UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

Graduado en Ingeniería del Software

Desarrollo de un sistema de asistencia al diagnóstico de tumores cerebrales basado en inteligencia artificial

Development of a brain tumor diagnosis assistance system based on artificial intelligence

Realizado por

Pablo Barranco Céspedes

Tutorizado por

Jamal Toutouh El Alamin

Departamento

Lenguajes y Ciencias de la Computación

MÁLAGA, junio de 2025

UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

GRADUADO EN INGENIERÍA DEL SOFTWARE

Desarrollo de un sistema de asistencia al diagnóstico de tumores cerebrales basado en inteligencia artificial

Segmentación de imágenes empleando redes neuronales para la detección de tumores cerebrales

Development of a brain tumor diagnosis assistance system based on artificial intelligence

Image segmentation using neural networks for brain tumors detection

Realizado por
Pablo Barranco Céspedes

Tutorizado por
Jamal Toutouh El Alamin

Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, JUNIO DE 2025

Fecha defensa: julio de 2025

Summary

This Final Degree Project (TFG) focuses on the development of an artificial intelligence (AI) system to assist in the diagnosis of brain tumors. Specifically, it explores the application of deep neural networks for the segmentation of magnetic resonance imaging (MRI) scans to accurately identify and delineate tumor regions. This work is part of a group project aimed at optimizing anomaly detection in medical imaging.

The main objective of this project is the analysis and evaluation of convolutional neural network (CNN)-based models, such as the U-Net architecture, for precise brain tumor segmentation. To achieve this, open-access datasets such as Brain MRI Segmentation and Brain MRI Images from Kaggle will be utilized. Various image processing and deep learning methodologies will be implemented, assessing their accuracy and efficiency in tumor detection.

Additionally, a web-based interface will be developed to allow users to interact with the segmentation models, visualize results, and facilitate the interpretation of the obtained data. The project will employ Python, TensorFlow, PyTorch, OpenCV, and FastAPI for model implementation, along with JavaScript and HTML/CSS for the web platform.

The ultimate goal is to provide an efficient tool that can support medical diagnosis by enhancing accuracy and reducing the time required for the identification of brain tumors in MRI scans.

Keywords:

- Artificial Intelligence
- Image segmentation
- Brain tumors
- Neural Networks
- Magnetic Resonance Image (MRI)

Resumen

El presente Trabajo de Fin de Grado (TFG) se centra en el desarrollo de un sistema de asistencia al diagnóstico de tumores cerebrales mediante el uso de inteligencia artificial (IA). En concreto, se explorará la aplicación de redes neuronales profundas para la segmentación de imágenes de resonancia magnética (MRI) con el objetivo de identificar y delimitar áreas tumorales. Este trabajo forma parte de un proyecto en grupo que busca optimizar la detección de anomalías en imágenes médicas.

El enfoque principal del TFG es el análisis y evaluación de modelos basados en redes neuronales convolucionales, como la arquitectura U-Net, para la segmentación precisa de tumores cerebrales. Para ello, se utilizarán bases de datos de acceso abierto, como Brain MRI Segmentation y Brain MRI Images, disponibles en Kaggle. Se implementarán diversas metodologías de procesamiento de imágenes y aprendizaje profundo, evaluando su precisión y eficiencia en la detección de tumores.

Además, como parte del desarrollo, se creará una interfaz web que permitirá a los usuarios interactuar con los modelos de segmentación, visualizar los resultados y facilitar la interpretación de los datos obtenidos. El proyecto empleará tecnologías como Python, PyTorch, OpenCV y Flask, junto con JavaScript y HTML/CSS para la implementación web.

El objetivo final es proporcionar una herramienta eficiente que pueda servir como apoyo en el diagnóstico médico, mejorando la precisión y reduciendo el tiempo necesario para la identificación de tumores cerebrales en imágenes de resonancia magnética.

Palabras clave:

- Inteligencia Artificial
- Segmentación de imágenes
- Tumores cerebrales

- Redes neuronales
- Resonancia magnética

Índice

1 Introducción

1.1. Contexto y motivación del problema

En los últimos años, la Inteligencia Artificial (IA) ha transformado la medicina, especialmente en el ámbito de la medicina de precisión. Técnicas como el aprendizaje automático y profundo están facilitando el trabajo médico, especialmente en el análisis de imágenes médicas. En el diagnóstico de tumores cerebrales, la IA permite detectar, segmentar y clasificar tumores con gran precisión. Este proyecto se centrará en el uso de redes neuronales profundas aplicadas a imágenes de resonancia magnética cerebral, evaluando y optimizando su desempeño en este contexto. En este capítulo se discute la justificación del proyecto, se resume los objetivos principales y presenta la estructura del documento.

1.2. Justificación del proyecto

El análisis de imágenes de resonancia magnética (MRI) es una tarea fundamental en la detección de tumores cerebrales. Sin embargo, este proceso requiere la intervención de especialistas altamente capacitados y, en muchos casos, puede ser subjetivo y consumir mucho tiempo. Además, las variaciones en el tamaño, forma y ubicación de los tumores dificultan su segmentación precisa. Por ello, surge la necesidad de desarrollar herramientas automáticas que puedan asistir en esta tarea, reduciendo tiempos de diagnóstico y aumentando la precisión [31].

1.3. Objetivos

Este proyecto tiene como objetivo desarrollar un sistema basado en inteligencia artificial para la segmentación automática de tumores cerebrales en imágenes de resonancia magnética. Se evaluarán diferentes modelos de redes neuronales convolucionales (CNN), con especial en-

foque en arquitecturas como U-Net, que han demostrado gran efectividad en la segmentación de imágenes médicas [8, 32]. Además, se diseñará una interfaz web para facilitar la interacción con los modelos y la visualización de los resultados, permitiendo que los especialistas puedan interpretar de manera más eficiente los análisis generados por la IA.

1.4. Estructura del documento

Este documento se estructura de la siguiente manera: en el Capítulo 1 se presenta el contexto del problema y los objetivos del proyecto. El Capítulo 2 aborda el estado del arte en el uso de inteligencia artificial para la segmentación de imágenes médicas. En el Capítulo 3 se describe la metodología utilizada, incluyendo la selección del conjunto de datos, el preprocesamiento de imágenes y el diseño del modelo de segmentación. El Capítulo 4 expone el desarrollo y análisis de los resultados. El Capítulo 5 presenta la interfaz web desarrollada y su integración con los modelos de Inteligencia Artificial, seguido del Capítulo 6, donde se analizan las conclusiones y futuras líneas de trabajo.

2

Estado del arte

En este capítulo se presenta una revisión de las principales tecnologías, herramientas y conceptos fundamentales utilizados a lo largo del desarrollo de este TFG. Se introduce el marco teórico imprescindible para comprender el funcionamiento de los modelos de segmentación automática, incluyendo las bases del aprendizaje profundo, las CNN y las arquitecturas específicas empleadas.

En este capítulo también se describen las librerías, entornos de desarrollo, *frameworks* y herramientas seleccionados, justificando la elección de los mismos en función de su idoneidad para tareas de análisis de imágenes médicas. Esta revisión proporciona el contexto para entender las decisiones que se adoptan en los capítulos posteriores.

2.1. Radiología y resonancia magnética en el diagnóstico de tumores cerebrales

La radiología es una especialidad médica fundamental en el diagnóstico y tratamiento de enfermedades, mediante el uso de imágenes obtenidas a través de diferentes técnicas, como los rayos X, la tomografía computarizada (TC) o la resonancia magnética (RM) [18]. Las Figuras 1, 2, 3 muestran ejemplos de imágenes del tipo MRI, TC y rayos X.

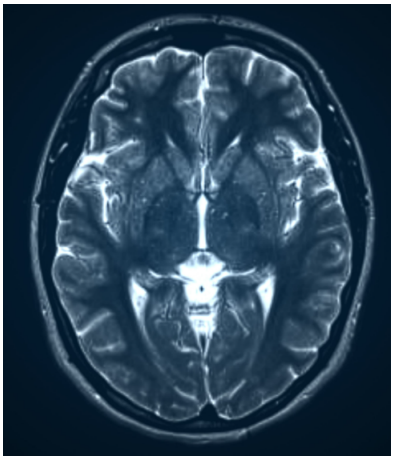

Figura 1: Imagen de ejemplo de MRI [3]

Figura 2: Ejemplo de imagen de TC [22]

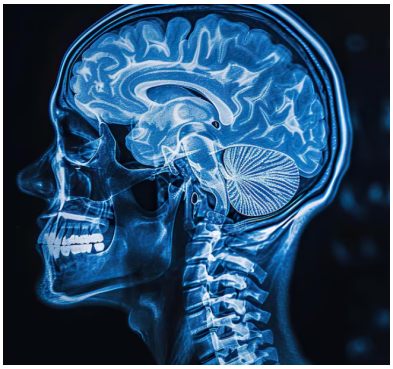

Figura 3: Ejemplo de imagen de Rayos X [11]

En el contexto del diagnóstico de tumores cerebrales, **la resonancia magnética** se ha consolidado como una de las herramientas más precisas y no invasivas, debido a su capacidad para generar imágenes detalladas del tejido cerebral sin emplear radiación ionizante [18].

La RM utiliza **campos magnéticos** y **ondas de radio** para producir imágenes tridimensionales del cerebro, lo que permite visualizar con claridad estructuras internas y posibles anomalías. A diferencia de otras técnicas, como la TC, la resonancia magnética ofrece un mayor contraste entre los diferentes tipos de tejidos blandos, lo que resulta especialmente útil para identificar masas tumorales, edema cerebral o desplazamientos de estructuras anatómicas.

En el caso de los tumores cerebrales, la RM permite no solo detectar la presencia del tumor, sino también estimar su **tamaño**, **forma**, **localización** y su relación con tejidos adyacentes. Además, gracias a secuencias específicas como **FLAIR**, **T1 con contraste** o **T2**, se pueden obtener distintos tipos de información que ayudan al especialista a realizar un diagnóstico más preciso y a planificar el tratamiento más adecuado, ya sea quirúrgico, farmacológico o por radioterapia [18].

En los últimos años, el avance de la tecnología en radiología ha permitido el desarrollo de imágenes funcionales, como la **RM funcional (fMRI)** o la **espectroscopía por RM**, que

ofrecen información adicional sobre la actividad metabólica del tejido cerebral. Estas herramientas pueden resultar de gran utilidad en la evaluación de la agresividad del tumor y su posible malignidad [18].

No obstante, la interpretación de las imágenes de RM sigue dependiendo de la experiencia del especialista, lo que puede introducir variabilidad en los diagnósticos. Por esta razón, se están explorando soluciones basadas en **IA** que ayuden a automatizar y mejorar la precisión de la segmentación y detección de tumores en estas imágenes, lo cual se abordará en las siguientes secciones [31].

2.2. Inteligencia Artificial en el ámbito médico

La IA ha emergido como una herramienta transformadora en numerosos sectores, y la medicina no es una excepción. En los últimos años, los avances en IA han permitido automatizar tareas complejas, mejorar la precisión diagnóstica y optimizar los tiempos de respuesta en distintos procesos clínicos [17, 31].

La IA médica se basa en algoritmos capaces de analizar grandes volúmenes de **datos clínicos**, **imágenes médicas** o **historiales electrónicos**, extrayendo patrones y relaciones útiles para la toma de decisiones. En este contexto, una de las ramas más destacadas es el **aprendizaje profundo (*Deep Learning*)**, que utiliza redes neuronales profundas para procesar datos de alta complejidad como imágenes médicas [1].

El uso de IA en medicina ha encontrado aplicaciones en campos como la **oncología**, **cardiología**, **dermatología** o **radiología**. En particular, ha demostrado gran potencial en tareas como la **clasificación de tejidos**, la **predicción de patologías**, la **segmentación de órganos** o la **detección de tumores** en MRI, TC o radiografías [25, 31].

La capacidad de los modelos de aprendizaje profundo para aprender representaciones jerárquicas directamente desde los datos ha permitido avances significativos en la **interpretación automatizada** de imágenes médicas. Esta automatización no pretende sustituir al especialista, sino ofrecer una herramienta de apoyo que aumente la eficiencia, reduzca errores humanos y permita un diagnóstico más rápido y preciso [12].

Dentro del aprendizaje profundo, los modelos de **CNN** han sido los más utilizados para la segmentación y análisis de imágenes médicas. Estas redes pueden aprender a identificar patrones visuales complejos que, en muchos casos, pueden pasar desapercibidos incluso para ojos expertos [8].

El presente proyecto se enmarca precisamente en esta línea de trabajo, utilizando técnicas de IA para la **segmentación automática de tumores cerebrales** en MRI, con el objetivo de facilitar el diagnóstico clínico y reducir la carga de trabajo de los profesionales médicos.

2.3. Redes neuronales profundas y CNNs en la segmentación de imágenes médicas

Las **CNN** han demostrado una alta eficacia en tareas de procesamiento de imágenes, siendo ampliamente utilizadas en aplicaciones de visión por computador como la **clasificación**, **detección** y **segmentación** de objetos [8, 32].

En el ámbito médico, estas redes se han convertido en una herramienta clave para la **segmentación de imágenes**, debido a su capacidad para aprender automáticamente características jerárquicas a partir de datos visuales complejos. Para visualizar una segmentación de una imagen médica, se ilustran las Figuras 4 y 5 que representan una imagen natural y una imagen con su segmentación superpuesta en color naranja.

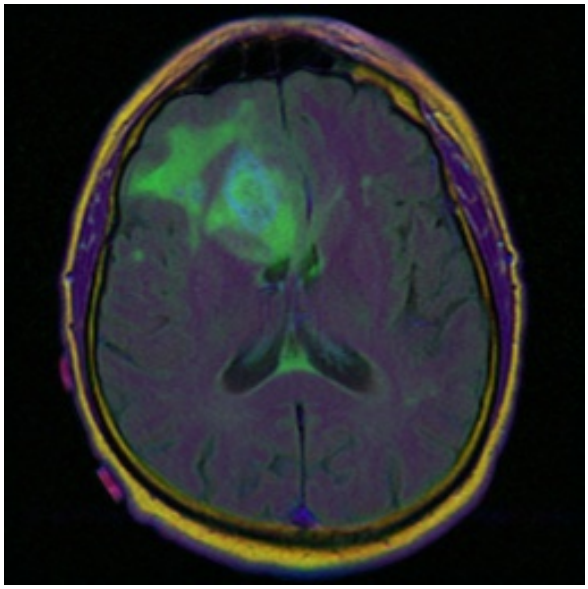

Figura 4: Ejemplo de MRI

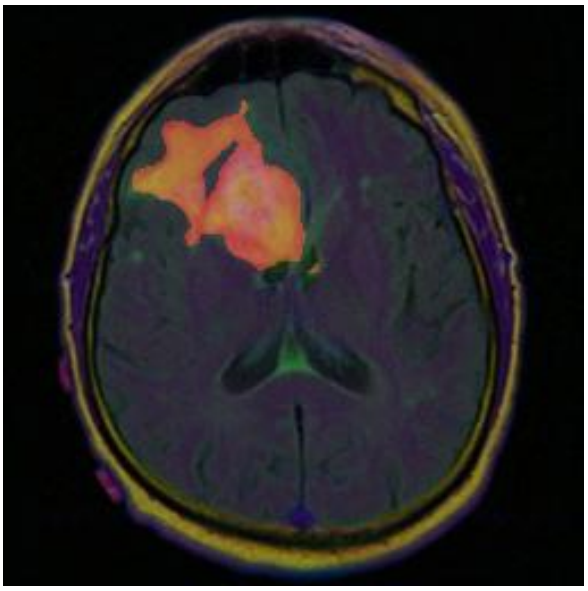

Figura 5: Ejemplo de MRI con su segmentación superpuesta

Una de las arquitecturas más representativas en segmentación médica es **U-Net**, una red neuronal convolucional diseñada específicamente para esta tarea. Su nombre proviene de la forma de su arquitectura, en forma de "U" como se muestra en la Figura 6. Consta de dos fases principales: **compresión (*encoder*)** y **expansión (*decoder*)**, conectadas mediante **rutas de salto (*skip connections*)** [32, 33].

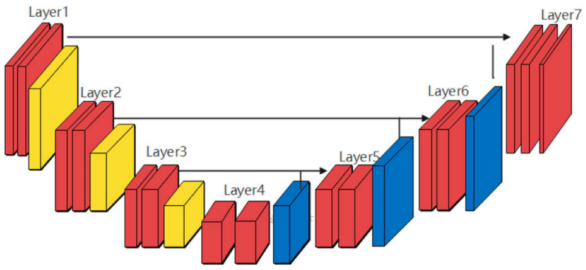

Figura 6: Visualización de una red U-Net [5].

