ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

MÁSTER UNIVERSITARIO EN INGENIERÍA DEL SOFTWARE E INTELIGENCIA ARTIFICIAL

Segmentación de Lesiones Cerebrales en Imágenes de Resonancia Magnética: Un Estudio de Optimización de Hiperparámetros de Detectron2

Brain Lesion Segmentation in Magnetic Resonance Imaging: A Detectron2 Hyperparameter Optimisation Study

Realizado por:

Alba Cano Lara

Tutorizado por:

Ezequiel López Rubio

Miguel Molina Cabello

Departamento:

Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA

MÁLAGA, Septiembre, 2025

Resumen. La esclerosis múltiple es una enfermedad neurológica crónica cuya detección temprana resulta fundamental para un diagnóstico adecuado y tratamientos oportunos. La resonancia magnética (*MRI*), constituye la principal técnica de monitorización de esta patología, ofreciendo imágenes de alta resolución que permiten identificar y cuantificar la progresión de las lesiones producidas por esta enfermedad en el cerebro. Sin embargo, la segmentación manual de éstas implica un elevado coste de tiempo y recursos.

En este trabajo de fin de máster se propone la aplicación de modelos de aprendizaje profundo basados en redes neuronales convolucionales y, en particular, la librería **Detectron2**, con el fin de evaluar su capacidad en tareas de segmentación de las lesiones producidas por la esclerosis múltiple. Para ello, se ha empleado el conjunto de datos **MsLesSeg**, que contiene imágenes MRI anotadas, sobre el cual se llevaron a cabo experimentos de *fine-tuning* de los modelos *pre-entrenados*. Asimismo, se han implementado técnicas de optimización como búsqueda aleatoria, algoritmos genéticos y validación cruzada, junto con estrategias de aumento de datos, a fin de adaptar los modelos a las particularidades del dominio.

Los resultados obtenidos permiten analizar de manera comparativa el rendimiento de diferentes configuraciones y aproximaciones de entrenamiento, identificando aquellas que logran un equilibrio más adecuado entre precisión y eficiencia computacional. En conclusión, este estudio contribuye a valorar la viabilidad del uso de herramientas de código abierto, originalmente diseñadas para tareas de visión por computador general, en el ámbito específico de la segmentación médica, aportando evidencia sobre sus fortalezas y limitaciones frente a arquitecturas específicamente diseñadas para esta tarea.

Palabras clave: Segmentación de Instancias, Aprendizaje Profundo, Redes Neuronales Convolucionales, Transferencia de Aprendizaje, Optimización de Hiperparámetros.

Abstract. Multiple sclerosis is a chronic neurological disease for which early detection is imperative for accurate diagnosis and timely treatment. Magnetic resonance imaging (*MRI*) is the primarly modality employed for the monitoring of this condition, providing high-resolution images that facilitate the identification and quantification of the progression of lesions caused by this disease in the brain. Nevertheless, the manual segmentation of these lesions is a time-consuming and resource-intensive process.

The present master's thesis puts forward the notion that the application of deep learning models base don convolutional neural networks, with a particular focus on the **Detectron2** library, holds considerable potential in evaluating their capacity for segmenting lesions caused by multiple sclerosis. To this end, the **MsLesSeg** dataset, which contains annotated MRI images, was used to carry out fine-tuning experiments on the pre-trained models. In order to adapt the models to the particularities of the domain, optimization techniques such as random search, genetic algorithms and cross-validation were implemented, along with data augmentation strategies.

The results obtained allow for a comparative analysis of the performance of different configurations and training approaches, identifying those that achieve more appropiate balance between accuracy and computational efficiency. In conclusion, the present study contributes to the assesment of the viability of using open-source tolos, originally designed for general computer vision tasks, in the specific field of medical segmentation. The study provides evidence of their strengths and limitations compared to architectures specifically designed for this task.

Keywords: Instance Segmentation, Deep Learning, Convolutional Neural Network, Transfer Learning, Hyperparameter Optimization.

ÍNDICE

1. Introducción

1.1 Motivación

La Esclerosis Múltiple es una enfermedad que afecta a millones de personas, por ello la detección temprana y la elaboración de un diagnóstico es crucial. Se utilizan diversas técnicas y biomarcadores como la Resonancia Magnética, o MRI (Magnetic Resonance Imaging). Esta técnica se ha vuelto clave para monitorizar el progreso de esta enfermedad y que profesionales de la salud sean capaces de responder con un tratamiento adecuado.

Gracias a las imágenes MRI y sus distintas modalidades de escaneo hace posible dar una visión en tres dimensiones del cerebro de los pacientes y potenciar ciertas zonas y características de este con el fin de contextualizar y mejorar la estimación de las lesiones por la MS (Multiple Sclerosis).

El reto central que se plantea consiste en desarrollar un sistema de detección que resulte automatizado, preciso y con un nivel de eficiencia adecuado. Por ello, se propone el uso de técnicas basadas en aprendizaje profundo.

En este ámbito, las redes neuronales convolucionales han evidenciado un alto rendimiento en aplicaciones de visión por computador y en la identificación de patrones complejos en imágenes. En la literatura se han propuesto diversas arquitecturas que implementan la base de una red convolucional junto con elementos específicos que tratan de optimizarlas con el objetivo de obtener mejores resultados para problemas concretos. Algunos de esos elementos permiten la segmentación de instancias, como las posibles lesiones en las imágenes MRI, de tal forma que son capaces de determinar con precisión el lugar donde se encuentran. Estas redes aprenden con la ingesta de grandes conjuntos de datos, usualmente anotados, de modo que sean capaces de generalizar y ser robustas ante cualquier tarea visual que se le pida para datos no antes vistos.

En el ámbito de este trabajo, se propone la utilización de un framework de código abierto apto para tareas de visión por computador y que tiene una extensa librería de modelos pre-entrenados de redes convolucionales basadas en regiones (R-CNN) sobre conjuntos de datos generales. Con este framework se pretende comprobar su rendimiento en el entorno de la segmentación de las lesiones por esclerosis múltiple y si es competente ante el resto de las arquitecturas diseñadas particularmente para el problema.

Para que los modelos sean capaces de detectar y dibujar el área de lesión en las imágenes, se hará uso del conjunto de datos anotados **MsLesSeg**, perteneciente a una competición de segmentación de lesiones de esclerosis múltiple organizado en 2024 principalmente por la Universidad de Catania (Italia) [1].

1.2 Objetivos

El propósito principal de este proyecto es implementar y optimizar el aprendizaje de diversos modelos de Detectron2 sobre el conjunto de datos seleccionado. Esto implicará adaptar y realizar varios entrenamientos de los modelos para ajustarlos a las particularidades de las imágenes cerebrales y características específicas de las lesiones de la esclerosis múltiple.

Como objetivos secundarios, este trabajo contempla el análisis del estado del arte de los modelos instanciación de imágenes y de segmentación por resonancia magnética, así como el desarrollo de un estudio de optimización de la herramienta de segmentación elegida sobre el conjunto de datos disponible. Dicho estudio se articula en varias fases:

- El desarrollo y diseño del preprocesamiento del conjunto de imágenes.
- La implementación del modelo de segmentación por instancias.
- La aplicación de diferentes técnicas de optimización del modelo.
- Comparación de los resultados obtenidos, con el propósito de determinar qué configuración del modelo proporciona mejores soluciones.

1.3 Organización

El trabajo está organizado en diferentes secciones. En la sección 2, se exponen los antecedentes de acuerdo con la propuesta desarrollada. Seguidamente, en la sección 3, se define con más detalle el problema que se pretende abordar. En la sección 4, se narra cómo se ha puesto en marcha la propuesta con la tecnología seleccionada. A continuación, la sección 5 recoge las pruebas realizadas, los resultados y la comparación con el resto de las soluciones existentes. Finalmente, en la sección 6 se exponen las conclusiones que se han formado tras completar el proyecto y posibles ideas que podrían incluirse en un trabajo futuro.

2. Antecedentes

En esta sección se describirán las principales redes neuronales utilizadas en la literatura en el estudio de detección y segmentación de instancias en imágenes de resonancia magnética cerebrales. Seguidamente, se detallarán las distintas técnicas de optimización de hiperparámetros y se estimará cuáles de éstas son las más adecuadas para el ámbito de este trabajo. Por último, se explicará la técnica de obtención de imágenes por resonancia magnética, sus diferentes tipos de secuencias y cómo se detectan las lesiones por esclerosis múltiple en ellas.

2.1 Redes Neuronales Aplicadas a la Visión por Computador

El desarrollo del aprendizaje profundo ha marcado un punto de inflexión en el campo de la visión por computador, impulsando notables progresos en áreas como la detección de objetos, el seguimiento de movimiento y la segmentación semántica. Entre las arquitecturas más representativas destacan las redes neuronales convolucionales, cuya aplicación ha permitido mejorar significativamente los resultados alcanzados en estas tareas a lo largo del tiempo.

Los modelos basados en redes convolucionales se distinguen por su eficacia en el procesamiento de información visual y en la identificación de patrones, apoyándose en una estructura compuesta por múltiples capas convolucionales, operadores de agrupamiento, funciones de activación como la *ReLU* o la *Sigmoide*, capas de normalización, combinaciones lineales y capas completamente conectadas como se ejemplifica en la *Figura 1*. En términos matemáticos, estas redes pueden describirse como una composición secuencial de funciones como se indica en *Función 1*, que a partir de un vector de entrada *x* y un conjunto de parámetros *W*, generan una representación transformada *x'*.

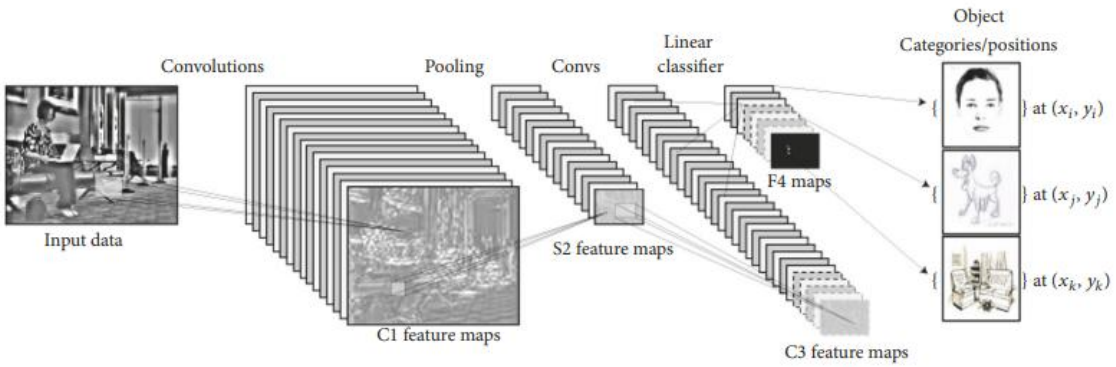

Figura 1. Esquema de la arquitectura de la CNN para la detección de objetos. [2]

$$f(x) = f_L(\cdots f_2(f_1(x_1, W_1); W_2)\cdots), W_L)$$

Función 1. Función compuesta que represta una red CNN. [3]

A continuación, se definirán las arquitecturas de redes convolucionales más utilizadas en la literatura y que han logrado resultados destacables en la segmentación de imágenes MRI.

2.1.1 UNet

Es una de las arquitecturas que más aparecen implementadas en los artículos y estudios acerca de la segmentación de entidades en entornos de la medicina. U-Net es una red neuronal convolucional que fue desarrollada específicamente para segmentación de imágenes biomédicas [4]. La propuesta de sus creadores fue construir a partir de la red totalmente convolucional [5] y sustituir los operadores originales de agrupamiento por unos de remuestreo. De manera que, la información resultante tras pasar por dichos operadores es de mayor resolución y las capas sucesivas que la reciben pueden aprender y detectar las características de forma más precisa.

Otro distintivo de esta red es la parte de sobremuestreo, donde se reúnen un gran número de canales de características que permiten a la red propagar la información contextual a las capas con mayor resolución. Como consecuencia, si se esquematiza la arquitectura de la red da lugar a una forma de U como se puede apreciar en la *Figura 2*, de ahí recibe el nombre de U-Net.

