



UNIVERSIDAD DE GRANADA

TRABAJO FIN DE GRADO INGENIERÍA INFORMÁTICA

Diagnóstico de tumores cerebrales a partir de IRM mediante aprendizaje profundo

Diseño e implementación de arquitecturas neuronales para la
clasificación y la segmentación

Autor
Jaime Castillo Uclés

Directora
Rosa María Rodríguez Sánchez



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE
TELECOMUNICACIÓN

—
Granada, Junio de 2024



BraTS UGR

Una interfaz de uso médico para la clasificación y
segmentación de tumores cerebrales

Diagnóstico de tumores cerebrales a partir de IRM mediante aprendizaje profundo : Diseño e implementación de arquitecturas neuronales para la clasificación y la segmentación

Jaime Castillo Uclés

Palabras clave: clasificación, segmentación, red convolucional, U-net, conexiones residuales, resonancia, slice, tumor, decoder, encoder, representación latente, aprendizaje no supervisado, transformer y autoencoder.

Resumen

Los tumores cerebrales son uno de los cánceres más letales y generalmente desafiantes a la hora de diagnosticar. En este proyecto exploraremos las capacidades que ofrecen las técnicas de aprendizaje profundo sobre imágenes de resonancia magnética dando como resultado la implementación de arquitecturas para la ayuda en su diagnóstico y una interfaz a modo de demostración de uso médico.

El desafío es obtener una interpretación más profunda de un órgano tan complejo como es el cerebro para sus formas de cáncer. El diagnóstico de un tumor cerebral tiene distintas tareas asociadas: la **segmentación** de estos tumores tiene los objetivos de agilizar el proceso de segmentación manual al personal médico, ser un elemento de seguridad en este proceso y eventualmente poder identificar especialmente los que pueden pasar desapercibidos incluso por un especialista humano. La **clasificación** entre los dos principales tipos de tumores, **glioblastomas** (más agresivos) y **meningiomas** (menos agresivos), puede guiar al personal médico en los tumores más difíciles de distinguir a tomar decisiones en base al tratamiento a seguir. De forma teórica y siguiendo la línea del trabajo se propone la solución a una tercera tarea: la **predicción de la evolución** de los tumores en una instancia a corto plazo de tiempo.

Estas tareas que conforman el diagnóstico de los tumores cerebrales se incluirán en este trabajo y se pondrán en valor junto con el estado del arte.

Brain tumor diagnosis from MRI images using Deep Learning : Design and neuronal architecture implementation for classification and segmentation

Jaime Castillo Uclés

Keywords: classification, segmentation, convolutional network, U-net, residual connections, MRI, slice, tumor, decoder, encoder, latent representation, unsupervised learning, transformer and autoencoder.

Abstract

Brain tumors are among the most lethal and generally challenging cancers to diagnose. In this project, we will explore the capabilities offered by deep learning techniques on magnetic resonance images, resulting in the implementation of architectures to assist in diagnosis and a demonstration interface for medical use.

The challenge is to obtain a deeper understanding of an organ as complex as the brain for its cancer forms. The diagnosis of a brain tumor involves various tasks: the **segmentation** of these tumors aims to expedite the manual segmentation process for medical staff, provide a safety element in this process, and eventually identify those that may go unnoticed even by a human specialist. The **classification** between the two main types of tumors, **glioblastomas** (more aggressive) and **meningiomas** (less aggressive), can guide medical staff in the more difficult-to-distinguish tumors to make decisions based on the treatment to follow. Theoretically, and in line with the work, a solution is proposed for a third task: the **prediction of the evolution** of the tumors in a short-term time frame.

These tasks that comprise the diagnosis of brain tumors will be included in this work and will be highlighted along with the state of the art.

Yo, **Jaime Castillo Uclés**, alumno de la titulación Grado en Ingeniería Informática de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 45924736S, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Jaime Castillo Uclés

Granada a 24 de Junio de 2024.

Dra. **Rosa María Rodríguez Sánchez**, Profesora del Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado *Diagnóstico de tumores cerebrales a partir de IRM mediante aprendizaje profundo , Diseño e implementación de arquitecturas neuronales para la clasificación y la segmentación* , ha sido realizado bajo su supervisión por **Jaime Castillo Uclés**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 25 de Junio de 2024.

La directora:

Agradecimientos

Desde el inicio de mi etapa universitaria y en este tan corto período de tiempo, puedo decir que los cambios en mí, en magnitud, han sido positivamente radicales. En este trabajo que representa el final de esta primera etapa, sólo puedo estar profundamente agradecido por todas las personas que se han cruzado de forma positiva en algún punto conmigo. A mi familia, que siempre ha creído en mí, dándome la oportunidad de poder formarme incluso cuando más me ha costado afrontar este proceso de cambio; a los buenos amigos que he hecho en el camino y a todos los buenos profesores de los que he recibido clase o apoyo, que han sido inspiradores para mí.

Índice general

1. Introducción	1
1.1. Objetivos	1
1.2. Metodología	4
1.2.1. Conjunto de datos	4
1.2.2. Línea de investigación	12
1.2.3. Planificación del trabajo	12
1.2.4. Metodología de desarrollo	13
2. Estado del arte	15
2.1. Revisión histórica de clasificación binaria de tumores	16
2.1.1. Métodos no profundos	16
2.1.2. Métodos basados en CNN	17
2.2. Revisión histórica de segmentación	19
2.2.1. Métodos que se enfocan en la arquitectura	21
2.2.2. Métodos que tratan el desbalanceo	25
2.2.3. Métodos que tratan la información multi-modal	28
2.3. Nuevas enfoques para la segmentación	30
2.3.1. Basados en Transformers	31
2.3.2. Basados en aprendizaje no supervisado	33
3. Metodología	35
3.1. Análisis de los recursos disponibles	35
3.2. Preprocesado de Datos	36
3.2.1. Elección de dimensionalidad de las entradas	37
3.2.2. Normalizado de las imágenes	37
3.2.3. Recortado de imagen	38
3.2.4. Undersampling	38
3.3. Elección de modelos	40
3.3.1. Codificador y representación latente	40
3.3.2. Modelo de clasificación	41
3.3.3. Modelo de segmentación	43
3.3.4. Uso de transfer learning	43
3.4. Diseño de las arquitecturas	44

3.4.1. Función de activación	44
3.4.2. Normalización por lotes	45
3.4.3. Arquitectura para la reconstrucción de imágenes	46
3.4.4. Arquitectura para clasificación	53
3.4.5. Arquitectura para segmentación	54
3.5. Optimización de las arquitecturas	55
3.5.1. Optimizador	55
3.5.2. Funciones de pérdida	57
3.5.3. One-Cycle Policy	60
3.6. Evaluación y métricas	62
3.6.1. Métricas para clasificación	62
3.6.2. Métricas para segmentación	63
3.7. Desarrollo para la predicción de la evolución	65
4. Experimentación	67
4.1. Bibliotecas y desarrollo de los experimentos	67
4.2. Construcción del codificador y representación latente	69
4.2.1. Arquitecturas con conexiones residuales: ResNet34 . .	69
4.2.2. Arquitecturas con filtros con distinto tamaño: Xception	71
4.3. Clasificación	73
4.3.1. Entrenamiento en clasificación	73
4.3.2. Validación en clasificación	76
4.3.3. Comparativa de clasificación con el estado del arte .	79
4.4. Segmentación	80
4.4.1. Entrenamiento en segmentación	80
4.4.2. Validación en segmentación	82
4.4.3. Comparativa de segmentación con el estado del arte .	85
5. Conclusiones y Trabajos Futuros	87
5.1. Conclusiones del trabajo	87
5.1.1. Resultados y recursos	87
5.1.2. Nuevo preprocesado en el problema	88
5.1.3. El poder de la reconstrucción previa	88
5.2. Trabajos futuros	88
5.2.1. Uso de transformers	89
5.2.2. Unificación de arquitecturas	89
5.2.3. Exploración de otras técnicas de aprendizaje no supervisado	89
Bibliografía	89

A. Códigos y repositorio	97
A.1. Repositorio de GitHub	97
A.2. Cuadernos en Kaggle	97
A.3. Modelos en Kaggle	98
A.4. Demostración de uso de la interfaz	98
B. Documentación de la interfaz	99
B.1. Estructura de la interfaz	99
B.1.1. Diagrama de clases	99
B.1.2. Documentación de funciones	100
B.2. Dependencias	102
C. Manual de uso de la interfaz	105
C.1. Diagrama de flujo de la interfaz	105
C.2. Tutorial sobre el flujo de la interfaz	106

Índice de figuras

1.1.	Porcentaje de Glioblastomas y Meningiomas en el conjunto de datos	5
1.2.	Cantidad de instancias temporales en el conjunto de datos . .	6
1.3.	Visualización de imágenes MRI de tumores en adultos de origen europeo y americano	7
1.4.	Visualización de imágenes MRI de glioblastomas en niños y adultos de origen africano	8
1.5.	Visualización de imágenes MRI de tumores en adultos de origen europeo y americano con su segmentación.	9
1.6.	Visualización de imágenes MRI de tumores en niños y adultos de origen africano con su segmentación.	9
1.7.	Distribución del tejido tumoral en Gliomas.	10
1.8.	Distribución del tejido tumoral en Meningiomas.	11
2.1.	Arquitectura usada por [Abiwinanda et al., 2019]	17
2.2.	Arquitectura usada por [Sultan et al., 2019]	18
2.3.	Arquitectura usada por [Díaz-Pernas et al., 2021]	18
2.4.	Evolución del estado del arte hasta 2021 [Liu et al., 2023]. .	20
2.5.	Comparación entre arquitecturas de una y múltiples trayectorias. Imagen de [Liu et al., 2023]	23
2.6.	Arquitectura de dos vías de [Havaei et al., 2017]	23
2.7.	Comparación arquitecturas encoder-decoder. [Liu et al., 2023]	25
2.8.	Estructura de método en cascada de [Wang et al., 2018] . .	26
2.9.	Arquitectura autoencoder regularizador de [Myronenko, 2019]	27
2.10.	Arquitectura de [Zhou et al., 2021]	29
2.11.	Modelo especializado en la correlación de las modalidades .	29
2.12.	Red [Zhou et al., 2021] de fusión de representaciones latentes	30
2.13.	Arquitectura de [Wenxuan et al., 2021]	31
2.14.	Arquitectura de [Hatamizadeh et al., 2021]	33
2.15.	Imagen con ruido y reconstruida [Ferreira et al., 2024] . . .	34
2.16.	Estrategia de aumento de datos de [Ferreira et al., 2024] . .	34
3.1.	Comparativa de rendimiento de las GPU disponibles	36

3.2. Esquema de la arquitectura empleada para inicializar el codificador y representación latente. [Subedi et al., 2022]	41
3.3. Esquema de la arquitectura empleada para el problema de clasificación	42
3.4. Esquema de la arquitectura empleada para el problema de segmentación	43
3.5. Arquitectura usada de ResNet34 [He et al., 2016]	47
3.6. Arquitectura usada de Xception [Srinivasan et al., 2021]	48
3.7. Arquitectura de la representación latente	49
3.8. Ejemplo de transformación mediante PCA	51
3.9. Arquitectura para la tarea de clasificación	53
3.10. Arquitectura para la tarea de segmentación	55
3.11. Interpretación del coeficiente de Similaridad Dice	63
3.12. Distancias de Hausdorff entre dos conjuntos	64
3.13. Arquitectura propuesta para predicción de la evolución de los tumores	66
4.1. Diagrama de clases de PyTorch	68
4.2. Curva de aprendizaje con ResNet34	69
4.3. Reconstrucción de las imágenes con ResNet34	70
4.4. Curva de aprendizaje con Xception	71
4.5. Reconstrucción de las imágenes con Xception	72
4.6. Curva de aprendizaje para clasificación parte convolucional congelada	73
4.7. Curva de aprendizaje para clasificación toda la red descongelada tras ajustar capas densamente conectadas	74
4.8. Curva de aprendizaje para clasificación todo el entrenamiento toda la red descongelada	75
4.9. Matriz de confusión de entrenamiento sin votación	76
4.10. Matriz de confusión de validación sin votación	77
4.11. Matriz de confusión de validación con votación: <i>Meningiomas < 5</i>	78
4.12. Matriz de confusión de validación con votación: <i>Meningiomas < 3</i>	78
4.13. Matriz de confusión de Test con votación	79
4.14. Curva de aprendizaje para la tarea de segmentación	81
4.15. Comparación de la salida del modelo respecto la real	82
4.16. Distribución de similaridad Dice en el conjunto de validación	83
4.17. Distribución de distancia Hausdorff en el conjunto de validación	84
4.18. Distribución de sensibilidad en el conjunto de validación	84
4.19. Distribución de similaridad Dice en el conjunto de test	84
4.20. Distribución de distancia Hausdorff en el conjunto de test	85
4.21. Distribución de sensibilidad en el conjunto de test	85

B.1. Diagrama de clases de toda la interfaz	100
C.1. Diagrama de flujo de la interfaz.	105
C.2. Inicio de la interfaz.	106
C.3. Seleccionar archivo a cargar en la interfaz.	106
C.4. Seleccionar prueba a cargar en la interfaz.	107
C.5. Visualizado de imágenes y 3D en la interfaz.	107
C.6. Segmentar con la interfaz y visualizado de la máscara resultado.	108
C.7. Clasificar con la interfaz y resultados de clasificación.	108
C.8. Exportar segmentación con la interfaz.	109
C.9. Seleccionar ruta a exportar en la interfaz.	109
C.10. Mensaje de exportación realizada correctamente.	110

Índice de cuadros

1.1.	Planificación del trabajo con tiempos y presupuesto.	13
3.1.	Porcentaje de imágenes conservadas tras undersampling. . . .	40
4.1.	Pérdida de entrenamiento y validación para la reconstrucción con ResNet34	69
4.2.	Pérdida de entrenamiento y validación para la reconstrucción de Xception	71
4.3.	Pérdida de entrenamiento y validación para clasificación con la parte convolucional congelada	72
4.4.	Pérdida de entrenamiento y validación para clasificación con la parte convolucional congelada	73
4.5.	Pérdida de entrenamiento y validación para clasificación toda la red descongelada	74
4.6.	Pérdida de entrenamiento y validación para clasificación para todo el entrenamiento toda la red descongelada	75
4.7.	Resultados de validación y entrenamiento sin votación	77
4.8.	Resultados de validación con votación y de la búsqueda . . .	79
4.9.	Comparativa resultados de test con la literatura	80
4.10.	Pérdida de entrenamiento y validación para segmentación toda la red descongelada	81
4.11.	Resultados de hold-out en validación y test para segmentación	83
4.12.	Comparativa de segmentación con el estado del arte	86
A.1.	Enlaces a los notebooks en Kaggle	98
A.2.	Enlaces a los checkpoints en Kaggle	98

Capítulo 1

Introducción

Los tumores cerebrales son una de las formas más letales de cáncer. Específicamente, los glioblastomas y sus variantes difusas son los más comunes y agresivos tipos de tumor del sistema nervioso central en adultos. Su alta heterogeneidad en apariencia, forma e histología los convierte en una de las patologías más difíciles de diagnosticar, de tratar y un reto para el campo de la imagen médica.

