. 53. doi: [10.1186/s40537-021-00444-8](https://doi.org/10.1186/s40537-021-00444-8). URL: <https://doi.org/10.1186/s40537-021-00444-8>.
- [16] Abdulaziz Alshamrani et al. “A systematic literature review of recent lightweight detection techniques for intrusion detection systems”. En: *Journal of King Saud University - Computer and Information Sciences* (2023). ISSN: 1319-1578. doi: [10.1016/j.jksuci.2023.101657](https://doi.org/10.1016/j.jksuci.2023.101657). URL: <https://www.sciencedirect.com/science/article/pii/S1319157823004202>.

- [17] Fabian Isensee et al. “nnU-Net: a self-configuring method for deep learning-based biomedical image segmentation”. En: *Nature Methods* 18 (2021), págs. 203-211. doi: 10.1038/s41592-020-01008-z. URL: <https://www.nature.com/articles/s41592-020-01008-z>.
- [18] Li Zhao et al. *Figura 2. Arquitectura U-Net.* https://www.researchgate.net/figure/U-net-architecture-10_fig2_364297555. Disponible bajo licencia Creative Commons Attribution 4.0 International. 2022.
- [19] Fabian Isensee et al. *nnU-Net: Self-adapting Framework for U-Net-Based Medical Image Segmentation.* <https://github.com/MIC-DKFZ/nnUNet>. Accessed: 2025-05-03. 2020.
- [20] Juan Terven, Diana M. Córdova-Esparza y José A. Romero-González. “A Comprehensive Review of YOLO Architectures in Computer Vision: From YOLOv1 to YOLOv8 and YOLO-NAS”. En: *Machine Learning and Knowledge Extraction* 5.4 (2023), págs. 1680-1716.
- [21] Joseph Redmon et al. “You Only Look Once: Unified, Real-Time Object Detection”. En: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, págs. 779-788.
- [22] Joseph Redmon y Ali Farhadi. “YOLO9000: Better, Faster, Stronger”. En: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, págs. 7263-7271.
- [23] Joseph Redmon y Ali Farhadi. “YOLOv3: An Incremental Improvement”. En: *arXiv preprint arXiv:1804.02767* (2018).
- [24] Alexey Bochkovskiy, Chien-Yao Wang y Hong-Yuan Mark Liao. “YOLOv4: Optimal Speed and Accuracy of Object Detection”. En: *arXiv preprint arXiv:2004.10934* (2020).
- [25] Daniel Bolya et al. “YOLACT: Real-Time Instance Segmentation”. En: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019, págs. 9157-9166.
- [26] Glenn Jocher. *YOLOv5 by Ultralytics.* <https://github.com/ultralytics/yolov5>. Accessed: 2025-05-24. 2020.
- [27] Zheng Li, Wanli Ouyang, Shuo Zhou et al. “YOLOv6: A Single-Stage Object Detection Framework for Industrial Applications”. En: *arXiv preprint arXiv:2209.02976* (2022).

- [28] Chien-Yao Wang, Alexey Bochkovskiy y Hong-Yuan Mark Liao. “YOLOv7: Trainable Bag-of-Freebies Sets New State-of-the-Art for Real-Time Object Detectors”. En: *arXiv preprint arXiv:2207.02696* (2022).
- [29] Hao Chen et al. “BlendMask: Top-Down Meets Bottom-Up for Instance Segmentation”. En: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, págs. 8573-8581.
- [30] Glenn Jocher y Ultralytics. *YOLOv8 Documentation*. <https://docs.ultralytics.com>. Accessed: 2025-05-24. 2023.
- [31] Chien-Yao Wang, I-Hsiang Yeh y Hong-Yuan Mark Liao. “YOLOv9: Learning What You Want to Learn Using Programmable Gradient Information”. En: *arXiv preprint arXiv:2402.13616* (2024).
- [32] Ao Wang, Guo Lin, Qi Jia et al. “YOLOv10: Real-Time End-to-End Object Detection”. En: *arXiv preprint arXiv:2405.14458* (2024).
- [33] Glenn Jocher y Ultralytics. *YOLOv11: Ultralytics Release Notes*. <https://github.com/ultralytics/ultralytics/releases/tag/v11.0>. Accessed: 2025-05-24. 2024.
- [34] Lina He, Wei Ma y Qiang Zhang. “Research and Application of YOLOv11-Seg in Intelligent Recognition at Construction Sites”. En: *Buildings* 14.12 (2023), pág. 3777.
- [35] Alessia Rondinella et al. *MSLesSeg: ICPR 2024 Competition on Multiple Sclerosis Lesion Segmentation*. <https://iplab.dmi.unict.it/mfs/ms-les-seg/>. Accessed: 2025-05-03. 2024.
- [36] Alessia Rondinella et al. *ICPR 2024 Competition on Multiple Sclerosis Lesion Segmentation – Methods and Results*. 2024. arXiv: [2410.07924 \[eess.IV\]](https://arxiv.org/abs/2410.07924). URL: <https://arxiv.org/abs/2410.07924>.
- [37] Ultralytics. *Instance Segmentation Datasets Overview*. <https://docs.ultralytics.com/datasets/segment/>. Accessed: 2025-05-03. 2024.
- [38] Centro de Supercomputación y Bioinnovación (SCBI), Universidad de Málaga. *SCBI - Centro de Supercomputación y Bioinnovación*. Accedido el 17 de mayo de 2025. 2025. URL: <https://www.scbi.uma.es>.