La parte izquierda de la red reduce el tamaño de la imagen para conservar características importantes, y la parte derecha recupera el tamaño original resaltando las regiones más significativas, como los tumores. U-Net se caracteriza por su capacidad de capturar tanto el

contexto global como los **detalles locales** de las imágenes médicas.

Una extensión de esta arquitectura es **Attention U-Net**, que incorpora mecanismos de atención mediante bloques llamados ***Attention Gates* (AG)**. Estos permiten que el modelo se centre únicamente en las regiones más relevantes para la segmentación, filtrando la información irrelevante. Esta mejora resulta especialmente útil para detectar **tumores pequeños** o con **bordes poco definidos**, reduciendo los falsos positivos y mejorando la sensibilidad del modelo [26].

Por otro lado, una arquitectura más moderna utilizada en este trabajo es **DeepLabV3+**, desarrollada por Google. Este modelo de segmentación semántica hace uso de **convoluciones dilatadas** y del módulo *Atrous Spatial Pyramid Pooling (ASPP)*, que permite capturar información en múltiples escalas sin pérdida de resolución espacial. En la Figura 7 [33] se muestra el funcionamiento general del modelo DeepLabV3+.

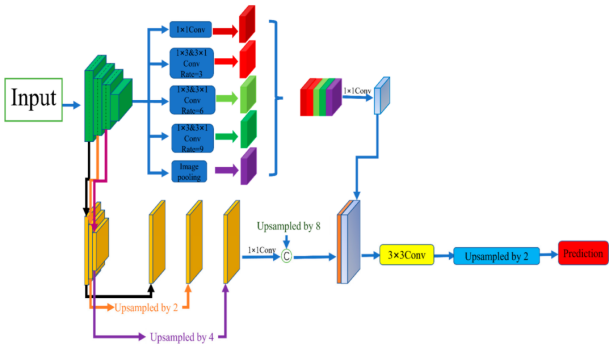

Figura 7: Visualización general de una red de DeepLabV3+. [9]

DeepLabV3+ ha sido adaptado en este proyecto con diferentes *backbones*, como **MobileNetV2** y **ResNet34**, aprovechando pesos preentrenados en conjuntos de datos generales como ImageNet mediante técnicas de *Transfer Learning*.

Estas tres arquitecturas —**U-Net**, **Attention U-Net** y **DeepLabV3+**— representan enfoques diversos dentro del campo de la segmentación automática, lo que permite realizar comparaciones de rendimiento y estudiar cuál se adapta mejor al problema específico de **detección de tumores cerebrales** en MRI.

2.4. Transfer Learning aplicado a imágenes médicas

El ***Transfer Learning*** o aprendizaje por transferencia es una técnica del aprendizaje profundo que consiste en aprovechar el conocimiento adquirido por un modelo previamente entrenado en una tarea o conjunto de datos, para aplicarlo en un nuevo problema con características similares. Esta estrategia es especialmente útil cuando se dispone de un volumen limitado de datos, como ocurre a menudo en el ámbito médico [31].

En este proyecto, se ha empleado *Transfer Learning* en la implementación de la arquitectura **DeepLabV3+**, utilizando **MobileNetV2** y **ResNet34** como *backbones* preentrenados en el conjunto de datos **ImageNet**. Esta técnica permite iniciar el entrenamiento con pesos ya optimizados para detectar patrones visuales generales, acelerando la convergencia del modelo y mejorando su capacidad para generalizar.

La integración de modelos preentrenados se ha realizado a través de la librería *segmentation_models_pytorch*, lo que ha facilitado la carga directa de arquitecturas avanzadas con sus respectivos pesos. En particular:

- **DeepLabV3+ con MobileNetV2** se ha utilizado por su eficiencia computacional, ya que requiere menos parámetros y es adecuado para entornos con recursos limitados.
- **DeepLabV3+ con ResNet34** ofrece mayor robustez y precisión, a costa de un mayor coste computacional.

Gracias al *Transfer Learning*, estos modelos han podido adaptarse rápidamente a la tarea de segmentación de tumores cerebrales en MRI, incluso con un conjunto de datos moderadamente reducido. Esta técnica ha sido clave para mejorar la **precisión** y **estabilidad** del entrenamiento, minimizando el riesgo de **sobreajuste** y reduciendo el **tiempo de entrenamiento** necesario.

3
Metodología

Este capítulo describe en detalle el enfoque metodológico seguido durante el desarrollo del proyecto. Se expone cómo se ha estructurado el trabajo en diferentes fases, desde la investigación hasta la integración del sistema en una aplicación web, detallando los procedimientos, técnicas y decisiones adoptadas en cada parte.

Se abordan aspectos clave como la obtención y preparación del conjunto de datos, el preprocesamiento de imágenes, el diseño e implementación de los modelos de segmentación, las métricas empleadas y las herramientas utilizadas. Este capítulo tiene como objetivo ofrecer una visión clara, ordenada y justificada del proceso completo seguido para alcanzar el objetivo del proyecto.

3.1. Planteamiento general del proyecto

El enfoque general del proyecto consiste en desarrollar un sistema basado en **IA** capaz de segmentar automáticamente **tumores cerebrales** en **MRI**. Este sistema se apoya en arquitecturas de **CNN** especializadas en segmentación semántica, como **U-Net**, **Attention U-Net** y **DeepLabV3+** [33].

El proyecto se ha dividido en tres fases principales:

1. **Fase de investigación y análisis**, en la que se estudian las técnicas de **aprendizaje profundo** aplicadas al procesamiento de imágenes médicas y se evalúan distintos enfoques y arquitecturas [1, 26, 8, 32].

2. **Fase de desarrollo y experimentación**, en la que se implementan y entrenan distintos modelos de segmentación utilizando imágenes reales de resonancia magnética. Se aplican técnicas de evaluación para medir la precisión de los modelos y se ajustan los parámetros para mejorar el rendimiento.

3. **Fase de integración**, en la que los modelos se incorporan en una **aplicación web** que permite la interacción con el sistema de segmentación. La interfaz permite al usuario subir imágenes, procesarlas mediante los modelos entrenados y visualizar los resultados obtenidos.

Además, el sistema ha sido diseñado con el objetivo de ser **modular y escalable**, permitiendo la incorporación de nuevos modelos o mejoras futuras sin necesidad de rehacer la arquitectura completa.

3.2. Dataset utilizado

Para el entrenamiento y validación de los modelos de segmentación se han utilizado dos conjuntos de datos públicos obtenidos de la plataforma **Kaggle**:

- **Brain MRI Segmentation** [20]
- **Brain MRI Images** [20]

Ambos conjuntos contienen **MRI cerebral**, junto con sus correspondientes **máscaras de segmentación de tumores**. Las imágenes originales se encuentran en formato .tif y las máscaras correspondientes comparten nombre con la imagen original pero añadiendo el sufijo _mask. Esto permite establecer un mapeo directo entre cada imagen y su máscara.

El proceso comenzó con la descarga de los conjuntos desde Kaggle y su posterior subida a **Google Drive**, desde donde se compartieron públicamente. Este paso se consideró esencial para asegurar la disponibilidad permanente de los datos, dado que podrían ser retirados de la plataforma en cualquier momento [19, 14].

La estructura del conjunto de datos está organizada en subcarpetas, y cada una contiene imágenes y máscaras correspondientes. Para automatizar el acceso a estos datos, se recorren las subcarpetas para extraer las rutas de las imágenes y de sus máscaras mediante **código Python**, lo que permite construir un *dataset* personalizado para el entrenamiento.

El tamaño del conjunto de datos es de entre 3100 y 3200 imágenes. Es un buen tamaño para obtener modelos de calidad decente, a la vez que manteniendo cierta eficiencia y velocidad en

el entrenamiento. Sin embargo, si se dispusiese de recursos más potentes, se podrían añadir muchas más imágenes para que el modelo sea mucho más preciso y eficaz.

El tipo de imágenes utilizadas es .tif, es un formato de imagen que presenta bastantes ventajas sobre otros formatos como .jpg o .png:

- Es un formato sin pérdida de información, a diferencia de .jpg que sí que tiene pérdida de información. Esto es principal en el ámbito de los tumores cerebrales, ya que se busca la mayor precisión posible.

- Tiene mayor profundidad de bits que los formatos .jpg y .png, los cuáles están limitados a 8 y 16 bits por canal respectivamente. En radiología, los matices de intensidad son muy importantes, y por ello esta característica es esencial.

- A pesar de que no se incluye en este TFG, el formato .tif puede incluir metadatos extendidos como los DICOM headers.

Las imágenes del conjunto de datos tienen un tamaño de 256x256 píxeles (que será el tamaño utilizado para el entrenamiento de los modelos, a excepción de DeepLabV3+ con *backbone* MobileNetV2) y el modo de color es RGB (*Red-Green-Blue*).

Cada una de las imágenes va asociada a una máscara de segmentación binaria en la que se puede visualizar fácilmente el tumor. Las características de dichas máscaras son relativamente similares a las imágenes naturales. La diferencia principal es el modo de color, ya que la máscara es en blanco y negro (en blanco el tumor y en negro el fondo).

Las Figuras 8 y 9 introduce un ejemplo de imagen del conjunto de datos y su respectiva máscara. Gracias a estas Figuras, se puede establecer una visión más clara de cómo es el conjunto de datos, ya que todas las imágenes del *dataset* siguen el mismo patrón:

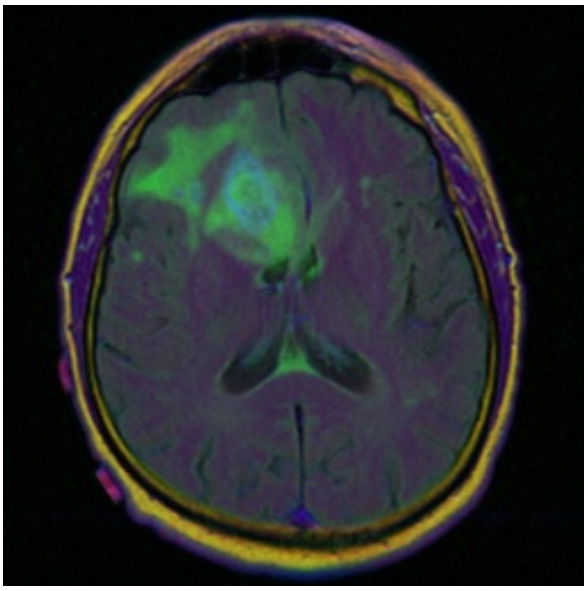

Figura 8: Imagen 1 Dataset. Ejemplo de MRI del conjunto de datos

Figura 9: Máscara 1 Dataset. Ejemplo de máscara de segmentación del conjunto de datos

Antes de alimentar estos datos al modelo, se realizó una **visualización inicial** con la librería **matplotlib**, lo que permitió confirmar que las imágenes y sus máscaras están correctamente alineadas y que el contenido es coherente.

3.3. Preprocesamiento de datos (normalización, segmentación, augmentación...)

Antes de entrenar cualquier modelo de segmentación, se llevó a cabo un proceso de **preprocesamiento** sobre las imágenes y sus máscaras. Estas transformaciones fueron fundamentales para garantizar la compatibilidad con las arquitecturas utilizadas y para mejorar el rendimiento del entrenamiento [27]. Aunque cada modelo tuvo algunas particularidades, existen una serie de pasos comunes aplicados en todos ellos.

Redimensionamiento

Las imágenes se redimensionaron a un tamaño fijo. Este tamaño varió según el modelo:

- En **U-Net**, **DeepLabV3+ con ResNet34** y **Attention U-Net**, se utilizó un tamaño de **256x256 píxeles**.
- En **DeepLabV3+ con MobileNetV2**, se usó un tamaño de **128x128 píxeles** para reducir los tiempos de entrenamiento.

Para visualizar las diferencias entre la calidad de una imagen de 128x128 píxeles y 256x256, las Figuras 10 y 11 muestran un ejemplo visual. Se pueden observar las diferencias de resolución entre las imágenes, la de 256x256 tiene mayor nivel de detalle y la de 128x128 tiene menor resolución, especialmente en los bordes:

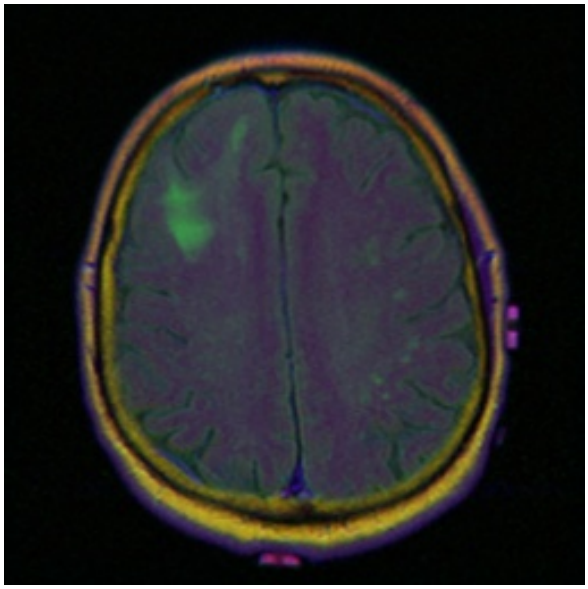

Figura 10: Ejemplo de MRI de 256x256 píxeles

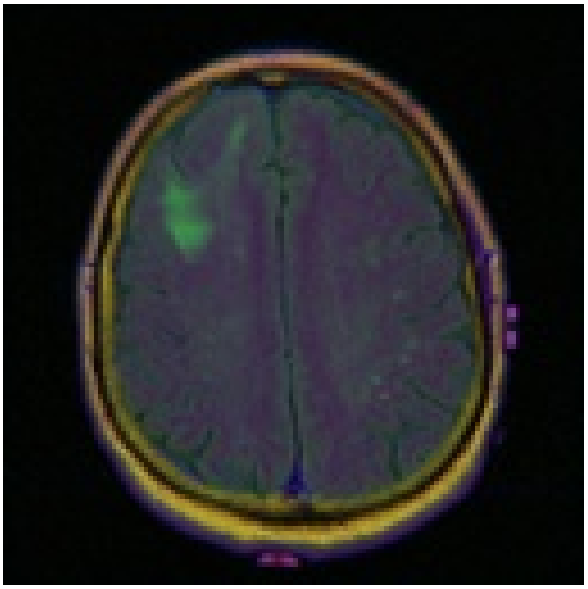

Figura 11: Ejemplo de MRI de 128x128 píxeles

Este redimensionamiento garantiza **uniformidad en las dimensiones de entrada**, requisito imprescindible para las CNN [8].

Conversión de color

Todas las imágenes fueron convertidas al espacio de color **RGB**, ya que **OpenCV** las carga inicialmente en **BGR** y este cambio es necesario para una visualización correcta y un tratamiento coherente en los modelos [27].

Normalización

Las imágenes se normalizaron al rango **[-1, 1]**, utilizando **medias y desviaciones estándar por canal**. Este proceso mejora la eficiencia del entrenamiento, facilitando la convergencia de los modelos.

Conversión a tensores

Tanto las imágenes como las máscaras fueron convertidas a **tensores**, utilizando las funciones de la librería **PyTorch** [28]. Esto permite su utilización directa en las redes neuronales.

Binarización de máscaras

Las máscaras de segmentación fueron **binarizadas**, asignando el valor **1** a las regiones tumorales y **0** al fondo. Esto transforma el problema en una tarea de **segmentación binaria**, común en imágenes médicas [8].

Ajuste de dimensiones

En las máscaras se añadió una **dimensión adicional de canal** para que pudieran ser procesadas en paralelo junto con las imágenes.

Aumentos de datos

Se aplicaron técnicas de ***data augmentation*** utilizando la librería **Albumentations**, especialmente en el caso de **DeepLabV3+ con MobileNetV2**. Las transformaciones utilizadas fueron:

- *Flip horizontal*
- Rotación aleatoria
- Ajustes de brillo y contraste

Estas transformaciones se aplicaron tanto a la imagen como a su máscara correspondiente de forma sincronizada, con el objetivo de aumentar la **variabilidad del conjunto de entrenamiento** y reducir el riesgo de **sobreajuste** [33].