Desde el punto de vista de la ingeniería y la informática, vemos como sin duda la aplicación de técnicas de Visión por Computador es una de las máximas para la investigación en imagen médica en la actualidad. Sólo considerando su aplicación en el diagnóstico de enfermedades, desde 2008 el número de publicaciones promedio realizadas por año se ha incrementado notablemente tanto que actualmente es diez veces mayor que en sus inicios.

Resultados notables como la inclusión de robots especializados para la cirugía [Cheng et al., 2022] o buenos resultados en competiciones de ciencia de datos que replican la precisión médica en el diagnóstico mediante imagen [Bulten et al., 2022] evidencian esta tendencia. El trabajo conjunto de personal médico e ingenieros promete seguir dando resultados que de forma separada eran inaccesibles.

1.1. Objetivos

Con este trabajo se persigue la creación de una arquitectura basada en aprendizaje profundo para equipar a un programa de uso médico, estudiarla y compararla junto a trabajos previos y estado del arte. Este programa tiene el objetivo de la ayuda en la evaluación del diagnóstico y pronóstico de un posible paciente de tumor cerebral y en caso afirmativo, la ayuda en la aplicación de la terapia por radiación.

Se seguirá un planteamiento similar al seguido en la competición **BraTS: Brain Tumor Segmentation 2023** [Baid et al., 2021] históricamente reconocida por ser un benchmark recurrente de las capacidades de las arquitecturas profundas en el campo de la imagen médica.

BraTS es una competición que se define como un conglomerado de diferentes tareas relevantes en el diagnóstico de los tumores cerebrales «Cluster of Challenges». En 2023 dando especialmente importancia a la generalidad de un modelo que mantenga los resultados anteriores para pacientes más diversos (de diferente origen étnico y de diferentes edades) y con diferentes tipos de tumores. En concreto para 2023, se contemplaron las siguientes tareas por separado: segmentación de glioblastomas en adultos, segmentación de meningiomas en adultos, segmentación de tumores pediátricos, segmentación de tumores de pacientes de origen africano, generación de pruebas faltantes y generación de partes de la imagen.

De forma análoga a esta competición, se plantea conseguir dicho objetivo a partir de la resolución de las siguientes tareas.

1. **Segmentación de los tumores.** La segmentación de la lesión tumoral de los glioblastomas y los meningiomas.
2. **Clasificación entre tipos de tumores.** Clasificación binaria entre glioblastomas y meningiomas. El programa indicará de qué tipo de tumor se trata.
3. **Predicción de la evolución.** La segmentación a corto plazo de la más probable instancia futura a partir de la resonancia actual. Ya no solo se pretende dar una segmentación y clasificación que pudieran aportar valor en las decisiones médicas, sino predecir una nueva segmentación a partir de la resonancia apoyándose en casos similares y en la segmentación de la metástasis de la resonancia actual.

A continuación, detallaremos de una forma más profunda la naturaleza de este planteamiento.

Sólo en los EEUU para 2024 se esperan 25400 nuevos casos de tumor cerebral. La supervivencia de estos a los cinco años es del 33.8% de los pacientes. [cancer.org, 2024]

El cerebro no tiene terminaciones nerviosas. Los pacientes no sienten dolor a causa de un tumor cerebral por sí mismo, lo cual hace que no exista una alerta sobre el paciente que lo motive a buscar ayuda médica en las primeras fases de la patología. Generalmente, acaban buscando ayuda médica por la aparición de otros indicios relacionados difíciles de distinguir de otras patologías agudas y de menor transcendencia como visión borrosa, pérdida

del control, etc. Además, los glioblastomas son tumores de muy rápido crecimiento pueden llegar a estar en una fase avanzada desde su inicio en tan solo 2-3 meses.

Por estos motivos, es común llegar tarde. Tomando mucha importancia el diagnóstico temprano para su superación. Con el objetivo de dar un diagnóstico temprano se han introducido máquinas basadas en aprendizaje profundo. Así en este proyecto evaluaremos las capacidades del aprendizaje profundo para la segmentación de tumores. Prestando mayor interés a aquellos tumores que en sus inicios podrían pasar desapercibidos por el especialista humano, e incluso dar una segmentación de zonas potenciales en las que podrían aparecer nuevos tumores.

En general, los tumores cerebrales son difíciles de tratar y son resistentes a terapias convencionales usadas en otros tipos de cánceres como la quimioterapia debido a los desafíos que presenta el cerebro para tolerar ciertos químicos, transportar medicamentos dentro de él y la alta importancia que tiene en este órgano la optimización del uso de tratamientos que puedan ser invasivos. En otras palabras, el uso de tratamientos basados en la extirpación o en la medicación pueden ser arriesgados. Por tanto, el tratamiento más común de estos está basado en la radioterapia.

A la hora de aplicar un tratamiento de radioterapia siempre se tiene el objetivo de ser lo menos invasivo posible. Para ello, el médico debe ser lo más preciso posible en introducir la segmentación correcta en la que se aplicarán los rayos.

Uno de los objetivos específicos para la ayuda en el tratamiento se basaría en el uso del modelo para automatizar esta tarea ya que podría suponer ahorrar costes en errores humanos y en tiempo a veces escaso para el personal médico cuya tarea se reduciría a corregir dicha segmentación. En nuestro caso, integrándolo en un programa de uso médico.

Por otro lado, de forma general podemos interpretar este trabajo como un **sistema de apoyo a la decisión médica** ya que en ningún caso respecto al desarrollo actual de este problema se pretende sustituir la decisión final del personal médico. Aunque sí aprovechar las capacidades que puede ofrecer el aprendizaje profundo en un mejor diagnóstico a través de caracterizar mejor el tejido afectado: segmentándolo y clasificándolo.

Este sistema de apoyo a la decisión médica se plasmaría en una aplicación de escritorio que el personal médico pueda usar como paso intermedio entre la recogida de las imágenes del escáner y un posible tratamiento de radioterapia.

1.2. Metodología

1.2.1. Conjunto de datos

Una de las **limitaciones frecuentes en el campo de la imagen médica** es la poca disponibilidad de datos. En general y también para nuestro problema, las dificultades que se presentan a la hora de construir un conjunto de datos médico grande son:

1. **Poca densidad de pruebas médicas.** A pesar de que la densidad de casos es alta, es frecuente que la cantidad de pruebas que se realizan sea mucho más baja. Especialmente, para datos médicos es frecuente encontrarse con pocos datos en magnitud con la necesidad de variabilidad en la muestra que tienen las técnicas típicas de optimización.
2. **La desvinculación de los pacientes de sus datos.** El tratamiento de un dato médico siempre supone la eliminación de cualquier identificativo que ponga en riesgo la privacidad de este. Suponiendo un trabajo adicional para el personal médico que no siempre se puede asumir.
3. **No existe una fuerte centralización de datos.** Al igual que los avances en imagen médica, el interés por la construcción de una base de datos única que recoja los máximo datos posibles es también reciente haciendo que en la actualidad la mayoría de los datos estén distribuidos en muchos centros médicos diferentes con formatos diferentes.

Partimos del conjunto de datos **BraTS** y serán los que únicamente utilizaremos para todo el trabajo ya que son los únicos que se pueden encontrar en la red pidiendo un acceso a ellos de una forma simple.

Hasta nuestro conocimiento, **BraTS** es el mayor conglomerado de resonancias magnéticas en la actualidad para los desafíos que planteamos en este trabajo. Otros datasets usados por la comunidad para este problema como el incluido en **Medical Segmentation Decathlon** descubrimos que es un subconjunto de **BraTS**.

BraTS está patrocinado y organizado por la ASNR (American Society of Neurology), MICCAI (Medical Image Computing and Computer Assisted Intervention Society), National Cancer Institute, la Universidad de Pensilvania e Intel entre otros.

Los datos de BraTS son heterogéneos ya no sólo externamente con diferentes tipos tumores (glioblastomas y meningiomas) y de pacientes de orígenes distintos y de diferentes edades sino también internamente con

lesiones más y menos avanzadas en el tiempo (low- and high-grade) y con datos de multitud de centros que han sido escaneados con distintos escáneres.

Definiremos como nuestro conjunto de datos X a un conjunto de resonancias magnéticas cerebrales completas (resonancias con imágenes de todo el cerebro, no una parte). Este lo tenemos en formato **nii : Neuroimaging Informatics Technology Initiative (NIfTI)**. Con este formato podemos trabajar directamente con su versión comprimida en **gz: GNU ZIP**. Para ello, usaremos la librería **nibabel** que nos permite la lectura y conversión de cada resonancia a un array 3D de la librería NumPy de Python. Cada resonancia se despliega en un array 3D de dimensiones $240 \times 240 \times 155$. Siendo 155 el número de imágenes del cerebro del paciente dividida en partes equidistantes. Cada imagen tiene una resolución 240×240 .

Por otro lado, definimos el conjunto de etiquetas Y como otro 3D-array de las mismas dimensiones que en el conjunto de datos X donde se muestra la segmentación (Ground Truth) de los tumores.

En términos numéricos, contando solo los datos que tenemos sobre adultos tenemos 1251 resonancias de glioblastomas de 1133 pacientes diferentes y 1000 resonancias de meningiomas de 944 pacientes diferentes. Adicionalmente, tenemos resonancias de glioblastomas pediátricos y de adultos de origen africano que brindan de una distribución mucho más rica al dataset, aunque estos son una minoría siendo 99 y 60 respectivamente.

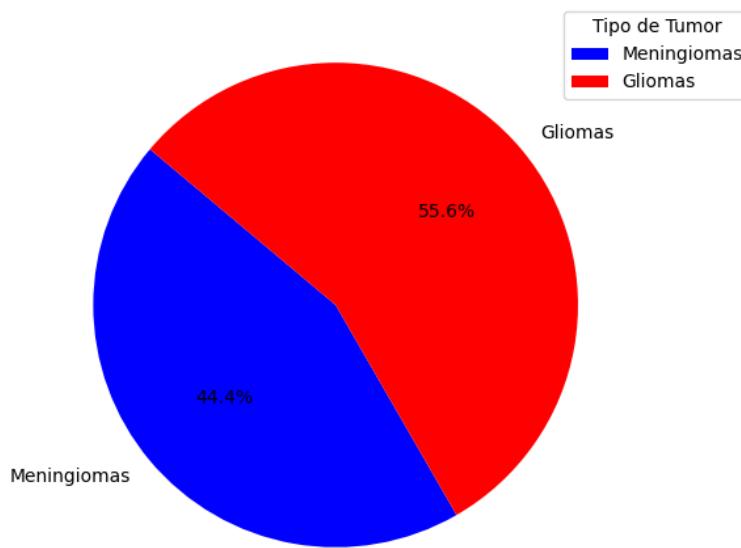


Figura 1.1: Porcentaje de Glioblastomas y Meningiomas en el conjunto de datos

Por otro lado, observamos los pacientes que tienen más de una resonancia versus los que tienen una única resonancia para caracterizar la cantidad de instancias temporales que tenemos en el dataset.

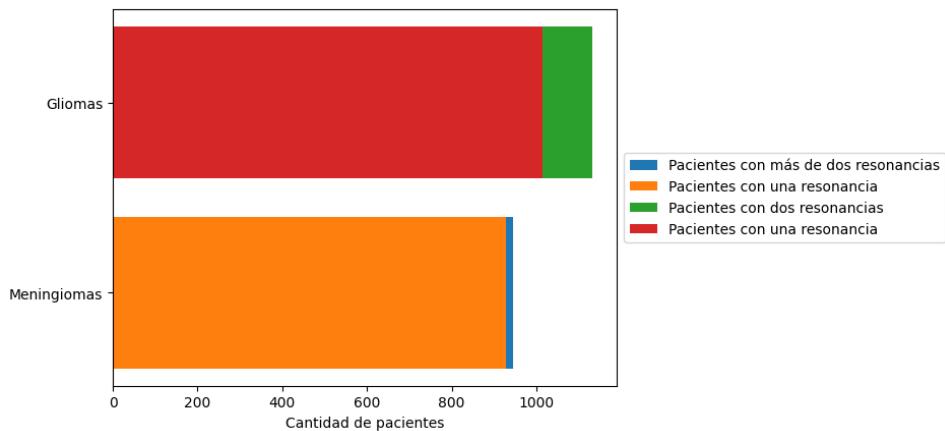


Figura 1.2: Cantidad de instancias temporales en el conjunto de datos

En comparativa, observamos como la mayoría de pacientes no tienen más de una resonancia magnética, dejando sólo una cantidad de 118 pacientes de glioblastoma y de 16 pacientes de meningiomas con varias resonancias en el tiempo.

Visualizado de datos

A continuación, detallaremos más profundamente los datos haciendo algunas visualizaciones.

Para todos los tumores del conjunto de datos X tenemos distintos tipos de muestras según las características de la frecuencia empleada en la toma de la resonancia, **T1-weighted** o **T2-weighted**. [Dominic LaBella, 2023]

El tiempo de repetición (TR) es la cantidad de tiempo entre secuencias de pulsos sucesivas aplicadas al mismo segmento. El tiempo hasta el eco (TE) es el tiempo entre la llegada del pulso sobre el tejido y la recepción de la señal de eco.

Las imágenes ponderadas en T1 se producen utilizando tiempos TE y TR cortos. Por el contrario, las imágenes ponderadas en T2 se producen utilizando tiempos TE y TR más largos.

Las imágenes ponderadas en T1 muestran con más detalle la anatomía normal del tejido blando y la grasa. Las imágenes ponderadas en T2 muestran con más detalle el líquido y alteraciones (p. ej., tumores, inflamación,

traumatismo).

En resumen, tenemos cuatro 3D-arrays por resonancia magnética según frecuencia de señal y aplicando o no un agente de contraste:

1. **T1N Pre-contrast T1-weighted** : Resonancia en frecuencia T1 sin suministrarle ningún agente de contraste al paciente.
2. **T1C Post-contrast T1-weighted** : Resonancia en frecuencia T1 suministrándole un agente de contraste al paciente.
3. **T2W T2-weighted** : Resonancia en frecuencia T2 convencional.
4. **T2F T2-weighted Fluid Attenuated Inversion Recovery** : Resonancia en frecuencia T2 en la que se anula la señal proveniente del líquido cefalorraquídeo.

A continuación, observamos las imágenes producidas por una resonancia magnética en las diferentes pruebas para los dos tipos de tumores que tenemos.

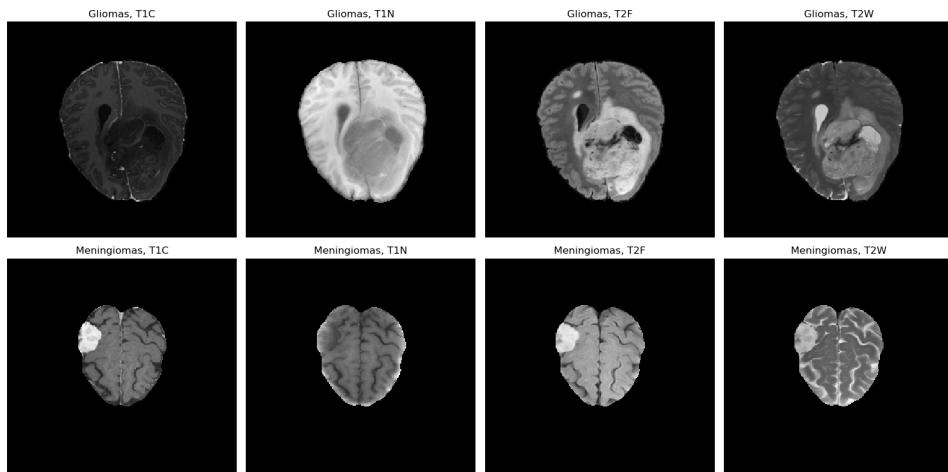


Figura 1.3: Visualización de imágenes MRI de tumores en adultos de origen europeo y americano

Por otro lado, podemos observar algunas imágenes de las resonancias más específicas que tenemos, pediátrica y de adultos de origen africano.