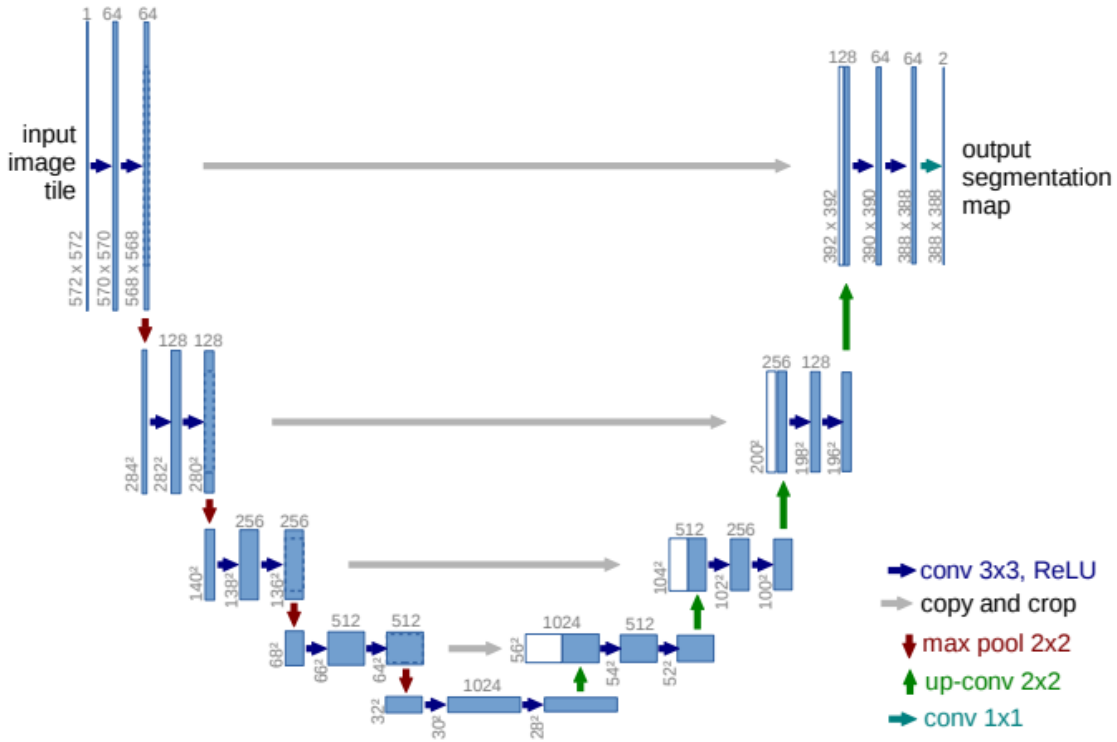

Figura 2. Arquitectura de la red convolucional U-Net (ejemplo de 32x32 píxeles a baja resolución) [4]

La estrategia de U-Net permite una segmentación excelente de imágenes en entornos biomédicos muy diferentes con conjuntos de datos anotados limitados y con un tiempo de entrenamiento eficaz. Por ello, gran parte de los estudios utilizan esta red convolucional para sus casos de uso e incluso introduciendo nuevas variaciones que mejoran su rendimiento. Como por ejemplo añadiendo conexiones residuales (Res-UNet) [6] o métodos de atención (Attention Res-UNet) [7].

2.1.2 Redes Convolucionales Basadas en Regiones (R-CNN)

En los últimos años, el aprendizaje profundo ha experimentado un notable progreso, en particular, gracias al desarrollo de arquitecturas convolucionales basadas en regiones, conocidas bajo la denominación de **R-CNN**. Estas variantes han marcado un hito en tareas de clasificación, localización y segmentación de objetos dentro de las imágenes. La razón principal de su éxito radica en que las CNN tradicionales eran capaces de identificar la categoría a la que pertenecía un objeto, pero no podían determinar su posición exacta en la imagen, lo cual se tornaba aún más complejo cuando coexistían múltiples instancias.

Las R-CNN abordan esta limitación al identificar regiones de interés denominadas **RoI** (*Region of Interest*), mediante un proceso de búsqueda selectiva apoyado en una red convolucional de base. El funcionamiento general de este enfoque puede observarse en la *Figura 3*, donde se distinguen cuatro fases principales.

1. **Red de Propuestas de Regiones (RPN):** Esta fase consiste en que el modelo genere un conjunto de regiones candidatas que tengan alta probabilidad de contener objetos de interés.
2. **Agrupación de Regiones de Interés (RoI):** Cada una de las regiones propuestas se recorta de la imagen original y se redimensiona a un tamaño fijo para su análisis posterior.
3. **Extracción de Características:** Las regiones normalizadas se introducen en una red convolucional previamente entrenada con el fin de obtener representaciones de alto nivel.
4. **Clasificación de objetos y Regresión de Cajas Delimitadoras:** Por último, las características extraídas permiten tanto la asignación de la clase correspondiente al objeto como la predicción de las coordenadas de la caja que lo enmarca.

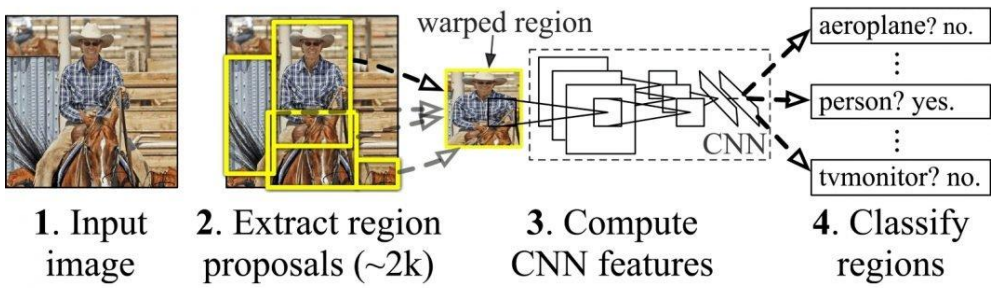

Figura 3. Esquema del funcionamiento interno de una R-CNN. [8]

Si bien que este enfoque demostró gran eficacia, el proceso de entrenamiento resultaba costoso y lento, lo que dio lugar al surgimiento de variantes como **Fast R-CNN**, **Faster R-CNN**, **YOLO** (*You Only Look Once*) o **Mask R-CNN**, diseñadas para mejorar tanto la precisión como la velocidad de cómputo.

2.1.3 Mask R-CNN

Dentro de estas evoluciones, **Mask R-CNN** introducido por [9] se consolidó como una de las arquitecturas más relevantes al ampliar las capacidades de **Faster R-CNN** [10]. Además de predecir la clase y la caja delimitadora de cada objeto candidato, este modelo incorpora

una rama adicional capaz de generar una máscara binaria precisa para cada región de interés. La *Figura 4* esquematiza el proceso de segmentación con Mask R-CNN.

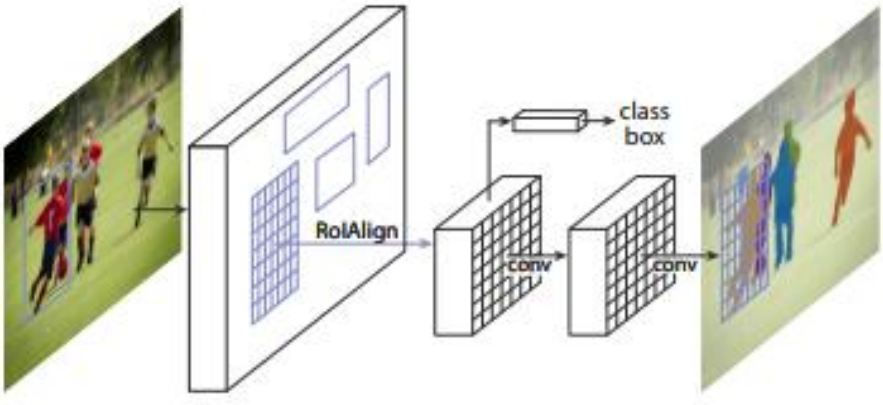

Figura 4. Esquema de segmentación con Mask R-CNN. [9]

Su diseño se apoya en una estructura de tipo **ResNet** [11], combinada con **FPN** (*Feature Pyramid Network*) [12], lo que le permite un mejor equilibrio entre velocidad y precisión. La arquitectura se organiza en dos componentes principales:

1) **Backbone** – es el encargado de la extracción jerárquica de las características a partir de la imagen completa.
2) **Head** – su función es integrar tanto las operaciones de clasificación y regresión de las cajas delimitadoras como la predicción de las máscaras.

Los resultados obtenidos resultaron notablemente satisfactorios, como se evidencia en la *Figura 5*.

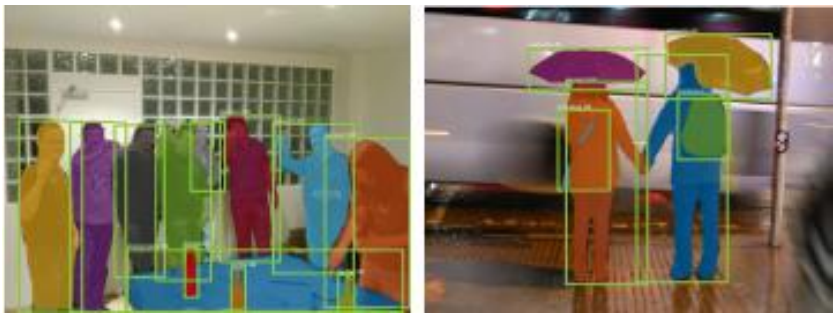

Figura 5. Resultados de Mask R-CNN sobre imágenes del conjunto COCO. [9]

2.1.4 YOLO

YOLO, "*You Only Look Once*" [13], es un modelo de detección de objetos ampliamente reconocido por su elevada velocidad y capacidad para identificar múltiples instancias en imágenes o secuencias de vídeo. Esta propuesta marcó un punto de inflexión en el ámbito de la detección, puesto que permitió localizar y clasificar diferentes objetos en una sola pasada de la red, a diferencia de los enfoques anteriores que requerían varios procesos consecutivos y un mayor coste computacional.

La arquitectura de **YOLO** se fundamenta en redes neuronales convolucionales y se caracteriza por procesar la imagen de entrada mediante su división en una malla o cuadrícula de celdas,

tal como se ilustra en la *Figura 6*. Cada una de estas celdas se encarga de generar un conjunto de predicciones que incluyen posibles cuadros delimitadores junto con la probabilidad de pertenencia a distintas clases de objetos. La descripción de dichos cuadros se realiza mediante las coordenadas de sus esquinas, mientras que las probabilidades predichas permiten estimar la categoría del objeto presente. En consecuencia, el proceso de detección se formula como un problema de regresión sobre las coordenadas de las cajas y clasificación simultánea sobre las categorías, lo que convierte a YOLO en un modelo capaz de realizar detecciones en una sola pasada de la red, con una notable eficiencia en términos de velocidad de inferencia.

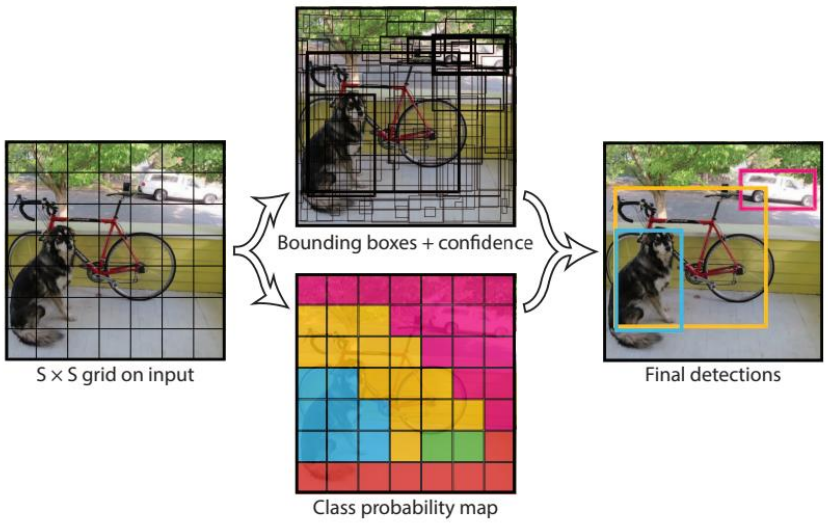

Figura 6. Detección del modelo YOLO. [13]

2.1.5 Detectron2

Detectron2 es un proyecto de código abierto desarrollado por *Facebook AI Research* [14] ampliamente utilizado para tareas de detección de objetos, la segmentación de instancias y la segmentación semántica. Este sistema se distingue por su diseño modular y flexible, que facilita la personalización de los diferentes componentes de la red, entre ellos el núcleo, el cuello y la cabeza, como se muestra en la *Figura 7*. Además, integra una extensa colección de modelos pre-entrenados que pueden ajustarse a distintos conjuntos de datos y objetivos específicos de la detección, lo que lo convierte en una herramienta de gran versatilidad.