La Figura 12 muestra un ejemplo de una imagen natural y sus posibles transformaciones, que son: la misma imagen con un *flip* horizontal, con una rotación aleatoria y con el brillo ajustado. En estas ilustraciones se puede observar las distintas transformaciones que se aplica al conjunto de datos para que el entrenamiento sea lo más eficaz posible. Estas transformaciones también pueden evitar que el modelo memorice en lugar de aprender:

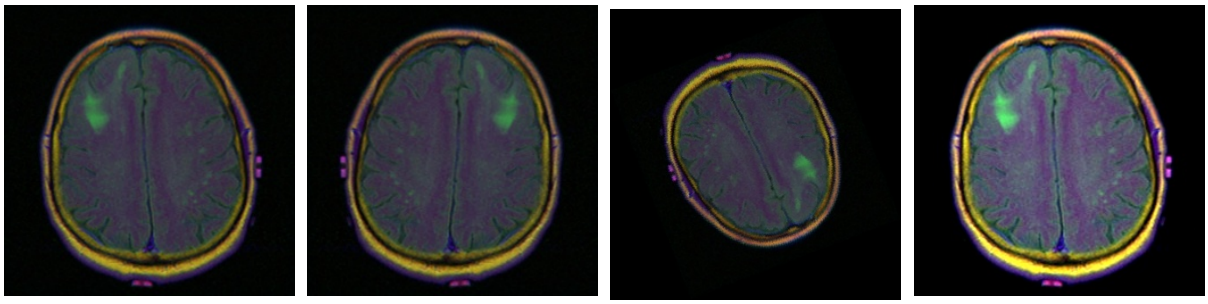

Imagen original | Flip horizontal | Rotación aleatoria | Brillo y contraste

Figura 12: Transformaciones aplicadas sobre una imagen médica para aumento de datos (data augmentation).

Visualización inicial

Antes de aplicar el preprocesamiento de forma masiva, se realizó una **visualización de las primeras imágenes y sus máscaras** utilizando **matplotlib**, para asegurar la correcta correspondencia y alineación de los datos. Esta validación inicial fue fundamental para confirmar que no existían errores en la carga o en la estructura del *dataset*.

3.4. Diseño e implementación de modelos

Durante el desarrollo del proyecto se implementaron cinco modelos de **segmentación semántica** basados en **CNN**. Estas arquitecturas fueron seleccionadas por su relevancia en el campo de la segmentación médica y se utilizaron como base para comparar distintos enfoques [8, 32, 26, 33, 1].

- **U-Net (modelo experimental)**
- **U-Net (modelo final)**

- **Attention U-Net**
- **DeepLabV3+ con MobileNetV2**
- **DeepLabV3+ con ResNet34**

Las implementaciones se realizaron utilizando **PyTorch** [28], y en el caso de **DeepLabV3+**, a través de la librería **segmentation_models_pytorch**.

U-Net - Modelo experimental

La primera implementación de **U-Net** fue desarrollada desde cero utilizando **PyTorch**. El objetivo principal de este modelo fue comprender en detalle el funcionamiento interno de una arquitectura de segmentación, sus componentes, flujo de datos y lógica de entrenamiento.

Esta red se construyó siguiendo la estructura clásica en forma de U, compuesta por una etapa de *encoder* y otra de *decoder*, conectadas mediante *skip connections* [32]. Cada bloque de *encoder* contiene dos capas convolucionales con funciones de activación **ReLU**, seguidas de una capa de ***max pooling***. El *decoder* replicaba esta estructura de forma simétrica utilizando ***upsampling***.

El número de filtros aumentaba progresivamente a medida que se profundizaba en el *encoder* (de 64 a 512), y disminuía en el *decoder* hasta volver al tamaño original. La salida final era una imagen del mismo tamaño que la entrada, con un único canal que representa la **máscara segmentada**.

U-Net - Modelo final

La segunda implementación de **U-Net** fue desarrollada con un enfoque más **modular**, manteniendo la arquitectura original pero utilizando una clase **UNet** parametrizable. Esta versión se utilizó para realizar comparaciones con otros modelos más avanzados.

La arquitectura mantenía los mismos componentes principales: **convoluciones dobles por bloque**, **capas de *pooling***, **capas de *upsampling***, y **conexiones de salto**. Además, se implementaron bloques auxiliares como **DoubleConv**, **Down**, **Up** y **OutConv** para mejorar la legibilidad y escalabilidad del código.

Este modelo también trabajaba con imágenes de tamaño **256x256 píxeles** y aceptaba máscaras binarizadas. Se utilizó una **función de pérdida combinada** que mezcla ***Binary Cross-Entropy (BCE)*** y ***Dice Loss***. El modelo fue entrenado utilizando el optimizador **Adam**.

Attention U-Net

El modelo **Attention U-Net** extiende la arquitectura clásica de U-Net incorporando **mecanismos de atención**. La principal diferencia respecto a U-Net es el uso de bloques denominados **AG**, que permiten al modelo centrarse en regiones relevantes de la imagen durante la decodificación [26].

Estos bloques se aplican antes de concatenar las características del *encoder* con las del *decoder*, **filtrando la información más útil** y suprimiendo el ruido irrelevante. La estructura general sigue una forma de U, con bloques simétricos de *encoder* y *decoder* y ***skip connections***, pero enriquecida con **atención espacial**.

La implementación se realizó mediante la definición de módulos **AttentionBlock** y **AttentionUNet** en PyTorch, manteniendo el enfoque **modular y reutilizable**. También utilizó imágenes de **256x256**, y se empleó **Dice Loss** como función de pérdida, junto con el optimizador **Adam**.

DeepLabV3+ con MobileNetV2

DeepLabV3+ es una arquitectura desarrollada por Google para **segmentación semántica** [33]. En esta implementación, se utilizó **MobileNetV2** como ***backbone***, lo que permite reducir la cantidad de parámetros y acelerar la inferencia, siendo adecuado para dispositivos con recursos limitados.

El modelo fue cargado mediante la librería **segmentation_models_pytorch**, que permite instanciar arquitecturas con pesos preentrenados en **ImageNet**. La arquitectura hace uso de **convoluciones dilatadas** y del módulo *Atrous Spatial Pyramid Pooling (ASPP)*, que permite capturar **información contextual a distintas escalas** sin reducir la resolución espacial.

La entrada al modelo se ajustó al tamaño **128x128 píxeles**, y se utilizó una función de pérdida basada en **Dice Loss**, con el optimizador **Adam**.

DeepLabV3+ con ResNet34

En esta variante de **DeepLabV3+** se utilizó **ResNet34** como *backbone*. A diferencia de MobileNetV2, ResNet34 tiene **mayor profundidad y capacidad de representación**, lo que la convierte en una opción más potente a costa de un mayor coste computacional.

La estructura principal del modelo sigue siendo DeepLabV3+, con ***ASPP*** y **convoluciones dilatadas**. El modelo se cargó también desde **segmentation_models_pytorch**, con pesos preentrenados.

Este modelo fue configurado para aceptar imágenes de **256x256 píxeles** y utilizar una función de pérdida **Dice**, con el optimizador **Adam**, al igual que las otras implementaciones.

3.5. Métricas de evaluación utilizadas

Durante el entrenamiento y validación de los modelos desarrollados en este proyecto, se han utilizado tres métricas principales para evaluar el rendimiento y la eficiencia de los sistemas de segmentación automática: **Dice Coefficient**, ***Intersection over Union*** **(IoU)** y el **tiempo de entrenamiento** [24, 29].

Dice Coefficient (también usado como Dice Loss)

El **Dice Coefficient** es una métrica ampliamente utilizada en tareas de segmentación médica. Mide el grado de coincidencia entre la segmentación realizada por el modelo y la segmentación real (máscara de referencia). Tiene en cuenta tanto los **verdaderos positivos** como los errores (**falsos positivos y falsos negativos**) y toma valores entre 0 y 1, donde 1 indica una coincidencia perfecta [24].

Matemáticamente se expresa como:

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

donde A representa la máscara predicha y B la máscara real.

En la práctica, se utiliza el **Dice Coefficient como función de pérdida** invirtiendo su valor (es decir, $1 - \text{Dice}$) para que su minimización durante el entrenamiento implique una mejora en la calidad de la segmentación.

Esta implementación personalizada incluye un término de **suavizado** smooth $= 1$ para evitar divisiones por cero, especialmente en imágenes donde el tumor puede estar ausente.

Intersection over Union (IoU)

La métrica **IoU** también mide la superposición entre la máscara predicha por el modelo y la máscara real. Es una métrica más estricta que el *Dice Coefficient*, ya que calcula el área de intersección entre predicción y verdad del terreno y la divide entre el área total de la unión de ambas [29].

Se define como:

$$\text{IoU} = \frac{|A \cap B|}{|A \cup B|}$$

En segmentación médica, se considera que un modelo empieza a ser útil a partir de un valor de **IoU de 0.7**, mientras que valores inferiores a **0.5** suelen interpretarse como indicativos de un rendimiento pobre.

Durante el entrenamiento, IoU se utiliza como **métrica de validación al final de cada época**, permitiendo evaluar la evolución del modelo y comparar diferentes arquitecturas de forma objetiva y cuantitativa.

Tiempo de entrenamiento

Además de las métricas de precisión, se ha registrado el **tiempo total de entrenamiento** de cada modelo. Esta medida permite valorar la **eficiencia computacional** de cada arquitectura y resulta especialmente útil en entornos con **recursos limitados**, donde el coste computacional puede ser un factor decisivo.

3.6. Herramientas, lenguajes y librerías empleadas

Durante el desarrollo del **Trabajo de Fin de Grado** se han utilizado diferentes herramientas de software, entornos de desarrollo, lenguajes de programación y librerías especializadas que han permitido abordar tareas como la carga y procesamiento de imágenes, entrenamiento

de modelos de IA, evaluación, visualización de resultados e implementación de una interfaz web funcional.

Entornos de desarrollo

- **Jupyter Notebook**: entorno interactivo ampliamente usado en ciencia de datos y aprendizaje automático, que permite combinar código, visualizaciones y texto en un único documento ejecutable [21].

- **Google Colab**: plataforma gratuita de Google basada en Jupyter Notebook que ofrece recursos en la nube, como acceso a GPU y TPU, útil para entrenar modelos complejos sin hardware local [2, 13].

- **Visual Studio Code**: editor de código fuente desarrollado por Microsoft, con soporte para múltiples lenguajes y extensiones, incluyendo herramientas de desarrollo web y Python [23].

Librerías y frameworks

Se han empleado distintas librerías tanto para el entrenamiento de los modelos como para su integración y visualización:

- **PyTorch**: *framework* de aprendizaje profundo desarrollado por *Facebook AI Research*, ampliamente utilizado en investigación y producción por su flexibilidad y soporte para redes neuronales dinámicas [28].

- **OpenCV**: librería de código abierto para procesamiento de imágenes y visión por computador, que ofrece funciones eficientes para lectura, conversión y manipulación de imágenes [27].

- **Albumentations**: biblioteca de aumento de datos diseñada para tareas de visión por computador, que proporciona transformaciones rápidas y flexibles [4].

- **Flask**: *microframework* de Python utilizado para crear aplicaciones web y APIs de forma sencilla y escalable [15].

- **FPDF**: librería para la generación de documentos PDF desde código Python, utilizada para construir informes automáticos con resultados del modelo [10].
- **Matplotlib** y **TQDM**: Matplotlib se ha utilizado para la visualización de imágenes y resultados, mientras que TQDM permite mostrar barras de progreso en tiempo real durante la ejecución de bucles [16, 7].

Recursos adicionales

- **Kaggle**: plataforma que proporciona *datasets* públicos, *notebooks* ejecutables y competencias de ciencia de datos. Se utilizaron *datasets* de resonancia magnética cerebral descargados desde esta plataforma [20].
- **GitHub**: sistema de control de versiones basado en Git que permite alojar código, gestionar colaboraciones y mantener el historial de versiones del proyecto [6].
- **Google Drive**: servicio de almacenamiento en la nube de Google utilizado como repositorio temporal para compartir conjuntos de datos y modelos durante el desarrollo [14].

4

Desarrollo y análisis

En este capítulo se detalla el proceso práctico de desarrollo del sistema de segmentación automática de tumores cerebrales, desde la exploración inicial de los datos hasta la evaluación final de los modelos implementados.

Se describe en primer lugar el análisis exploratorio realizado sobre el conjunto de datos, con el objetivo de verificar su calidad y estructura. A continuación, se presentan los distintos modelos entrenados, tanto aquellos implementados desde cero como los basados en arquitecturas preentrenadas, explicando su funcionamiento interno, configuración y proceso de entrenamiento.

Además, se lleva a cabo una comparativa cuantitativa y cualitativa entre modelos, atendiendo a métricas como la *Dice Loss*, el IoU y el tiempo de entrenamiento. Finalmente, se discuten las mejoras aplicadas durante el proceso y se reflexiona sobre la utilidad práctica de los modelos en un entorno real.

4.1. Exploración y análisis de datos

Antes de iniciar el entrenamiento de los modelos de segmentación, se realizó un análisis exploratorio detallado del conjunto de datos con el fin de garantizar su calidad, entender su estructura y asegurar la correcta correspondencia entre imágenes y máscaras.

Visualización inicial del dataset

El primer paso consistió en realizar una visualización manual de algunas muestras del conjunto de datos. Esto permitió verificar que las imágenes y sus correspondientes máscaras estaban correctamente alineadas y en buen estado.

Se utilizó la librería **matplotlib** para mostrar en paralelo una MRI y su máscara binaria, representando las regiones tumorales como se enseña en las Figuras 13 y 14:

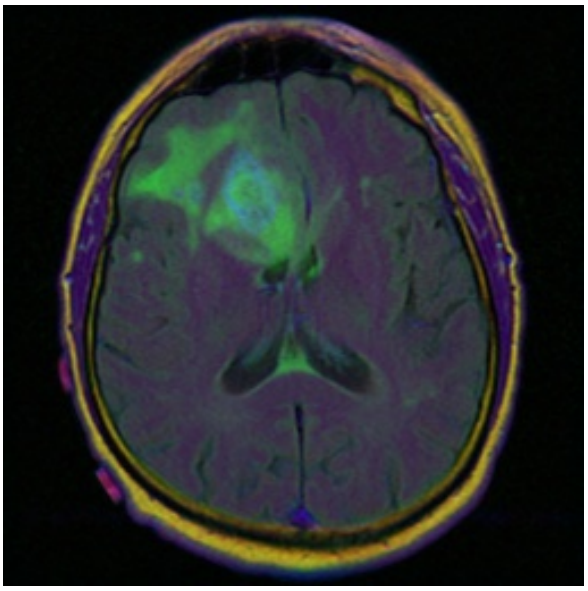

Figura 13: Ejemplo de visualización de imagen del conjunto de datos

Figura 14: Ejemplo de visualización de máscara de segmentación del conjunto de datos

```
import cv2
import matplotlib.pyplot as plt

img = cv2.imread(image_paths[0])
mask = cv2.imread(mask_paths[0], cv2.IMREAD_GRAYSCALE)

plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.title("Imagen")
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.axis('off')

plt.subplot(1, 2, 2)
plt.title("M\'ascara")
plt.imshow(mask, cmap='gray')
plt.axis('off')
```

```
plt.show()
```

Gracias a este procedimiento se pudo confirmar que:

- Las rutas de imagen y máscara estaban correctamente mapeadas.
- Las máscaras presentaban una codificación binaria simple (región tumoral vs. fondo).
- Las dimensiones eran coherentes y apropiadas para el tratamiento posterior.

Formato y estructura de los archivos

El conjunto de datos se descargó desde Kaggle e incluye imágenes en formato ṫif, divididas en subcarpetas. Cada imagen tiene una máscara asociada con el mismo nombre y el sufijo _mask. A continuación, se muestra el código utilizado para extraer las rutas:

```
image_paths = []
mask_paths = []
data_dir = "dataset/lgg-mri-segmentation/kaggle_3m"

for folder in os.listdir(data_dir):
    folder_path = os.path.join(data_dir, folder)
    for file in os.listdir(folder_path):
        file_path = os.path.join(folder_path, file)
        if "_mask" in file:
            mask_paths.append(file_path)
        else:
            image_paths.append(file_path)
```

Esto permitió construir dos listas ordenadas con las rutas de las imágenes y sus respectivas máscaras, asegurando la correspondencia entre ellas.

Comprobación del mapeo

Para garantizar que cada imagen tenía su máscara correspondiente, se empleó una comprobación de igualdad entre nombres de archivo, que solo difieren por el sufijo _mask. Esto sirvió como paso previo para la creación de un *dataset* personalizado.

4.2. Entrenamiento de modelos desde cero

En esta sección se describe el proceso completo de entrenamiento de los modelos desarrollados manualmente desde cero, sin utilizar arquitecturas preentrenadas. Se incluyen tres implementaciones diferentes: U-Net (dos versiones) y Attention U-Net. Cada modelo se ha construido utilizando la librería PyTorch y ha seguido una metodología sistemática, abarcando desde la preparación del entorno hasta la evaluación en el conjunto de validación.