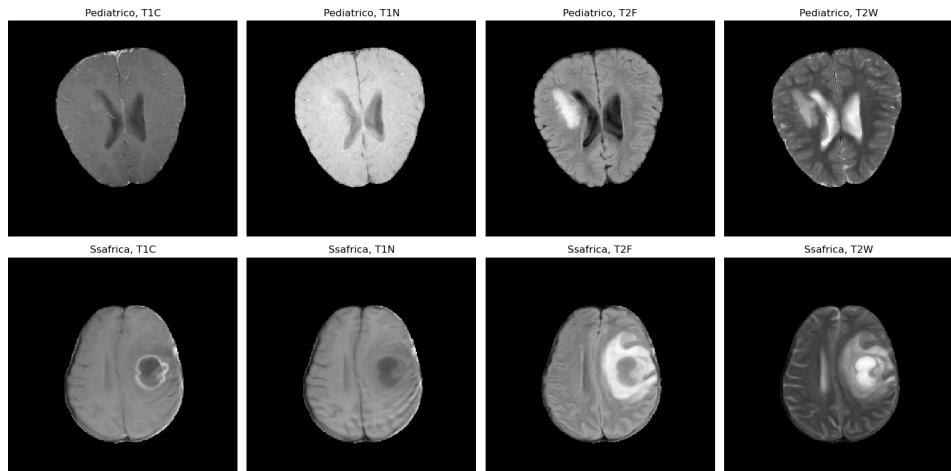


Figura 1.4: Visualización de imágenes MRI de glioblastomas en niños y adultos de origen africano

A continuación, exploraremos el **etiquetado de las resonancias**, es decir, su segmentación realizada por el personal médico colaborador en **BraTS**.

La etiquetas de la segmentación pueden tomar cuatro valores, los del intervalo $[0, 3]$ que están relacionados con el tipo de tejido que segmentan. Tenemos tres tipos de tejidos que se relacionan con el valor de etiquetas en el array.

1. **Etiqueta 0. Sano:** Tejido sano.
2. **Etiqueta 1. NCR:** Tejido Necrótico. Núcleo del tumor tejido sin vida y usualmente reseco.
3. **Etiqueta 2. ED:** Edema peritumoral. Tejido afectado por expansión del tumor generalmente acumulación de líquido y tejido sano desplazado.
4. **Etiqueta 3. ET:** Enhancing tumor. Tejido donde se encuentra la principal actividad de expansión del tumor. Área de actividad tumoral más agresiva o prolífica.

La no existencia de tejido también recibe la etiqueta 0 junto con el tejido sano.

Podemos ver la segmentación de las resonancias que se mostraron en Figura 1.3 y Figura 1.4.

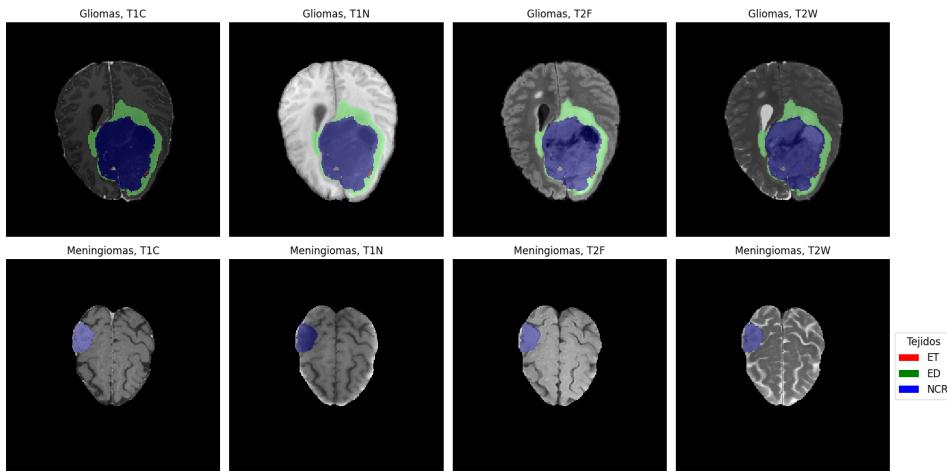


Figura 1.5: Visualización de imágenes MRI de tumores en adultos de origen europeo y americano con su segmentación.

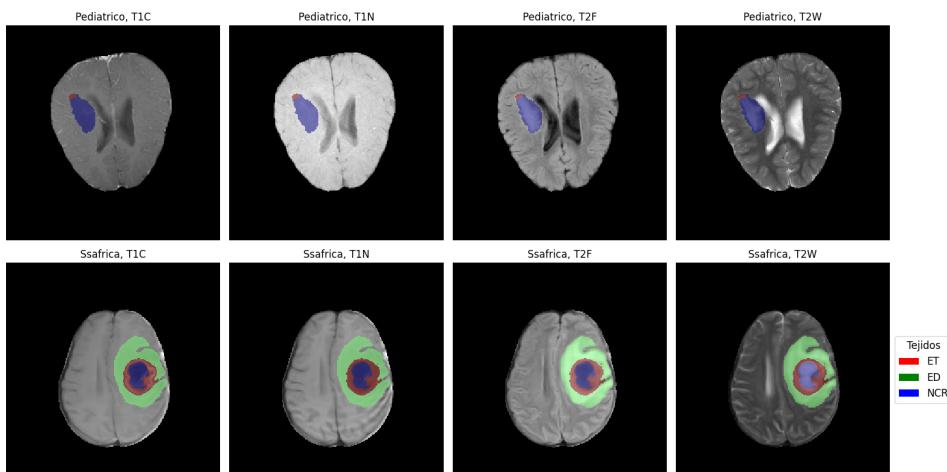


Figura 1.6: Visualización de imágenes MRI de tumores en niños y adultos de origen africano con su segmentación.

A continuación, exploraremos la **localización de los tumores** en todo el conjunto de datos. Buscando responder a la siguiente pregunta significativa para el tránscurso del trabajo: ¿Existe alguna zona del cerebro especialmente afectada? Para responderlo podemos intentar visualizarlo. Crearemos un mapa de calor para los 150 primeros slices marcando la presencia de la lesión, ponderaremos de forma lineal a los tejidos según su importancia.

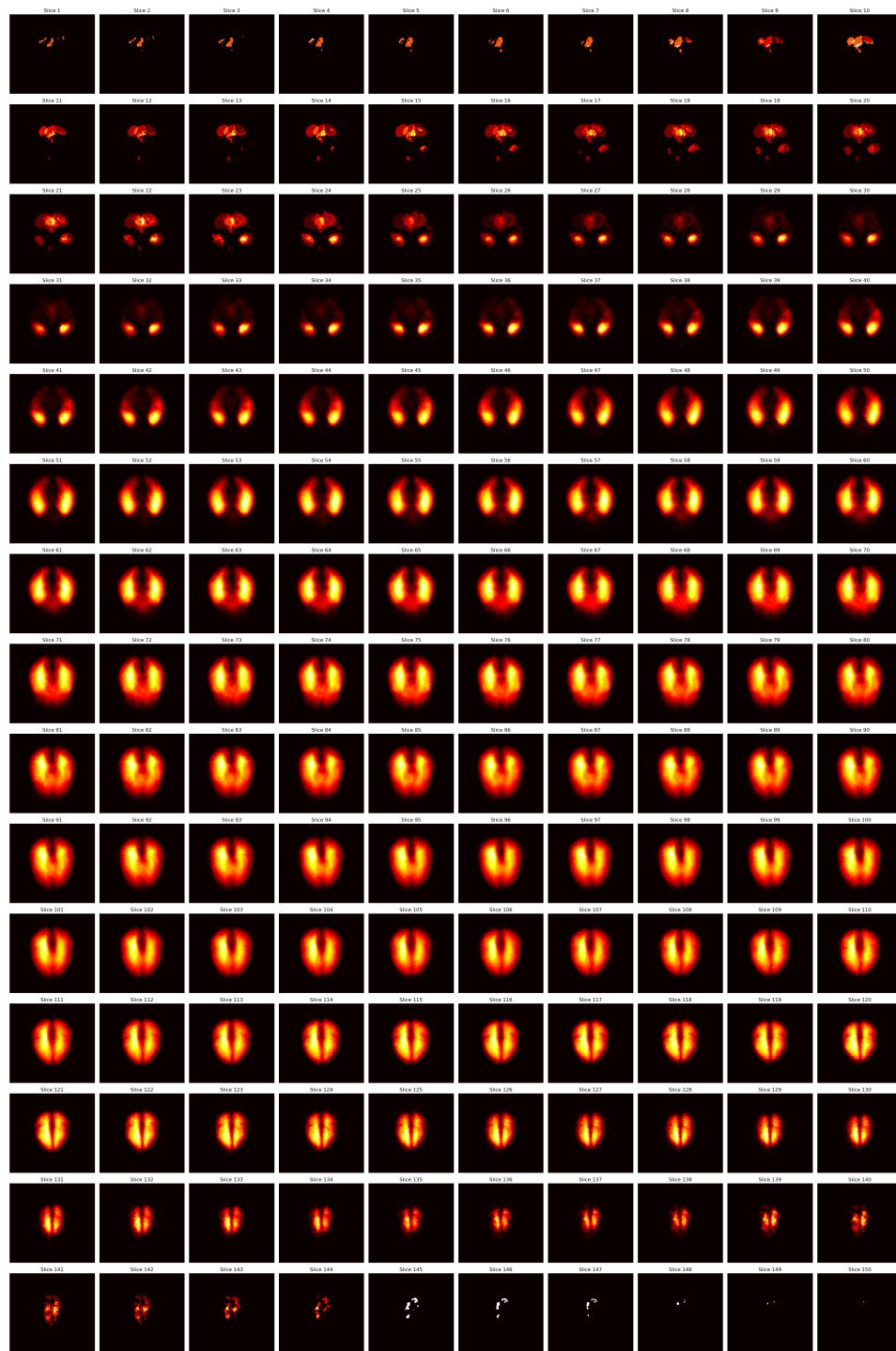


Figura 1.7: Distribución del tejido tumoral en Gliomas.

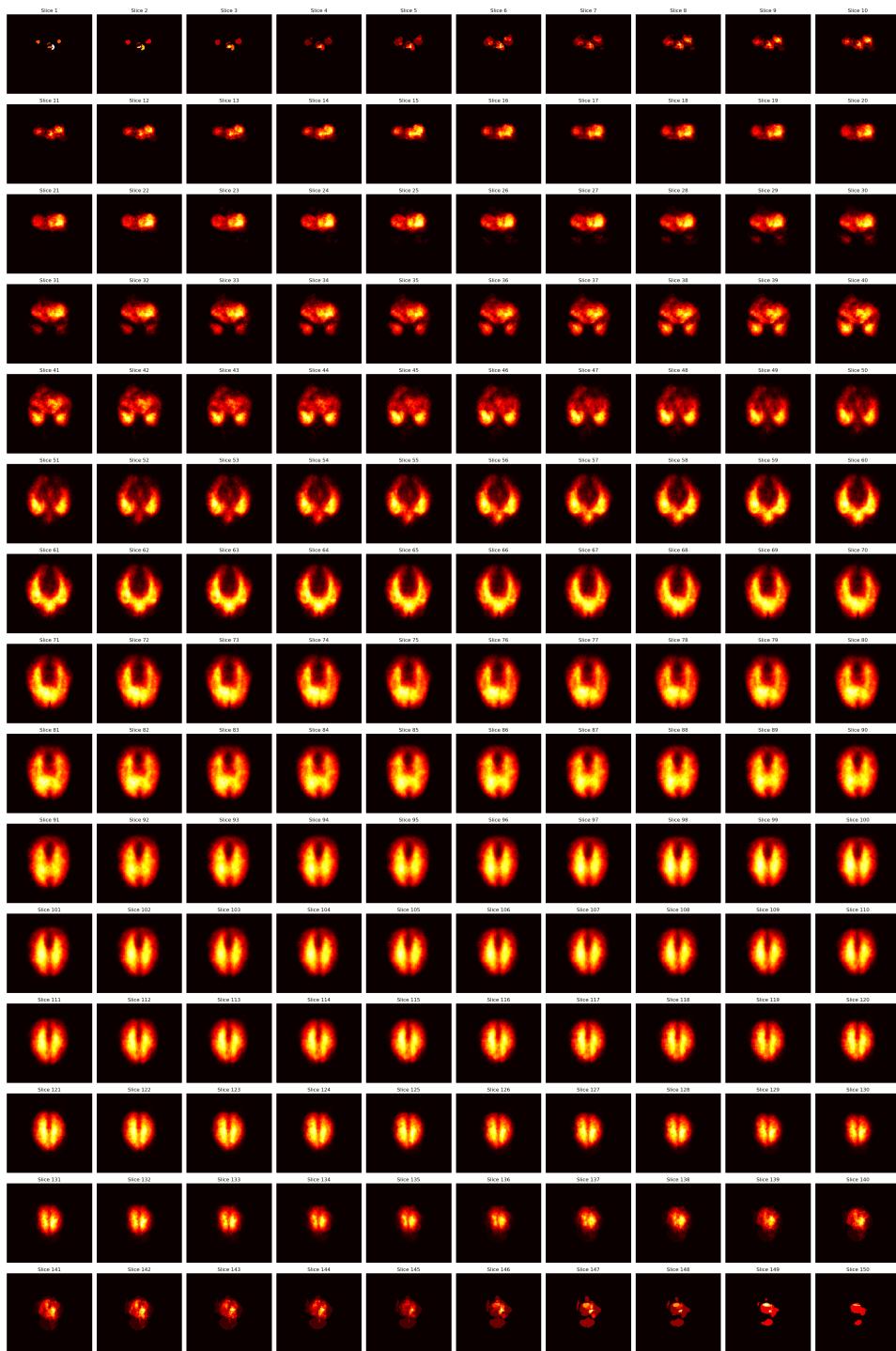


Figura 1.8: Distribución del tejido tumoral en Meningiomas.

Tras la visualización, no se observa una zona especialmente marcada ni

en glioblastomas ni en meningiomas. Aunque sí ciertos detalles, como era de esperar en los primeros slices (que son de la base del cerebro) la zona de los meningiomas solo presenta tejido tumoral en la parte posterior esto es debido que a los meningiomas son tumores de desarrollo entre el cráneo y el cerebro, no habiendo ahí tejido posible de esta forma. Esto también puede explicar porque los gliomas tienen una distribución más centrada en los núcleos de los dos hemisferios del cerebro y los meningiomas más distribuido pero con una ligera tendencia en la parte frontal del cerebro donde más líquido cefalorraquídeo se concentra. A parte, de estos detalles íntegros a la naturaleza de estos tumores vemos como siguen una **localización independientemente distribuida**.