Su arquitectura mantiene similitudes con las implementaciones derivadas de **Mask R-CNN**, organizándose en tres bloques principales:

1) **La parte troncal (Backbone)**, responsable de generar y extraer mapas de características en múltiples escalas.
2) **La Red de Propuestas de Regiones (RPN)**, que identifica áreas candidatas basadas en dichas características multiescala.
3) **Los ROI Heads**, que se encargan de refinar la información obtenida mediante capas totalmente conectadas que ajustan tanto las coordenadas de las cajas delimitadoras como la clasificación de los objetos detectados.

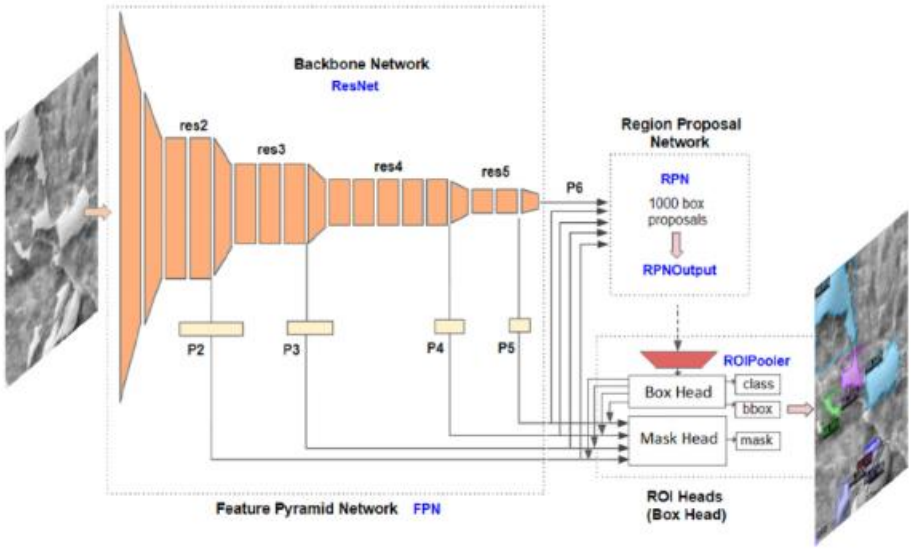

Figura 7. Esquema de la arquitectura de Detectron2. [15]

2.2 Técnicas de Optimización

En este estudio se aplicarán diversas técnicas de optimización orientadas a mejorar el análisis de imágenes por resonancia magnética. Estas metodologías permiten afrontar limitaciones habituales como la escasez de datos disponibles, la complejidad computacional de los modelos y la necesidad de garantizar una adecuada capacidad de generalización. En particular, se emplearán estrategias de *transfer learning*, *data augmentation*, métodos de validación más robustos como la validación cruzada, así como procedimientos de optimización de hiperparámetros tales como la búsqueda aleatoria y los algoritmos genéticos para explorar configuraciones en problemas de alta complejidad.

2.2.1 Transfer Learning

Transfer learning, o aprendizaje por transferencia es una técnica de aprendizaje automático que permite emplear el conocimiento capturado por un modelo previamente entrenado en una tarea para mejorar el rendimiento y acelerar el aprendizaje en una nueva tarea más específica [16]. Entrenar un modelo desde cero es costoso, conlleva bastante tiempo y en tareas como la visión por ordenador el procesamiento de la información resulta en un incremento aún mayor en el tiempo. Aplicar esta técnica es útil cuando se disponen de pocos datos en el dominio objetivo, de tal forma que el modelo puede reutilizar conocimiento adquirido anteriormente [17].

2.2.2 Data Augmentation

Los datos en las tareas de entrenamiento de modelos de inteligencia artificial son cruciales para el éxito de éstas. Por ello, cuando son insuficientes se aplica la técnica *data augmentation* que enriquece artificialmente un conjunto de datos mediante transformaciones. Entre las estrategias básicas se incluyen transformaciones geométricas como la rotación, el volteo, escalado, traslaciones o recortes; ajustes en el espacio de color como modificando el brillo, el contraste o la saturación; y la adición de ruido como aplicando un filtro Gaussiano sobre la imagen. Estas variaciones aumentan la diversidad de los datos, ayudan a mitigar el sobreajuste y mejoran la capacidad del modelo para generalizar ante variaciones reales. [18]

2.2.3 K-Fold Cross Validation

La validación cruzada *k-fold* es una metodología estadística de evaluación que se puede emplear en modelos de inteligencia artificial que ayuda a estimar la capacidad de generalización de estos. Esta técnica consiste en dividir el conjunto de datos en k subconjuntos de un tamaño homogéneo. Seguidamente, el modelo se entrena k veces, en las que en cada iteración se toma un subconjunto k como conjunto de validación y los k-1 como conjunto de entrenamiento. Finalmente, tras las k iteraciones se realiza un promedio de los resultados obtenidos que significará en una métrica representativa del rendimiento y generalización del modelo ante diversas divisiones del conjunto de datos [19]. Con esto se pretende evitar sesgos introducidos por la partición aleatoria de los datos, mejorar la detección de sobreajuste y demostrar una mayor fiabilidad ante una simple división de entrenamiento y test.

2.2.4 Random Search

El método de búsqueda aleatoria se define como un método matemático de optimización de hiperparámetros en el cual se seleccionan de manera aleatoria diversas combinaciones de soluciones dentro de unos rangos especificados. Debido a su aleatoriedad es capaz de generar secuencias de aproximaciones e incrementar la probabilidad de encontrar configuraciones óptimas locales en espacios de búsqueda de alta combinatoria [20].

2.2.5 Algoritmo Genético

Los algoritmos genéticos pertenecen a la familia de las metaheurísticas que se inspiran en los mecanismos de la selección natural. Son algoritmos basados en la búsqueda poblacional, que utilizan el concepto de la supervivencia del más apto asignando una puntuación a cada individuo de la población, fitness [21]. En cada iteración se trabaja con una población de soluciones candidatas que van evolucionando a lo largo de varias generaciones, o iteraciones, con el uso de operadores de selección, de cruce para la reproducción, mutación, reemplazo y evaluación de los individuos. Estos algoritmos proporcionan un marco general para abordar cuestiones complejas relacionados con la combinatoria y la optimización de sistemas, siendo independientes del contexto y el tipo de problemas [22].

A continuación, en *Algoritmo 1*, se presenta el pseudocódigo del esquema general de un algoritmo genético.

Input: Tamaño de la población, n. Número máximo de generaciones de la población, G

Output: Mejor solución, Ybest

```
Empezar
    Generar población inicial de tamaño n Yi (i=1, 2, …, n)
    Evaluar población inicial
    Establecer contador t a 0
    Mientras que (t < G) entonces
        Seleccionar progenitores de la población inicial
        Cruzar los progenitores
        Aplicar mutación al individuo hijo acorde a una probabilidad de mutación
        Reemplazar la población con los hijos obtenidos
        Incrementar t
    Terminar
    Devolver el mejor individuo de la población, Ybest
Terminar
```

Algoritmo 1. Estructura básica de un Algoritmo Genético. [21]

2.3 Imágenes de Resonancia Magnética (MRI)

Para comprender mejor el contexto del problema, a continuación, se detallan qué son las imágenes MRI que tenemos como conjunto de datos, cómo se obtienen, los tipos de escaneos y cuáles son las características visuales principales de las lesiones por esclerosis múltiple.

2.3.1 Introducción a la Resonancia Magnética

La resonancia magnética es una técnica de imagen médica ampliamente utilizada en neurología. Su principal ventaja radica en su capacidad de proporcionar detalles rigurosos de la anatomía del órgano a estudiar en tres planos: axial, sagital y coronal [23]. En la *Figura 8* se muestra un ejemplo de los diferentes planos en una resonancia magnética cerebral.

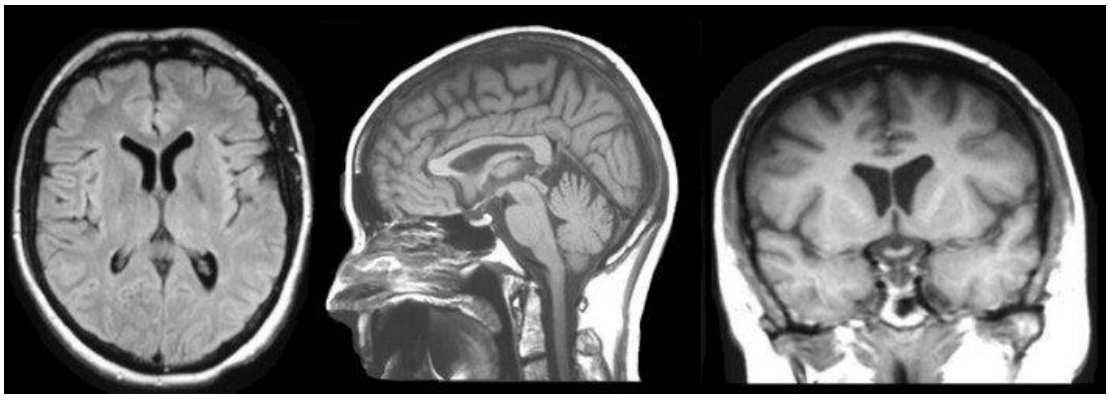

Figura 8. Planos axial, sagital y coronal de una resonancia magnética cerebral. [23]

Esta técnica se basa en las propiedades de magnetización de los núcleos atómicos, principalmente los protones de hidrógeno, abundantes en el agua y en la grasa del cuerpo humano. El proceso implica la utilización de un potente campo magnético externo que obliga a los protones de los núcleos de agua del tejido examinado a alinearse con dicho campo. Seguidamente se introduce energía de radiofrecuencia en el paciente, de modo que los protones se estimulan y pierden el equilibrio, resistiéndose a la fuerza del campo magnético

[24]. Una vez se apaga la señal, los núcleos regresan a su alineación de reposo mediante procesos de relajación, emitiendo energía de radiofrecuencia detectada por los sensores de la resonancia magnética. Después de un tiempo determinado, se miden las señales emitidas. Aplicando la transformación de Fourier se convierte la información de frecuencia de la señal en niveles de intensidad, mostrada como un rango de tonos grises en una matriz de píxeles para formar la imagen [23].