U-Net – Primera implementación

Este primer modelo se desarrolló con fines exploratorios y de aprendizaje. Se trata de una implementación manual de la arquitectura U-Net, diseñada para entender en profundidad la estructura *encoder-decoder*, las conexiones de salto y el flujo de datos en segmentación médica.

Preparación del entorno

```
import os
import numpy as np
import cv2
import torch
import torchvision.transforms as transforms
from torch.utils.data import Dataset, DataLoader
import torch.nn as nn
import matplotlib.pyplot as plt
```

Carga y visualización de datos

```
image_paths = []
mask_paths = []
data_dir = "dataset/lgg-mri-segmentation/kaggle_3m"

for folder in os.listdir(data_dir):
    folder_path = os.path.join(data_dir, folder)
    for file in os.listdir(folder_path):
        file_path = os.path.join(folder_path, file)
```

```
        if "_mask" in file:
            mask_paths.append(file_path)
        else:
            image_paths.append(file_path)
```

Preprocesamiento

```
def preprocess_image(image_path, mask_path, img_size=(256, 256)
    ):
    image = cv2.imread(image_path)
    image = cv2.resize(image, img_size)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

    mask = cv2.imread(mask_path, cv2.IMREAD_GRAYSCALE)
    mask = cv2.resize(mask, img_size)
    mask = (mask > 0).astype(np.uint8)

    transform = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.5]*3, std=[0.5]*3)
    ])

    image = transform(image)
    mask = torch.tensor(mask, dtype=torch.float32).unsqueeze(0)
    return image, mask
```

Dataset personalizado

```
class BrainTumorDataset(Dataset):
    def __init__(self, image_paths, mask_paths):
        self.image_paths = image_paths
        self.mask_paths = mask_paths

    def __len__(self):
```

```
        return len(self.image_paths)

    def __getitem__(self, idx):
        image, mask = preprocess_image(self.image_paths[idx],
   self.mask_paths[idx])
        return image, mask
```

Definición del modelo

```
def conv_block(in_channels, out_channels):
    return nn.Sequential(
        nn.Conv2d(in_channels, out_channels, 3, padding=1),
        nn.ReLU(inplace=True),
        nn.Conv2d(out_channels, out_channels, 3, padding=1),
        nn.ReLU(inplace=True)
    )

class UNet(nn.Module):
    def __init__(self):
        super().__init__()
        self.enc1 = conv_block(3, 64)
        self.enc2 = conv_block(64, 128)
        self.pool = nn.MaxPool2d(2)
        self.bottleneck = conv_block(128, 256)
        self.upconv = nn.ConvTranspose2d(256, 128, 2, stride=2)
        self.dec = conv_block(256, 64)
        self.final = nn.Conv2d(64, 1, 1)

    def forward(self, x):
        e1 = self.enc1(x)
        e2 = self.enc2(self.pool(e1))
        b = self.bottleneck(self.pool(e2))
```

```
        d = self.dec(torch.cat([self.upconv(b), e1], dim=1))
        return self.final(d)
```

Función de pérdida

```
class DiceLoss(nn.Module):
    def forward(self, inputs, targets, smooth=1):
        inputs = torch.sigmoid(inputs)
        inputs = inputs.view(-1)
        targets = targets.view(-1)
        intersection = (inputs * targets).sum()
        dice = (2. * intersection + smooth) / (inputs.sum() +
   targets.sum() + smooth)
        return 1 - dice
```

Entrenamiento

```
optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)
```

Entrenamiento completo

```
def train(model, loader, optimizer, criterion, epoch_num=None):
    model.train()
    total_loss = 0
    for batch_idx, (x, y) in enumerate(loader):
        x, y = x.to(DEVICE), y.to(DEVICE)
        preds = model(x)
        loss = criterion(preds, y)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
        if batch_idx % 5 == 0:
            print(f"Epoch {epoch_num} - Batch {batch_idx + 1} -
    Loss: {loss.item():.4f}")
    return total_loss / len(loader)
```

U-Net - Segunda implementación

Esta segunda versión de U-Net fue diseñada para ser más robusta y escalable que la primera, manteniendo la arquitectura original pero con una implementación completamente modular y orientada a reutilización de código. Esta versión ha sido usada en la comparativa final del proyecto.

Preparación del entorno

El modelo fue implementado y entrenado en Google Colab, aprovechando aceleración por GPU. Las librerías utilizadas fueron:

- `torch`, `torchvision`
- `albumentations` (para data augmentation)
- `cv2` (OpenCV)
- `matplotlib`
- `tqdm` (barras de progreso)

Arquitectura del modelo

Se implementaron bloques auxiliares para definir la red de forma clara y mantenible:

```
def conv_block(in_channels, out_channels):
    return nn.Sequential(
        nn.Conv2d(in_channels, out_channels, 3, padding=1),
        nn.BatchNorm2d(out_channels),
        nn.ReLU(inplace=True),
        nn.Conv2d(out_channels, out_channels, 3, padding=1),
        nn.BatchNorm2d(out_channels),
        nn.ReLU(inplace=True)
    )
```

Red principal

```
class UNet(nn.Module):
    def __init__(self):
```

```
        super().__init__()
        self.enc1 = conv_block(3, 64)
        self.enc2 = conv_block(64, 128)
        self.enc3 = conv_block(128, 256)
        self.enc4 = conv_block(256, 512)

        self.pool = nn.MaxPool2d(2)
        self.bottleneck = conv_block(512, 1024)

        self.upconv4 = nn.ConvTranspose2d(1024, 512, 2, stride
=2)
        self.dec4 = conv_block(1024, 512)

        self.upconv3 = nn.ConvTranspose2d(512, 256, 2, stride
=2)
        self.dec3 = conv_block(512, 256)

        self.upconv2 = nn.ConvTranspose2d(256, 128, 2, stride
=2)
        self.dec2 = conv_block(256, 128)

        self.upconv1 = nn.ConvTranspose2d(128, 64, 2, stride=2)
        self.dec1 = conv_block(128, 64)

        self.final = nn.Conv2d(64, 1, 1)

    def forward(self, x):
        e1 = self.enc1(x)
        e2 = self.enc2(self.pool(e1))
        e3 = self.enc3(self.pool(e2))
        e4 = self.enc4(self.pool(e3))
```

```
        b = self.bottleneck(self.pool(e4))

        d4 = self.dec4(torch.cat([self.upconv4(b), e4], dim=1))
        d3 = self.dec3(torch.cat([self.upconv3(d4), e3], dim=1)
    )
        d2 = self.dec2(torch.cat([self.upconv2(d3), e2], dim=1)
    )
        d1 = self.dec1(torch.cat([self.upconv1(d2), e1], dim=1)
    )

        return self.final(d1)
```

Función de pérdida

```
class DiceLoss(nn.Module):
    def forward(self, inputs, targets, smooth=1):
        inputs = torch.sigmoid(inputs)
        inputs = inputs.view(-1)
        targets = targets.view(-1)
        intersection = (inputs * targets).sum()
        dice = (2. * intersection + smooth) / (inputs.sum() +
    targets.sum() + smooth)
        return 1 - dice
```

Optimización

```
optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)
```

Entrenamiento

```
def train(model, loader, optimizer, criterion, epoch_num=None):
    model.train()
    total_loss = 0
    for batch_idx, (x, y) in enumerate(loader):
```

```
        x, y = x.to(DEVICE), y.to(DEVICE)
        preds = model(x)
        loss = criterion(preds, y)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
        if batch_idx % 5 == 0:
            print(f"Epoch {epoch_num} - Batch {batch_idx + 1} -
    Loss: {loss.item():.4f}")
    return total_loss / len(loader)
```

Resultados del entrenamiento

Las Figuras 15 y 16 ilustran 2 gráficas que muestran de manera visual todo el proceso de entrenamiento. Por una parte, se dispone de una gráfica que representa el *loss* con respecto al número de época, y por otra parte, se dispone de la evolución del IoU con respecto al número de época. En este gráfico se aprecia el aprendizaje de este modelo. Se puede observar cómo la pérdida disminuye a lo largo de las épocas, convergiendo al final del entrenamiento. Se podría haber logrado alguna pequeña mejoría entrenándolo durante más épocas pero nada increíblemente significativo ya que converge. El segundo gráfico muestra la evolución del IoU con respecto al número de épocas. También se puede concluir que el modelo está aprendiendo conforme al paso de las épocas, hasta que converge. Se podría mejorar algo el resultado entrenándolo durante más épocas pero no sería una mejoría muy significativa debido a esta convergencia.

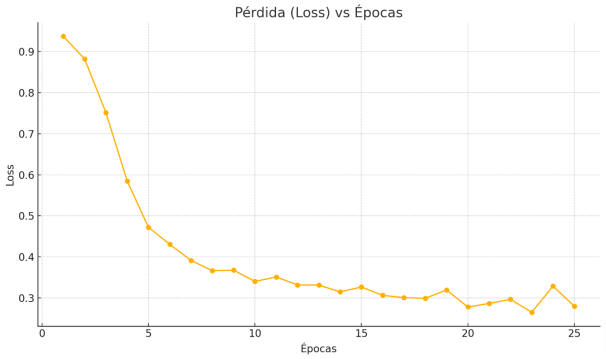

Figura 15: Gráfica 1 U-Net. Resultados del entrenamiento, pérdida con respecto al número de época

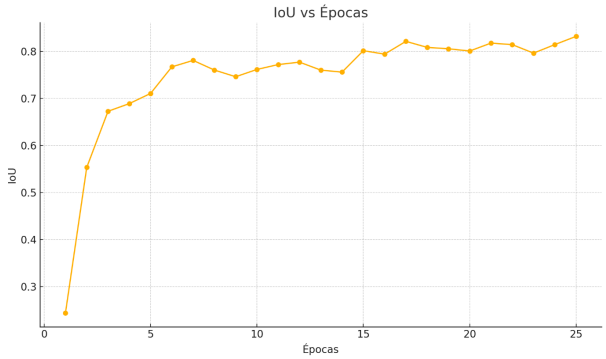

Figura 16: Gráfica 2 U-Net. Resultados del entrenamiento, IoU con respecto al número de época

Ejemplo de predicción del modelo

En la Figuras 17 se muestra un ejemplo de la predicción que realiza el modelo U-Net. Por un lado se muestra la imagen natural con la máscara generada por el modelo superpuesta en color naranja, y por otro lado se muestra el zoom sobre dicha máscara para una mejor visualización. En dichas Figuras se puede observar que el modelo sitúa bien el tumor, pero aún es algo mejorable. En los próximos modelos que se presentan, se obtienen resultados bastante más notables:

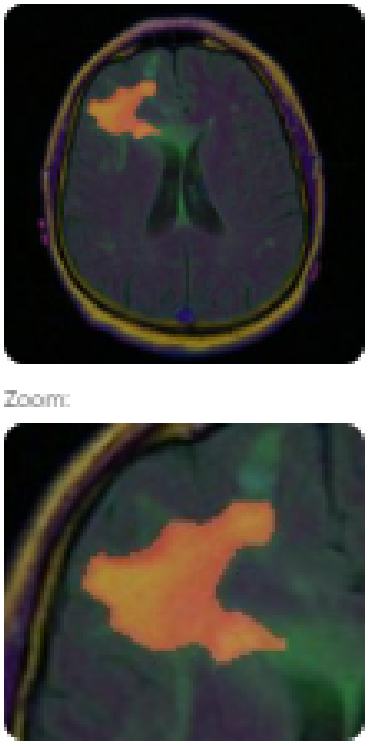

Figura 17: Predicción 1 U-Net. Ejemplo de predicción con U-Net y zoom

Este segundo modelo U-Net constituye una versión madura y modular de la arquitectura original, permitiendo una mejor interpretación del flujo de datos y ofreciendo una precisión superior a la implementación experimental. Gracias a su estructura clara y a su eficiencia, este modelo ha sido incluido en la comparativa final de resultados del TFG y es uno de los candidatos integrados en la interfaz web.

Attention U-Net

El tercer modelo desarrollado desde cero en este proyecto es una implementación de la arquitectura **Attention U-Net**, una evolución del U-Net clásico diseñada para mejorar la precisión de segmentación en casos complejos, como tumores pequeños, poco definidos o con bordes difusos.

Esta arquitectura mantiene la estructura en forma de U, pero introduce un componente clave: los **AG**, que permiten al modelo enfocar su atención en las regiones relevantes de la imagen, suprimir ruido y mejorar la sensibilidad en áreas críticas.

Descripción general

- **Objetivo:** mejorar el enfoque del modelo sobre las regiones anatómicamente importantes.
- **Ventaja principal:** mayor precisión en la segmentación de tumores pequeños y estructuras difíciles de identificar.
- **Complejidad:** moderada, apta para ejecución en Google Colab con GPU.
- **Entrada:** imágenes RGB de 256x256.
- **Salida:** máscara binaria del mismo tamaño que la imagen de entrada.

Preprocesamiento

Se aplica el mismo *pipeline* de preprocesamiento usado en los modelos U-Net:

- Redimensionado a 256x256.
- Conversión de color BGR $\longrightarrow$RGB.
- Normalización a rango $[-1, 1]$.
- Conversión a tensores.
- Binarización de máscaras.

Arquitectura: Integración de bloques de atención

A nivel estructural, Attention U-Net conserva el *encoder* y *decoder* de U-Net, pero modifica las conexiones tipo *skip* añadiendo bloques de atención antes de la concatenación.

Implementación del AttentionBlock

```
class AttentionBlock(nn.Module):
    def __init__(self, F_g, F_l, F_int):
        super().__init__()
        self.W_g = nn.Sequential(
            nn.Conv2d(F_g, F_int, kernel_size=1),
            nn.BatchNorm2d(F_int)
        )
        self.W_x = nn.Sequential(
```

```
            nn.Conv2d(F_l, F_int, kernel_size=1),
            nn.BatchNorm2d(F_int)
        )
        self.psi = nn.Sequential(
            nn.Conv2d(F_int, 1, kernel_size=1),
            nn.BatchNorm2d(1),
            nn.Sigmoid()
        )
        self.relu = nn.ReLU(inplace=True)

    def forward(self, g, x):
        g1 = self.W_g(g)
        x1 = self.W_x(x)
        psi = self.relu(g1 + x1)
        psi = self.psi(psi)
        return x * psi
```

Entrenamiento del modelo

El flujo de entrenamiento sigue la estructura típica de PyTorch, reutilizando el código del modelo U-Net clásico con ligeras modificaciones.

Configuración técnica:

- **Épocas:** 25
- **Tamaño de *batch*:** 8
- **Función de pérdida:** solo *Dice Loss*
- **Optimizador:** AdamW con regularización

```
optimizer = torch.optim.AdamW(model.parameters(), lr=1e-4,
   weight_decay=1e-5)
```

Dice Loss

```
class DiceLoss(nn.Module):
    def forward(self, inputs, targets, smooth=1):
        inputs = torch.sigmoid(inputs)
        inputs = inputs.view(-1)
        targets = targets.view(-1)
        intersection = (inputs * targets).sum()
        dice = (2. * intersection + smooth) / (inputs.sum() +
  targets.sum() + smooth)
        return 1 - dice
```

La función `train()` y el bucle de entrenamiento son equivalentes a los anteriores modelos, salvo que las salidas se procesan con mayor precisión gracias a los Attention Gates integrados.