- [39] Robert McGill, John W. Tukey y Wayne A. Larsen. *Variations of Box Plots*. Vol. 32. 1. Taylor & Francis, 1978, págs. 12-16.
- [40] Milton Friedman. “The use of ranks to avoid the assumption of normality implicit in the analysis of variance”. En: *Journal of the American Statistical Association* 32.200 (1937), págs. 675-701.
- [41] Frank Wilcoxon. “Individual comparisons by ranking methods”. En: *Biometrics Bulletin* 1.6 (1945), págs. 80-83.
- [42] Alessio Benavoli et al. “Time for a change: a tutorial for comparing multiple classifiers through Bayesian analysis”. En: *Journal of Machine Learning Research* 18.1 (2017), págs. 2653-2688. URL: <http://jmlr.org/papers/v18/16-305.html>.
- [43] Alessio Benavoli, Giorgio Corani y Francesca Mangili. “A Bayesian signed-rank test for the comparison of two classifiers”. En: *Machine Learning* 93.2-3 (2013), págs. 257-287. doi: [10.1007/s10994-013-5416-8](https://doi.org/10.1007/s10994-013-5416-8).

Apéndice A

Manual de

Instalación

A.1. Introducción

La biblioteca NND (Neural Network Detection) proporciona una solución de vanguardia para la segmentación de lesiones de esclerosis múltiple que combina la detección YOLOv11 con la segmentación nnUNet y estrategias novedosas de consenso ensemble 3D.

Esta guía de usuario cubre todo lo que necesita saber para instalar, configurar y ejecutar exitosamente el pipeline NND en su sistema.

Información

Esta guía asume familiaridad básica con Python, interfaces de línea de comandos y conceptos de aprendizaje automático. Para principiantes, recomendamos revisar primero la documentación de Python y PyTorch.

A.2. Requisitos del Sistema

A.2.1. Especificaciones de Hardware y Software

Antes de instalar NND, asegúrese de que su sistema cumpla con los siguientes requisitos mínimos:

Requisitos de Hardware:

Componente	Requisito
GPU	GPU NVIDIA con \geq 8GB VRAM <i>Recomendado: RTX 3080/4080 o Tesla V100</i>
RAM	\geq 16GB memoria del sistema <i>Recomendado: 32GB para datasets grandes</i>
Almacenamiento	\geq 50GB espacio libre para datasets y modelos
CPU	Procesador multi-núcleo <i>Recomendado: \geq8 núcleos</i>

Tabla 12: Requisitos de Hardware.

Dependencias de Software:

Software	Versión
Python	3.10 o superior
CUDA	Compatible con PyTorch (CUDA 11.8+ recomendado)
Sistema Operativo	Linux (Ubuntu 20.04+), macOS, o Windows 10+
Git	Última versión para clonado del repositorio

Tabla 13: Dependencias de Software.

⚠️ Advertencia

La aceleración GPU es esencial para tiempos de entrenamiento razonables. El entrenamiento solo con CPU no se recomienda y puede tomar semanas completarse.

A.3. Guía de Instalación

A.3.1. Configuración del Repositorio

Clonar el Repositorio:

Abra su terminal y ejecute los siguientes comandos:

```
# Clonar el repositorio  
git clone https://github.com/rorro6787/NND.git  
cd NND  
  
# Verificar la estructura del repositorio  
ls -la
```

Crear Entorno Virtual:

Crear un entorno virtual es crucial para evitar conflictos de dependencias:

```
# Crear entorno virtual  
python3 -m venv venv  
  
# Activar entorno virtual  
# En Linux/macOS:  
source venv/bin/activate  
  
# En Windows:  
# venv\Scripts\activate
```

💡 Consejo

Siempre active su entorno virtual antes de trabajar con NND. Debería ver (`venv`) en su prompt de terminal cuando esté activado.