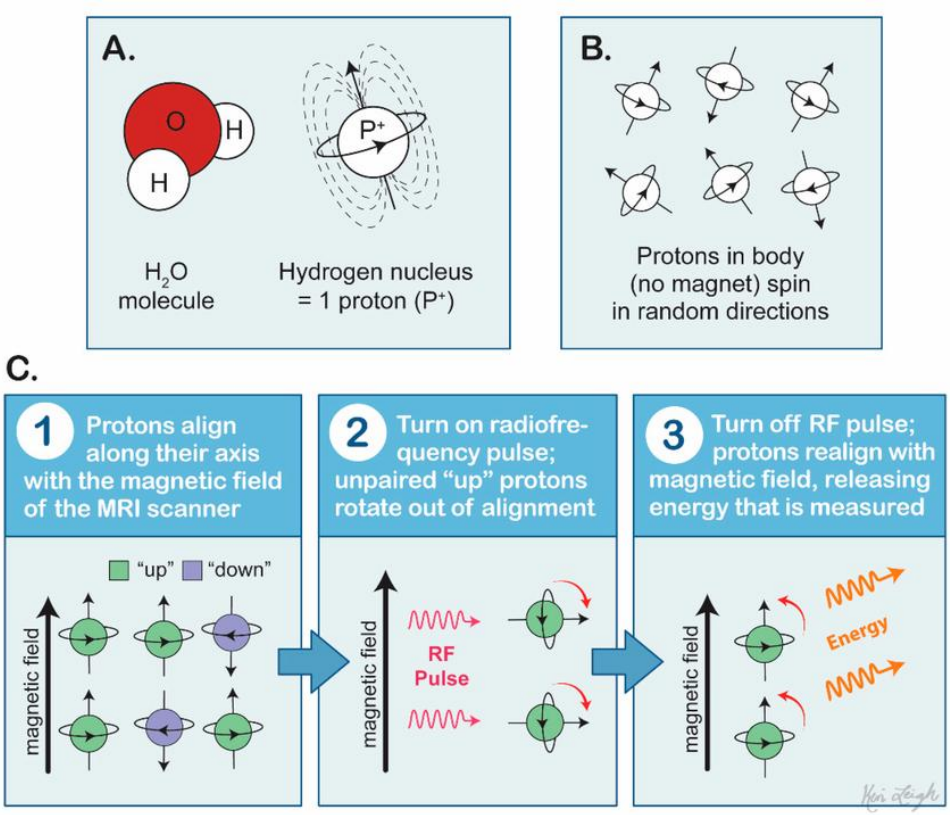

Figura 9. Proceso esquematizado del mecanismo de MRI. (A) Muestra los protones de hidrógeno. (B) Los protones están alineados al azar en el tejido a estudiar. (C) Al aplicar el campo magnético los protones se alinean paralelamente al eje del campo; al emitir pulsos de radiofrecuencia dichos protones se realinean perpendicularmente al campo magnético; cuando deja de emitirse radiofrecuencia, los protones liberan energía captada por los sensores MRI. [25]

2.3.2 Tipos de Escaneos

La creación de diferentes tipos de imágenes se logra variando la secuencia de pulsos de radiofrecuencia aplicados y recogidos. Hay tres parámetros clave en esta variación:

- **Tiempo de Repetición (TR):** Cantidad de tiempo entre sucesivas secuencias de pulsos aplicadas a la misma sección [23].
- **Tiempo de Eco (TE):** Tiempo que transcurre entre la entrega del pulso de radiofrecuencia y la recepción de la señal de eco [23].
- **Tiempo de Inversión (TI):** Utilizado en secuencias FLAIR para anular la señal de fluidos específicos [26].

Por otro lado, las propiedades de relajación de los tejidos, denominados T_1 y T_2, son factores que influyen también en la diferenciación de tejidos en las imágenes.

- **T_1 (Tiempo de relajación longitudinal):** es la constante de tiempo que determina la velocidad a la que los protones excitados tardan en realinearse con el campo magnético externo [23]. Las imágenes potenciadas en T_1 son útiles para evaluar la corteza cerebral, identificar el tejido graso y obtener información morfológica del órgano [27].
- **T_2 (Tiempo de relajación transversal):** es la constante de tiempo que determina la velocidad a la que los protones en rotación alcanzan el equilibrio o se desfasan entre sí [23]. Las imágenes potenciadas en T_2 son eficaces en detectar inflamaciones y revelar lesiones de la sustancia blanca [27].

Dados estos parámetros y constantes se pueden construir diferentes tipos de secuencias de resonancia magnética. En este trabajo se definirán los tres más comunes y que están presentes en el conjunto de datos, *T_1-Weighted*, *T_2-Weighted* y *FLAIR*.

Imágenes potenciadas en T_1 Weighted

Se obtienen ajustando los tiempos de repetición (TR) y eco (TE) hacia valores cortos (por ejemplo, TR ≈ 500 ms y TE ≈ 14 ms [23]), priorizando el contraste basado en las propiedades de relajación T_1 del tejido. Estas imágenes reflejan la anatomía con gran fidelidad, mostrando el líquido, como el líquido cefalorraquídeo (LCR) con baja señal (píxeles más oscuros), la materia gris con una señal intermedia (gris), la materia blanca más brillante (blanquecina), y la grasa con alta intensidad de señal. El músculo también aparece con señal intermedia, mientras que los focos de inflamación suelen verse oscuros [28]. Al aplicar *gadolinio*, un agente paramagnético que reduce el tiempo T_1, las estructuras que captan contraste se vuelven especialmente brillantes, lo que realza áreas con alteración de la barrera hematoencefálica, como tumores o sitios inflamatorios [23].

Imágenes potenciadas en T_2 Weighted

Utilizan TR y TE mucho más largos (por ejemplo, TR ≈ 4000 ms y TE ≈ 90 ms [23]), destacando tejidos con alta agua libre, lo que las hace excelentes para detectar edemas, inflamaciones y lesiones en materia blanca. En estas imágenes, el líquido como el LCR aparece brillante, la materia gris y el músculo se mantienen con señal intermedia, la materia blanca se muestra más oscura que la gris, y la grasa sigue siendo brillante. De esta manera las zonas inflamadas se ven intensamente más brillantes, y gracias a la supresión grasa es aún más fácil la identificación de líquido en presencia de grasa que presentan los edemas [28].

Secuencia FLAIR (Fluid Attenuated Inversion Recovery)

Es una variante de T_2 que a diferencia de ésta incorpora un tiempo de inversión (TI) ajustado para anular la señal del líquido cefalorraquídeo (LCR), haciéndolo oscuro, y con tiempos TR y TE muy largos (aproximadamente TR ≈ 9000 ms y TE ≈ 114 ms). Así, cualquier patología permanece luminosa y se distingue claramente del líquido circundante [23]. El aspecto de la corteza aparece similar al de una T_2, con materia gris más luminosa que la blanca. A simple vista puede confundirse con una imagen T_1 (por el LCR oscuro), pero la clave para diferenciarlas yace en el contraste entre materia gris y blanca. En T_1, la gris es más oscura que la blanca y en FLAIR (como en T_2), la blanca es más oscura que la gris [26]. Clínicamente, FLAIR forma parte de casi todos los protocolos de MRI cerebral, siendo especialmente útil en

patologías como infartos, traumatismos craneoencefálicos o esclerosis múltiple, puesto que resalta cambios sutiles en regiones adyacentes al LCR.

2.3.3 Características de las Lesiones por Esclerosis Múltiple

Las lesiones causadas por la esclerosis múltiple son áreas del sistema nervioso central donde la cubierta protectora de las fibras nerviosas ha sido dañada, interrumpiendo así la transmisión de señales nerviosas. Estas lesiones, también conocidas como placas, implican inflamación y la infiltración de células inmunes [29]. Pueden aparecer en diversas partes del sistema nervioso central, siendo comunes en la sustancia blanca del cerebro, el nervio óptico y la médula espinal [30].

En las secuencias del conjunto de datos se pueden localizar dada una serie de características que pueden ser más diferenciables en T_2 y FLAIR que en T_1, a la cual se le debe de aplicar un contraste posterior para mejorar la detección de las lesiones.

En las imágenes potenciadas en T_2, las lesiones aparecen como áreas hiperintensas, blancas y brillantes [31]. Estas señales brillantes permiten al modelo aprovechar tanto la intensidad como patrones de textura para distinguir correctamente las lesiones. La forma suele ser ovalada o circular, ubicándose en zonas características como la sustancia blanca cercana a los ventrículos o adyacente a la corteza [30].

Las secuencias FLAIR, similares a las imágenes ponderadas en T_2, pero con un mejor contraste con respecto al tejido cerebral circundante. Esto hace que las secuencias FLAIR sean especialmente eficaces para detectar lesiones periventriculares y corticales, que son las localizaciones características de las placas de EM. Las imágenes FLAIR son esenciales para diagnosticar la EM y diferenciarla de otras afecciones neurológicas [31].

Por último, en las imágenes T_1 sin contraste, las lesiones de EM se manifiestan como áreas hipointensas, oscuras, conocidas como "*agujeros negros*". Estos reflejan daño tisular profundo e irreversible, y tienen una fuerte asociación con discapacidad a largo plazo, siendo más frecuentes y extensos en formas progresivas de la enfermedad [30]. En las secuencias T_1 post-contraste, las lesiones activas que se encuentran donde existe ruptura de la barrera hematoencefálica, se realzan y se vuelven brillantes, lo que actúa como biomarcador clave de actividad inflamatoria en curso. Esta distinción entre lesiones activas e inactivas es fundamental para segmentar correctamente y para seguir la progresión o respuesta al tratamiento [31].

3. Descripción del Problema

En este trabajo de investigación se pretende comprobar el rendimiento de modelos generales pre-entrenados en conjuntos de imágenes MRI donde las instancias a segmentar son de un tamaño reducido y con unas características casi imperceptibles. El principal desafío de los modelos será la correcta detección y segmentación de las lesiones por esclerosis múltiple. Estas lesiones pueden representar una mínima parte de las imágenes, teniendo más casos de "no lesión" o elementos clasificados como fondo (*background*). En proporción, el número de píxeles clasificados como lesión es mucho menor y de menos dimensión que los píxeles "no lesión". Como consecuencia, los modelos experimentarán dificultades a la hora de aprender y diferenciar con precisión.

Las redes neuronales convolucionales que se proponen para el problema en cuestión están basadas en regiones como explicado anteriormente. Principalmente son Mask R-CNN, las cuales tienen dos fases bien diferenciadas al proceder en el entrenamiento de sus capas. Al introducir una imagen como entrada en el flujo de este tipo de redes, en la primera fase tratarán de encontrar las zonas de interés denominadas ROI. Llevarán a cabo las tareas de clasificación y detección de los objetos, reportando las dimensiones de estos en áreas rectangulares y un grado de confianza de que ese objeto pertenece a la clase objetivo. Según el umbral de confianza determinado, el modelo dará como válida dicha detección. Posteriormente, en la siguiente fase, teniendo en cuenta dichos rectángulos actúan las capas dedicadas a la segmentación de píxeles. En cada rectángulo se determinarán qué píxeles son considerados objetivos delimitando así un área más irregular en el espacio de la imagen resultante.

Otro reto al que nos enfrentamos en este proyecto es la detección parcial de las lesiones en las imágenes, dado que éstas pueden ser múltiples debido al contexto del problema. Dada una imagen MRI, es probable que presente varias zonas de lesión por esclerosis múltiple en distintas zonas de la imagen, o en otras ocasiones, adyacentes entre sí. Ante esto, es probable que los modelos no sean competentes para segmentar al completo y correctamente las imágenes recibidas si no se les proporcionan suficientes ejemplos.

Por tanto, para que el modelo sea capaz de ir más allá de indicar que la imagen contiene la clase objetivo y establecer las diversas zonas dañadas se plantean varios procedimientos para mejorar el rendimiento de la segmentación del modelo. Primeramente, se aplicarán técnicas fundamentales de pre-procesamiento del conjunto de datos, como normalización de los valores de los píxeles o el redimensionamiento homogéneo. También se probará qué tipo de escaneo de las imágenes MRI ofrece mayor rendimiento a los modelos, y si la combinación entre ellos podría contribuir a ello. Por último, se pondrán en marcha diversas técnicas de optimización de hiperparámetros adecuadas para los modelos de aprendizaje profundo puesto que otro de los desafíos de este trabajo es la gran combinatoria de posibles configuraciones de los modelos de entrenamiento. Algunos de los parámetros que se modificarán serán la arquitectura base del modelo, las iteraciones, el aumento de la resolución de las imágenes, cuántas de ellas se procesan a la vez en cada vuelta, o la ratio de aprendizaje entre otros.

4. Detalles de la Propuesta

A lo largo de este punto, se detallará el conjunto de datos de estudio y su previo procesamiento para su entrenamiento. Seguidamente, se describirá cómo se ha establecido el flujo de trabajo y por último la tecnología y las técnicas utilizadas.

4.1 Conjunto de Datos

La estructura del conjunto de datos de **MSLesSeg** [1] viene diferenciada entre los escaneos enfocados a entrenamiento y los de test, los cuales contienen un número determinado de pacientes, siendo mayor en el conjunto de entrenamiento. Dentro éste, se les ha realizado en distintos momentos a los pacientes hasta tres tomas de resonancias magnéticas, indicándose como T1, T2, o T3 si se fuese el caso. En cada toma se incluyen las secuencias previamente mencionadas en la sección 2.3.2 Tipos de Escaneos, T_1-Weighted, T_2-Weighted y FLAIR, a las cuales se han anotado sus correspondientes máscaras en archivos denominados MASK. De tal forma que, la estructura general del conjunto de datos puede esquematizarse como se ve en *Tabla 1*. Cabe destacar el Paciente 30, anteriormente perteneciente al conjunto de entrenamiento, y que al no poseer las anotaciones de las lesiones, se ha estimado oportuno trasladarlo al conjunto de test.