Resultados del entrenamiento

Las Figuras 18 y 19 muestran 2 gráficas que representan de manera visual todo el proceso de entrenamiento del modelo Attention U-Net. Por una parte, se dispone de una gráfica que representa el *loss* con respecto al número de época, y por otra parte, se dispone de la evolución del IoU con respecto al número de época. Al igual que en el modelo anterior, se observa una disminución progresiva de la pérdida y un aumento del IoU a lo largo de las épocas, lo que refleja un aprendizaje efectivo. Si bien la precisión podría haberse mejorado con un mayor número de épocas, la convergencia alcanzada indica que las ganancias adicionales serían marginales:

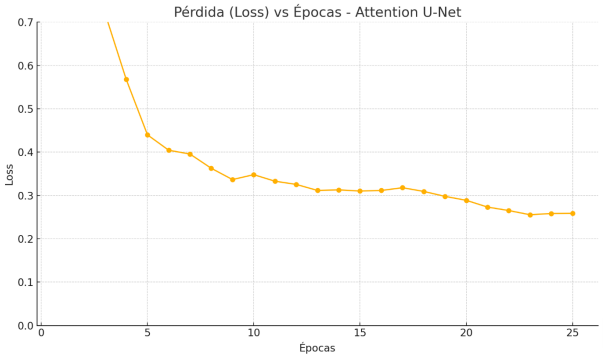

Figura 18: Gráfica 1 Attention U-Net. Resultados del entrenamiento, pérdida con respecto al número de época

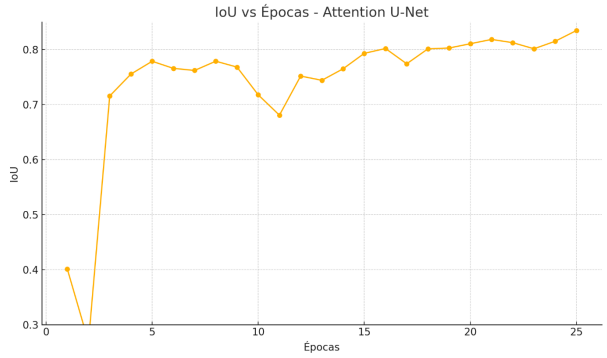

Figura 19: Gráfica 2 Attention U-Net. Resultados del entrenamiento, IoU con respecto al número de época

Ejemplo de predicción del modelo

La Figura 20 muestra un ejemplo de la predicción que realiza el modelo Attention U-Net. La imagen muestra, por un lado, la máscara generada por el modelo superpuesta en color naranja sobre la imagen natural, y por otro, un acercamiento que facilita su análisis. El modelo localiza correctamente el tumor y, en comparación con el U-Net clásico, logra una segmentación más precisa en cuanto a forma. Al reducir los falsos positivos, la región segmentada es ligeramente más pequeña, lo que también implica una menor inclusión de área no tumoral:

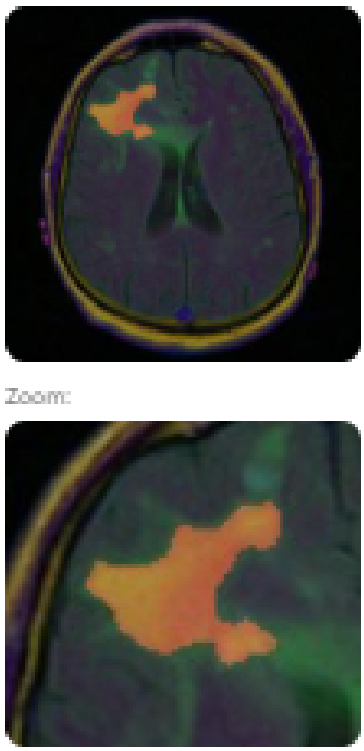

Figura 20: Predicción 2 Attention U-Net. Ejemplo de predicción con Attention U-Net y zoom

Conclusión

Attention U-Net representa una mejora significativa en entornos clínicos donde la precisión es crítica. Gracias a sus mecanismos de atención, ofrece:

- Mayor sensibilidad en zonas pequeñas o con bajo contraste.
- Menor cantidad de falsos positivos.
- Mejor adaptación a formas irregulares.

Este modelo ha demostrado un rendimiento notable y ha sido incluido en el análisis comparativo de resultados, con visualizaciones y métricas que validan su efectividad.

4.3. Implementación de modelos preentrenados

En esta sección se describen los dos modelos basados en la arquitectura **DeepLabV3+**, que fueron implementados utilizando la librería `segmentation_models_pytorch`. Ambos modelos

emplean ***backbones* preentrenados en ImageNet**, lo cual permite acelerar el entrenamiento y mejorar la generalización, especialmente en escenarios con conjuntos de datos limitados.

Los modelos desarrollados son:

- DeepLabV3+ con MobileNetV2 (modelo ligero)
- DeepLabV3+ con ResNet34 (modelo robusto)

Ambas versiones comparten una misma arquitectura base de segmentación semántica, cuya característica más relevante es la inclusión del módulo *ASPP*.

Arquitectura DeepLabV3+

La arquitectura DeepLabV3+ fue desarrollada por Google para tareas de segmentación semántica avanzada. Su estructura combina:

- *Backbone* convolucional preentrenado (MobileNetV2 o ResNet34).
- Módulo *ASPP*, que aplica varias convoluciones dilatadas en paralelo con diferentes tasas de expansión. Esto permite capturar contexto a múltiples escalas sin reducir la resolución espacial.
- *Decoder* simple, que recupera la resolución espacial para producir una máscara del mismo tamaño que la imagen de entrada.

La arquitectura fue definida mediante el siguiente bloque de código genérico:

```
model = smp.DeepLabV3Plus(
    encoder_name="mobilenet_v2" or "resnet34",
    encoder_weights="imagenet",
    in_channels=3,
    classes=1
).to(DEVICE)
```

DeepLabV3+ con MobileNetV2

Este modelo fue concebido como una solución eficiente y ligera, especialmente adecuada para entornos con recursos limitados.

Características técnicas:

- *Backbone*: `mobilenet_v2` preentrenado en ImageNet.
- Parámetros: bajo número de parámetros, lo que lo hace ideal para entrenamiento rápido.
- Tamaño de entrada: 128x128 píxeles (elegido para reducir tiempos de entrenamiento).
- *ASPP*: incluido por defecto en la arquitectura para multiescala.
- Ventaja: menor consumo de GPU y memoria.
- Desventaja: menor calidad espacial en las segmentaciones finales debido al redimensionamiento.

Transformaciones aplicadas:

```
train_transform = A.Compose([
    A.Resize(128, 128),
    A.HorizontalFlip(),
    A.RandomBrightnessContrast(),
    A.Rotate(limit=15),
    A.Normalize(),
    ToTensorV2()
])
```

Resultados del entrenamiento

Una vez descrito el modelo, las Figuras 21 y 22 muestran 2 gráficas que representan de manera visual todo el proceso de entrenamiento del modelo DeepLabV3+ MobileNetV2. Una muestra la evolución de la pérdida y la otra la del IoU a lo largo de las épocas. En este modelo, la pérdida desciende bruscamente entre la primera y segunda época, estabilizándose rápidamente. Al tratarse de un modelo preentrenado, el aprendizaje adicional durante el entrenamiento es limitado, ya que converge en pocas épocas. No obstante, a pesar de este aprendizaje más

superficial en comparación con modelos entrenados desde cero, las métricas obtenidas son notablemente buenas. De igual forma que en el anterior gráfico, en el segundo, se aprecia una leve mejoría a lo largo de las épocas por la misma razón que se comenta en el primero. El resultado es muy bueno pero la realidad es que el modelo converge muy pronto y por tanto, no aprende tanto como los otros.

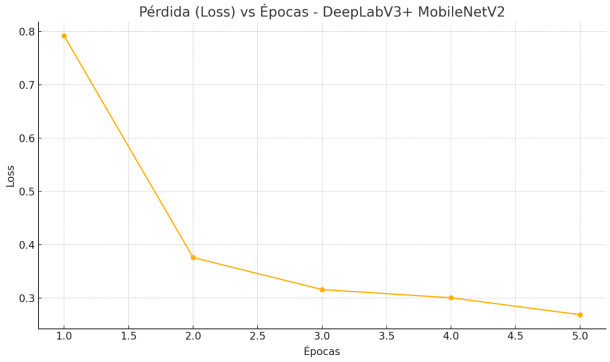

Figura 21: Gráfica 1 DeepLabV3+ MobileNetV2. Resultados del entrenamiento, pérdida con respecto al número de época

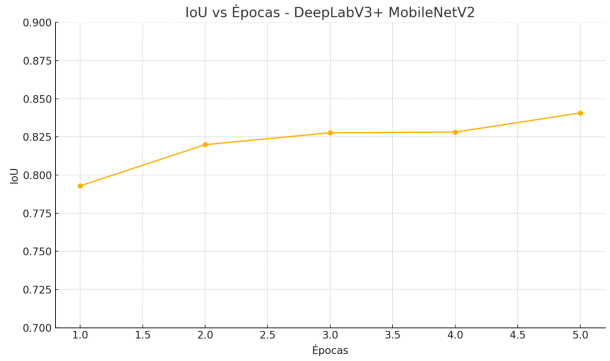

Figura 22: Gráfica 2 MobileNetV2. Resultados del entrenamiento, IoU con respecto al número de época

Ejemplo de predicción del modelo

Para una mejor visualización sobre lo que hace el modelo, la Figura 23 muestra un ejemplo de la predicción que realiza el modelo DeepLabV3+ con MobileNetV2. Se muestra la imagen original con la máscara generada superpuesta en color naranja, junto a un acercamiento que permite una mejor visualización. En este caso, la máscara presenta menor resolución, ya que

fue procesada a 128×128 píxeles y, al redimensionarse a 256×256, pierde nitidez:

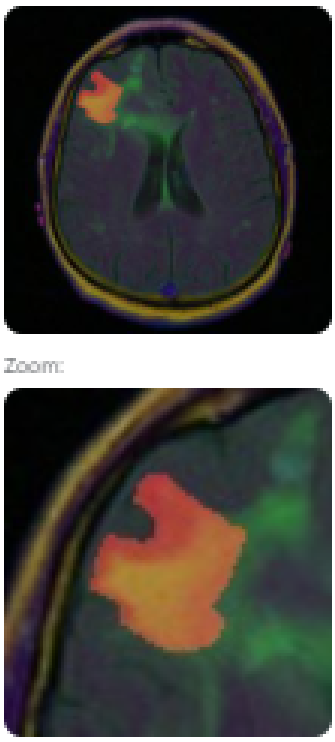

Figura 23: Predicción 3 DeepLabV3+ con MobilenetV2. Ejemplo de predicción con DeepLabV3+ con MobileNetV2 y zoom

DeepLabV3+ con ResNet34

Este modelo representa una alternativa más potente al anterior, utilizando una arquitectura más profunda para lograr segmentaciones más detalladas y suaves.

Características técnicas:

- *Backbone*: `resnet34` preentrenado en ImageNet.
- Parámetros: considerablemente más alto que MobileNetV2.
- Tamaño de entrada: 256x256 píxeles, lo que mejora la resolución de las predicciones.
- Ventaja: mayor capacidad para representar estructuras complejas y segmentar bordes con precisión.

- Desventaja: incremento sustancial del tiempo de entrenamiento y del consumo de recursos computacionales.

Transformaciones adaptadas:

```
train_transform = A.Compose([
    A.Resize(256, 256),
    A.HorizontalFlip(),
    A.RandomBrightnessContrast(),
    A.Rotate(limit=15),
    A.Normalize(),
    ToTensorV2()
])
```

Resultados del entrenamiento

Las Figuras 24 y 25 muestran 2 gráficas que representan de manera visual todo el proceso de entrenamiento del modelo DeepLabV3+ ResNet34. Se presentan las gráficas de pérdida e IoU a lo largo de las épocas, donde se observa una rápida convergencia, similar a la del modelo anterior. Al igual que en el gráfico previo, el IoU no muestra un aumento significativo con el paso de las épocas, lo que refuerza la conclusión de que el modelo aprende poco por época. No obstante, al tratarse de un modelo robusto, prediseñado y preentrenado, alcanza los mejores resultados en términos de precisión y mantiene un buen desempeño en IoU a pesar del aprendizaje limitado.

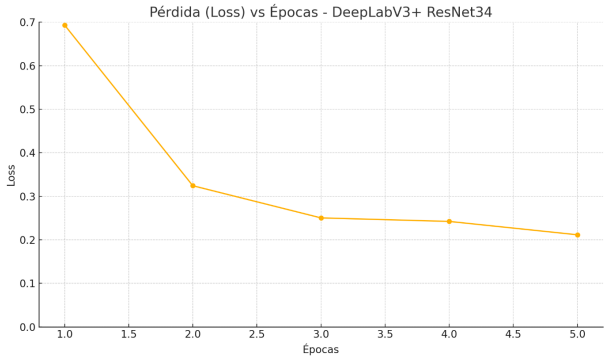

Figura 24: Gráfica 1 DeepLabV3+ ResNet34. Resultados del entrenamiento, pérdida con respecto al número de época

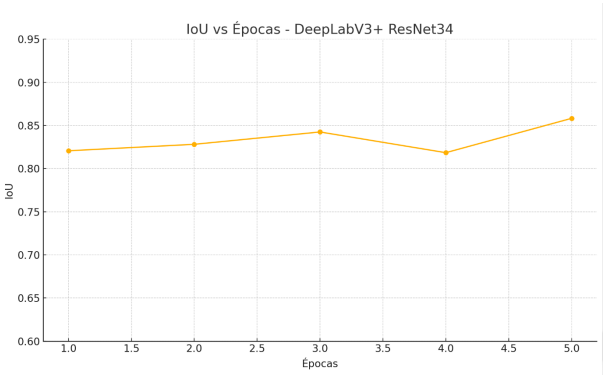

Figura 25: Gráfica 2 ResNet34. Resultados del entrenamiento, IoU con respecto al número de época

Ejemplo de predicción del modelo

Y para este último modelo, también se muestra un ejemplo de la predicción que realiza el modelo DeepLabV3+ con ResNet34 en la Figura 26. Se muestra la imagen original con la máscara generada superpuesta en color naranja, junto a un acercamiento que facilita su análisis. Este modelo ofrece la predicción más precisa de todas, gracias a su robustez y a las excelentes métricas obtenidas tanto en entrenamiento como en validación. En este caso, la segmentación del tumor cerebral es especialmente precisa:

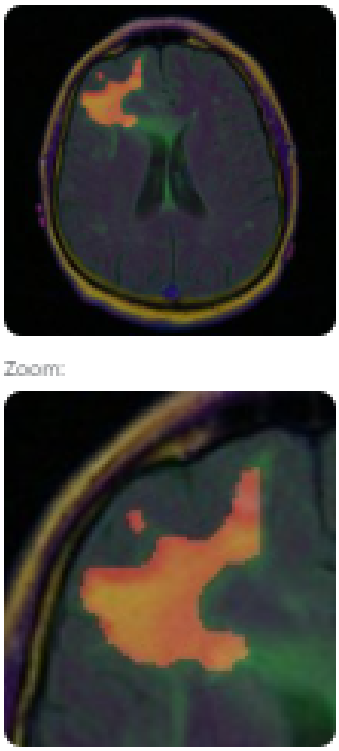

Figura 26: Predicción 4 DeepLabV3+ con ResNet34. Ejemplo de predicción con DeepLabV3+ con ResNet34 y zoom

Comparación estructural

Atributo	**MobileNetV2**	**ResNet34**
Tipo de arquitectura	Ligera	Profunda
Tamaño de entrada	128x128	256x256
Número de parámetros	Bajo	Medio-alto
Velocidad de entrenamiento	Rápida	Lenta
Precisión potencial	Moderada	Alta
Aplicaciones recomendadas	Producción ligera	Clínicas exigentes

En ambos casos, el uso del *Transfer Learning* ha resultado clave para minimizar el tiempo de entrenamiento y evitar el sobreajuste, aprovechando el conocimiento adquirido en el entrenamiento previo de los *backbones* en ImageNet.

4.3.1. Errores en la predicción de los modelos

En esta sección se muestran ejemplos en los que los modelos fallan en las predicciones del tumor cerebral. Estos errores pueden ser tanto una detección de un tumor que en realidad no está, como la no detección de un tumor que sí está.

Ejemplos de errores

En la Figura 27 se muestra la imagen de entrada seleccionada para realizar la predicción, en este caso errónea.

Figura 27: Predicción Errónea 1. Imagen de entrada

Por otra parte, se muestra la Figura 28, en la que se refleja un resultado de una predicción errónea para todos los modelos. Ninguno de ellos es capaz de encontrar el tumor correctamente, ya que los modelos DeepLabV3+ no consiguen encontrarlo, y los modelos U-Net y Attention U-Net muestran una predicción que es errónea:

Figura 28: Predicción Errónea 2. Resultados de las predicciones

En la Figura 29 se muestra la imagen de entrada seleccionada para realizar lasegunda predicción, en este caso también errónea.