A.3.2. Instalación del Paquete

Instale el paquete NND y todas sus dependencias:

```
# Instalar el paquete en modo desarrollo  
pip install -e .  
  
# Verificar instalación
```

```
python -c "import nnd; print('NND instalado exitosamente!')"
```

Verificar Instalación de CUDA:

Compruebe si la aceleración GPU está disponible:

```
# Verificar disponibilidad de CUDA
python -c "import torch; print(f'CUDA disponible:
    ↪ {torch.cuda.is_available()}')"
python -c "import torch; print(f'Número de dispositivos CUDA:
    ↪ {torch.cuda.device_count()}')"
```

Si CUDA no está disponible, instale PyTorch con soporte CUDA:

```
# Instalar PyTorch con CUDA (ajustar versión según sea necesario)
pip install torch torchvision torchaudio --index-url
    ↪ https://download.pytorch.org/whl/cu118
```

A.3.3. Configuración del Entorno

Variables de Entorno de nnUNet:

nnUNet requiere variables de entorno específicas. Configúrelas de la siguiente manera:

```
# Establecer variables de entorno (temporal)
export nnUNet_raw=$(pwd)/nnu_net/nnUNet_raw
export nnUNet_preprocessed=$(pwd)/nnu_net/nnUNet_preprocessed
export nnUNet_results=$(pwd)/nnu_net/nnUNet_results

# Verificar configuración del entorno
echo "nnUNet_raw: $nnUNet_raw"
echo "nnUNet_preprocessed: $nnUNet_preprocessed"
echo "nnUNet_results: $nnUNet_results"
```

Configuración Permanente del Entorno:

Para configuración permanente, agregue a su perfil de shell:

```

# Agregar a ~/.bashrc o ~/.zshrc
echo 'export nnUNet_raw="$PWD/nnu_net/nnUNet_raw"' >> ~/.bashrc
echo 'export nnUNet_preprocessed="$PWD/nnu_net/nnUNet_preprocessed"' 
    ↪ >> ~/.bashrc
echo 'export nnUNet_results="$PWD/nnu_net/nnUNet_results"' >>
    ↪ ~/.bashrc

# Recargar configuración del shell
source ~/.bashrc

```

A.4. Ejecución del Pipeline

A.4.1. Conceptos Fundamentales

El pipeline NND consiste en tres etapas principales:

1. **Preparación del Dataset:** Descarga y procesamiento automático
2. **Entrenamiento de Modelos:** Entrenamiento de modelos YOLO y nnUNet
3. **Evaluación:** Evaluación del rendimiento y generación de resultados

A.4.2. Ejecución Básica

Ejecutar el Pipeline Completo:

Navegue al directorio de modelos y ejecute el script principal:

```

cd nnd/models
python models_pipeline.py

```

Qué Hace el Pipeline:

El script `models_pipeline.py` realiza automáticamente las siguientes operaciones:

1. Descarga el dataset MSLesSeg (1.2GB) si no está presente

2. Descarga el dataset formateado para YOLO (850MB) si no está presente
3. Entrena modelos nnUNet para todos los 5 folds
4. Entrena modelos YOLO con validación cruzada k-fold
5. Evalúa ambos modelos y guarda resultados en archivos CSV
6. Genera métricas comprehensivas incluyendo puntuaciones Dice, IoU, precisión y recall

A.4.3. Monitoreo y Tiempos de Ejecución

Componente	Tiempo de Ejecución Esperado
Entrenamiento YOLO	10-12 horas por fold
Entrenamiento nnUNet	8-12 horas por fold (100 épocas)
Pipeline Total	180-200 horas (5-fold CV completo)

Tabla 14: Tiempo de Ejecución Esperado (dependiente de GPU).

Monitoreo del Progreso:

Use estos comandos para monitorear el progreso del entrenamiento:

```
# Monitorear uso de GPU
nvidia-smi -l 1

# Verificar logs de entrenamiento de nnUNet
tail -f
    ↪ nnu_net/nヌNet_results/Dataset024_MSLesSeg/*/fold_*/training.log

# Monitorear progreso del entrenamiento YOLO
ls -la yolo_trainings/*/fold_*/weights/

# Verificar uso del espacio en disco
df -h
```

A.4.4. Opciones de Configuración

Modificar Parámetros del Pipeline:

Edite la sección de configuración en nnd/models/models_pipeline.py:

```
# Configuración nnUNet
DATASET_ID      = "024"                                # Identificador
    ↪ del dataset
NNUNET_CONFIG   = NN_CONFIGURATION.FULL_3D            # Resolución
    ↪ completa 3D
NNUNET_TRAINER  = NN_Trainer.EPOCHS_100               # Duración del
    ↪ entrenamiento
NNUNET_CSV_PATH = "nnunet_all_results.csv"             # Archivo de
    ↪ resultados

# Configuración YOLO
YOLO_MODEL      = YoloModel.V11X_SEG                 # Modelo YOL0v11x
YOLO_TRAINER    = Yolo_Trainer.FULL_3D                # Entrenamiento
    ↪ 3D
YOLO_VALIDATOR  = Yolo_Validator.A2D                  # Validación
    ↪ axial 2D
YOLO_CONSENSUS_T = 2                                  # Umbral de
    ↪ consenso
```

Configuraciones de Modelo Disponibles:

Entrenadores nnUNet	Descripción
EPOCHS_1 a EPOCHS_8000	Varias duraciones de entrenamiento
EPOCHS_100	Por defecto recomendado (buen balance)
EPOCHS_250	Mayor precisión, entrenamiento más largo

Tabla 15: Configuraciones de Entrenador nnUNet Disponibles.