Conjunto de Entrenamiento	Conjunto de Test
P1 (Paciente 1) T1 T1.nii T2.nii FLAIR.nii Masks.nii T2 T1.nii T2.nii FLAIR.nii Masks.nii T3 T1.nii T2.nii FLAIR.nii Masks.nii P2 (Paciente 2) T1 T1.nii T2.nii FLAIR.nii Masks.nii T2 T1.nii T2.nii FLAIR.nii Masks.nii … P53 (Paciente 53)	P30 (Paciente 30) T1.nii T2.nii FLAIR.nii P54 (Paciente 54) T1.nii T2.nii FLAIR.nii P55 (Paciente 55) T1.nii T2.nii FLAIR.nii P56 (Paciente 56) … P75 (Paciente 75)

Tabla 1. Estructura del conjunto de datos MSLesSeg.

Estas imágenes médicas vienen con formato *.nii*, común en este tipo de datos, por lo que para su correcto procesamiento por parte de los modelos proporcionados por Detectron2 deben transformarse a formato *.png*. Para ello, el procedimiento a seguir consiste en extraer cada fragmento del escaneo MRI, al que hemos denominado *slice*, seguido de un número del 0 al 181 recorriendo de principio a fin el plano sagital del cerebro del paciente. A cada fragmento se le aplica una normalización en el rango de los píxeles de valores entre 0 y 255, y manteniendo el dimensionamiento original homogéneo de 182 píxeles de ancho y 218 de alto.

Una vez obtenidas las imágenes en el formato requerido, se propuso dividir el conjunto de datos de entrenamiento original en dos subconjuntos: entrenamiento y validación, con proporciones del 80% y 20%, respectivamente. El objetivo de esta división es evaluar el rendimiento de los modelos aplicando las métricas de evaluación que se detallarán más adelante y, a partir de ello, comparar los resultados con otras soluciones propuestas para este mismo conjunto de datos. No obstante, cabe resaltar que las evaluaciones realizadas sobre los conjuntos de validación no son estrictamente comparables con las calificaciones alcanzadas por otros grupos en la clasificación oficial de este conjunto de datos. Esto se debe a que dichas calificaciones se calcularon utilizando el conjunto de test y sus correspondientes anotaciones, las cuales no se disponen en este trabajo.

Posteriormente, para cada subconjunto de datos se deben transformar sus anotaciones al formato permitido por el framework Detectron2 para el entrenamiento de los modelos, denominado COCO-JSON [32].

```
función anotación_imagenes(mask, image_id, annotation_id, category_id):
  # Obtención de los contornos de las máscaras binarias
  contours = cv2.encontrarContorno(máscara, cv2.RETR_EXT, cv2.CHAIN_APPROX_SIMPLE)
  annotations = []

  # Creación de las anotaciones para cada contorno
  para cada contorno en contornos:
    bbox = cv2.boundingRect(contour)
    area = cv2.contourArea(contour)
    segmentation = contour.flatten().tolist()
     annotation = {
        "iscrowd": 0,
         "id": annotation_id,
         "image_id": image_id,
         "category_id": category_id,
         "bbox": bbox,
          "area": area,
          "segmentation": [segmentation],
    }
    si area > 0:
       annotations.append(annotation)
       annotation_id += 1

  retornar annotations, annotation_id
```

Algoritmo 2. Anotación de las máscaras de cada imagen.

```
función procesar_máscaras(dir_mask, dest_json):
  global image_id, annotation_id
  image_id = 1
  annotation_id = 1

  # Inicialización del archivo COCO JSON con sus categorías
  coco_output = {
    "info": {},
    "licenses": [],
    "images": [],
    "categories": [{"id": 1, "name": “MS Lesion”}],
    "annotations": [],
  }

  # Creación de las secciones para las imágenes y sus anotaciones
  para cada máscara en dir_mask:
    coco_output["images"].append({
      "id": image_id,
      "file_name": mask_path.name.replace('MASK', modality),
      "width": width,
      "height": height
    })
    anns, annotation_id = anotación_imágenes(máscara, image_id, annotation_id)
    coco_output["annotations"].extend(anns)
    image_id += 1

  # Guardar el archivo COCO JSON
  guardarJSON(coco_format, dest_json)
```

Algoritmo 3. Creación del archivo JSON y procesamiento de las máscaras.

Los *Algoritmo 2* y *Algoritmo 3* descritos anteriores tienen como función principal transformar las máscaras binarias del conjunto de imágenes en anotaciones en formato *COCO JSON*. Para ello, se extraen los contornos de cada lesión utilizando herramientas de procesamiento de imágenes proporcionadas por **OpenCV**. A partir de estos contornos, es posible derivar las coordenadas poligonales correspondientes y calcular las cajas delimitadoras, que resultan fundamentales en la fase de entrenamiento de los modelos. Una vez procesadas todas las imágenes, se genera un archivo en formato **.json** que recopila la información de segmentación junto con las categorías definidas, que en este caso se limitan a una única clase, “Lesión”. Un ejemplo de la estructura de este archivo se presenta en el *Código 1*.


```
{
        "info": {},
        "licenses": [],
        "images": [
          {
           "id": 1,
           "file_name": "P10_T1_FLAIR_slice_0_0.png",
           "width": 182,
           "height": 218
          },
```

```
        {
          "id": 2,
          "file_name": "P10_T1_FLAIR_slice_0_1.png",
          "width": 218,
          "height": 182
        }, . . . ]
      "annotations": [
        {
          "id": 1,
          "image_id": 25,
          "category_id": 1,
          "segmentation": [
            [ 104, 105, 104, 106, 103, 107, 103, 111, 104, 110, 104, 109, 105, 108, 104, 107 ]
          ],
          "bbox": [ 103, 105, 3, 7 ],
          "area": 5.0,
          "iscrowd": 0
        },
        {
          "id": 2,
          "image_id": 25,
          "category_id": 1,
          "segmentation": [
            [ 98, 73, 98, 75, 102, 75, 103, 74, 102, 73 ]
          ],
          "bbox": [ 98, 73, 6, 3 ],
          "area": 9.0,
          "iscrowd": 0
        }, . . . ]
}
```

Código 1. Ejemplo de anotación transformada a COCO JSON.

4.2 Diseño Experimental

Una vez se tiene el conjunto de datos preparado para la experimentación, se procede a diseñar las etapas de la propuesta. Primeramente, se determinan los equipos que se van a utilizar durante el proceso, en nuestro caso, se hará uso de dos entornos de trabajo. Por un lado, se ha hecho uso de los servicios de computación en línea que ofrece *Google Colab* en los que se han realizado las ejecuciones iniciales. Por otro lado, en las fases más avanzadas, se ha llevado a cabo la experimentación intensiva en un servidor especializado y con capacidad de computación gráfica del grupo de investigación *ICAI* (Inteligencia Computacional y Análisis de Imágenes) de la Universidad de Málaga. En la *Tabla 2* y *Tabla 3* se exponen los componentes de los equipos empleados en la experimentación, con el propósito de proporcionar un contexto adecuado a todas las pruebas realizadas.

Componente	Modelo
Memoria RAM	12.7 GB
Disco Duro	78.2 GB
GPU	Tesla T4 15 GB

Tabla 2. Especificaciones del Equipo de Google Colab.

Componente	Modelo
Memoria RAM	32 GB
Disco Duro	3 TB
GPU	Nvidia RTX 3080 10 GB

Tabla 3. Especificaciones del Equipo del servidor del grupo de investigación ICAI.

Posteriormente, se diseñará el flujo de trabajo principal, el cual deberá ser capaz de ejecutar el proceso completo a partir de una serie de parámetros definidos en una estructura de datos. Este planteamiento proporcionará la flexibilidad necesaria para llevar a cabo futuras ejecuciones con cambios mínimos en la configuración. Dichos cambios pueden incluir, por ejemplo, la selección de una arquitectura de red alternativa o la modificación del tamaño de las imágenes de entrada, lo que permite explorar de manera sistemática distintas variantes del modelo.

La siguiente fase consiste en implementar las técnicas de *fine-tuning* descritas previamente en la sección *2.2 Técnicas de Optimización* **¡Error! No se encuentra el origen de la referencia.**. Al disponer de un flujo de trabajo general ya establecido, la aplicación de estos métodos al conjunto de datos resultará más eficiente y facilitará la recolección de resultados de manera semi-automatizada, reduciendo así la probabilidad de errores manuales y mejorando la reproducibilidad de los experimentos.

Finalmente, con dichos resultados se identificarán las configuraciones más prometedoras y se someterán al proceso de validación cruzada. De esta forma, el rendimiento de éstas será más representativo con el objetivo de establecer comparaciones fiables con otras soluciones propuestas en la competición asociada al conjunto de datos.

4.3 Implementación

En este proyecto, se ha hecho uso de la herramienta de código abierto **Detectron2**. La cual proporciona varios modelos pre-entrenados según la tarea de visión por computador. En nuestro caso, se ha optado por emplear los modelos de segmentación de instancias basados en Mask R-CNN. Detectron2 ofrece diferentes variantes de *backbone* para Mask R-CNN, cuyos nombres hacen referencia a la arquitectura de red base:

- **C4:** utiliza de manera más directa las capas intermedias de *ResNet*, en concreto la cuarta etapa de convoluciones, para generar las predicciones. Fue la configuración original de *Faster R-CNN*.
- **DC5 o Dilated-C5:** se apoya en las capas más profundas de *ResNet*, en este caso en la quinta etapa, pero aplicando convoluciones dilatadas para capturar más contexto en la imagen sin aumentar demasiado el coste computacional.
- **FPN o Feature Pyramid Network:** combina información de varias profundidades de *ResNet* mediante una red piramidal. Lo cual permite detectar con más eficacia objetos de distintos tamaños y ofrece la mejor relación entre precisión y velocidad.

Además de las diferencias estructurales, Detectron2 proporciona modelos entrenados con distintos esquemas de entrenamiento. El denominado *3x schedule* equivale según la documentación de la herramienta a unas 37 épocas sobre el conjunto COCO [32], mientras que *1x schedule* serían unas 12 épocas, algo insuficiente para un entrenamiento completo pero que aun así se pueden tener en cuenta para las fases iniciales de experimentación.

De esta manera, la nomenclatura de los modelos disponibles sigue el patrón *[Backbone]-[Variante]-[Schedule].* Así, un modelo llamado **R50-FPN-3x** correspondería a una arquitectura Mask R-CNN con una base de **ResNet-50**, empleando una red **FPN** y entrenado durante **37 épocas**. El listado completo de modelos que se ejecutarán es el siguiente:

- R50-C4-1X
- R50-DC5-1X
- R50-FPN-1X
- R101-C4-3X
- R101-DC5-3X
- R50-C4-3X
- R50-DC5-3X
- R50-FPN-3X
- R101-FPN-3X
- X101-FPN-3X

Para utilizar la herramienta Detectron2 se debe instalar el repositorio correspondiente de *GitHub* junto con las dependencias y librerías que precise. Seguidamente, se instancian los pesos del modelo pre-entrenado elegido como se muestra en *Código 2*.

```
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url(
                        "COCO-InstanceSegmentation/mask_rcnn_R_50_FPN_3x.yaml")
```

Código 2. Ejemplo de Instanciación de un modelo de Detectron2.

Más adelante se definen los parámetros a modificar tanto de la arquitectura del modelo como el comportamiento del entrenamiento o de los datos de origen. En el *Código 3* se presenta un ejemplo de algunos parámetros de configuración.

```
cfg.SOLVER.BASE_LR = 0.001
cfg.SOLVER.MAX_ITER = 5000
cfg.SOLVER.IMS_PER_BATCH = 2
cfg.SOLVER.STEPS = [2000,]
cfg.INPUT.MIN_SIZE_TRAIN = 512 # Redimensionamiento de las imágenes de entrenamiento
cfg.INPUT.MAX_SIZE_TRAIN = 1024
cfg.MODEL.ROI_HEADS.NUM_CLASSES = 1  # Solo hay una clase, MS Lesión
cfg.INPUT.MIN_SIZE_TRAIN_SAMPLING = "choice"
cfg.INPUT.RANDOM_FLIP = “vertical”
cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.7
```

Código 3. Configuración de entrenamiento de Detectron2.