Figura 29: Predicción Errónea 3. Imagen de entrada 2

Por otra parte, se muestra la Figura 30, en la que se refleja un resultado de una predicción errónea para todos los modelos. Ninguno de ellos es capaz de encontrar el tumor correctamente. En este caso el que está más cerca de segmentarlo bien es el DeepLabV3+ MobileNetV2 ya que encuentra las 2 áreas tumorales pero no segmentándolas correctamente. Por otra parte, el modelo ResNet34 directamente no encuentra el tumor y los modelos U-Net y Attention U-Net si encuentran una de las áreas tumorales pero no segmentándolas correctamente:

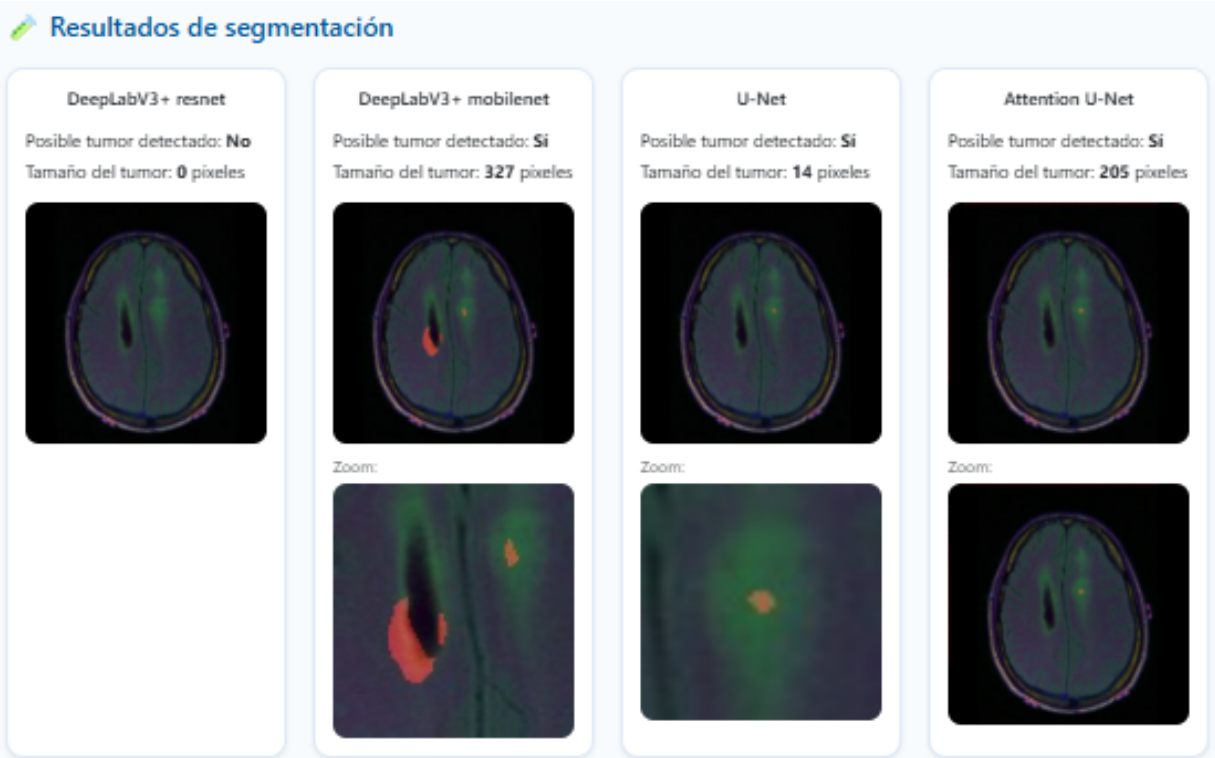

Figura 30: Predicción Errónea 4. Resultados de las predicciones 2

4.4. Comparación de resultados

Una vez entrenados todos los modelos, se realizó una comparativa exhaustiva utilizando las métricas previamente descritas: ***Dice Loss***, **IoU** y **tiempo total de entrenamiento**.

En primer lugar, las Figuras 31 y 32 enseñan 2 gráficas comparativas entre todos los modelos sobre la pérdida y el IoU con respecto al número de época en la que se pueden sacar las conclusiones que ya se han descrito previamente sobre cada uno de los modelos. Es interesante visualizar todos los resultados en la misma gráfica:

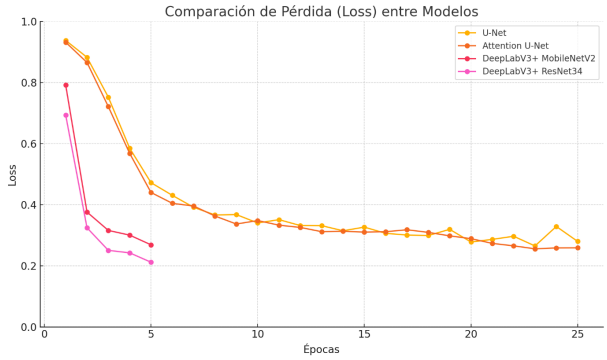

Figura 31: Gráfica 1 Comparativa general de modelos. Resultados del entrenamiento, *loss* con respecto al número de época

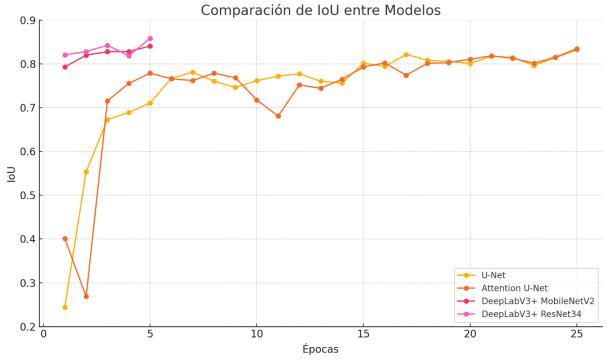

Figura 32: Gráfica 2 Comparativa general de modelos. Resultados del entrenamiento, IoU con respecto al número de época

En cuanto a los resultados finales obtenidos, se resumen en la Tabla 1:

Modelo	**Dice Loss**	**IoU**	**Tiempo de entrenamiento**
U-Net (segunda implementación)	0.2796	0.8319	1h 15min
DeepLabV3+ (128) MobileNetV2	0.2625	0.8407	35min
DeepLabV3+ (256) ResNet34	**0.2115**	**0.8582**	**5h**
Attention U-Net	0.2586	0.8346	1h 45min

Cuadro 1: Comparativa de resultados entre modelos

A partir de esta comparativa, se pueden destacar diversas observaciones clave:

- **DeepLabV3+ con ResNet34** ha sido el modelo con mejor rendimiento global, logrando la menor pérdida (*Dice Loss* = 0.2115) y el mayor IoU (0.8582). Esto es atribuible a su *backbone* ResNet34, una red profunda que permite capturar representaciones ricas y detalladas. Sin embargo, su tiempo de entrenamiento es considerablemente mayor (5 horas), lo que podría limitar su viabilidad en entornos con recursos computacionales limitados.

- **DeepLabV3+ con MobileNetV2**, en cambio, ofrece un excelente compromiso entre precisión y eficiencia. Con un IoU de 0.8407 y un tiempo de entrenamiento de solo 35 minutos, se posiciona como una opción ideal para despliegues rápidos o aplicaciones móviles. No obstante, al utilizar imágenes de tamaño 128x128, pierde cierto nivel de detalle en la segmentación.

- **Attention U-Net** demuestra una capacidad notable para segmentar regiones pequeñas o poco definidas. Con un IoU de 0.8346 y una pérdida menor que U-Net clásico, este modelo valida la eficacia de los bloques de atención, destacando especialmente en imágenes con estructuras complejas y bajo contraste.

- **U-Net (segunda implementación)** sirve como *baseline* para el resto de comparativas. Aunque sus métricas son las más discretas, ofrece resultados aceptables con una arquitectura sencilla y fácilmente entendible. Su tiempo de entrenamiento también es razonable, lo que lo convierte en una buena opción educativa o de prototipado.

En resumen:

- Si se busca la **mayor precisión**, la mejor elección es DeepLabV3+ con ResNet34.

- Si se prioriza la **velocidad y eficiencia**, destaca DeepLabV3+ con MobileNetV2.

- Si el foco está en **segmentaciones sensibles**, especialmente en tumores pequeños, Attention U-Net resulta superior.

- U-Net es útil como modelo base, de bajo coste y fácil de interpretar.

Este análisis confirma que no existe un único modelo ideal, sino que la selección depende del contexto de aplicación, recursos disponibles y objetivos clínicos concretos.

4.5. Ajustes y mejoras aplicadas

Durante el proceso de diseño e implementación de los modelos de segmentación, se llevaron a cabo diversos ajustes con el objetivo de mejorar tanto la precisión como la eficiencia del entrenamiento. Estas mejoras se enfocaron en aspectos clave como la arquitectura de red, las técnicas de regularización, los parámetros de entrenamiento y el preprocesamiento de datos.

Optimización del entrenamiento

Se implementaron varias estrategias orientadas a mejorar la convergencia de los modelos y reducir el riesgo de sobreajuste:

- **Uso exclusivo de *Dice Loss***: Esta función de pérdida fue elegida por su alta sensibilidad ante clases desbalanceadas, como ocurre en la segmentación de tumores. Su implementación se mantuvo constante en todos los modelos desarrollados desde cero y también en los modelos preentrenados, lo que permitió una comparación equitativa de resultados.

- **Regularización mediante AdamW**: En el caso del modelo Attention U-Net se optó por el optimizador AdamW, que desacopla el *weight decay* del gradiente. Esta mejora permitió una generalización superior, especialmente en estructuras pequeñas y poco contrastadas.

- **Tamaño del *batch* ajustado a recursos disponibles**: Se utilizó un valor común de 8 para el tamaño de lote. Este número ofreció un equilibrio entre aprovechamiento de memoria y estabilidad del entrenamiento.

- **Ajuste del número de épocas**: Según el modelo y su complejidad, el número de épocas osciló entre 5 y 25. Los modelos más ligeros como MobileNetV2 se entrenaron durante menos tiempo, mientras que arquitecturas más complejas como Attention U-Net o DeepLabV3+ con ResNet34 se entrenaron durante 20-25 épocas.

Mejoras en el preprocesamiento y transformaciones

Se refinaron las transformaciones aplicadas a los datos de entrada para mejorar la robustez del entrenamiento:

- **Tamaño de entrada personalizado por arquitectura**:
  - Modelos ligeros como MobileNetV2 se entrenaron con imágenes de `128x128` píxeles.
  - Modelos más complejos como Attention U-Net y ResNet34 utilizaron imágenes de `256x256` píxeles, mejorando la resolución de la segmentación final.
- **Albumentations para aumento de datos**: Se aplicaron transformaciones aleatorias como rotaciones, cambios de brillo y contraste, y volteo horizontal para aumentar la variedad del conjunto de datos y prevenir el sobreajuste.
- **Normalización al rango [-1, 1]**: Esta estrategia se utilizó en todos los modelos con el fin de estabilizar el entrenamiento y facilitar la convergencia de los pesos.

Cambios estructurales en modelos

Algunos de los ajustes más significativos afectaron directamente a la arquitectura de los modelos:

- **Incorporación de AG** en la versión Attention U-Net, que permitió al modelo centrarse en regiones relevantes de la imagen y suprimir ruido no informativo. Este cambio mejoró considerablemente la segmentación de tumores pequeños o con bordes difusos.
- **Uso de BatchNorm2d en U-Net versión 2**: A diferencia de la primera implementación, se introdujo normalización por lotes en cada bloque convolucional para estabilizar el flujo de gradientes y acelerar el aprendizaje.
- **Modularización de la arquitectura**: La segunda implementación de U-Net se diseñó de forma más estructurada, dividiendo la red en bloques reutilizables. Esto facilitó la integración de nuevas mejoras y fomentó el mantenimiento del código.

4.6. Evaluación final del rendimiento de los modelos

Una vez completado el entrenamiento y validación de todos los modelos, se procede a una evaluación final que consolida tanto las métricas cuantitativas como los aspectos cualitativos observados durante el desarrollo del proyecto. Esta evaluación tiene como objetivo extraer

conclusiones objetivas sobre el rendimiento de cada modelo y su aplicabilidad en entornos reales.

Resumen comparativo de rendimiento

A continuación se sintetizan los resultados más relevantes obtenidos por cada modelo:

- **U-Net (segunda implementación)**: Sirve como base de comparación. Obtuvo una *Dice Loss* de 0.2796 y un IoU de 0.8319. Aunque no lidera en precisión, es un modelo fácil de interpretar, con entrenamiento razonable (1h 15min) y buena generalización en estructuras claras.

- **Attention U-Net**: Se posiciona como una mejora directa del U-Net, gracias a los mecanismos de atención. Con una *Dice Loss* de 0.2586 e IoU de 0.8346, destaca especialmente en la detección de tumores pequeños o mal definidos. Su entrenamiento fue de 1h 45min, lo que representa un equilibrio adecuado entre precisión y coste computacional.

- **DeepLabV3+ con MobileNetV2**: Modelo eficiente y ligero, ideal para sistemas de bajo consumo o con restricciones de tiempo. Su IoU fue de 0.8407 y su entrenamiento apenas duró 35 minutos. Sin embargo, el tamaño reducido de entrada (128x128) compromete la calidad espacial de las segmentaciones.

- **DeepLabV3+ con ResNet34**: Modelo más preciso del estudio, con una *Dice Loss* de 0.2115 y un IoU de 0.8582. Utiliza imágenes de 256x256 y un *backbone* profundo que permite captar mejor la estructura de los tumores. Su principal desventaja es el tiempo de entrenamiento (5 horas) y el mayor consumo de recursos.

Consideraciones sobre la utilidad práctica

Además de las métricas, se ha valorado la calidad visual de las máscaras generadas y el comportamiento de los modelos frente a distintos tipos de tumores. Attention U-Net, por ejemplo, ha demostrado ser especialmente eficaz en la segmentación de masas irregulares o de bajo contraste, mientras que DeepLabV3+ con ResNet34 proporciona segmentaciones más suaves y ajustadas al contorno real del tumor.

Conclusión

La evaluación final pone de manifiesto que no existe un modelo único ideal. Cada arquitectura presenta ventajas y limitaciones que deben ponderarse según el contexto de aplicación:

- Si se busca **máxima precisión**, la opción recomendada es **DeepLabV3+ con ResNet34**.
- Para **sistemas con recursos limitados**, **MobileNetV2** ofrece un buen rendimiento con bajo coste.
- Si el interés está en **tumores difíciles de identificar**, **Attention U-Net** destaca por su sensibilidad.
- El modelo **U-Net clásico** es útil como punto de partida o base experimental.

Esta evaluación global servirá como guía para seleccionar el modelo más adecuado en función del entorno clínico, los recursos disponibles y los requisitos del sistema final

5

Interfaz de usuario y aplicación final

Este capítulo describe el diseño, desarrollo e integración de la aplicación web que permite interactuar con el sistema de segmentación automática de tumores cerebrales. El objetivo principal de esta aplicación es proporcionar una herramienta accesible, intuitiva y funcional para usuarios que deseen cargar MRIs y obtener segmentaciones precisas sin necesidad de conocimientos técnicos en IA.

Se presentan los componentes principales del sistema: la interfaz de usuario, el *backend* desarrollado con Flask, y el *pipeline* de inferencia que conecta el modelo entrenado con la imagen cargada por el usuario. Además, se explican los flujos de trabajo, las funcionalidades implementadas y las decisiones de diseño adoptadas para garantizar la usabilidad y la eficiencia del sistema.

Este capítulo también incluye capturas de pantalla, ejemplos de uso y consideraciones prácticas sobre el despliegue del sistema.

5.1. Requisitos funcionales

La aplicación web desarrollada tiene como objetivo proporcionar una herramienta sencilla e intuitiva que permita a profesionales del ámbito clínico aprovechar las capacidades de los modelos de segmentación desarrollados. Para ello, se han definido los siguientes requisitos funcionales:

- **Subida de imágenes:** El usuario debe poder subir MRIs cerebrales en formato `.tif`.
- **Selección del modelo:** El sistema permite seleccionar entre los distintos modelos implementados (U-Net, Attention U-Net, DeepLabV3+ con MobileNetV2 y ResNet34).

- **Ejecución del modelo:** La aplicación debe ejecutar el modelo seleccionado sobre la imagen proporcionada, generando una máscara segmentada correspondiente a la región tumoral.
- **Visualización del resultado:** El sistema debe mostrar la imagen original, la máscara generada y la superposición de ambas de forma clara y comprensible.
- **Zoom automático:** Debe generarse una versión recortada de la imagen centrada en la región segmentada, con mayor nivel de detalle.
- **Extracción de estadísticas:** La aplicación debe calcular el área segmentada (en número de píxeles) y proporcionar información básica sobre la región tumoral detectada.
- **Descarga de informe:** El usuario debe poder descargar un informe en formato PDF que contenga la imagen segmentada, el área estimada, el modelo utilizado y las estadísticas obtenidas.

Estos requisitos buscan garantizar la utilidad práctica de la aplicación, permitiendo su uso tanto en contextos académicos como en posibles entornos clínicos experimentales.

5.2. Diseño de la interfaz

La interfaz de usuario desarrollada para esta aplicación tiene como principal objetivo ofrecer una experiencia clara, intuitiva y accesible, especialmente para profesionales médicos sin conocimientos técnicos en inteligencia artificial. El diseño se centra en simplificar el flujo de trabajo desde la carga de la imagen hasta la visualización de los resultados de la segmentación tumoral.

Estructura de la interfaz

La interfaz está compuesta por los siguientes paneles funcionales:

- **Zona de carga:** permite al usuario subir una imagen médica desde su ordenador o arrastrarla directamente al área habilitada.
- **Visualización principal:** tras la inferencia, se muestran la imagen original y las máscaras generadas por cada modelo entrenado (U-Net, Attention U-Net, DeepLabV3+).