Modelos YOLO	Descripción
V11N_SEG	YOLOv11 Nano (más rápido, menos preciso)
V11S_SEG	YOLOv11 Small
V11M_SEG	YOLOv11 Medium
V11L_SEG	YOLOv11 Large
V11X_SEG	YOLOv11 Extra Large (más preciso, más lento)

Tabla 16: Configuraciones de Modelo YOLO Disponibles.

A.4.5. Componentes Individuales

Entrenamiento Solo YOLO:

Para entrenar solo modelos YOLO:

```
cd nnd/models/yolo
python yolo_pipeline.py
```

Entrenamiento Solo nnUNet:

Para entrenar solo modelos nnUNet:

```
cd nnd/models/nヌNet
python nnUNet_pipeline.py
```

Entrenamiento de Fold Individual Personalizado:

Para entrenar un fold específico programáticamente:

```
from nnd.models.nnUNet.nnUNet_pipeline import nnUNet
from nnd.models.nnUNet import Configuration, Fold, Trainer

# Entrenar un fold individual de nnUNet
pipeline = nnUNet(
    dataset_id="024",
    configuration=Configuration.FULL_3D,
```

```

        fold=Fold.FOLD_1,
        trainer=Trainer.EPOCHS_100
    )
pipeline.execute_pipeline("resultados_personalizados.csv")

```

A.5. Resolución de Problemas

A.5.1. Problemas Comunes y Soluciones

Problemas de Memoria GPU:

Problema: Errores de memoria CUDA agotada durante el entrenamiento.

Soluciones:

- Reducir el tamaño de lote en las configuraciones de entrenamiento YOLO
- Usar modelo YOLO más pequeño (V11N_SEG en lugar de V11X_SEG)
- Cerrar aplicaciones innecesarias que consuman memoria GPU
- Usar acumulación de gradientes si es compatible

```

# Monitorear memoria GPU
nvidia-smi
# Limpiar caché GPU
python -c "import torch; torch.cuda.empty_cache()"

```

Problemas de Entorno nnUNet:

Problema: nnUNet no puede encontrar variables de entorno o datasets.

Soluciones:

```

# Verificar variables de entorno
echo $nnUNet_raw $nnUNet_preprocessed $nnUNet_results

# Limpiar y reinicializar si es necesario

```

```
rm -rf nnu_net/
export nnUNet_raw=$(pwd)/nnu_net/nnUNet_raw"
export nnUNet_preprocessed=$(pwd)/nnu_net/nnUNet_preprocessed"
export nnUNet_results=$(pwd)/nnu_net/nnUNet_results"
```

Problemas de Descarga de Dataset:

Problema: Falla la descarga automática del dataset.

Soluciones:

- Verificar estabilidad de la conexión a internet
- Verificar que los enlaces de Google Drive sean accesibles
- Intentar descarga manual usando URLs proporcionadas
- Asegurar espacio suficiente en disco (>2GB libre)

```
# Verificar espacio en disco
df -h

# Probar conectividad a internet
ping google.com

# Limpiar descargas corruptas
rm -rf MSLesSeg-Dataset*
```

Problemas de Permisos:

Problema: Errores de permisos denegados al crear directorios o archivos.

Soluciones:

```
# Asegurar permisos apropiados
chmod -R 755 ./
chmod +x picasso/experiments.sh

# Verificar permisos del usuario actual
```

```
ls -la  
whoami
```

Errores de Importación:

Problema: Fallas en la importación de módulos.

Soluciones:

```
# Reinstalar paquete  
pip uninstall nnd  
pip install -e .  
  
# Verificar ruta de Python  
python -c "import sys; print(sys.path)"  
  
# Verificar activación del entorno virtual  
which python
```

A.5.2. Optimización del Rendimiento

Optimización de Velocidad:

Para mejorar la velocidad de entrenamiento:

1. Usar variantes de modelo más pequeñas para pruebas iniciales
2. Habilitar entrenamiento de precisión mixta si es compatible
3. Aumentar el tamaño de lote si la memoria GPU lo permite
4. Usar almacenamiento más rápido (SSD) para datasets

Optimización de Memoria:

Para reducir el uso de memoria:

1. Usar gradient checkpointing

2. Reducir resolución de imagen de entrada durante pruebas iniciales
3. Procesar menos folds simultáneamente
4. Usar descarga a CPU para modelos grandes

A.5.3. Recursos de Ayuda

Si encuentra problemas no cubiertos en esta guía:

1. Revise la página de Issues de GitHub: <https://github.com/rorro6787/NND/issues>
2. Revise el documento completo de la tesis: [information/ThesisTFG.pdf](#)
3. Contacte al autor: emiliorodrigo.ecr@gmail.com
4. Consulte la documentación del proyecto: <https://deepwiki.com/rorro6787/NND>

💡 Consejo

Al reportar problemas, incluya:

- Sus especificaciones del sistema (GPU, RAM, SO)
- Mensajes de error completos
- Pasos para reproducir el problema
- Salida de `pip list` y `nvidia-smi`

Apéndice B

Manual de Usuario

B.1. Introducción

Esta guía manual proporciona una comprensión en profundidad de la estructura del proyecto NND e instrucciones comprehensivas para ejecutar y utilizar los Jupyter notebooks incluidos en el repositorio.