A partir de la configuración inicial establecida, es posible definir el flujo general completo del proceso de entrenamiento del modelo. En primer, lugar, se procede al registro y preparación del conjunto de datos junto con sus anotaciones correspondientes, asegurando que estén disponibles en un formato compatible con el framework del estudio. Posteriormente, se instancia y configura el modelo de Detectron2 elegido en función de los parámetros previamente definidos, modificando así la arquitectura seleccionada y los hiperparámetros relevantes para la tarea de segmentación en el entrenamiento. Una vez establecida esta configuración, se inicia la fase de entrenamiento, donde el modelo ajusta sus pesos durante las iteraciones estipuladas buscando la minimización de la función de pérdida y mejorando la capacidad de éste en la segmentación de lesiones.

Finalizado el entrenamiento, los nuevos pesos aprendidos se emplean en la fase de inferencia, en la cual el modelo genera predicciones sobre el conjunto de validación. Dichas predicciones son posteriormente evaluadas mediante métricas calculadas por los evaluadores *COCOEvaluator* y *F1Evaluator* que en la siguiente sección se explicarán en más detalle. En el

Código 4 se presenta en pseudocódigo que sintetiza las distintas fases descritas. Sin embargo, la implementación detallada de este flujo puede consultarse tanto en los archivos de código fuente entregados como en el repositorio de *GitHub*[1] asociado a este trabajo.

```
def flujo_entrenamiento(config_eedd):
    register_coco_instances(config_eedd.trainData, config_eedd.valData)
    cfg = inicializarDetectron2(config_eedd.model, config_eedd.paramsModel)
    trainer = DefaultTrainer(cfg)
    trainer.entrenar()
    predictor = DefaultPredictor(cfg)
    inferencia(predictor)
    resultados = evaluacion(COCOEvaluator(cfg), F1Evaluator(cfg))
    guardar_resultados(resultados)
```

Código 4. Flujo General de Entrenamiento.

Tras unas pruebas comprobando su funcionamiento con los diversos modelos, se obtienen los primeros resultados que segmentan y detectan las lesiones como se puede ver en las *Figura 10* y *Figura 11*. Aunque no es del todo preciso, puesto que en otras ocasiones detecta lesiones donde originalmente no hay, o donde sí hay zona lesionada no es capaz de segmentarla. Por ello, en las secciones a continuación se estudia cómo implementar las técnicas de optimización y mejorar estas predicciones.

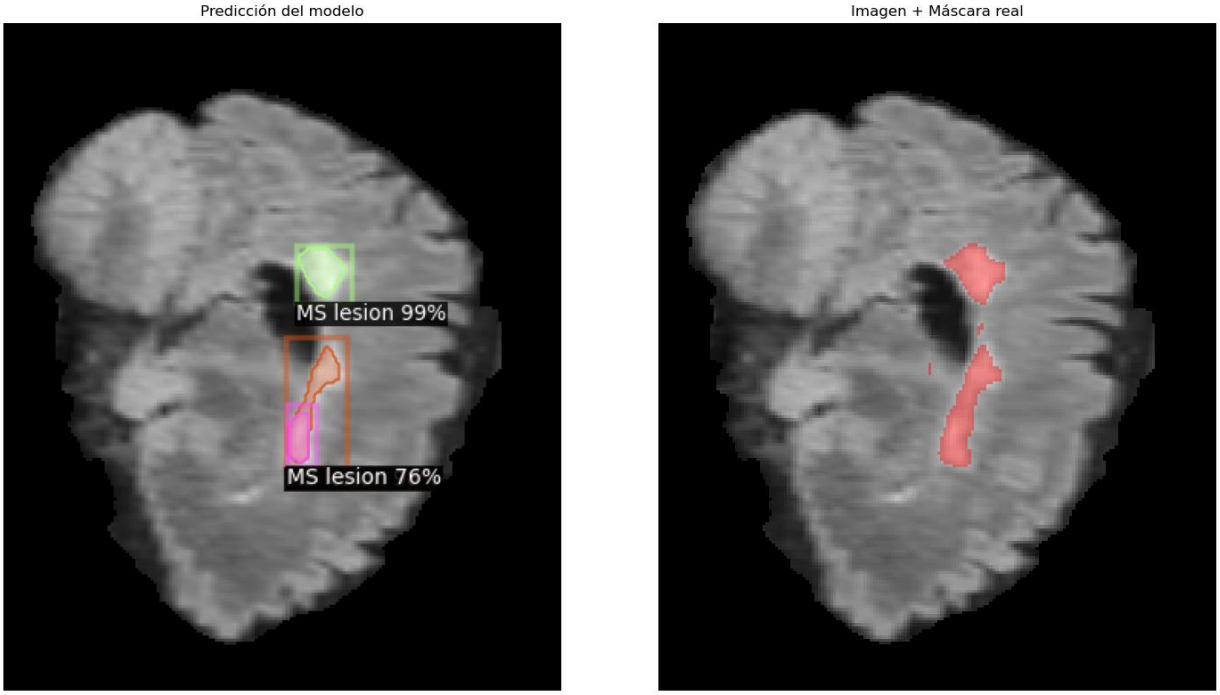

Figura 10. Ejemplo de Segmentación tras entrenamiento con modelo C4-101-3x.

[1] *https://github.com/acl00111/TFM-Scripts*

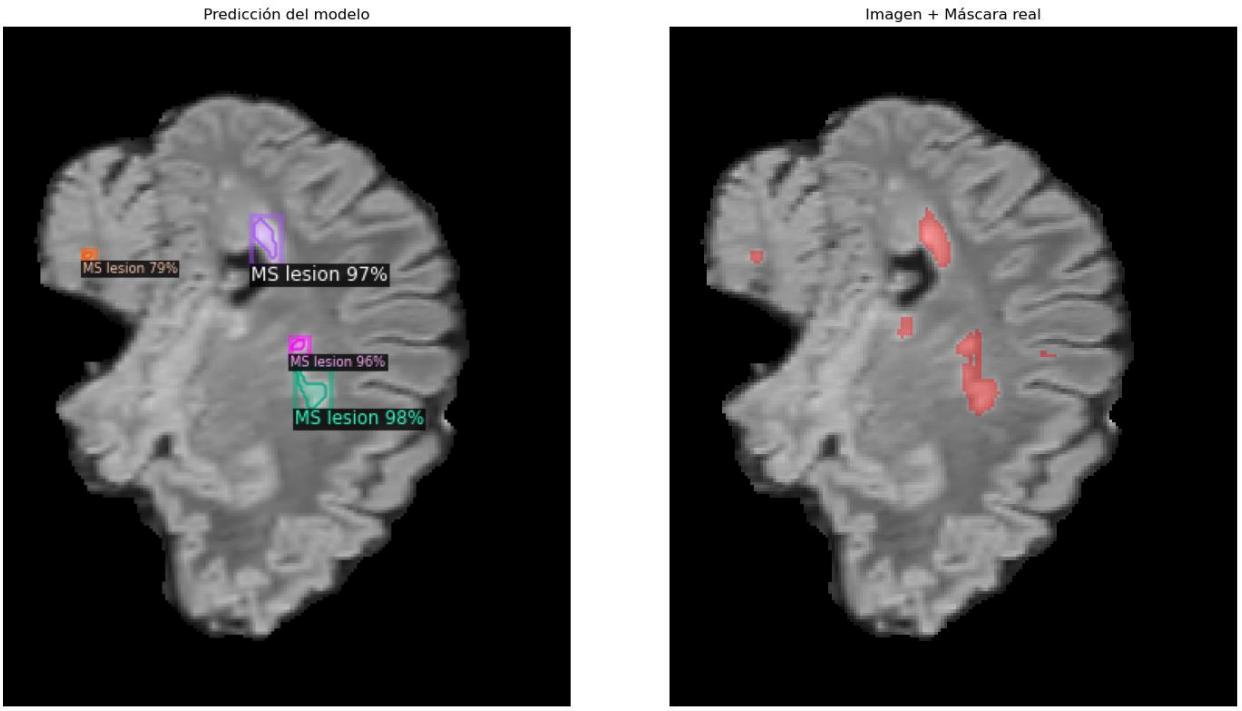

Figura 11. Ejemplo de Segmentación tras entrenamiento con modelo DC5-101-3x.

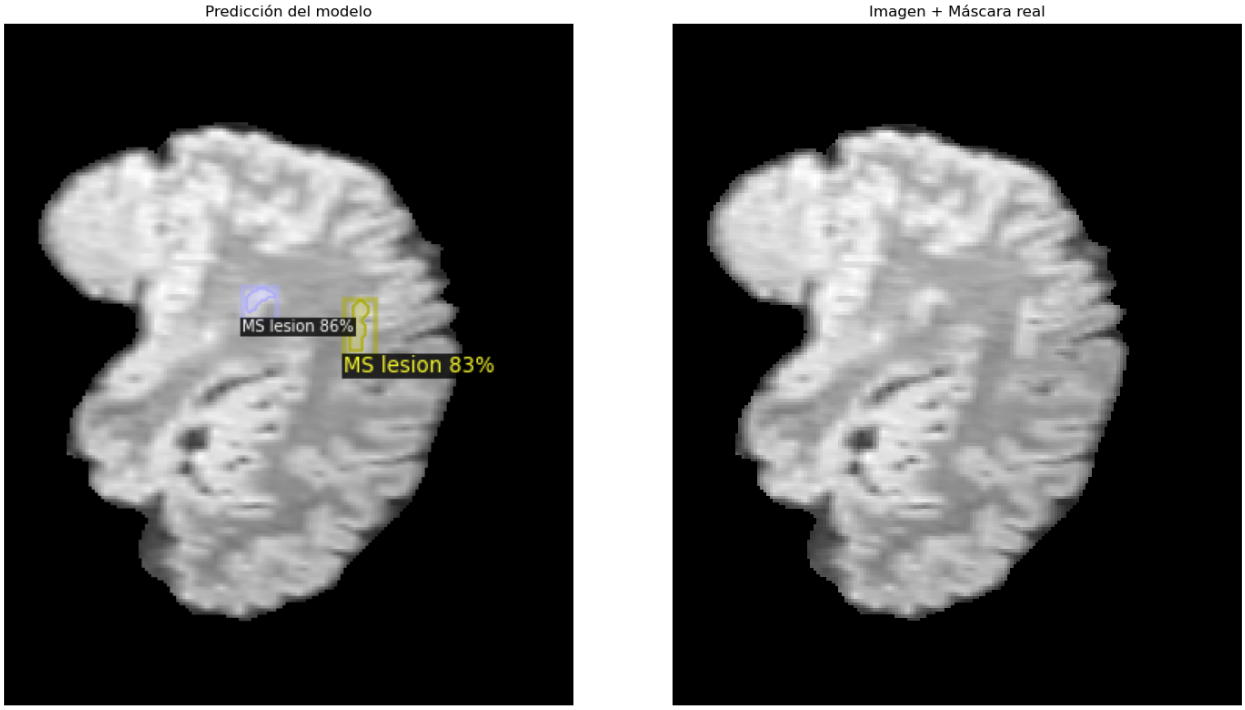

Figura 12. Ejemplo de Segmentación tras entrenamiento con modelo FPN-101-3x.

4.4 Optimización “Fine-Tuning”

Antes de abordar en detalle las técnicas de optimización implementadas en este estudio, es necesario señalar las principales limitaciones que condicionaron el proceso de ajuste de los modelos. Estas restricciones han tenido un impacto directo en la selección y variabilidad de los hiperparámetros explorados, lo que en consecuencia limitó el alcance del *fine-tuning* realizado.

La limitación más significativa estuvo relacionada con la capacidad de cómputo disponible, en particular, con la memoria de las unidades de procesamiento gráfico GPU. Debido al elevado consumo de recursos asociado al entrenamiento de los modelos de segmentación de imágenes, se alcanzó rápidamente en varios casos los máximos de memoria permitidos por el hardware. Este hecho supuso un condicionante en la elección del tamaño y resolución de las imágenes procesadas, en la cantidad de ejemplos incluidos en cada lote de las iteraciones (*batch-size*), la complejidad o densidad de capas de los modelos pre-entrenados y, de manera indirecta, en el tiempo necesario para completar una ejecución completa de entrenamiento.