1.2.2. Línea de investigación

En este trabajo definiremos como línea base la construcción de un arquitectura encoder-decoder totalmente convolucional que se ajuste a los datos de entrenamiento. Para posteriormente utilizar el codificador y representación latente del autoencoder para todas las tareas. Conectándole:

1. **Para clasificación** añadiendo una red densamente conectada.
2. **Para segmentación** un decodificador totalmente convolucional transformando la arquitectura en un autoencoder ResU-net (U-net con conexiones residuales).
3. **Para predicción** un decodificador basado en modelos sequence-to-sequence como los transformadores debido a la naturaleza secuencial que tienen diferentes resonancias de un mismo paciente en el tiempo.

La tarea de predicción sólo se planteará a nivel teórico por una baja existencia de datos con información temporal.

1.2.3. Planificación del trabajo

A continuación, mostramos la planificación de llevar a cabo este trabajo en un entorno real a través de una tabla de Gantt que recoja el presupuesto y tiempo dedicado a en cada fase del desarrollo de la investigación y de su software.

Tarea	Semanas	Presupuesto medio
Revisión del estado del arte	1-2	300
Definición de línea de investigación	1	300
Implementación de los experimentos	1-2	300
Ejecución de los experimentos	3-4	900
Validación de los experimentos	1-2	310
Diseño de la interfaz	1	200
Implementación de la interfaz	1-2	300
Documentación	1-2	200
Total	10-16	2810

Cuadro 1.1: Planificación del trabajo con tiempos y presupuesto.

El presupuesto medio necesario ha sido calculado en base a lo que costaría pagar a un becario durante ese tiempo para esa tarea asumiendo que el becario cobra **800** euros al mes más el coste del uso del hardware empleado en los experimentos.

Para el cálculo del hardware no asumimos que usamos las gráficas gratuitas de Kaggle sino que las alquilamos en Lambda Cloud con un precio de uso de ≈ 5 euros por hora. De esta forma, estimamos el presupuesto medio de la tabla.

1.2.4. Metodología de desarrollo

En este trabajo nos intentaremos ajustar lo máximo posible a una metodología de desarrollo ágil, **SCRUM**. Contemplando las siguientes características a cumplir.

1. **Enfoque en roles y procesos:** SCRUM define roles. En nuestro caso, la tutora líder del proyecto (tomando las decisiones estructurales) y el alumno implementador (llevándolo a la práctica y tomando el resto de decisiones). También define procesos estructurados como las reuniones de planificación de sprint y la revisión de sprint. Esto se traducen en las tutorías y comunicación durante el desarrollo del proyecto.
2. **Iterativo e incremental:** SCRUM se basa en iteraciones cortas (sprints) para desarrollar de manera incremental. En nuestro caso, comenzando por clasificación y posteriormente por segmentación. Siguiendo la planificación de trabajo e informando de los avances.
3. **Flexibilidad en las prácticas técnicas:** SCRUM permite adaptarse. Esta característica es muy importante en este proyecto ante pruebas sin éxito o cambios de enfoque que sean necesarios.

Capítulo 2

Estado del arte

En este capítulo estudiaremos y analizaremos los diferentes enfoques presentados históricamente en la literatura para nuestros objetivos, **clasificación de tumores cerebrales** y **la segmentación de tumores cerebrales**. Abordando desde el inicio del estudio del problema pasando por la explosión de métodos basados en Aprendizaje profundo con la construcción de **BraTS** hasta nuestros días. Se pondrá especial énfasis a las soluciones actuales comparándolas desde sus diferencias en metodología y perspectiva.

Por un lado, la clasificación entre los dos tipos de tumores no es tan relevante a la hora de diseñar un sistema de ayuda a la toma de decisión ya que clínicamente sí existe una característica diferencial entre ambos, su **localización**. Los meningiomas aparecen entre el cráneo y el cerebro no internamente en el cerebro como los glioblastomas. Esto hace que un médico pueda distinguirlos sin requerir una gran asistencia para la mayoría de los casos. No obstante, la clasificación puede ayudar a la toma de decisiones en el tratamiento ya que debido a la naturaleza difusa de los glioblastomas estos pueden aparecer en la misma localización que algunos meningiomas y pueden ser confundidos. Sin embargo, la clasificación entre tipos de tumores no es el elemento crítico para la supervivencia del paciente que depende de la eliminación del tumor donde su segmentación toma un papel crucial. Por esto, veremos como el estado del arte del problema de segmentación es mucho mayor que el del problema de clasificación.

Por otro lado, la tarea de predicción de la evolución del tumor debido a la baja densidad de instancias temporales de los datos existentes hacen que este problema no sea tratado en la literatura.

Los grandes esfuerzos se han realizado en entorno a la segmentación, ya que otras tareas que conforman su diagnóstico se verían arrastradas.

2.1. Revisión histórica de clasificación binaria de tumores

Dentro de los problemas de clasificación que se pueden plantear para caracterizar a los tumores cerebrales se puede distinguir en: clasificación multi-clase para diferenciar entre tipos de tumores (glioblastomas, meningiomas u otros tumores más raros y menos frecuentes) o clasificación multi-grado (low- or high- grade) para caracterizar el estado de avance del tumor.

En este trabajo solo nos centraremos en clasificación multi-clase para los dos tipos de tumores más frecuentes y de los que disponemos de etiquetas por **BraTS**, glioblastomas y meningiomas. Adicionalmente en todos los trabajos que revisaremos también incluyen a otro tipo de tumor, el **tumor pituitario** y en un conjunto de datos menor que BraTS, **T1-CE image dataset**.

A continuación, se pretende recoger los trabajos más relevantes realizados entorno a esta clasificación multi-clase ordenados en el tiempo, haciendo especial hincapié en métodos basados en aprendizaje profundo.

2.1.1. Métodos no profundos

Los primeros métodos para abordar la clasificación de tumores aparecen en 2015. En ese momento ya existía una fuerte investigación en el problema de segmentación. Y ni siquiera los primeros métodos utilizaban aprendizaje profundo para abordarlo, como tiempo atrás se había hecho con su tarea hermana la segmentación. A continuación, los dos principales trabajos que abordan la clasificación de tumores con métodos no profundos, es decir, extrayendo las características de la imagen a mano o a través de un proceso automático de procesamiento de imagen.

Basados en SVM

En [Cheng et al., 2015] se propone un método de clasificación basado en técnicas de aumento de la región del tumor en subregiones y la extracción de características usando tres diferentes métodos: histograma de intensidades, la matriz de concurrencia de niveles de grises (GLCM) y bag-of-words (BoW). Utilizó tres modelos: SVM (máquinas de vectores soporte), SRC (sparse representation) y kNN (k-vecinos más cercanos). Obteniendo los mejores resultados con BoW y SVM.

Basados en Fischer Kernel

Otro enfoque interesante que sigue la línea de construir algoritmos precisos a través del procesamiento de imagen es intentar medir la similaridad de las imágenes del dataset para así poder clasificar en base a esta similaridad. En [Cheng et al., 2016] se utiliza un kernel Fischer con esta finalidad. La premisa de este trabajo es que la imagen test debe parecerse al conjunto de entrenamiento. La clasificación de una imagen se basa en determinar, usando el kernel, a cual subconjunto de imágenes de entre todas las clases se parece más.

Este trabajo fue evaluado en un conjunto de datos muy pequeño. Sin embargo, este método permite no solo clasificar sino entender las relaciones entre las similitudes de las imágenes.

2.1.2. Métodos basados en CNN

[Abiwinanda et al., 2019] es una de las primeras aproximaciones que usan redes neuronales convolucionales. En este trabajo se prueba con arquitecturas no muy profundas (para los estándares actuales). Crean 5 arquitecturas a probar consiguiendo sus mejores resultados con la siguiente arquitectura.



Figura 2.1: Arquitectura usada por [Abiwinanda et al., 2019]

Observamos como sólo utilizan dos capas convolución con función de activación **ReLU**, maxpolling para reducir la dimensionalidad y una capa fully-connected para clasificar.

En [Pashaei et al., 2018], prueban a variar ligeramente la arquitectura de [Abiwinanda et al., 2019] añadiéndole una mayor profundidad. Sin embargo, posiblemente motivado por falta de datos obtienen peores resultados al tener una arquitectura más compleja y por tanto más difícil de ajustar. En este mismo trabajo ante esto, prueban a preprocesar las entradas con un clasificador KELM obteniendo mejores resultados.

[Sultan et al., 2019] proponen una arquitectura de 16 capas de convolución que por si misma a excepción de los trabajos anteriores obtiene resultados mucho más competitivos. La novedad de su trabajo es la inclusión de algún tipo de capa normalizadora en la red, en este caso **cross channel normalization**.

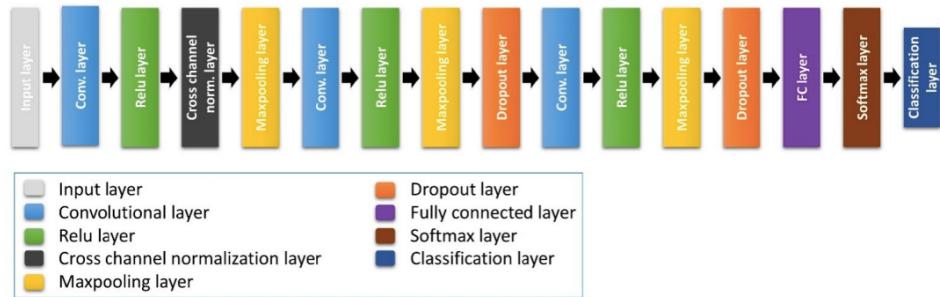


Figura 2.2: Arquitectura usada por [Sultan et al., 2019]

Para finalizar esta revisión, incluimos en la revisión al estado del arte del problema. [Díaz-Pernas et al., 2021] utiliza una red CNN con múltiples capas con kernels de distintos tamaños y capas FCN como clasificador final. Este es un enfoque simple únicamente neuronal ya que no utiliza otros tipos clasificadores. Su estrategia es utilizar características multi-escala para capturar mejor las características de la lesión. En concreto, utiliza filtros 11×11 , 7×7 y 3×3 .

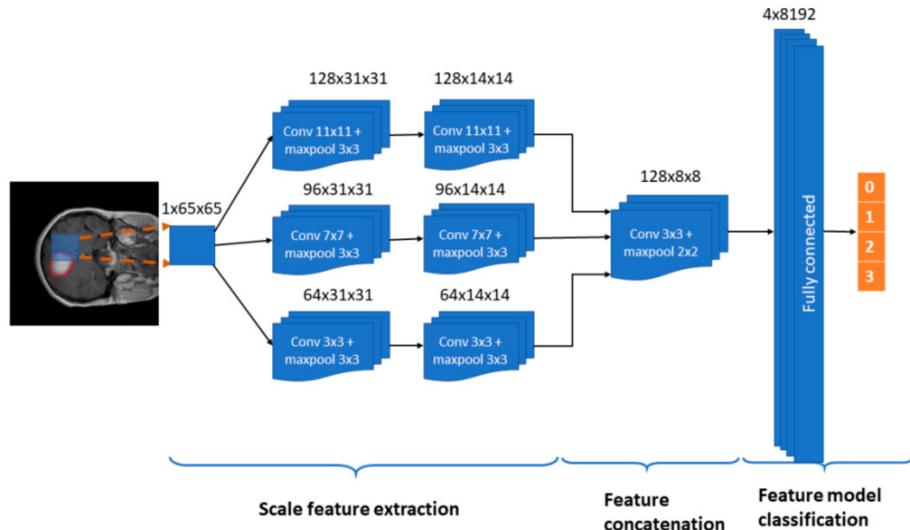


Figura 2.3: Arquitectura usada por [Díaz-Pernas et al., 2021]

Como vemos esta arquitectura no es lineal sino que combina diferentes ramas con diferentes tamaño de filtros. Podemos apreciar algunos de las propiedades que tiene:

- **Extracción jerárquica de características:** El uso de múltiples ra-

mas con diferentes tamaños de filtro permite la extracción de características a diferentes escalas, lo cual es crucial para capturar diferentes aspectos y detalles de las imágenes de tumores cerebrales.

- **Concatenación de características:** La concatenación de las características de diferentes ramas permite combinar información de diversas escalas, mejorando así la capacidad del modelo para aprender representaciones ricas y diversas de las imágenes.
- **Clasificación final:** Las capas completamente conectadas al final de la red realizan la tarea de clasificación, transformando las características extraídas en una predicción final de la clase del tumor.

2.2. Revisión histórica de segmentación

A pesar de los años de desarrollo, el problema de la segmentación de tumores cerebrales es un reto para los investigadores y los algoritmos propuestos a día de hoy deben ser mejorados. Algunas de las características que hacen que esta tarea sea tan difícil son las siguientes:

1. **Incertidumbre en la localización :** Como vimos no existe una zona concreta en general para la aparición de los tumores cerebrales. A excepción, de los meningiomas localizados en zonas superficiales del cerebro y aún siendo una región muy amplia, incluso ya desarrollado un tumor pueden aparecer otros localizados en regiones muy distintas de la original.
2. **Incertidumbre en la morfología :** A diferencia de otras patologías, cada tumor cerebral presenta un tamaño y forma completamente distintas y donde en principio no se puede apreciar un patrón distintivo. Esto hace que sea muy complicado y generalmente aporte malos resultados. Por eso algunos métodos usan algoritmos basados en reglas o aproximaciones alternativas que no incluyen módulos de aprendizaje.
3. **Bajo contraste :** Una buena resolución y contraste son características muy importantes para entender la información de una imagen. Las imágenes IRM producidas en una resonancia debido a proyecciones de imagen y procesos de tomografía usualmente ofrecen una baja resolución y contraste haciendo más difícil la definición de bordes entre diferentes tejidos de la imagen. Una segmentación precisa es difícil de conseguir.
4. **Sesgo en las etiquetas.** Existen indicios para pensar que las etiquetas proporcionadas pueden presentar ruido. El proceso de segmentado

por parte del personal médico depende de su experiencia profesional lo cual puede llevar a cometer errores. Por ejemplo, se han presentado eventualmente discrepancias entre distintos anotadores: algunos tienden a conectar todas las pequeñas regiones de un tejido mientras que otros las segmentan de forma más precisa y separada.

5. **Desbalanceo en el tejido** : Dentro de la segmentación entre los diferentes tipos de tejidos, usualmente el tejido enfermo y que compone la lesión tumoral es usualmente más pequeño que el tejido sano. Esto podría afectar en el proceso de aprendizaje haciendo más difícil la identificación del tejido enfermo.
6. **Desbalanceo entre pacientes** : En el conjunto de datos tenemos muchos pacientes de norteamérica y de ascendencia blanca, pero pocos de otros orígenes como el africano. Además, de tener un sesgo claro de edad ya que existen pocos casos en niños. Esta falta de datos puede impedir que exista una buena generalización para estos casos más aislados.

A continuación, se presenta una revisión histórica sobre la segmentación de tumores cerebrales hasta 2021 [Liu et al., 2023]. Se presenta una línea del tiempo con los principales trabajos de estudio.