- **Zoom automático:** si se detecta una región tumoral, se genera y muestra una vista aumentada de la zona segmentada, para facilitar su inspección visual.
- **Panel de resultados:** incluye estadísticas básicas como:
  - Detección del tumor (sí/no).
  - Área estimada del tumor en píxeles.
- **Botón de descarga de informe:** permite generar un informe en PDF con los resultados de la segmentación, incluyendo imágenes, métricas y el nombre del modelo utilizado.

Tecnologías utilizadas

- **HTML y CSS:** para la estructura y el diseño visual de la interfaz.
- **JavaScript:** para permitir interacción dinámica con el usuario (como previsualización de imágenes y carga asincrónica).
- **Flask:** como *backend* ligero que gestiona la lógica del servidor, conecta con los modelos IA y devuelve los resultados al *frontend*.

Estilo visual

El aspecto visual se ha mantenido sencillo y funcional, priorizando la legibilidad y la organización. Los colores neutros predominan en la interfaz para no interferir con la visualización de las imágenes médicas. La disposición en columnas facilita la navegación y reduce la carga cognitiva del usuario.

5.3. Integración con el modelo IA

El *backend* de la aplicación está desarrollado en Python utilizando Flask. Al iniciar la aplicación, se cargan automáticamente todos los modelos previamente entrenados (archivos `.pth`) junto a sus parámetros de normalización y umbral de binarización. Para ello, se emplea una función llamada `get_model_by_name()`, que reconstruye la arquitectura necesaria según el nombre del modelo:

```
def get_model_by_name(name):
    if name == "unet":
        return UNet()
    elif name == "attention_unet":
        return AttentionUNet()
    elif name == "deeplab_mobilenet":
        return smp.DeepLabV3Plus(encoder_name="mobilenet_v2",
   ...)
    elif name == "deeplab_resnet":
        return smp.DeepLabV3Plus(encoder_name="resnet34", ...)
```

La inferencia se realiza aplicando las transformaciones necesarias a la imagen (redimensionamiento, normalización y conversión a tensor), y posteriormente ejecutando el modelo correspondiente. La máscara generada se superpone a la imagen original mediante OpenCV, y se genera una versión aumentada de la región segmentada si se detecta un tumor.

5.4. Casos de uso y ejemplos

Durante el proceso de validación de la aplicación web, se realizaron múltiples pruebas utilizando imágenes reales del conjunto de datos empleado en el entrenamiento de los modelos. El objetivo fue comprobar la funcionalidad de cada uno de los módulos del sistema y asegurar que los resultados ofrecidos eran coherentes y útiles desde el punto de vista clínico.

A continuación se describen algunos de los casos de uso más representativos:

Caso 1: Imagen con tumor claramente visible

El usuario carga una imagen donde el tumor se presenta de forma nítida y con buena delimitación. Los modelos segmentan correctamente la masa tumoral. La interfaz genera una superposición clara sobre la imagen original y un recorte ampliado de la zona afectada. Se calcula el área tumoral y se genera un informe en PDF, incluyendo nombre del modelo, área detectada, imagen segmentada y estadística básica.

Al entrar a la aplicación se ve la imagen de la Figura 33:

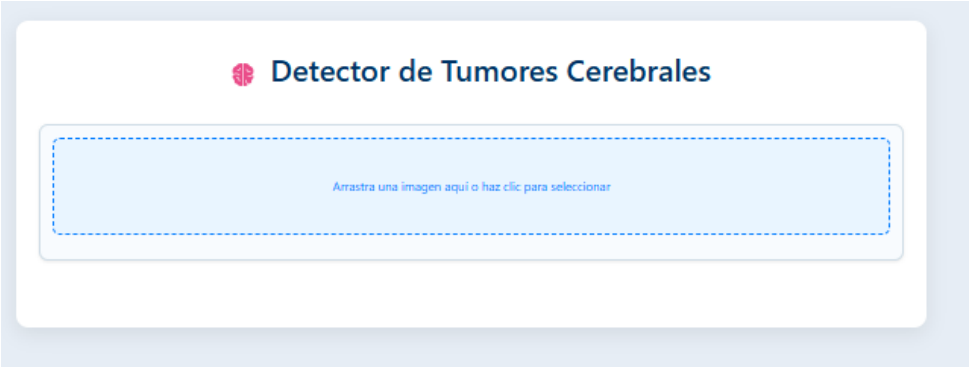

Figura 33: Imagen 1 Interfaz. Primera vista de la interfaz web

Se arrastra una MRI o se selecciona una imagen de las carpetas del PC correspondiente.

Aparece la pantalla de la Figura 34, en la que está:

- La imagen de entrada.
- Los resultados de la segmentación con los siguientes datos:
    - La imagen completa con la máscara de segmentación superpuesta en color rojo.
    - Zoom sobre dicha máscara.
    - Tumor detectado: Sí/No.
    - Tamaño del tumor detectado en píxeles.
- Además, podemos descargar un informe en PDF sobre estos datos.

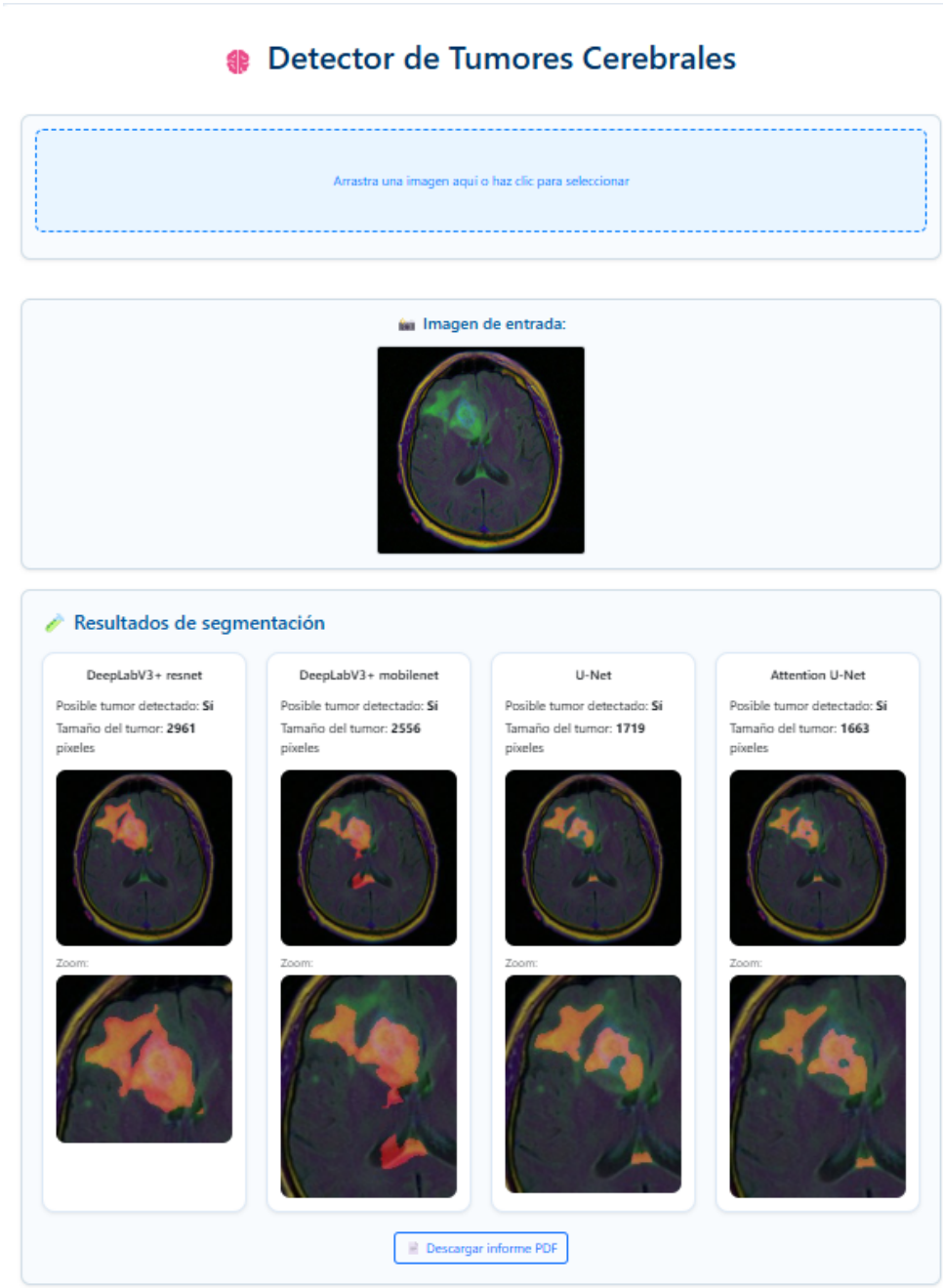

Figura 34: Imagen 2 Interfaz. Vista de las predicciones de los modelos, zooms y estadísticas básicas de cada modelo

Descarga del informe generado: Para consultar el informe detallado correspondiente al caso de uso 1, puedes descargar el siguiente documento PDF:

[Descargar Informe de Segmentación - Caso 1]

Caso 2: Imagen sin tumor

En la Figura 35, se realiza una prueba con una imagen que no contiene tumor visible. Algunos modelos como U-Net generan pequeñas regiones segmentadas erróneamente. Esto permite comprobar la sensibilidad de cada modelo.

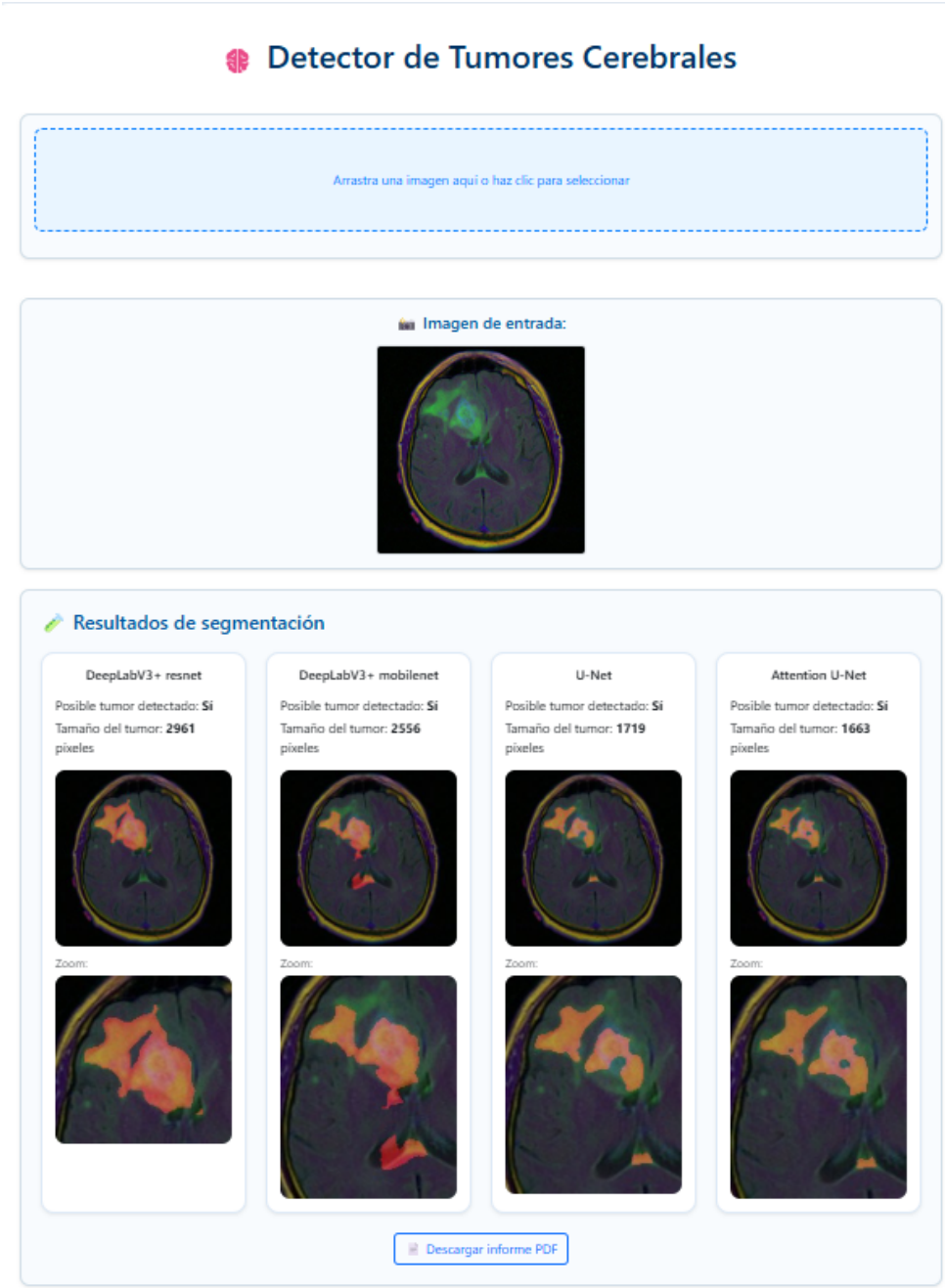

Figura 35: Imagen 3 Interfaz. Vista de las predicciones de los modelos, zooms y estadísticas básicas de cada modelo con una imagen sin tumor

Caso 3: Tumor pequeño o de contorno difuso

Este caso es bastante más complicado, hay menos imágenes en el conjunto de datos de este tipo, y los modelos U-Net y Attention U-Net pueden fallar precisamente por esto. En estos casos, la vista con zoom es muy útil para poder visualizar el tamaño del tumor.

En la Figura 36 se muestra el **caso en el que la segmentación es correcta de todos los modelos:**

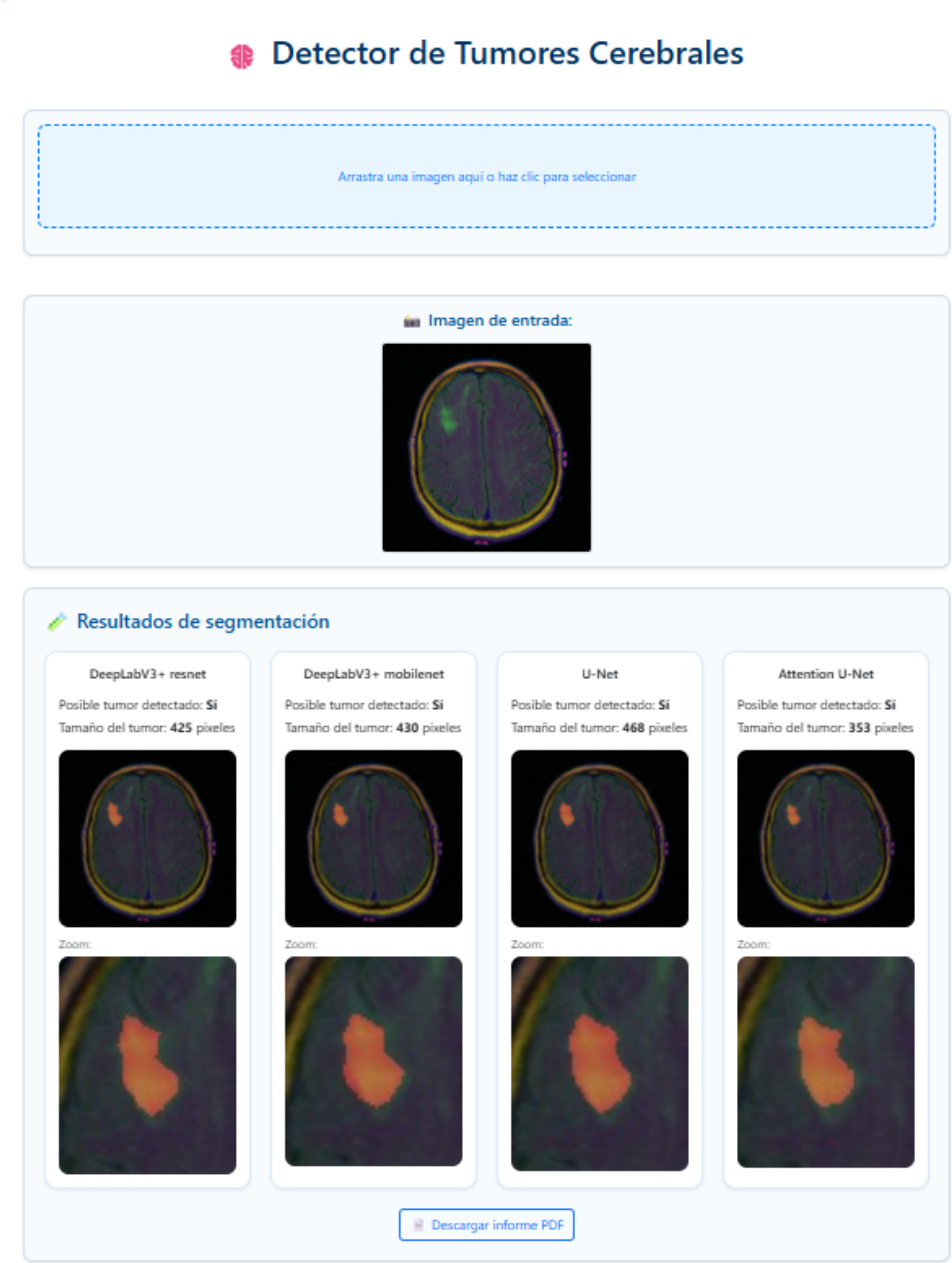

Figura 36: Imagen 4 Interfaz. Vista de las predicciones de los modelos, zooms y estadísticas básicas de cada modelo con tumores pequeños, casos de éxito en todos los modelos

En la Figura 37 se muestra el **caso en el que DeepLabV3+ ResNet y MobileNet son capaces de identificar que no hay tumor:**

Figura 37: Imagen 5 Interfaz. Vista de las predicciones de los modelos, zooms y estadísticas básicas de cada modelo con tumores pequeños, casos de éxito en los modelos DeepLabV3+

En estos casos en los que no hay tumor, pero puede haber alguna parte confusa de la imagen, el modelo U-Net o incluso el modelo Attention U-Net pueden fallar al ser modelos entrenado. No obstante, el modelo Attention U-Net tiene mejores resultados en la mayoría de predicciones que el modelo U-Net clásico.