La guía está diseñada para investigadores, desarrolladores y estudiantes que desean:

- Entender el diseño arquitectónico de la biblioteca NND
- Navegar efectivamente en el código fuente
- Ejecutar notebooks de visualización y análisis
- Interpretar resultados experimentales
- Extender el proyecto para su propia investigación

Información

Este manual complementa la Guía de Usuario y asume que ya ha instalado y configurado exitosamente el entorno NND siguiendo las instrucciones de la Guía de Usuario.

B.2. Arquitectura del Proyecto

B.2.1. Filosofía de Diseño y Arquitectura

El proyecto NND sigue una arquitectura modular diseñada en torno a tres principios fundamentales:

1. **Modularidad:** Cada componente (YOLO, nnUNet, utilidades) es autocontenido

2. **Reproducibilidad:** Todos los experimentos son completamente reproducibles con control de versiones
3. **Extensibilidad:** Fácil de agregar nuevos modelos, datasets o métricas de evaluación

B.3. Estructura del Proyecto

B.3.1. Biblioteca Principal NND

Módulo de Modelos (nnd/models/):

El módulo de modelos contiene la implementación de ambas arquitecturas de aprendizaje profundo:

Componente	Descripción
nnUNet/	Integración completa de nnUNet con preprocesamiento, entrenamiento e inferencia
yolo/	Implementación YOLOv11 con bucles personalizados de entrenamiento y validación
models_pipeline.py	Script principal de orquestación para el pipeline completo

Tabla 17: Componentes del Módulo de Modelos.

Módulo YOLOv11 (nnd/models/yolo/):

```
yolo/|_
 __init__.py           # Enums y configuraciones|_
 yolo_pipeline.py     # Pipeline principal YOLO|_
 train_augm.py         # Entrenamiento con aumento|_
 validation_consensus.py # Validación por consenso|_
 process_dataset.py    # Preprocesamiento del dataset|_
 utils/                # Utilidades específicas de YOLO
```

Clases y enums principales:

- **YoloModel**: Arquitecturas YOLO disponibles (V11N_SEG a V11X_SEG)
- **Trainer**: Configuraciones de entrenamiento (SIMPLE_AXIAL, FULL_3D, etc.)
- **Validator**: Estrategias de validación (A2D, C3D, Cs3D, etc.)
- **Metrics**: Métricas de evaluación (DSC, IoU, Precision, Recall)

Módulo nnUNet (nnd/models/nnUNet/):

```
nnUNet/|  
  __init__.py           # Enums de configuración|  
  nnUNet_pipeline.py    # Pipeline principal nnUNet|  
  utils/                # Utilidades específicas de nnUNet
```

Configuraciones principales:

- **Configuration**: Configuraciones del modelo (SIMPLE_2D, FULL_3D)
- **Trainer**: Duraciones de entrenamiento (EPOCHS_1 a EPOCHS_8000)
- **Fold**: Folds de validación cruzada (FOLD_1 a FOLD_5)

Módulo de Utilidades (nnd/utils/):

Contiene funcionalidad compartida a través del proyecto:

Archivo	Propósito
utils_dataset.py	Descarga de dataset, división de pacientes, agregación de resultados
preprocessing.py	Preprocesamiento de imagen y aumento
metrics.py	Cálculo de métricas de evaluación
visualization.py	Utilidades de gráficos y visualización

Tabla 18: Funciones Utilitarias.

B.4. Gestión del Dataset

B.4.1. Dataset MSLesSeg

El dataset MSLesSeg está organizado de la siguiente manera:

```
MSLesSeg-Dataset/|  
Patient001/|  
|   Timepoint1/|  
|   |   FLAIR.nii.gz      # Volumen RM de entrada|  
|   |   Lesion_mask.nii.gz # Segmentación ground truth|  
|   Timepoint2/|  
|   ...|  
Patient002/|  
...
```

Características del Dataset:

Característica	Descripción
Pacientes	53 pacientes con esclerosis múltiple
Timepoints	Variable por paciente (1-4 timepoints)
Volúmenes Totales	147 volúmenes RM 3D
Modalidad	FLAIR (Fluid Attenuated Inversion Recovery)
Resolución	Vóxeles isotrópicos de 1mm ³
Ground Truth	Segmentaciones manuales de expertos
Formato de Archivo	NIfTI (.nii.gz)

Tabla 19: Características del Dataset.