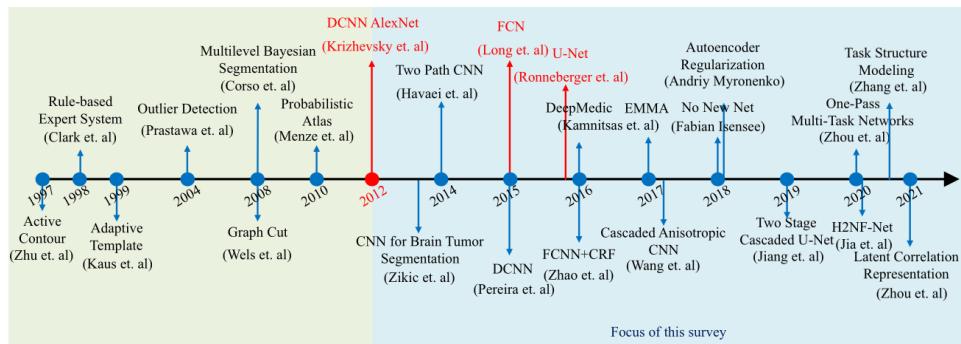


Figura 2.4: Evolución del estado del arte hasta 2021 [Liu et al., 2023].

En la década de 1990 investigadores como [Zhu and Yan, 1997] fueron pioneros al utilizar una red Hopfield con un modelo de contornos activos para extraer los bordes del tumor. Sin embargo, incluso el entrenamiento de una pequeña red como esta era algo computacionalmente costoso por las limitaciones de la época. Desde 1990 hasta 2012, los métodos que iban surgiendo para la segmentación de tumores cerebrales estaban basados en métodos clásicos de aprendizaje con características extraídas a mano, sistemas expertos que se apoyaban en los histogramas de la imagen, plantillas para la segmentación y modelos gráficos.

A pesar de ser un gran paso inicial, tenían grandes deficiencias. Por ejemplo, la mayoría de ellos sólo se centraba en la segmentación de todo el tumor lo cual lleva a un modelo poco útil. Por otro lado, en los modelos basados en características extraídas se hacía muy tedioso poder usarlos eficazmente ya que este paso de extracción dependía de conocimiento previo experto que en ningún momento se pudo llegar a representar en un modelo. En último lugar, los mismos problemas que compartimos hoy en día sobre el desbalanceo y la incertidumbre del problema eran mucho más notables.

Tras 2012 con la revolución del Deep Learning, se introducen nuevas tecnologías (Redes neuronales convolucionales y U-net) que mejorarán los resultados obtenidos hasta el momento. Se empezarán a construir arquitecturas encoder-decoder convolucionales para conseguir pipelines completos para la segmentación. El aprendizaje profundo toma el problema de lleno proclamándose el enfoque que define el estado del arte.

Las soluciones basadas en aprendizaje profundo se pueden clasificar en tres categorías que se corresponden con el problema que quieren resolver y que describiremos a continuación. Sin embargo, como veremos en las soluciones lo ideal es tratar con los tres problemas.

2.2.1. Métodos que se enfocan en la arquitectura

Para poder obtener redes que automáticamente extraen características discriminativas a altas dimensiones es necesario un efectivo diseño de módulos y arquitecturas. Por un lado, se pretende que la arquitectura sea capaz de aprender las características distintivas de los tejidos y a localizar regiones de interés por medio de añadir profundidad a la red, a través de mecanismos de atención o la fusión de características entre las resonancias. Por otro lado, se pretende minimizar la cantidad de parámetros entrenables de la red o conseguir un entrenamiento más rápido.

Diseño de bloques especializados

Los primeros trabajos que tenían este objetivo comenzaron por basarse en arquitecturas bien conocidas como AlexNet o VGGNet a través del uso de una única imagen de la resonancia completa como entrada de la red.

Para la mejora de resultados, se optó por introducir todas la secuencia de imágenes de una resonancia como entrada de la red y añadir más capas

convolucionales. Con ello, teníamos redes más profundas pero que pronto empezaban a sufrir los problemas de la explosión y desvanecimiento del gradiente durante el proceso de entrenamiento. Para ayudar a lidiar con estos problemas, se introdujo a las redes, **conexiones residuales** [Chang, 2016]. Conectando la entrada de la red con su salida, convergiendo más rápido y con arquitecturas de la familia ResNet.

Este proceso de aumento de profundidad con conexiones residuales no sería definitivo porque también conlleva el sacrificio de resolución espacial. Se reemplazaría en trabajos siguientes, el uso de la convolución simple por convoluciones dilatadas. El **uso de convoluciones dilatadas** traería el aumento del espacio receptivo (ya que se aplica una convolución a un espacio mayor de la imagen) sin necesidad de introducir parámetros a la red. La convolución dilatada se vería especialmente útil por ejemplo en la segmentación de áreas grandes como suele ocupar el tejido ED (edema tumoral).

Respecto a conseguir una buena eficiencia en tiempo de entrenamiento se aplica un reordenamiento en memoria de las imágenes de la resonancia similares (p. ej. el mismo slice en las 4 pruebas) de forma, que se reduzcan la comunicación entrada-salida con GPU. Adicionalmente, autores como [Brügger et al., 2019] utilizan **conexiones reversibles** en la red de forma que durante el proceso de backpropagation (backward pass) no se necesite memoria adicional para guardar las activaciones intermedias. Por último, para ahorrar en eficiencia se sustituye la convolución standard por la combinación de **convoluciones separables**.

Diseño de arquitecturas efectivas

La mayoría de los trabajos de recorrido histórico se encasillan en alguno de los siguientes dos enfoques de arquitectura: **redes neuronales convolucionales** para extraer características de la imagen y clasificar los patches o píxeles de la imagen según las etiquetas de los tejidos posibles o **redes encoder-decoder** en las cuales se puede definir un pipeline completo convolucional sin la necesidad de la agregación de capas totalmente conectadas.

1. Redes neuronales convolucionales de una/múltiples trayectorias

A diferencia de una red convolucional de una única trayectoria, las redes de trayectoria múltiples tienen la capacidad de extraer diversas características a diferentes escalas. Estas características se combinan para su posterior procesamiento, usualmente en capas totalmente conectadas, permitiendo a las redes aprender tanto características globales como locales.

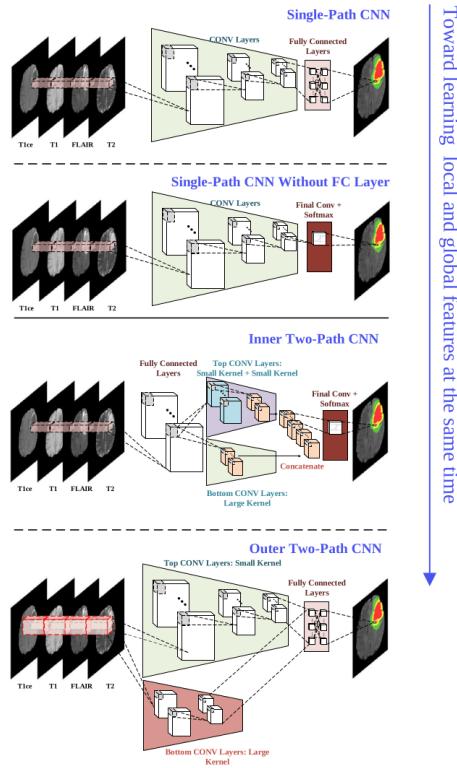


Figura 2.5: Comparación entre arquitecturas de una y múltiples trayectorias.
Imagen de [Liu et al., 2023]

Por ejemplo, [Havaei et al., 2017] desarrollaron una estructura de dos vías que integra información tanto local como global del tumor, utilizando núcleos de convolución de diferentes tamaños.

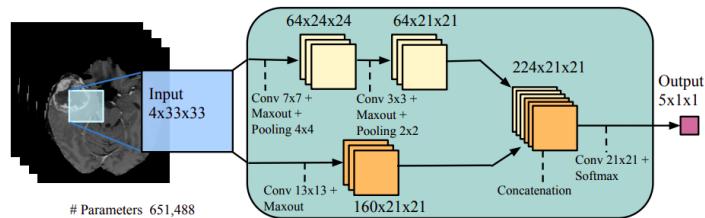


Figura 2.6: Arquitectura de dos vías de [Havaei et al., 2017]

Otros enfoques, como el de [Kamnitsas et al., 2017], optan por aprender información global y local desde la entrada misma, utilizando redes de doble vía, patches de diferentes tamaños y pequeños núcleos de convolución.

Este tipo de arquitecturas fueron una de las primeras aproximaciones que empezaban adaptarse con éxito a las complejidades de la segmentación de tumores cerebrales. Sin embargo, veremos como la dificultad de un buen ajuste en el diseño de estas arquitecturas todavía seguía siendo un problema.

2. Arquitecturas Encoder-Decoder

Las redes de una/múltiples trayectorias toman como input un patch de una cierta región de la imagen y dan como output la clasificación del tejido que existe en ese patch. Este enfoque hace que obtener una buena arquitectura que haga la transformación de los patches a información categórica sea complicado por varios motivos:

- a) Existe una gran **dependencia** entre el tamaño y calidad de los patches, y los resultados que ofrecería la arquitectura.
- b) Toda la transformación de características visuales (aunque, reducidas) a información categórica estaría concentrada en las capas totalmente conectadas. Las capas totalmente conectadas de un tamaño razonables para una capacidad de memoria usualmente utilizada **no puede totalmente representar un espacio de características tan grande**.
- c) Si necesitamos tener distintas redes separadas, el proceso de ajuste de cada una de ellas es independiente. Esto lo podemos interpretar como un coste añadido en términos de **eficiencia**.

Para superar estos problemas en los siguientes trabajos se empieza a utilizar **FCN Redes neuronales totalmente convolucionales** y **U-net** basadas en arquitecturas encoder-decoder, de forma que se establece un pipeline completo desde la imagen a la segmentación.

Una de los tipos más importantes de FCN para este problema es U-net. U-net consiste en la creación de conexiones entre el encoder y el decoder. Permitiendo una vinculación directa en el proceso de reducción y ampliación de dimensionalidad. Estas conexiones reciben el nombre de **Skip Connections** y pueden ayudar a las capas del decoder a recuperar detalles visuales aprendidos en el encoder, llevando a una segmentación más precisa.

[Isensee et al., 2018] utilizan una U-Net dándole aún más énfasis a la tarea de una segmentación utilizando una función de pérdida basada en la similaridad Dice.

Similar a las skip connections antes mencionado, el uso de conexiones residuales y skip connections permiten el paso de características de alto y bajo nivel para una mejor segmentación final.

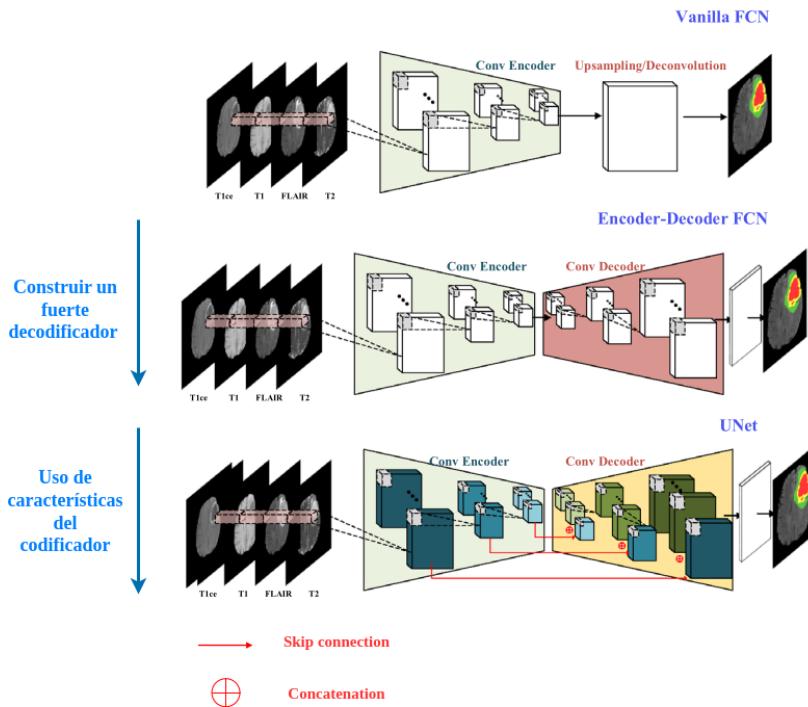


Figura 2.7: Comparación arquitecturas encoder-decoder. [Liu et al., 2023]

2.2.2. Métodos que tratan el desbalanceo

Como anunciábamos anteriormente el alto desbalanceo de los diferentes tejidos presentes en el cerebro de un paciente puede tener un impacto negativo en el proceso de entrenamiento. Motivados por métodos como los sistemas multi-expertos, se empezó a construir métodos específicos para este problema.

Podemos diferenciar en:

1. **Diseños sobre la arquitectura:** Redes en cascada, ensamblado de modelos y arquitecturas multi-tarea.
2. **Mejorar el entrenamiento:** Funciones de pérdida especializadas.

Redes en cascada

Una red en cascada es un conjunto de redes más pequeñas ordenadas en las cuales el output de la red anterior sirve como una input a la siguiente, formando una «cascada de redes». De esta forma, podemos tener redes especializadas en distintos niveles.

Las primeras redes en cascada están especializadas en características de más alto nivel y las siguientes de más bajo nivel.

Por ejemplo, en [Wang et al., 2018] se utilizan tres redes especializadas para los tres regiones de tejidos definidas por **BraTS**. Empezando por la región más grande hasta la más pequeña.

Su primera red WNet segmenta a Whole Tumor, toda la lesión. La siguiente TNet segmenta al núcleo del tumor. Finalmente, Enet a la parte activa del tumor.

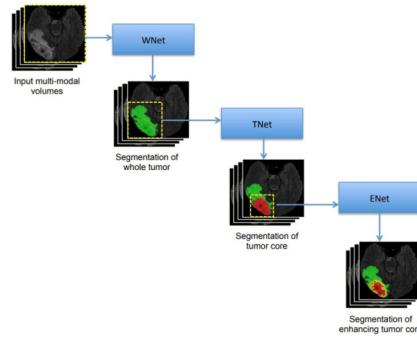


Figura 2.8: Estructura de método en cascada de [Wang et al., 2018]

La ventaja de este modelo es evitar la interferencia de las clases desbalanceadas, ya que cada red trata su clase como un problema de segmentación binaria.

Sin embargo, hace que las redes dependientes de otras dependan también de sus resultados. Si la primera red obtiene malos resultados, todas las siguientes redes se verán afectadas por ella.

Ensamblado de modelos

Una de las consecuencias que tiene el uso de una sola red es que está altamente influenciada por la elección de sus hiperparámetros. Con el objetivo de obtener un modelo más robusto y general modelo para la segmentación se puede combinar la salida de múltiples redes, ensamblarlas.

El ensamblado de modelos aumentaría el espacio de hipótesis del modelo final evitando la caída en óptimos locales debido a el desbalanceo de datos.

EMMA de [Kamnitsas et al., 2018] es uno de los primeros modelos para segmentación de tumores que es un ensamblado de varias redes. EMMA utiliza tres modelos: DeepMedic, una red FCN y una U-net para dar el output de los tres con una mayor confianza.

[Jiang et al., 2020] ganadores de BraTS2019 adoptaron una estrategia de

ensamblado con 12 modelos obteniendo entorno 0.6 – 1 % mejores resultados el mejor resultado usando un único modelo.