Por otra parte, se visualiza que los modelos DeepLabV3+ tienen resultados muy buenos en

prácticamente todos los casos.

En todos los casos se valida:

- El funcionamiento correcto de la carga de imágenes.
- La ejecución de los cuatro modelos integrados.
- La visualización de resultados y máscaras.
- La generación del informe en PDF al pulsar sobre el botón correspondiente.

Estos ejemplos demuestran la versatilidad de la aplicación y su potencial utilidad como herramienta de apoyo al diagnóstico clínico.

6

Conclusiones y Líneas Futuras

En este capítulo se recogen las principales conclusiones extraídas tras el desarrollo del proyecto, poniendo en perspectiva los objetivos planteados inicialmente y los resultados alcanzados. Se reflexiona sobre las fortalezas y limitaciones del sistema de segmentación desarrollado, así como sobre el impacto que podría tener en contextos clínicos reales.

Asimismo, se presentan diversas líneas de trabajo futuro que podrían mejorar o ampliar este proyecto. Estas propuestas incluyen desde mejoras técnicas en los modelos hasta estrategias de validación más exhaustivas, integración con sistemas hospitalarios o adaptaciones para otras modalidades de imagen médica.

Este apartado busca no solo cerrar el trabajo realizado, sino también ofrecer una base sólida para investigaciones posteriores o aplicaciones prácticas más avanzadas.

6.1. Conclusiones generales del trabajo

Este Trabajo de Fin de Grado ha permitido comprobar la eficacia del uso de redes neuronales profundas aplicadas a la segmentación de tumores cerebrales en MRIs. A lo largo de este proyecto, se han explorado diferentes arquitecturas, tanto diseñadas desde cero, como U-Net o Attention U-Net, como basadas en modelos prediseñados, como DeepLabV3+ ResNet34 y MobileNetV2. Gracias a esto, se ha podido realizar un análisis comparativo entre todas las arquitecturas en base a su rendimiento.

Los resultados obtenidos muestran que los modelos DeepLabV3+, en concreto el que tiene como *backbone* ResNet34, ofrecen una precisión superior en la segmentación, logrando unas métricas globales muy buenas, aunque a costa de tiempos de entrenamiento mayores y mayor

consumo de recursos. Por otro lado, el modelo DeepLabV3+ MobileNetV2 demuestra ser una alternativa eficiente para entornos con restricciones de capacidad de cómputo, manteniendo un rendimiento decente, aunque está preparado para imágenes de entrada de 128×128 píxeles y, por tanto, genera máscaras de menor calidad. En el caso de Attention U-Net, es un modelo entrenado desde cero, algo menos eficaz que los modelos preentrenados, pero con ciertas mejoras respecto al U-Net clásico, que sirve de base experimental y es un buen modelo si se requiere de una solución básica.

A nivel de metodología, se ha llevado a cabo un preprocesamiento del conjunto de datos, incluyendo normalización, binarización y *data augmentation*, lo que ha permitido mejorar la generalización de los modelos. Se han aplicado técnicas de *Transfer Learning* en los modelos DeepLabV3+, los cuales no se entrenan directamente desde cero, sino que aprovechan los pesos preentrenados y la información adquirida por el *encoder* ResNet34 o MobileNetV2 para la segmentación de imágenes de tumores cerebrales.

Otro aspecto importante ha sido la creación de una aplicación web práctica y funcional, capaz de ejecutar los modelos desarrollados y presentar los resultados de forma visual e intuitiva para el usuario final. La posibilidad de generar informes automáticos y calcular estadísticas básicas convierte esta herramienta en un sistema de apoyo que en el futuro podría llegar a ser útil para profesionales clínicos.

En resumen, el trabajo realizado demuestra que la IA, cuando se implementa adecuadamente, puede convertirse en una herramienta potente (que no una solución) para la asistencia médica, mejorando tanto la precisión como la eficiencia en el diagnóstico de patologías complejas como los tumores cerebrales.

6.2. Propuestas de mejora y desarrollo futuro

A lo largo del desarrollo de este trabajo han surgido diversas ideas que podrían contribuir a mejorar y ampliar el sistema presentado, tanto desde el punto de vista técnico como clínico. A continuación se describen algunas de las principales líneas futuras que podrían explorarse:

1. Mayor tamaño de imagen y resolución: Uno de los principales retos en este trabajo ha sido el uso de imágenes de tamaño reducido, lo que limita la calidad de las segmentaciones

obtenidas. Sería deseable utilizar imágenes con mayor definición y resolución, que pueden encontrarse en grandes bases de datos médicas como los sistemas PACS y VNA del Servicio Andaluz de Salud, que dispone de uno de los mayores repositorios de imágenes médicas del mundo [18].

2. Validación por expertos clínicos: Es fundamental incorporar a profesionales médicos en el proceso de validación de los modelos. Esto permitiría realizar ajustes humanos post-entrenamiento, corrigiendo errores y mejorando la sensibilidad del sistema en base al conocimiento experto.

3. Diagnóstico asistido por doble vía: Una propuesta interesante sería implementar un método similar al doble ciego, donde el diagnóstico se realice de forma independiente tanto por el modelo de IA como por un profesional clínico. Si ambos coinciden, se podría asumir una alta probabilidad de certeza diagnóstica. En caso de discrepancia, se escalaría el caso a un nivel superior con profesionales más experimentados. Este enfoque podría optimizar recursos humanos y reducir costes manteniendo la seguridad del paciente [31].

4. Adaptación ética y social a la IA en salud: Actualmente, existe una barrera social y psicológica para aceptar que una IA pueda participar en decisiones médicas. La confianza en estos sistemas dependerá de su fiabilidad, transparencia y capacidad para reducir errores médicos. A medida que la IA avance y se perfeccionen sus mecanismos de aprendizaje, los modelos podrán alcanzar o incluso superar la capacidad diagnóstica humana en ciertas áreas específicas [25, 12].

5. Entrenamiento con mejores recursos e infraestructuras: Los modelos de aprendizaje profundo requieren infraestructuras adecuadas para su entrenamiento. El uso de GPU potentes o incluso supercomputadores permitiría entrenar modelos más complejos sobre conjuntos de datos de gran escala. El Servicio Andaluz de Salud, con sus inmensos repositorios, representa una oportunidad estratégica para entrenar modelos de segmentación con aplicación directa en la práctica clínica real [30].

En definitiva, la IA debe ser entendida no como una solución definitiva, sino como una herramienta complementaria que potencie la capacidad del profesional sanitario. A medio plazo, se espera que emerjan perfiles híbridos en el ámbito de la salud, con conocimientos técnicos

en IA y formación médica, que lideren esta transformación en los entornos hospitalarios.

Referencias

[1] *Aprendizaje profundo*. https://es.wikipedia.org/wiki/Aprendizaje_profundo. Wikipedia. 2025.

[2] Ekaba Bisong. *Google Colaboratory*. Apress, 2019, págs. 59-64. URL: https://link.springer.com/chapter/10.1007/978-1-4842-4470-8_7.

[3] Bunyos. *MRI Brain Scan*. Imagen de resonancia magnética del cerebro humano. © Bunyos / Dreamstime.com. 2013. URL: https://thumbs.dreamstime.com/b/exploraci%C3%B3n-de-cerebro-de-mri-28673367.jpg.

[4] Alexander Buslaev, Vladimir I. Iglovikov et al. "Albumentations: Fast and Flexible Image Augmentations". En: *Information* 11.2 (2020), pág. 125. URL: https://www.mdpi.com/2078-2489/11/2/125.

[5] Sijing Cai, Yi Wu y Guannan Chen. *Figura 1: Diagrama de la estructura de UNet*. Imagen extraída del artículo .ᴬ Novel Elastomeric UNet for Medical Image Segmentation", Frontiers in Aging Neuroscience, 14:841297. DOI: 10.3389/fnagi.2022.841297. 2022. URL: https://www.frontiersin.org/files/Articles/841297/fnagi-14-841297-HTML-r2/image_m/fnagi-14-841297-g001.jpg.

[6] Scott Chacon y Ben Straub. *Pro Git*. Apress, 2014. URL: https://git-scm.com/book/en/v2.

[7] Casper da Costa-Luis. *tqdm: A Fast, Extensible Progress Bar for Python*. https://github.com/tqdm/tqdm. 2023.

[8] DataScientest. *U-NET: todo lo que tienes que saber sobre la red neuronal de Computer Vision*. https://datascientest.com/es/u-net-lo-que-tienes-que-saber. 2024.

[9] Anbu Valluvan Devadasan, Saptarshi Sengupta y Mohammad Masum. *Figura 1: Arquitectura del modelo de aprendizaje profundo para monitoreo no invasivo de la presión arterial*. Imagen extraída del artículo "Deep Learning for Non-Invasive Blood Pressure Monitoring: Model Performance and Quantization Trade-Offs", Electronics, 14(7), 1300. DOI: 10.3390/electronics14071300. 2025. URL: https://www.mdpi.com/electronics/

electronics-12-01300/article_deploy/html/images/electronics-12-01300-g001.png.

[10] *FPDF for Python*. https://pyfpdf.github.io/fpdf2/. 2023.

[11] Freepik. *Imagen médica de rayos X de cabeza humana y cerebro: concepto de diagnóstico en neurología*. Imagen de Freepik. 2025. URL: https://img.freepik.com/fotos-premium/imagen-medica-rayos-x-cabeza-humana-cerebro-concepto-diagnostico-neurologia_928695-3011.jpg?w=996.

[12] Matthew A Gianfrancesco et al. "Building trust in artificial intelligence for healthcare". En: *JAMA* 321.3 (2019), págs. 215-216.

[13] *Google Colab*. https://colab.research.google.com/. Google. 2025.

[14] *Google Drive Help*. https://support.google.com/drive/. 2023.

[15] Miguel Grinberg. *Flask Web Development: Developing Web Applications with Python*. O'Reilly Media, 2018. URL: https://flask.palletsprojects.com/.

[16] John D. Hunter. "Matplotlib: A 2D graphics environment". En: *Computing in science & engineering* 9.3 (2007), págs. 90-95. URL: https://ieeexplore.ieee.org/document/4160265.

[17] *Inteligencia Artificial*. https://es.wikipedia.org/wiki/Inteligencia_artificial. Wikipedia. 2025.

[18] Junta de Andalucía. *Sistemas PACS y VNA en el Servicio Andaluz de Salud*. https://www.juntadeandalucia.es/servicioandaluzdesalud. Acceso en mayo de 2025. 2022.

[19] Kaggle. *Kaggle: Your Machine Learning and Data Science Community*. 2025. URL: https://www.kaggle.com.

[20] *Kaggle: Your Machine Learning and Data Science Community*. https://www.kaggle.com/. 2023.

[21] Thomas Kluyver et al. *Jupyter Notebooks—a publishing format for reproducible computational workflows*. IOS Press, 2016. URL: https://eprints.soton.ac.uk/403913/1/papermache.pdf.

[22] Aida Fernández-Lebrero Manuel Arias Emilio Rodríguez-Castro e Iria A. López-Dequidt. *Figura 1a: Tomografía computarizada cerebral (inicial), corte axial.* Imagen extraída del artículo "Hypertrophic pachymeningitis secondary to IgG4-related disease: Case report and review of the literature". 2015. URL: https://www.researchgate.net/profile/Manuel-Arias-8/publication/283736278/figure/fig1/AS:339879159451676@1458044914702/Figura-1-a-Tomografia-computarizada-cerebral-inicial-corte-axial-donde-se-observa.png.

[23] Microsoft. *Visual Studio Code*. 2023. URL: https://code.visualstudio.com/.

[24] Fausto Milletari, Nassir Navab y Seyed-Ahmad Ahmadi. "V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation". En: *arXiv preprint arXiv:1606.04797* (2016). URL: https://arxiv.org/abs/1606.04797.

[25] Jessica Morley et al. "Ethical and social implications of AI in medicine". En: *The Lancet Digital Health* 2.4 (2020), e205-e207.

[26] Ozan Oktay et al. "Attention U-Net: Learning Where to Look for the Pancreas". En: *arXiv preprint arXiv:1804.03999* (2018). URL: https://arxiv.org/abs/1804.03999.

[27] *OpenCV*. https://opencv.org/. OpenCV. 2025.

[28] *PyTorch*. https://pytorch.org/. PyTorch. 2025.

[29] Md Rahman y Yang Wang. "Optimizing Intersection-over-Union in Deep Neural Networks for Image Segmentation". En: *arXiv preprint arXiv:1606.05650* (2016). URL: https://arxiv.org/abs/1606.05650.

[30] Servicio Andaluz de Salud. *Infraestructuras Big Data en el SAS: una apuesta por la medicina personalizada*. https://www.sspa.juntadeandalucia.es. Acceso en mayo de 2025. 2023.

[31] Eric J Topol. "Artificial intelligence for clinical decision support: challenges and opportunities". En: *Nature Medicine* 25.1 (2019), págs. 44-56.

[32] *U-Net*. https://es.wikipedia.org/wiki/U-Net. Wikipedia. 2025.

[33] Y. Zhang, H. Li y X. Wang. "Improved U-Net neural network structure". En: (2021). ResearchGate.

Apéndice A
Manual de Instalación

A continuación, se describe el procedimiento necesario para instalar y ejecutar la aplicación desarrollada, basada en el framework Flask, en un entorno local sobre el sistema operativo Windows.

1. En primer lugar, se debe acceder a la carpeta del proyecto, denominada `tfg_flask_app`, donde se encuentran los archivos fuente de la aplicación.

2. Es necesario verificar que Python está instalado en el sistema. Para ello, se puede abrir una terminal (por ejemplo, escribiendo `CMD` en el buscador de Windows dentro de la carpeta del proyecto) y ejecutar el siguiente comando:

   ```
   python --version
   ```

   En caso de que Python no esté instalado, se puede descargar desde el sitio oficial: https://www.python.org/downloads/. Durante la instalación, es fundamental marcar la opción `Add Python to PATH` para que el intérprete sea accesible desde la terminal.

3. A continuación, se procede a la creación de un entorno virtual que permita aislar las dependencias del proyecto. Esto se realiza ejecutando los siguientes comandos en la terminal:

   ```
   python -m venv venv
   venv\Scripts\activate
   ```

4. Con el entorno virtual activado, se deben instalar las dependencias especificadas en el archivo `requirements.txt`, mediante el comando:

```
pip install -r requirements.txt
```

5. Finalmente, se ejecuta la aplicación utilizando el siguiente comando:

   ```
   python app.py
   ```

   Si la instalación ha sido exitosa, en la terminal aparecerá un mensaje indicando que el servidor está en ejecución en la dirección local:

   ```
   Running on http://127.0.0.1:5000/
   ```

6. Para acceder a la interfaz web de la aplicación, basta con abrir un navegador e introducir en la barra de direcciones la URL indicada anteriormente: http://127.0.0.1:5000/.

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga

E.T.S. DE INGENIERÍA INFORMÁTICA