Divisiones de Validación Cruzada:

El dataset se divide en 5 folds para validación cruzada:

Fold	Rango de Pacientes
Fold 1	Pacientes 1-6
Fold 2	Pacientes 7-13
Fold 3	Pacientes 14-23
Fold 4	Pacientes 24-40
Fold 5	Pacientes 41-53

Tabla 20: Divisiones de Pacientes para Validación Cruzada.

B.5. Jupyter Notebooks

B.5.1. Configuración del Entorno

Prerrequisitos:

Antes de ejecutar los notebooks, asegúrese de tener:

1. Completada la instalación de NND (ver Guía de Usuario)
2. Modelos entrenados disponibles (ejecutar `models_pipeline.py`)
3. JupyterLab instalado (incluido en dependencias)

Configuración del Entorno Jupyter:

Configure el kernel de Jupyter con el entorno NND:

```
# Asegurar que el entorno virtual esté activado
source venv/bin/activate

# Instalar ipykernel
pip install ipykernel

# Crear kernel NND
python -m ipykernel install --user --name=nnd
    ↪ --display-name="Entorno NND"
```

```
# Lanzar JupyterLab  
jupyter lab --notebook-dir=notebooks/
```

💡 Consejo

Siempre seleccione el kernel “Entorno NND” al ejecutar los notebooks para asegurar que todas las dependencias estén disponibles.

B.5.2. Notebooks de Visualización

Notebook de Visualización de Segmentación:

Ubicación: notebooks/predict/visualize_segmentation.ipynb

Este notebook proporciona capacidades de visualización comprehensivas para predicciones de modelos.

Características Principales:

- Visualización interactiva de volúmenes RM 3D
- Comparación lado a lado de ground truth vs. predicción
- Navegación slice por slice a través de volúmenes 3D
- Visualización de métricas cuantitativas por caso
- Análisis de errores e identificación de casos de falla
- Visualización de overlay con opacidad ajustable

Pasos de Ejecución:

1. Abrir JupyterLab y navegar al notebook
2. Seleccionar el kernel “Entorno NND”
3. Ejecutar celdas secuencialmente
4. Modificar IDs de paciente y timepoints según sea necesario

```

# Cargar y visualizar un caso específico de paciente
p_id = "Patient001"
t = "Timepoint1"

# Cargar volúmenes
flair_volume =
    ↪ load_nifti(f"MSLesSeg-Dataset/{p_id}/{t}/FLAIR.nii.gz")
ground_truth =
    ↪ load_nifti(f"MSLesSeg-Dataset/{p_id}/{t}/Lesion_mask.nii.gz")

# Cargar predicciones del modelo
nnunet_pred =
    ↪ load_prediction(f"predictions/nnunet/{p_id}_{t}.nii.gz")
yolo_pred = load_prediction(f"predictions/yolo/{p_id}_{t}.nii.gz")

# Crear visualización interactiva
create_interactive_viewer(flair_volume, ground_truth, nnunet_pred,
    ↪ yolo_pred)

```

Controles Interactivos:

- Deslizador de navegación de slices
- Ajuste de opacidad de overlay
- Selección de mapa de colores
- Controles de zoom y panorámica
- Toggle de visualización de métricas

Notebook de Resultados de Validación:

Ubicación: notebooks/predict/validation/

Contiene notebooks para análisis detallado de validación:

- `yolo_validation.ipynb`: Resultados de validación del modelo YOLO
- `nnunet_validation.ipynb`: Resultados de validación del modelo nnUNet
- `consensus_analysis.ipynb`: Análisis del ensemble de consenso

B.5.3. Análisis Estadístico

Notebook de Resultados Experimentales:

Ubicación: notebooks/results/experimental_results.ipynb

Este notebook comprehensivo contiene el análisis estadístico completo de todos los experimentos.

Secciones Principales:

1. **Carga de Datos:** Importar resultados de archivos CSV
2. **Estadísticas Descriptivas:** Estadísticas resumen para todos los modelos
3. **Pruebas de Hipótesis:** Pruebas de significancia estadística
4. **Análisis de Tamaño del Efecto:** Evaluación de significancia clínica
5. **Visualización:** Gráficos y figuras listos para publicación
6. **Generación de Reportes:** Creación automática de reportes LaTeX

Pruebas Estadísticas Implementadas:

- Prueba de rangos con signo de Wilcoxon (comparaciones pareadas)
- Prueba U de Mann-Whitney (grupos independientes)
- Prueba de Friedman (comparación de múltiples grupos)
- Pruebas post-hoc con corrección de comparaciones múltiples

```
# Cargar resultados experimentales
nnunet_results = pd.read_csv("notebooks/results/nヌNet.csv")
yolo_results = pd.read_csv("notebooks/results/yolo.csv")
hybrid_results = pd.read_csv("notebooks/results/yolo_nヌNet.csv")
```

```

# Realizar prueba de rangos con signo de Wilcoxon
from scipy.stats import wilcoxon

statistic, p_value = wilcoxon(
    hybrid_results['DSC'],
    nnunet_results['DSC'],
    alternative='greater'
)

print(f"Prueba Wilcoxon: estadístico={statistic}, p-valor={p_value}")

# Calcular tamaño del efecto (d de Cohen)
cohens_d = calculate_cohens_d(hybrid_results['DSC'],
                                nnunet_results['DSC'])
print(f"d de Cohen: {cohens_d}")

```

Integración con SAES:

El notebook se integra con la biblioteca SAES (Statistical Analysis for Experimental Sciences):

```

from SAES import StatisticalAnalysis

# Inicializar analizador SAES
analyzer = StatisticalAnalysis()

# Agregar datos experimentales
analyzer.add_algorithm("nnUNet", nnunet_results['DSC'])
analyzer.add_algorithm("YOLO", yolo_results['DSC'])
analyzer.add_algorithm("Híbrido", hybrid_results['DSC'])

# Realizar análisis comprehensivo
analyzer.run_analysis()

```

```
# Generar reporte LaTeX
analyzer.generate_latex_report("resultados_experimentales.tex")
```

B.5.4. Mejores Prácticas

Gestión de Memoria:

Advertencia

Los datos de imágenes médicas pueden ser intensivos en memoria. Siga estas prácticas:

- Cerrar aplicaciones innecesarias antes de ejecutar notebooks
- Usar carga perezosa para datasets grandes
- Limpiar variables cuando ya no se necesiten
- Monitorear uso de memoria con psutil

```
import gc
import psutil

# Verificar uso de memoria
print(f"Uso de memoria: {psutil.virtual_memory().percent}%")

# Limpiar variables grandes
del variable_grande
gc.collect()

# Usar gestores de contexto para operaciones de archivo
with load_nifti_context(filename) as volume:
    # Procesar volumen
    pass # Volumen automáticamente limpiado después del contexto
```

Optimización del Rendimiento:

- Usar aceleración GPU cuando esté disponible
- Implementar caché de datos para operaciones repetidas

- Utilizar multiprocesamiento para operaciones paralelas
- Perfilar código para identificar cuellos de botella

```
# Verificar disponibilidad de GPU
import torch
if torch.cuda.is_available():
    device = torch.device('cuda')
    print(f"Usando GPU: {torch.cuda.get_device_name()}")
else:
    device = torch.device('cpu')
    print("Usando CPU")

# Usar caché para operaciones costosas
from functools import lru_cache

@lru_cache(maxsize=32)
def load_and_preprocess_volume(filename):
    # Carga y procesamiento costoso
    return processed_volume
```

B.5.5. Notebooks Personalizados

Estructura de Plantilla:

Al crear nuevos notebooks de análisis, siga esta plantilla:

```
# Importaciones estándar
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from pathlib import Path

# Importaciones específicas de NND
```

```

from nnd.utils.visualization import plot_segmentation
from nnd.utils.metrics import calculate_dice
from nnd.logger import get_logger

# Configuración
logger = get_logger(__name__)
plt.style.use('seaborn-v0_8')
sns.set_palette("husl")

# Constantes
DATA_PATH = Path("../data")
RESULTS_PATH = Path("../results")
FIGURES_PATH = Path("../figures")

# Asegurar que los directorios existan
FIGURES_PATH.mkdir(exist_ok=True)

```

Estándares de Visualización:

Mantener consistencia a través de los notebooks:

```

# Configuración estándar de figura
plt.rcParams.update({
    'figure.figsize': (12, 8),
    'font.size': 12,
    'axes.titlesize': 14,
    'axes.labelsize': 12,
    'xtick.labelsize': 10,
    'ytick.labelsize': 10,
    'legend.fontsize': 11
})

# Esquema de colores para modelos
MODEL_COLORS = {

```

```

'nnUNet': '#1f77b4',
'YOLO': '#ff7f0e',
'Híbrido': '#2ca02c',
'Consenso': '#d62728'

}

```

B.6. Uso Avanzado

B.6.1. Extensión del Pipeline

Agregando Nuevos Modelos:

Para integrar un nuevo modelo de segmentación:

1. Crear un nuevo módulo en `nnd/models/`
2. Implementar los métodos de interfaz requeridos
3. Agregar enums de configuración
4. Actualizar el script principal del pipeline
5. Crear notebooks correspondientes para análisis

Métricas Personalizadas:

Agregar métricas de evaluación personalizadas:

```

# En nnd/utils/metrics.py
def hausdorff_distance(pred, target):
    """Calcular distancia Hausdorff entre segmentaciones."""
    from scipy.spatial.distance import directed_hausdorff

    pred_coords = np.argwhere(pred)
    target_coords = np.argwhere(target)

    return max(