Arquitecturas multi-tarea

Todo lo descrito en esta revisión histórica gira entorno a la segmentación de tumores. Sin embargo, la desventaja que puede tener enfocarnos en esta sola tarea es que quizás los modelos específicos para segmentación ignoran información útil en las imágenes para otras tareas, que indirectamente pueda ayudar a obtener una mejor generalización en la segmentación de tumores.

Por un lado, esta idea radica en la suposición de que los modelos que aprenden más tareas están aumentando su aprendizaje en el dominio del problema y esto debería ser beneficioso para todas las tareas. Por otro lado, de una forma más justificada, sabemos que nos enfrentamos a cierto ruido que desconocemos en los datos y etiquetas por tanto si entrenamos para múltiples tareas en conjunto el modelo aprende representaciones más generales reduciendo el riesgo de sobreajuste. Añadir tareas a la arquitectura y aprenderlas en conjunto podría tener **un efecto regularizador**.

Un claro ejemplo de esto es [Myronenko, 2019] que usa como tarea complementaria la reconstrucción de la resonancia de entrada mediante un autoencoder. Teniendo un efecto regularizador sobre los parámetros compartidos del encoder que a diferencia de regularizaciones L1 o L2 que explícitamente añaden una penalización para evitar el sobreajuste, la tarea nueva añade una penalización en la dirección en la que ambas tareas son optimizadas reduciendo el espacio de búsqueda de los parámetros entrenables de la red.

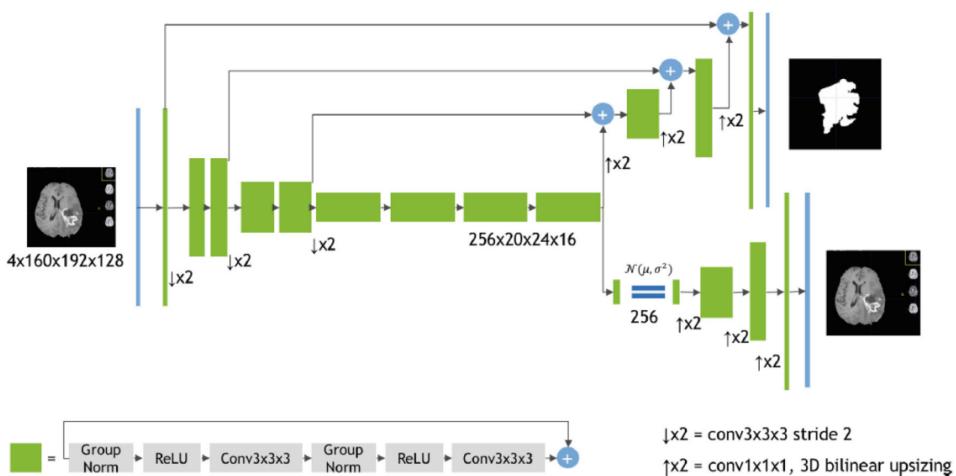


Figura 2.9: Arquitectura autoencoder regularizador de [Myronenko, 2019]

Funciones de pérdida especializadas

De forma más detallada, el problema del desbalanceo entre los diferentes tejidos se manifiesta durante el proceso de entrenamiento, en un gradiente excesivamente influenciado por los tejidos mayoritarios. Por ello, atacando directamente al problema multitud de trabajos proponen funciones de pérdida especializadas.

Funciones de pérdida estándar en este problema incluyen categorical cross-entropy, cross-entropy y dice loss D_L .

Una de las aproximaciones es el uso de utilizar una función de pérdida balanceada. Por ejemplo, añadir una penalización en función de la presencia del tejido segmentado para mitigar su escasa presencia respecto el total.

Otro enfoque se basa en la combinación de diferentes funciones de pérdida en una nueva. Por ejemplo, una nueva función de pérdida de cross-entropy a nivel de píxel y dice loss podría ser su media.

En general, funciones de pérdida que eviten el desbalanceo y mejoren el nivel de atención de las arquitecturas es beneficioso a todo tipo de problemas. Por ello, a diferencia de seguir funciones clásicas como cross-entropy, [Lin et al., 2017] proponen una nueva función llamada **Focal Loss** que ha sido utilizada en años recientes en combinación con Dice Loss para diversos problemas de segmentación.

2.2.3. Métodos que tratan la información multi-modal

Las imágenes asociadas a una resonancia contienen diferentes tipos de imagen según las características de la frecuencia y contraste suministrado al paciente en su toma. Esta forma de proceder en la toma de las resonancias es debido a las limitaciones de las imágenes IRM de poder representar y al menos para el ojo humano, visualizar todos los tejidos importantes en el diagnóstico. Por ello, surge como idea clave tener métodos que tengan los objetivos de poder fusionar, relacionar y incluso distinguir en importancia las diferentes modalidades de imagen.

Otras arquitecturas basadas en autoencoders como [Myronenko, 2019] únicamente fusionan las cuatro modalidades como los canales de una imagen para un mismo slice concatenando las cuatro pruebas en la misma entrada, obteniendo entradas de dimensiones $H \times W \times 4$ en caso de 2D y $H \times W \times D \times 4$ en caso de 3D.

Sin embargo, usar concatenación o adición como método de fusión de los cuatro métodos no permitiría a la red de una forma directa aprender semánticamente la relación entre ellas. Por ello, en trabajos recientes se han adoptado mecanismos de atención aplicados a hacer aprender a la red de

forma más robusta las diferentes modalidades e información espacial.

[Zhou et al., 2021] proponen también una arquitectura encoder-decoder con la particularidad de crear un encoder y decoder específico para cada una de las cuatro posibles representaciones, teniendo un espacio latente donde se fusiona la información de salida de los cuatro encoder dando un tratamiento especial a la fusión de las diferentes pruebas.

A continuación, podemos ver la arquitectura específica usada.

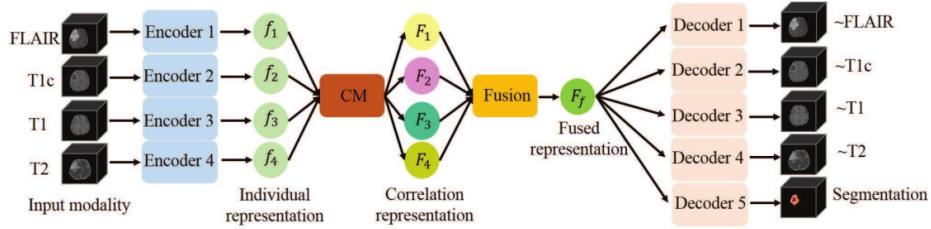


Figura 2.10: Arquitectura de [Zhou et al., 2021]

Por un lado, transforma las representaciones individuales a representaciones correlacionadas. A través de lo que denominan **correlation model**.

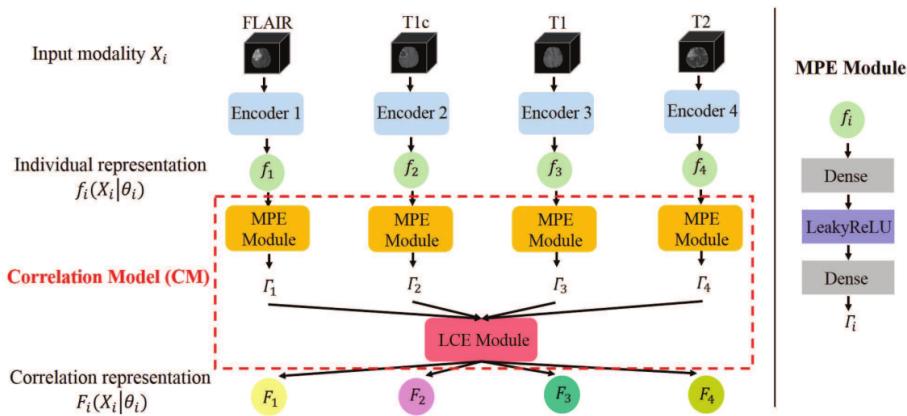


Figura 2.11: Modelo especializado en la correlación de las modalidades

El **correlation model** se compone de dos partes: módulo de estimación de parámetros (MPE) y un módulo de expresión de correlación lineal (LCE).

El módulo de estimación de parámetros se compone de una red totalmente conectada que vincula cada representación salida de cada encoder con unos parámetros $\Gamma_i = \{\alpha_i, \beta_i, \gamma_i, \delta_i\}$

El módulo de expresión de la correlación lineal (LCE) utiliza estos parámetros para obtener una versión correlacionada de cada representación individual aplicando:

$$F_i(X_i|\theta_i) = \alpha_i \odot \gamma_i f_j(X_j|\theta_j) + \beta_i \odot f_k(X_k|\theta_k) + \gamma_i \odot f_m(X_m|\theta_m) + \delta_i$$

Tras ello, se fusiona las representaciones correlacionadas resultado. Permitiendo al modelo manejar de forma explícita la información multi-modal y dándole robustez ante pruebas faltantes.

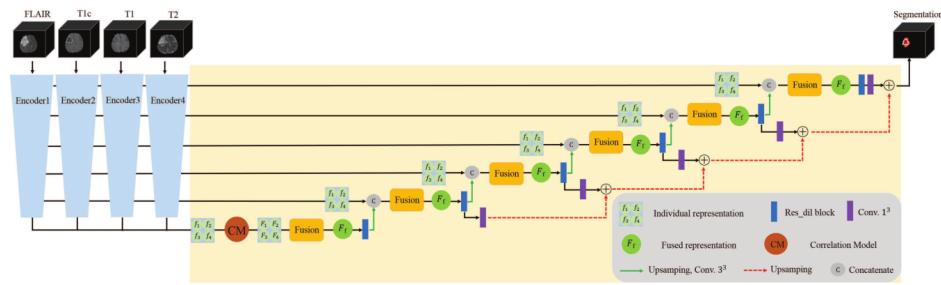


Figura 2.12: Red [Zhou et al., 2021] de fusión de representaciones latentes

Si bien esta arquitectura da ligeramente peores resultados que [Myronenko, 2019] define un paso más en el estado del arte al usar menos recursos computacionales.

2.3. Nuevas enfoques para la segmentación

Las soluciones más relevantes presentadas en la revisión histórica que se ha hecho anteriormente se basan en la aplicación de la convolución sobre las imágenes de resonancia magnética. En el diagnóstico de tumores cerebrales ha tenido largo recorrido el uso de redes neuronales convolucionales.

Con la inclusión de las arquitecturas transformadoras se planteó un nuevo modelo que podía traer ventajas significativas. No siendo la imagen médica y en concreto este problema una excepción.

Con la adaptación de los transformers al campo de la visión, Vision Transformers podría ser un modelo más unificador, paralelizable y que ofreciera mejores resultados que las redes convolucionales al romper con la localidad que supone el uso de convoluciones.

En las soluciones más recientes de la segmentación de tumores cerebrales se introduce el uso de Vision Transformers con estas expectativas.

2.3.1. Basados en Transformers

A continuación, se presentan las soluciones principales que hacen uso de una arquitectura basada en Transformers para la segmentación de tumores cerebrales.

TransBTS

[Wenxuan et al., 2021] introduce la novedad del uso de un transformer como representación latente. El objetivo de esto es tener una representación del conjunto de datos que no depende de las relaciones de localidad de las convoluciones convencionales. En el corazón de este transformer dos capas: Multi-Head Attention (MHA) y Feed Forward Network (FFN).

A continuación, podemos ver la arquitectura que diseña.

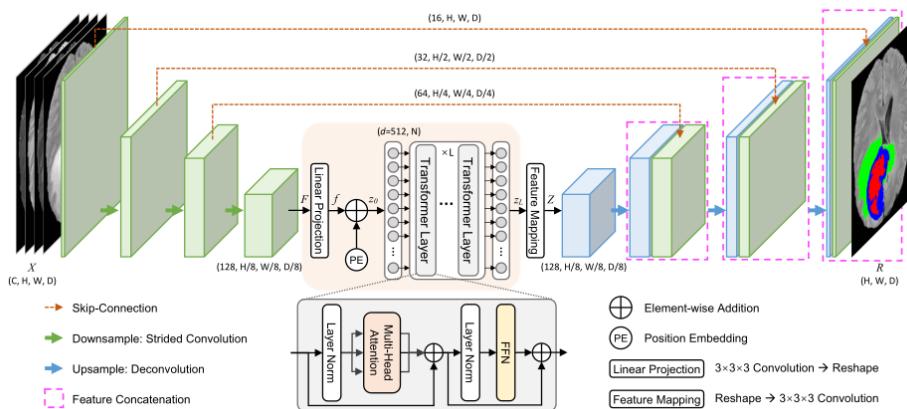


Figura 2.13: Arquitectura de [Wenxuan et al., 2021]

SwinUnet

Una de las ideas más rompedoras es el uso de una arquitectura transformadora como encoder, ya que romper con las convoluciones de un codificador convencional puede suponer un avance al segmentar de forma más precisa los tumores especialmente los difusos. En [Hatamizadeh et al., 2021] usan esta idea para construir una arquitectura encoder-decoder a través de un fuerte codificador basado en Swin Transformers.

Los Swin Transformers [Liu et al., 2021] son un tipo específico de arquitectura transformadora donde se introduce el concepto de ventanas deslizantes (sliding windows) para dividir la imagen de entrada en regiones no

solapadas de tamaño fijo, denominadas ventanas locales. Cada ventana local se procesa independientemente mediante un mecanismo de autoatención (self-attention) dentro de la ventana, lo que reduce significativamente la complejidad computacional en comparación con los Transformadores tradicionales, que aplican la autoatención a toda la imagen de entrada.

El diseño de los Swin Transformers sigue una estructura jerárquica que permite la construcción de representaciones a múltiples escalas. En cada nivel jerárquico, las ventanas locales se combinan y se procesan utilizando una estrategia de atención desplazada (shifted window attention), que permite capturar relaciones entre diferentes regiones de la imagen. Esta estrategia introduce una conexión entre ventanas adyacentes, asegurando que la información pueda fluir a través de toda la imagen de manera eficiente.

Las principales características de los Swin Transformers incluyen:

- **Atención local:** La autoatención se calcula dentro de cada ventana local, lo que reduce la complejidad computacional de $O(n^2)$ a $O(n)$, donde n es el número de píxeles en una ventana.
- **Ventanas Desplazadas:** En cada nivel jerárquico, las ventanas locales se desplazan para capturar relaciones entre ventanas adyacentes, permitiendo una mejor integración de la información a lo largo de la imagen.
- **Estructura jerárquica:** Los Swin Transformers construyen representaciones a múltiples escalas, lo que es crucial para tareas como la segmentación y la detección de objetos, donde se necesita comprender tanto detalles pequeños como contextos globales.
- **Flexibilidad de tamaño:** El tamaño de las ventanas locales y el desplazamiento se pueden ajustar para diferentes aplicaciones, permitiendo una gran flexibilidad.

Gracias a estas características, Swin Transformers han demostrado un rendimiento superior que Transformers ya siguen mantienen cierta relación de localidad.

A continuación, comentamos la arquitectura de [Hatamizadeh et al., 2021].