Como consecuencia, determinadas configuraciones resultaron inviables desde el punto de vista computacional, lo que ha obligado a restringir el rango de valores explorados en algunos hiperparámetros y a descartar configuraciones que podrían haber ofrecido un mayor potencial en las tareas de segmentación. Aunque con estas limitaciones se ha podido llevar a cabo un estudio de optimización, era necesario señalar que la búsqueda intensiva en el espacio de parámetros estuvo limitada por estas condiciones y, por ende, afecta a la interpretación y comparación futura de los resultados.

El proceso de optimización comienza estipulando los parámetros de configuración y los rangos de valores que estos pueden adoptar. En esta primera fase se prueba con la técnica de búsqueda aleatoria o *Random Search*, en la cual nos cerca dichos valores al ejecutar configuraciones. De esta forma, se reduce el espacio combinatorio haciendo más factible encontrar un óptimo local en menos tiempo.

Para la implementación práctica, los parámetros y sus valores candidatos se definieron en un archivo de configuración en formato ***.yaml***. Este archivo actuará como repositorio central de las opciones de entrenamiento para el resto de las técnicas implementadas, puesto que facilita la modificación y flexibilidad del rango de búsqueda. Una vez cargado el archivo, el código genera todas las posibles combinaciones derivadas de los valores definidos, a partir de las cuales, en este primer paso, el algoritmo de búsqueda aleatoria selecciona al azar un subconjunto representativo que será entrenado.

Los parámetros escogidos para la optimización se pueden observar en el *Código 5* que ejemplifica la representación de éstos en un archivo de configuración.

```
param_space:
  modalidad:    [FLAIR, AUG]
  modelo:       [mask_rcnn_R_50_FPN_3x.yaml,
                 mask_rcnn_R_101_FPN_3x.yaml,
                 mask_rcnn_R_50_C4_3x.yaml,
                 mask_rcnn_R_101_C4_3x.yaml,
                 mask_rcnn_R_50_DC5_3x.yaml,
                 mask_rcnn_R_101_DC5_3x.yaml,
```

```
                mask_rcnn_X_101_32x8d_FPN_3x.yaml,]
maxiter_steps:
              - { maxiter: 2000,  steps: [1000] }
              - { maxiter: 5000,  steps: [2500] }
              - { maxiter: 10000, steps: [5000] }
              - { maxiter: 15000, steps: [7500] }
min_size:       [182, 364, 512, 1024]
base_lr:        [0.001]
gamma:          [0.5]
batch_size:     [2, 4, 6, 8]
flip:           [horizontal, vertical, none]
weight_decay:   [0.0001]
```

Código 5. Ejemplo de archivo de configuración para las técnicas de optimización.

Estos parámetros, aun siendo unos pocos en comparación a la gran personalización que proporciona Detectron2 en su documentación, resulta en un espacio muy amplio de combinaciones y que en la extensión de este estudio pueden ser recogidas. En primer lugar, el parámetro común entre todos los modelos de aprendizaje profundo son las épocas, en este caso las iteraciones y pasos intermedios (*maxiter* y *steps*). Éstos regulan la duración y estabilidad del entrenamiento, de tal forma que se debe encontrar el punto entre no subestimar con valores bajos que puedan limitar el aprendizaje, ni tampoco aumentar los valores induciendo a un posible sobreajuste y un elevado coste computacional. Del mismo modo, el parámetro *min_size* define la resolución mínima de entrada de las imágenes y Detectron2 se encarga de aplicar el redimensionamiento de forma interna. De esta forma, se busca equilibrar el detalle anatómico de posibles lesiones pequeñas y la carga de memoria que puede suponer aumentar el tamaño de las imágenes.

Otros parámetros que se han considerado son la tasa de aprendizaje y su decaimiento controlado por gamma. Ambos inciden directamente en la velocidad y estabilidad de la convergencia. El tamaño del lote o *batch size*, afecta en el entrenamiento dado que si se definen lotes reducidos puede aportar mayor variabilidad y generalización suponiendo un mayor tiempo de cómputo. Mientras que con lotes grandes se puede estabilizar el aprendizaje, pero se requieren más recursos de memoria. La regularización se realiza a través del parámetro *weight decay*, el cual previene el sobreajuste penalizando aquellos valores excesivos en los pasos. Por último, otra técnica interna de Detectron2 es *flip*, mediante el cual se aplica *data augmentation* sobre el conjunto de datos con volteos horizontales o verticales.

Tras la fase de búsqueda aleatoria se comprobó que determinadas configuraciones de modelos con ciertos valores de *batch size* y *min size* excedían los límites de la memoria disponible en la GPU. Esta limitación estuvo asociada tanto al elevado tamaño de algunos modelos, caracterizados por una mayor profundidad en sus capas, como a las demandas de recursos computacionales impuestas por dichos parámetros. Asimismo, estas configuraciones conllevaban tiempos de entrenamiento considerablemente superiores en comparación con otros modelos, lo que restringió su viabilidad práctica en el trabajo. Dichos modelos en cuestión son los que tienen como arquitectura base C4, DC5 y en particular, X101-FPN. Por lo cual, se decidió tener un archivo de configuración con un rango menor de valores para estos modelos, y otro con valores más altos para los modelos con arquitectura FPN.

La siguiente fase del proceso de optimización consistió en la aplicación de principios propios de las metaheurísticas, en concreto de los algoritmos genéticos, con el objetivo de identificar configuraciones óptimas locales dentro del espacio de búsqueda de combinaciones de hiperparámetros. Por tanto, fue necesaria la adaptación de las particularidades del problema de optimización a la naturaleza del algoritmo evolutivo.

En lo relativo a la representación de los individuos, se mantuvo la estructura de datos para los parámetros previamente empleada en la fase de búsqueda aleatoria, donde cada individuo corresponde a una configuración concreta de parámetros. A su vez, a cada uno se le asignó una medida de aptitud o *fitness*, una evaluación de su material genético y su desempeño ante el problema, y que cuya función de evaluación es sencillamente la aplicación de dichos parámetros en el flujo general de entrenamiento. En este estudio, el *fitness* se evaluó exclusivamente mediante la **métrica F1-Score**, priorizando así la precisión en la segmentación de lesiones frente a otros posibles criterios de optimización o la conjunción de éstos, como la eficiencia computacional o el tiempo de entrenamiento.

Respecto a los operadores propios del algoritmo genético se seleccionaron aquellos que ofrecían una mejor adecuación al problema y la representación establecida de los individuos. La selección de los progenitores se realizó mediante torneo, asegurando un equilibrio entre exploración y explotación del espacio de búsqueda. El cruce se implementó de forma aleatoria, combinando los valores de los parámetros de ambos padres para generar nuevas configuraciones, favoreciendo así la diversidad genética de la población. Por otra parte, las mutaciones se aplicaron respetando los rangos de valores definidos en los archivos de configuración, evitando de esta manera combinaciones inviables. Finalmente, en la fase de reemplazo se optó por una estrategia de carácter estacionario, en la que los nuevos individuos sustituyen aleatoriamente a miembros de un subconjunto reducido de peores individuos de la población, buscando un equilibrio entre preservación de soluciones prometedoras y la exploración de nuevas alternativas. En el siguiente *Algoritmo 4* se muestra de forma resumida el comportamiento del algoritmo evolutivo y sus operadores personalizados para nuestro problema.

```
def inicializar_poblacion(combinaciones, tamaño):
    población = []
    usados = {}

    mezclar(combinaciones)
    para cada config en combinaciones:
        longitud(población) >= tamaño
            terminar
        si config no está en usados:
            población.añadir(Individuo(config))
            usados[hash(config)] = None

    mientras longitud(población) < tamaño:
        config = elegir_aleatorio(combinaciones)
        si config no está en usados:
            población.añadir(Individuo(config))
            usados[hash(config)] = None

    devolver población, usados
```

```
def evaluar_población(población, usados):
    para cada individuo en población:
        individuo.fitness = ejecutar_entrenamiento(individuo.config)
        usados[hash(individuo.config)] = individuo.fitness

def seleccionar_padres(población, num_padres, k=3):
    padres = []
    para i en rango(num_padres):
        torneo = elegir_k_aleatorio(población, k)
        ganador = máximo(torneo, clave=fitness)
        padres.añadir(ganador)

    devolver padres

def cruzar_padres(padres):
    hijo_config = {}
    para cada param en padres[0].config:
        hijo_config[param] = elegir_aleatorio([ padres[0].config[param], padres[1].config[param] ])

    devolver Individuo(hijo_config)

def mutar_hijo(hijo, prob=0.01):
    para cada param en hijo.config:
       si azar() < prob:
           hijo[param] = mutar_valor(param, hijo.config[param]

     devolver hijo

def reemplazo(población, hijo, num_reemplazables):
    población.añadir(hijo)
    ordenar(población, descendente=fitness)

    peores = últimos(población, num_reemplazables)
    eliminado = elegir_aleatorio(peores)
    población.quitar(eliminado)

def algoritmo_evolutivo(combinaciones, tamaño, generaciones):
     población, usados = inicializar_población(combinaciones, tamaño)
     evaluar_población(población, usados)

     para i en rango(generaciones):
         padres = seleccionar_padres(población)
         hijo = cruzar_padres(padres)
         hijo = mutar_hijo(hijo)
         hijo.fitness = ejecutar_entrenamiento(hijo.config)
         reemplazo(población, hijo, 3)

    mejor = máximo(población, clave=fitness)
    devolver mejor
```

Algoritmo 4. Pseudocódigo del Algoritmo Genético y operadores para la optimización de hiperparámetros de Detectron2.

Después de la implementación del algoritmo genético y de la obtención de las configuraciones más prometedoras a partir de las evaluaciones, se procedió a entrenarlas nuevamente utilizando un conjunto de datos enriquecido mediante técnicas de aumento de datos. En concreto, se aplicó la rotación de las imágenes en 90º, 180º y 270º, lo que permitió multiplicar significativamente el volumen de anotaciones disponibles para el proceso de entrenamiento. De este modo, el conjunto inicial de 16.744 imágenes se amplió hasta un total de 66.976, incrementando tres veces la cantidad de datos de la que se partía. Esto supone el favorecimiento de la diversidad de patrones a los que se expone el modelo y su capacidad de generalización a la hora de segmentar las lesiones.

Finalmente, a partir de aquellas configuraciones que mostraron un mejor desempeño tanto con el conjunto de datos original como en el conjunto enriquecido, se implementó un procedimiento de validación cruzada con el objetivo de obtener resultados más sólidos y representativos. Esta técnica permitió evaluar la estabilidad de los modelos frente a diferentes particiones del conjunto de datos, proporcionando una estimación más fiable y robusta de su rendimiento ante imágenes no vistas anteriormente.

5. Análisis Resultados

5.1 Métricas de Evaluación

Las métricas de evaluación constituyen un elemento esencial en la valoración del rendimiento y la eficacia de los modelos de aprendizaje automático y profundo. Su función radica en proporcionar información cuantitativa que permite medir la calidad de un modelo, establecer comparaciones entre diferentes arquitecturas o algoritmos y, en consecuencia, orientar la toma de decisiones en cuanto a ajustes y mejoras. De este modo, facilitan discernir qué modelo resulta más adecuado y eficiente para abordar la tarea planteada. En el contexto específico de la segmentación, dichas métricas se calculan a partir de la clasificación y segmentación de cada píxel por parte del modelo.

Para este trabajo se han seleccionado como métricas principales el **F1 Score**, la **Precisión** (*precision*) y la **Sensibilidad** (*recall*), aunque también se han considerado otras medidas complementarias como la **Intersección sobre Unión** (*IoU*) y la precisión media promedio **mAP50** (*mean average precision IoU=0.5*). No obstante, antes de proceder con la descripción de los cálculos de cada una de estas métricas, es necesario introducir una serie de conceptos fundamentales:

- **Verdaderos Positivos (TP):** corresponde al número de instancias que el modelo clasifica correctamente como pertenecientes a la clase interés. Por ejemplo, cuando el sistema detecta una lesión y efectivamente existe en la imagen, se contabiliza un verdadero positivo.
- **Verdaderos Negativos (TN):** son las instancias correctamente clasificadas como no pertenecientes a la clase de interés. Por ejemplo, cuando la red no identifica ninguna lesión en un área donde existe, se considera un verdadero negativo.
- **Falsos Positivos (FP):** representan las instancias que el modelo clasifica incorrectamente como pertenecientes a la clase de interés. Por ejemplo, si se detecta una lesión en una región donde no hay patología, se cuenta como un falso positivo.
- **Falsos Negativos (FN):** se refieren a las instancias de la clase objetivo que no son identificadas correctamente. Por ejemplo, si el modelo no detecta una lesión presente en la imagen, se registra como un falso negativo.