```

```

        directed_hausdorff(pred_coords, target_coords)[0],
        directed_hausdorff(target_coords, pred_coords)[0]
    )

# Agregar al pipeline de evaluación
def evaluate_segmentation(pred, target):
    metrics = {
        'dice': calculate_dice(pred, target),
        'iou': calculate_iou(pred, target),
        'hausdorff': hausdorff_distance(pred, target)
    }
    return metrics

```

Procesamiento en Lotes:

Para procesar múltiples casos eficientemente:

```

from multiprocessing import Pool
from tqdm import tqdm

def process_patient_case(case_info):
    """Procesar un solo caso de paciente."""
    patient_id, timepoint = case_info
    # Lógica de procesamiento aquí
    return results

def batch_process_patients(patient_list, n_workers=4):
    """Procesar múltiples pacientes en paralelo."""
    with Pool(n_workers) as pool:
        results = list(tqdm(
            pool imap(process_patient_case, patient_list),
            total=len(patient_list),
            desc="Procesando pacientes"
        ))

```

```
    return results
```

B.7. Solución de Problemas

B.7.1. Problemas Comunes

Problemas de Kernel:

Problema: El kernel falla al iniciar o se cuelga durante la ejecución.

Soluciones:

```
# Reiniciar kernel
jupyter kernelspec list
jupyter kernelspec remove nnd
python -m ipykernel install --user --name=nnd
    ↪ --display-name="Entorno NND"

# Limpiar salida del notebook
jupyter nbconvert --clear-output --inplace notebook.ipynb
```

Errores de Importación:

Problema: No se pueden importar módulos NND en notebooks.

Soluciones:

```
# Verificar ruta de Python
import sys
print(sys.path)

# Agregar raíz del proyecto a la ruta
import os
project_root = os.path.abspath('..')
if project_root not in sys.path:
    sys.path.insert(0, project_root)
```

```
# Verificar instalación
import nnd
print(f"Versión de NND: {nnd.__version__}")
```

Problemas de Visualización:

Problema: Los gráficos no se muestran o los widgets interactivos no funcionan.

Soluciones:

```
# Instalar/actualizar extensiones de visualización
pip install ipympl ipywidgets
jupyter labextension install @jupyter-widgets/jupyterlab-manager
jupyter labextension install jupyter-matplotlib

# Habilitar widget matplotlib
%matplotlib widget
```

Nota

Para asistencia adicional o escenarios de uso avanzado no cubiertos en este manual, consulte el documento completo de la tesis ([information/ThesisTFG.pdf](#)) o contacte directamente al autor.

B.8. Monitoreo y Optimización del Rendimiento

B.8.1. Perfilado del Rendimiento de Notebooks

Usar herramientas de perfilado integradas:

```
# Cronometrar celdas individuales
%%time
# Su código aquí

# Perfilar uso de memoria
```

```

%%memit
# Su código aquí

# Perfilado línea por línea
%load_ext line_profiler
%lprun -f nombre_funcion llamada_funcion()

# Perfilado de memoria
%load_ext memory_profiler
%mprun -f nombre_funcion llamada_funcion()

```

B.8.2. Monitoreo de Recursos

Monitorear recursos del sistema durante la ejecución del notebook:

```

import psutil
import time

def monitor_resources(duration=60, interval=5):
    """Monitorear recursos del sistema por duración especificada."""
    for _ in range(duration // interval):
        cpu_percent = psutil.cpu_percent()
        memory = psutil.virtual_memory()
        gpu_info = get_gpu_info() # Función personalizada

        print(f"CPU: {cpu_percent}%, RAM: {memory.percent}%, GPU:
        ↪ {gpu_info}")
        time.sleep(interval)

# Ejecutar monitoreo en segundo plano
import threading
monitor_thread = threading.Thread(target=monitor_resources,
        ↪ args=(300, 10))

```

```
monitor_thread.start()
```

B.9. Referencia Rápida

B.9.1. Ubicaciones de Archivos Importantes

Componente	Ubicación
Pipeline Principal	nnd/models/models_pipeline.py
Visualización	notebooks/predict/visualize_segmentation.ipynb
Análisis Estadístico	notebooks/results/experimental_results.ipynb
Configuración	nnd/models/*/__init__.py
Utilidades	nnd/utils/

Tabla 21: Referencia Rápida: Ubicaciones de Archivos.

B.9.2. Comandos Comunes

```
# Activar entorno
source venv/bin/activate

# Lanzar JupyterLab
jupyter lab --notebook-dir=notebooks/

# Ejecutar pipeline principal
cd nnd/models && python models_pipeline.py

# Verificar estado de GPU
nvidia-smi

# Monitorear logs
tail -f nnunet/nnUNet_result/Dataset024_MSLesSeg/fold_*/training.log
```