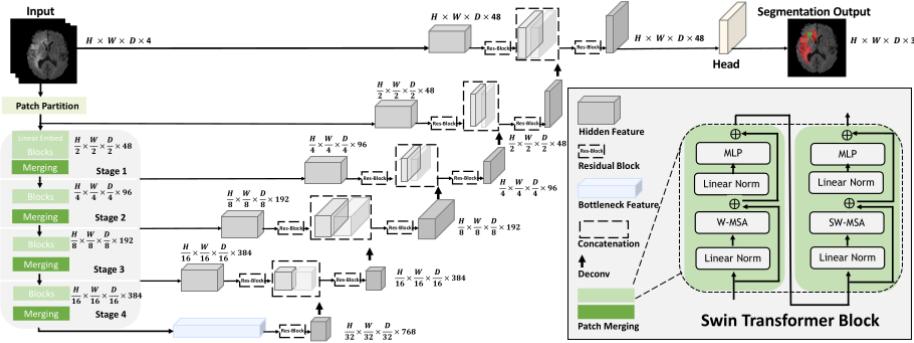


Figura 2.14: Arquitectura de [Hatamizadeh et al., 2021]

Implementan una U-net con un representación latente y decodificador convolucionales pero un codificador basado en Transformer. Esto permite capturar dependencias globales a través de la atención propia, lo cual es beneficioso para identificar relaciones espaciales complejas en las imágenes. Además de esto, la combinación del transformador y las CNNs permite la extracción de características tanto locales como globales, mejorando la precisión de la segmentación. En otras palabras, se aprovecha las capacidades de atención global de los transformadores y la habilidad de las CNNs para capturar detalles locales.

En años posteriores a estos trabajos se incluirá comúnmente la fusión de arquitecturas transformadoras y CNN tomando las ventajas de ambas. Como es el siguiente trabajo que define el estado del arte para 2023.

2.3.2. Basados en aprendizaje no supervisado

En esta revisión histórica muchos trabajos incluyen aprendizaje no supervisado, comúnmente utilizando la preparación del encoder y la representación latente para el aprendizaje de características previas a través de la reducción de la dimensionalidad.

En [Ferreira et al., 2024] utilizan **aumento de datos** a partir de redes generativas adversarias para generar nuevos tumores con buenos resultados. A continuación, mostramos y comentamos el esquema seguido.

Para generar datos nuevos a partir de los originales, toman la imagen y añaden ruido aleatorio en la superposición de su segmentación etiqueta con las imágenes de la resonancia magnética. A continuación, ver cómo añaden el ruido a la imagen y como un generador se encarga de reconstruir la imagen.

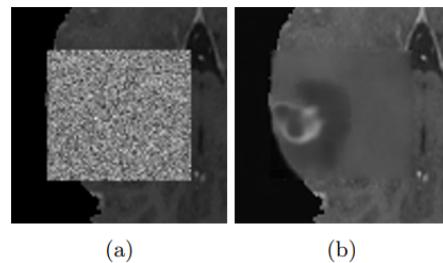


Figura 2.15: Imagen con ruido y reconstruida [Ferreira et al., 2024]

Las redes generativas adversarias tienen dos partes: el **generador G** y **discriminador D** . En este caso, el generador que se encarga de hacer realistas las imágenes a través de una reconstrucción y el discriminador que decide este realismo. Tras un proceso suficiente de entrenamiento el generador aprende a hacer reconstruir imágenes de forma realista suficientemente bien como para ser empleado para aumentar los datos.

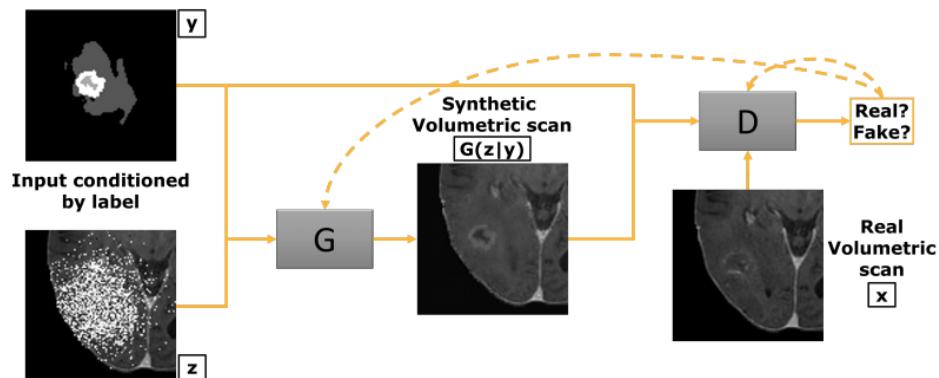


Figura 2.16: Estrategia de aumento de datos de [Ferreira et al., 2024]

Una vez tienen preparado este generador tienen una red que les generará imágenes más variables para las imágenes del dataset originales y pueden generar muchas más entradas a una red como las mencionadas para segmentación. Como arquitectura de segmentación usan la implementación de la Swin U-net de la librería MONAI, una arquitectura muy similar a la mencionada anteriormente, **SwinUnet**.

Capítulo 3

Metodología

En este capítulo describimos en profundidad todos los pasos seguidos en los métodos empleados en el trabajo y su justificación. Posteriormente, se aplicarán en la experimentación.

3.1. Análisis de los recursos disponibles

Para la realización de este trabajo debemos considerar los recursos hardware disponibles para la inferencia de los modelos pero sobre todo para el entrenamiento de los modelos.

1. **Hardware en entrenamiento.** Para el desarrollo de toda la experimentación, entrenamiento de los modelos y validación nos valdremos de los recursos que gratuitamente ofrece Kaggle, una plataforma de ciencia de datos propiedad de Google.

El recurso más importante que ofrece Kaggle y razón de su uso es que nos permite el uso de su gráfica NVIDIA Tesla P100 por 30 horas semanales. Con ella, podemos entrenar los modelos y hacer una inferencia rápida para validación en tiempo razonable.

Los recursos locales que dispongo son un ordenador personal que aunque con mejor capacidad de memoria en disco $\approx 2\ TB$ que la ofrecida por Kaggle $\approx 100\ GB$, nuestra gráfica NVIDIA GeForce RTX 2060 tiene inmensamente menores prestaciones que la ofrecida en Kaggle. A continuación, mostramos una gráfica de rendimiento sobre las características de ambos dispositivos para cuantificar este hecho.

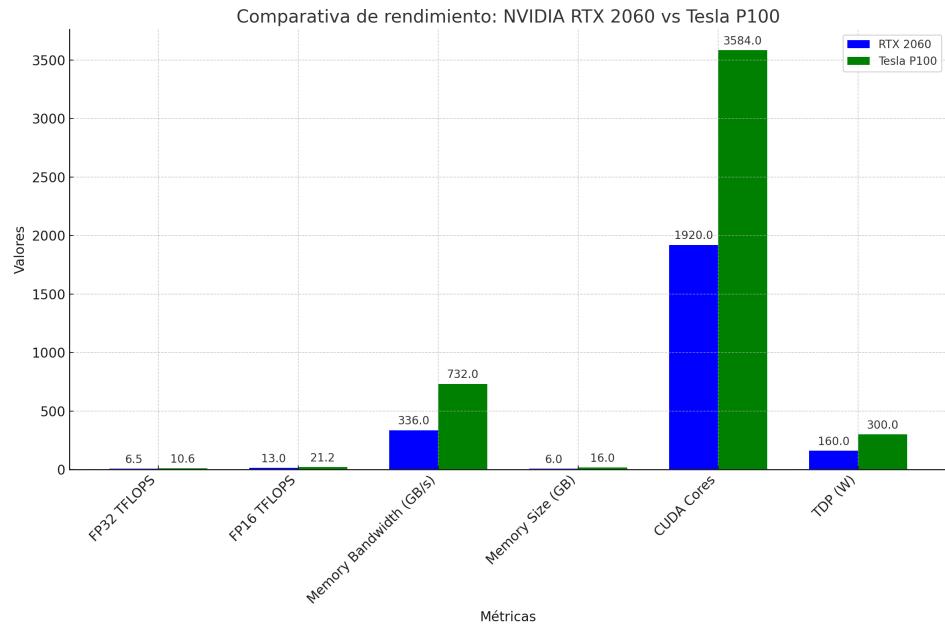


Figura 3.1: Comparativa de rendimiento de las GPU disponibles

En todas las características la gráfica ofrecida por Kaggle supera a la nuestra. Por lo que optaremos por usarla en todo el trabajo.

2. **Hardware para inferir con los modelos.** Para la construcción de la interfaz y su uso si hemos usado nuestro dispositivo personal.

De forma general, el hardware necesario para inferir con los modelos es cualquier PC de uso personal que disponga de una GPU de rendimiento similar o mejor que el nuestro y cumpla las siguientes dependencias (link a apéndice de uso del programa).

3.2. Preprocesado de Datos

En este apartado se explicará el preprocesamiento que se ha aplicado a las resonancias magnéticas para convertirlas a entradas de los modelos.

Partiendo de nuestro conjunto de datos que presentamos en la introducción obtenido de la competición **BraTS** en Synapse. Ya vemos como las resonancias presentan características favorables para ser una entrada a la red.

1. **Dimensiones estandarizadas :** Todas las resonancias (adultos, niños, diferente tipo de tumor) presentan las mismas dimensiones.

2. **Imágenes estandarizadas** : Todas las resonancias se han hecho con el mismo estándar de escáner, todas presentan el mismo rango para su visualización.
3. **No existen valores faltantes** : Observamos como el conjunto de datos es completo en su definición, todas las resonancias de cada paciente tienen las mismas cuatro pruebas.

3.2.1. Elección de dimensionalidad de las entradas

Una de las elecciones cruciales al inicio del trabajo es la dimensionalidad de las entradas de la red (determina la forma en la que se procesan los datos). ¿Es mejor trabajar en 2D con una única imagen como entrada o en 3D con todo el conjunto de imágenes de una resonancia como entrada?

En un primer momento debido a que la mayoría de literatura actual trabaja en 3D, intentamos trabajar en 3D. Para ello, construimos un primer modelo inicial de arquitectura para 2D y una forma de transformarlo a 3D es simplemente duplicar cada capa por la cantidad de imágenes en cada resonancia. Por lo que, el tamaño de nuestro modelo inicial 2D $SIZE_{2D}$ sólo debíamos multiplicarlo por la cantidad de imágenes en una resonancia $SIZE_{3D} = SIZE_{2D} \times 155$. Debido que el máximo de memoria RAM que disponíamos en la Tesla P100 de Kaggle es 16 GB siguiendo esa regla, el máximo tamaño que podría ocupar un modelo inicial 2D sería:

$$MAXSIZE_{2D} = \frac{16 \text{ GB}}{155} = 0.10323 \text{ GB} = 105.7 \text{ MB}$$

Esta cantidad de memoria para una arquitectura que obtenga resultados competentes en la actualidad de técnicas que existen no es viable. Viéndonos obligados ante la escasez de memoria en GPU a enfocar nuestros esfuerzos en una dimensionalidad 2D. Tendremos como entrada a las arquitecturas **una única imagen la correspondiente a la vista axial de las resonancias**.

3.2.2. Normalizado de las imágenes

Las imágenes que componen las resonancias son mapas en escala de gris donde un píxel de la imagen puede tomar un valor de gris en el intervalo [0, 256]. Entre las imágenes de distintas resonancias se encuentra una misma distribución de valores de píxeles para representar la misma información. Sin embargo, el proceso de entrenamiento no deja de ser un proceso de optimización y puede que este rango sea aún demasiado grande.

Adicionalmente, para evitar posibles píxeles erróneos en la toma de las imágenes que podamos interpretar como outliers que tengan un impacto

negativo en el entrenamiento y para hacer las imágenes más interpretables se aplica a las imágenes normalización Z-score o estandarización.

$$X_{std}^i = \frac{x^i - mean}{std}$$

3.2.3. Recortado de imagen

Podría ser razonable reducir las dimensiones de las imágenes para hacer que los datos sean menos pesados. Sin embargo, se opta por no hacerlo por seguridad y escalabilidad. **BraTS** fija esas dimensiones en base al estándar en una resonancia magnética, así para cualquier paciente se garantiza que la imagen de su cerebro se puede representar en una resonancia en unas condiciones de resolución iguales al resto de pacientes.

Si recortamos las imágenes de forma cuadrada al cerebro más grande de todas las resonancias, podríamos encontrarnos en inferencia con un cerebro mayor que no se podría representar en una imagen. Es necesario dejar cierto margen, optando por respetar el margen inicial que marcan los organizadores médicos de **BraTS**.

3.2.4. Undersampling

En el estado del arte ya mencionamos que existía un desbalanceo entre tejido sano y tejido enfermo. En la mayoría de resonancias existe una mayor proporción de tejido sano que de tejido enfermo. Esto no sólo podía introducir un sesgo en los algoritmos de segmentación sino que aumenta mucho el coste computacional de entrenar a los modelos por el exceso de imágenes que no contienen la información de una lesión tumoral.

El tratamiento del desbalanceo mediante undersampling siempre es una medida agresiva ya que podría eliminar información que a priori no consideramos relevante y si lo es.

Sin embargo, en nuestro problema aplicaremos undersampling con el principal objetivo de reducir los tiempos de entrenamiento y poder tener una arquitectura más profunda manteniendo tiempos razonables. Intentando aliviar de paso el problema del desbalanceo. A continuación explicamos en detalle como se ha llevado a cabo.

Para todo nuestro conjunto de datos X creamos archivos CSV para cada partición (entrenamiento, validación y test) donde cada fila de cada archivo CSV representa a una imagen o entrada a la red.

Estos archivos en formato CSV contendrán únicamente el conjunto de imágenes que contienen lesión tumoral de todas las resonancias N más una

parte seleccionada aleatoriamente de imágenes sin lesión de tamaño $\frac{|N|}{2}$. De esta forma, nos quedamos con todas las imágenes con información de lesión y con una parte representativa y balanceada sin lesión para no sesgar al modelo a segmentar en todas las imágenes.

Este es el pseudo-código de creación de los archivos CSV para este proceso de undersampling.

Algorithm 3.1 Undersampling de las imágenes a usar

```

1: Entrada: mri_paths, name
2: Salida: Archivo CSV con columnas (t1c, t1n, t2f, t2w, slice, etiqueta)
3: Iniciar lista vacía data
4: for cada mri_path en mri_paths do
5:   label_path  $\leftarrow$  mri_path[1]
6:   label_img  $\leftarrow$  Cargar label_path
7:   for slice_num en rango(5, 150) do
8:     if hay algún tumor en label_img[:, :, slice_num] then
9:       label  $\leftarrow$  0 si "MEN"  $\in$  label_path, sino 1
10:      Añadir {t1c, t1n, t2f, t2w, slice, etiqueta} a data
11:    end if
12:   end for
13:   Liberar label_img y recolectar basura
14: end for
15: notumores_size  $\leftarrow$  longitud de data dividido 2
16: for cada i en rango(0, notumores_size) do
17:   slice_num  $\leftarrow$  número aleatorio entre 5 y 150
18:   idx  $\leftarrow$  índice aleatorio entre 0 y longitud de mri_paths - 1
19:   if el slice y los paths no están en data then
20:     Añadir {t1c, t1n, t2f, t2w, slice, etiqueta} a data con etiqueta = 2
21:   end if
22: end for
23: Convertir data a DataFrame df
24: Guardar df como archivo CSV con nombre name
  
```

En estos archivos CSV existen las siguientes columnas:

1. **Rutas absolutas** : En 4 columnas están las rutas absolutas a los archivos .nii de cada prueba de cada resonancia.
2. **Número de slice** : Se guarda el número de slice en la que se localiza esa imagen dentro de la resonancia. Este campo es necesario para poder extraer la imagen.
3. **Etiqueta** : Para el problema de clasificación es necesario guardar la etiqueta que identifica a cada imagen, 0 para Meningioma, 1 para

Glioblastoma y 2 para No Tumor.