Estos conceptos suelen representarse mediante una matriz de confusión, la cual ofrece una visión global del comportamiento del modelo en términos de aciertos y errores, como se ilustra en la *Figura 13*.

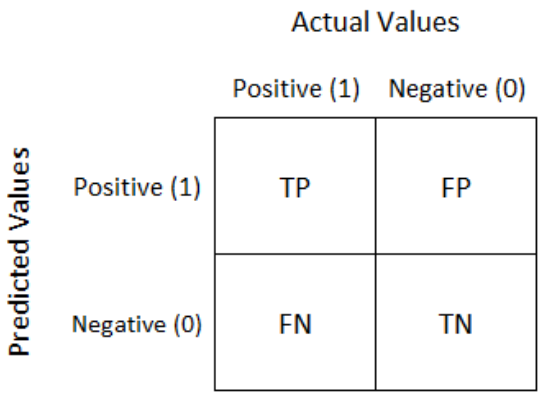

Figura 13. Esquema de matriz de confusión. [33]

Una vez introducidos los conceptos fundamentales, resulta posible comprender con mayor claridad el modo en que se calculan las métricas de evaluación empleadas:

- **F1 Score:** Evalúa el rendimiento de los modelos en contextos con clases desbalanceadas. En nuestro caso es de gran utilidad, dado que la proporción entre la clase a segmentar, "lesión MS", es mucho menos frecuente que el fondo, o la "no lesión" en las imágenes. La *Ecuación 1* expresa el cálculo de esta medida, definida por la media armónica entre la precisión y la recuperación. El F1 Score es especialmente valioso porque busca un equilibrio entre ambos aspectos [34].

$$F1 = 2 \times \frac{precisión \times recuperación}{precisión + recuperación}$$

Ecuación 1. Cálculo del F1 Score. [34]

- **Precisión:** Esta métrica cuantifica la proporción de predicciones correctas dentro del total de detecciones realizadas por el modelo. Un valor elevado de esta métrica implica que el sistema comete pocos errores al clasificar regiones como positivas. Matemáticamente, su definición se resume en la *Ecuación 2*.

$$Precisión = \frac{True\ Positives}{True\ Positives + False\ Positives}$$

Ecuación 2. Cálculo de la precisión. [35]

- **Recall:** El recall evalúa la capacidad del modelo para identificar correctamente las instancias positivas presentes en el conjunto de datos. Un valor alto indica que la red logra detectar la mayor paprte de las lesiones presentes en las imágenes. Su expresión matemática se recoge en la *Ecuación 3*.

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

Ecuación 3. Cálculo del recall. [35]

Análisis comparativo entre resultados. Cuantitativo con las métricas y también cualitativo con fotos de las inferencias hechas por los modelos.

6. Conclusión

Bibliografía

[1] «MSLesSeg 2024». Accedido: 12 de agosto de 2025. [En línea]. Disponible en: https://iplab.dmi.unict.it/mfs/ms-les-seg/

[2] A. Voulodimos, N. Doulamis, A. Doulamis, y E. Protopapadakis, «Deep Learning for Computer Vision: A Brief Review», *Comput. Intell. Neurosci.*, vol. 2018, n.º 1, p. 7068349, 2018, doi: 10.1155/2018/7068349.

[3] M. A. Ponti, L. S. F. Ribeiro, T. S. Nazare, T. Bui, y J. Collomosse, «Everything You Wanted to Know about Deep Learning for Computer Vision but Were Afraid to Ask», en *2017 30th SIBGRAPI Conference on Graphics, Patterns and Images Tutorials (SIBGRAPI-T)*, oct. 2017, pp. 17-41. doi: 10.1109/SIBGRAPI-T.2017.12.

[4] O. Ronneberger, P. Fischer, y T. Brox, «U-Net: Convolutional Networks for Biomedical Image Segmentation», 18 de mayo de 2015, *arXiv*: arXiv:1505.04597. doi: 10.48550/arXiv.1505.04597.

[5] J. Long, E. Shelhamer, y T. Darrell, «Fully Convolutional Networks for Semantic Segmentation», 8 de marzo de 2015, *arXiv*: arXiv:1411.4038. doi: 10.48550/arXiv.1411.4038.

[6] K. He, X. Zhang, S. Ren, y J. Sun, «Deep Residual Learning for Image Recognition», en *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, jun. 2016, pp. 770-778. doi: 10.1109/CVPR.2016.90.

[7] D. Maji, P. Sigedar, y M. Singh, «Attention Res-UNet with Guided Decoder for semantic segmentation of brain tumors», *Biomed. Signal Process. Control*, vol. 71, p. 103077, ene. 2022, doi: 10.1016/j.bspc.2021.103077.

[8] Y. Wang, W. Zhang, R. Gao, Z. Jin, y X. Wang, «Recent advances in the application of deep learning methods to forestry», *Wood Sci. Technol.*, vol. 55, sep. 2021, doi: 10.1007/s00226-021-01309-2.

[9] K. He, G. Gkioxari, P. Dollar, y R. Girshick, «Mask R-CNN», presentado en Proceedings of the IEEE International Conference on Computer Vision, 2017, pp. 2961-2969. Accedido: 15 de junio de 2024. [En línea]. Disponible en: https://openaccess.thecvf.com/content_iccv_2017/html/He_Mask_R-CNN_ICCV_2017_paper.html

[10] S. Ren, K. He, R. Girshick, y J. Sun, «Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks», *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, n.º 6, pp. 1137-1149, jun. 2017, doi: 10.1109/TPAMI.2016.2577031.

[11] K. He, X. Zhang, S. Ren, y J. Sun, «Deep Residual Learning for Image Recognition», presentado en Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770-778. Accedido: 15 de junio de 2024. [En línea]. Disponible en: https://openaccess.thecvf.com/content_cvpr_2016/html/He_Deep_Residual_Learning_CVPR_2016_paper.html

[12] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, y S. Belongie, «Feature Pyramid Networks for Object Detection», 19 de abril de 2017, *arXiv*: arXiv:1612.03144. doi: 10.48550/arXiv.1612.03144.

[13] J. Redmon, S. Divvala, R. Girshick, y A. Farhadi, «You Only Look Once: Unified, Real-Time Object Detection», en *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA: IEEE, jun. 2016, pp. 779-788. doi: 10.1109/CVPR.2016.91.

[14] *facebookresearch/detectron2*. (19 de junio de 2024). Python. Meta Research. Accedido: 19 de junio de 2024. [En línea]. Disponible en: https://github.com/facebookresearch/detectron2

[15] M. Ackermann, D. İren, S. Wesselmecking, D. Shetty, y U. Krupp, «Automated segmentation of martensite-austenite islands in bainitic steel», *Mater. Charact.*, vol. 191, p. 112091, jul. 2022, doi: 10.1016/j.matchar.2022.112091.

[16] F. Zhuang *et al.*, «A Comprehensive Survey on Transfer Learning», 23 de junio de 2020, *arXiv*: arXiv:1911.02685. doi: 10.48550/arXiv.1911.02685.

[17] A. Farahani, B. Pourshojae, K. Rasheed, y H. R. Arabnia, «A Concise Review of Transfer Learning», 5 de abril de 2021, *arXiv*: arXiv:2104.02144. doi: 10.48550/arXiv.2104.02144.

[18] «Data augmentation», *Wikipedia*. 19 de julio de 2025. Accedido: 26 de agosto de 2025. [En línea]. Disponible en: https://en.wikipedia.org/w/index.php?title=Data_augmentation&oldid=1301343127
[19] «Cross-validation (statistics)», *Wikipedia*. 9 de agosto de 2025. Accedido: 11 de agosto de 2025. [En línea]. Disponible en: https://en.wikipedia.org/w/index.php?title=Cross-validation_(statistics)&oldid=1305062291
[20] B. Yang *et al.*, «Comprehensive overview of maximum power point tracking algorithms of PV systems under partial shading condition», *J. Clean. Prod.*, vol. 268, p. 121983, sep. 2020, doi: 10.1016/j.jclepro.2020.121983.
[21] S. Katoch, S. S. Chauhan, y V. Kumar, «A review on genetic algorithm: past, present, and future», *Multimed. Tools Appl.*, vol. 80, n.º 5, pp. 8091-8126, feb. 2021, doi: 10.1007/s11042-020-10139-6.
[22] M. A. K. Raiaan *et al.*, «A systematic review of hyperparameter optimization techniques in Convolutional Neural Networks», *Decis. Anal. J.*, vol. 11, p. 100470, jun. 2024, doi: 10.1016/j.dajour.2024.100470.
[23] «MRI Basics». Accedido: 13 de agosto de 2025. [En línea]. Disponible en: https://case.edu/med/neurology/NR/MRI%20Basics.htm
[24] «Magnetic Resonance Imaging (MRI)», National Institute of Biomedical Imaging and Bioengineering. Accedido: 13 de agosto de 2025. [En línea]. Disponible en: https://www.nibib.nih.gov/science-education/science-topics/magnetic-resonance-imaging-mri
[25] «Mechanism of magnetic resonance imaging. (A) Hydrogen atoms are...», ResearchGate. Accedido: 14 de agosto de 2025. [En línea]. Disponible en: https://www.researchgate.net/figure/Mechanism-of-magnetic-resonance-imaging-A-Hydrogen-atoms-are-dispersed-within-the_fig2_352367301
[26] M. T. Niknejad, «Fluid attenuated inversion recovery | Radiology Reference Article | Radiopaedia.org», Radiopaedia. Accedido: 13 de agosto de 2025. [En línea]. Disponible en: https://radiopaedia.org/articles/fluid-attenuated-inversion-recovery
[27] «Magnetic resonance imaging», *Wikipedia*. 18 de julio de 2025. Accedido: 14 de agosto de 2025. [En línea]. Disponible en: https://en.wikipedia.org/w/index.php?title=Magnetic_resonance_imaging&oldid=1301121182
[28] F. Gaillard, «MRI sequences (overview) | Radiology Reference Article | Radiopaedia.org», Radiopaedia. Accedido: 13 de agosto de 2025. [En línea]. Disponible en: https://radiopaedia.org/articles/mri-sequences-overview
[29] «What Do MS Lesions on a Brain MRI Indicate?», Biology Insights. Accedido: 14 de agosto de 2025. [En línea]. Disponible en: https://biologyinsights.com/what-do-ms-lesions-on-a-brain-mri-indicate/
[30] O. I. Alomair, «Conventional and Advanced Magnetic Resonance Imaging Biomarkers of Multiple Sclerosis in the Brain», *Cureus*, vol. 17, n.º 3, p. e79914, doi: 10.7759/cureus.79914.
[31] «Multiple sclerosis(MS) MRI | Radiology Article on Multiple sclerosis», mrimaster. Accedido: 15 de agosto de 2025. [En línea]. Disponible en: https://mrimaster.com/multiple-sclerosisms-mri/
[32] «COCO - Common Objects in Context». Accedido: 22 de junio de 2024. [En línea]. Disponible en: https://cocodataset.org/#format-data
[33] F. Izco, *Base de datos corporativa de personas*. Accedido: 25 de junio de 2024. [En línea]. Disponible en: https://bookdown.org/f_izco/BDC-POC/metricas.html
[34] «F-score», *Wikipedia*. 19 de junio de 2025. Accedido: 11 de agosto de 2025. [En línea]. Disponible en: https://en.wikipedia.org/w/index.php?title=F-score&oldid=1296408310
[35] A. L. Batallas, J. Bermeo Paucar, J. Paredes Quevedo, y H. Torres Ordoñez, «Una revisión de las métricas aplicadas en el procesamiento de imágenes», *RECIMUNDO Rev. Científica Investig. El Conoc.*, vol. 4, n.º 3, pp. 267-273, 2020.