Tras aplicar este undersampling nos quedamos con 74487 imágenes reducidas I_{RED} en entrenamiento, 31899 en validación y 45354 en test respecto a un total de imágenes $I_{TOTAL} = \text{Pacientes} \times \text{Slices}$. En la siguiente tabla podemos ver recogida esta información también en términos de porcentaje.

Partición	I_{TOTAL}	I_{RED}	Porcentaje %
Entrenamiento	1033×155	74487	46.52
Validación	442×155	31899	46.56
Test	632×155	45354	46.3

Cuadro 3.1: Porcentaje de imágenes conservadas tras undersampling.

3.3. Elección de modelos

A continuación pasamos a discutir los modelos y técnicas empleadas para la creación de las arquitecturas. En este trabajo como al igual que en parte del estado del arte combinaremos técnicas de aprendizaje no supervisado y supervisado.

3.3.1. Codificador y representación latente

Ponemos ahora el foco en un modelo en principio pensado para el aprendizaje sin etiquetas, los **autoencoders**.

Los autoencoders son arquitecturas encoder-decoder con la finalidad de aprender las características de un conjunto de datos o distribución. Por ejemplo, siendo esto útil para obtener modelos generativos como los autoencoders variacionales.

Los autoencoders se formularon inicialmente como una generalización no lineal del análisis de componentes principales (PCA) por su poder para reducir la dimensionalidad. En este trabajo lo incluiremos como modelo base para aplicar aprendizaje no supervisado y que teóricamente presentarían notables ventajas de cara obtener una mayor convergencia y generalización en el proceso de entrenamiento.

A continuación, mostramos un esquema explicativo de las partes implicadas en la arquitectura para la reconstrucción de las imágenes necesaria para el construir el codificador y representación latente.

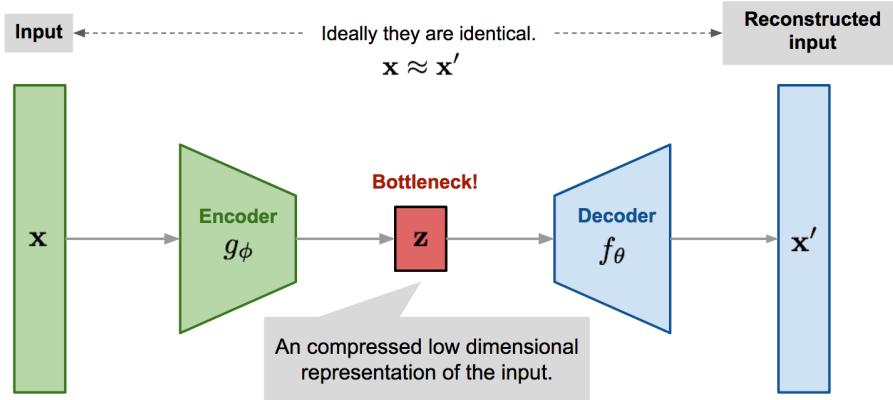


Figura 3.2: Esquema de la arquitectura empleada para inicializar el codificador y representación latente. [Subedi et al., 2022]

El objetivo tras aplicar este esquema es construir un fuerte codificador que reduzca la dimensionalidad y una representación latente (bottleneck) que comprima las características principales de las imágenes del conjunto de entrenamiento.

En términos del proceso de optimización que es el entrenamiento, de forma no supervisada se están inicializando los pesos de la red para parecerse al conjunto de imágenes. [Zeiler and Fergus, 2014] estudia como los diferentes filtros de las redes neuronales convolucionales al entrenarse ante una tarea de clasificación acaban replicando las características generales y específicas de las imágenes de las que son entrenados. Por ello, se llega a la conclusión de que una heurística importante dentro de las redes neuronales convolucionales es que **los filtros se parecen a las imágenes**.

Esta razón explica de forma teórica como un proceso de ajuste previo al conjunto de entrenamiento elimina el coste computacional de búsqueda de los pesos via descenso del gradiente y backpropagation que requiere el ajuste de la red por sí misma a las imágenes sólo a partir de las etiquetas.

3.3.2. Modelo de clasificación

A continuación, presentamos el modelo de clasificación.

La entrada x es el vector de características iniciales. En nuestro caso, son las tres imágenes correspondientes a las 3 pruebas. El codificador es una red neuronal que transforma la entrada de alta dimensionalidad x en una representación de menor dimensionalidad z . El proceso de codificación se puede ver como una función g_ϕ parametrizada por ϕ , que aprende a extraer las características más relevantes de los datos de entrada. La parte final del

esquema es una red neuronal densa que toma la representación comprimida z y realiza la tarea específica del modelo

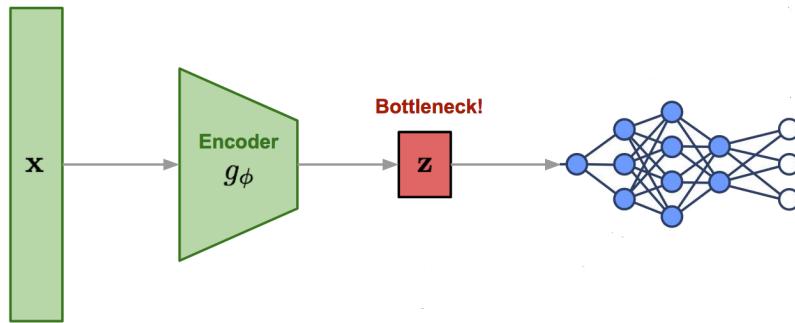


Figura 3.3: Esquema de la arquitectura empleada para el problema de clasificación

Tras ello tenemos un modelo capaz de distinguir entre las etiquetas **glioblastomas, meningiomas o no tumor** para imágenes donde aparecer uno de los esos dos tumores o no aparezca ninguno. Para transformarlo en un clasificación binaria para todas las imágenes de una resonancia aplicamos el siguiente esquema de votación.

Esquema de votación

Tras la salida de las neuronas en las 3 etiquetas creadas para trabajar en 2D: **glioblastomas, meningiomas o no tumor**. Necesitamos una transformación de las predicciones individuales de cada imágenes a toda la resonancia. En la mayoría de los casos ante este tipo de problema se usa un **esquema de votación**.

Consideraremos un esquema de votación que utilice solo la cantidad de predicciones cuando exista tumor, es decir, cuando la etiqueta sea **Glioblastoma** o **Meningioma**. Mostraremos el esquema de votación que usamos en el siguiente pseudo-código.

Algorithm 3.2 Predicción del tipo de tumor de toda la resonancia a partir de las predicciones de todas las imágenes de este

```

1: Function Votación( $Meningioma_{PREDs}$ , threshold)
2: if  $Meningioma_{PREDs} < threshold$  then
3:   return "Meningioma"
4: else
5:   return "Glioblastoma"
6: end if

```

Tras este método sólo queda saber cual es el parámetro *threshold* óptimo. Para ello, simplemente se ajustará a validación tras una búsqueda a través de fuerza bruta.

3.3.3. Modelo de segmentación

A continuación, presentamos el modelo para la tarea de segmentación. Muy similar al esquema para la reconstrucción tenemos un diferente y totalmente nuevo decodificador S_σ específico para la tarea de crear una máscara de segmentación. Se añaden conexiones (concatenaciones) entre el codificador y decodificador para poder preservar la precisión de las características de la imagen original. Finalmente la salida son las etiquetas Y las cuales son las máscaras segmentación reales proporcionadas por **BraTS** y umbralizadas a toda la lesión tumoral.

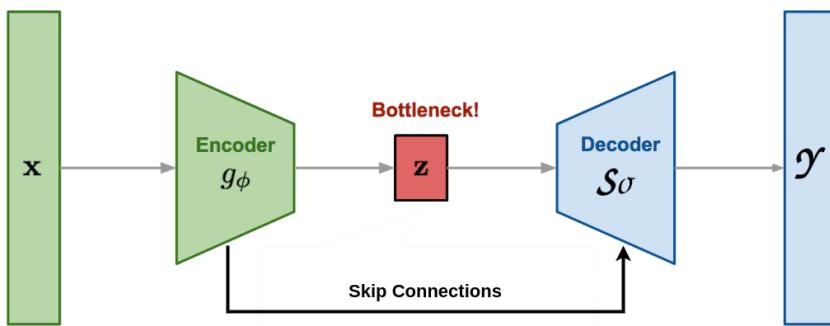


Figura 3.4: Esquema de la arquitectura empleada para el problema de segmentación

3.3.4. Uso de transfer learning

Dentro de las estrategias que podemos utilizar para una mayor convergencia en el aprendizaje y disminuir los tiempos de entrenamiento incluiremos el uso de **transfer learning**.

Debemos tener en cuenta que el entrenamiento en la tarea de la reconstrucción de las imágenes también es una estrategia de transfer learning, ya que transferimos el aprendizaje útil de la tarea de la reconstrucción a las tareas específicas del problema. Sin embargo, en esta sección nos enfocamos en usar transfer learning previamente a esa tarea.

La única parte que usaremos de la arquitectura usada en el autoencoder será el codificador por tanto sólo nos preocupa el uso de transfer learning en esa parte de la red. En la búsqueda de un buen codificador g_ϕ podemos encontrar arquitecturas usadas en clasificación que capturan características

visuales generales, de millones de imágenes diferentes. Es común para multitud de problemas el uso de arquitecturas previas entrenadas con el dataset **ImageNet**.

Optamos por tomar como codificador una arquitectura previamente entrenada en este dataset, las arquitecturas candidatas que probaremos en el desempeño del problema serán:

1. **ResNet34**: Por su multitud de conexiones residuales reportadas en el estado del arte como factor positivo. Con una profundidad de 34 capas está entre sus arquitecturas hermanas ResNet18 y ResNet50 como una arquitectura no muy profunda que ofrece buen desempeño para variedad de problemas.
2. **Xception**: A diferencia de **ResNet34** ofrece kernels de distinto tamaño que podrían favorecer la mejor caracterización de tumores de diferente tamaño.

3.4. Diseño de las arquitecturas

En el siguiente apartado detallaremos los módulos y capas que componen a las arquitecturas así como las componentes importantes.

3.4.1. Función de activación

Para todas las arquitecturas (segmentación, clasificación y el autoencoder para construir el codificador), se optará por el uso de la función de activación ReLU (Rectified Linear Unit), definida como:

$$\text{ReLU}(x) = \max(0, x)$$

Donde x es la entrada a la función ReLU.

A continuación, enunciamos algunas de las razones de su elección y su reconocida robustez para una gran amplitud de problemas en aprendizaje profundo.

1. **No Linealidad**: ReLU introduce no linealidad en las redes neuronales, lo cual es crucial para que las redes puedan aprender y modelar relaciones y características complejas en los datos. Esta no linealidad es esencial para tareas como la clasificación y la segmentación, donde las relaciones entre los datos son inherentemente no lineales.

2. **Gradiente Constante:** Para valores positivos de x , la derivada de ReLU es constante e igual a 1. Esto evita el problema del desvanecimiento del gradiente en redes profundas, donde el gradiente puede volverse extremadamente pequeño en funciones de activación saturadas como la sigmoide y la tangente hiperbólica.

$$\frac{\partial \text{ReLU}(x)}{\partial x} = \begin{cases} 1 & \text{si } x > 0 \\ 0 & \text{si } x \leq 0 \end{cases}$$

Esto facilita el entrenamiento de redes más profundas y la convergencia más rápida durante el proceso de optimización.

3. **Eficiencia Computacional:** La función ReLU es eficiente en términos computacionales. Su implementación es simple (una comparación y una operación de máximo) y no involucra cálculos costosos como funciones exponenciales.

La elección de ReLU como función de activación común a través de estas arquitecturas se basa en sus propiedades matemáticas que promueven la eficiencia, la no linealidad y la estabilidad del gradiente.

3.4.2. Normalización por lotes

Antes de aplicar la función de activación, aplicaremos **normalización por lotes**. PyTorch que es la biblioteca que usaremos implementa la normalización por lotes descrita en [Ioffe and Szegedy, 2015].

La normalización por lotes se aplica a la salida de una capa antes de aplicar la función de activación. Supongamos que tenemos una capa con activaciones \mathbf{x} , donde $\mathbf{x} \in \mathbb{R}^{m \times d}$, siendo m el tamaño del lote y d el número de características en cada ejemplo del lote.

Se calcula la media para cada característica a lo largo del lote:

$$\mu_j = \frac{1}{m} \sum_{i=1}^m x_{ij}$$

donde μ_j es la media de la característica j y x_{ij} es el valor de la característica j del ejemplo i en el lote.

Se calcula la varianza para cada característica a lo largo del lote:

$$\sigma_j^2 = \frac{1}{m} \sum_{i=1}^m (x_{ij} - \mu_j)^2$$

donde σ_j^2 es la varianza de la característica j .

Los datos se normalizan utilizando la media y la varianza calculadas:

$$\hat{x}_{ij} = \frac{x_{ij} - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}}$$

donde \hat{x}_{ij} es la característica j del ejemplo i normalizada, y ϵ es una pequeña constante (generalmente 10^{-5}) para evitar la división por cero y mejorar la estabilidad numérica.

Después de la normalización, se aplica un escalamiento y un sesgo:

$$y_{ij} = \gamma_j \hat{x}_{ij} + \beta_j$$

donde y_{ij} es la salida final para la característica j del ejemplo i , γ_j es un parámetro de escala aprendido, y β_j es un parámetro de sesgo aprendido.

Durante el entrenamiento, tanto γ_j como β_j se optimizan junto con los otros parámetros de la red neuronal mediante el descenso de gradiente.

El uso de normalización por lotes tiene algunos **beneficios**.

- **Estabilización del entrenamiento:** La normalización por lotes ayuda a reducir los problemas de desvanecimiento y explosión del gradiente, permitiendo que las redes neuronales más profundas se entrenen de manera más efectiva.
- **Regularización:** Actúa como una forma leve de regularización al introducir una varianza controlada en el proceso de entrenamiento, lo que a menudo conduce a una mejor generalización del modelo.

3.4.3. Arquitectura para la reconstrucción de imágenes

Como anunciábamos usamos una arquitectura bien conocida como codificador, probaremos la bondad de **ResNet34** y **Xception** en la experimentación. Tras ello, construimos un intuitivo espacio de capas que sean nuestra representación latente.

Arquitectura del codificador ResNet34

A continuación, mostramos y comentamos la arquitectura de uno de los posibles codificadores que usaremos ResNet34.

La innovación clave de ResNet es el bloque residual, que introduce atajos o conexiones directas entre capas no adyacentes de la red. Estos bloques permiten que el flujo de información y los gradientes se propaguen directamente a través de la red, mitigando los problemas de desaparición y explosión del gradiente.

Un bloque residual típico se representa como:

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + \mathbf{x}$$

Aquí, \mathbf{x} es la entrada del bloque, $\mathcal{F}(\mathbf{x}, \{W_i\})$ es la transformación no lineal aprendida (que puede incluir capas como convoluciones, normalización y activaciones), y \mathbf{x} se suma a esta transformación para obtener la salida \mathbf{y} . Esta suma es la conexión residual.

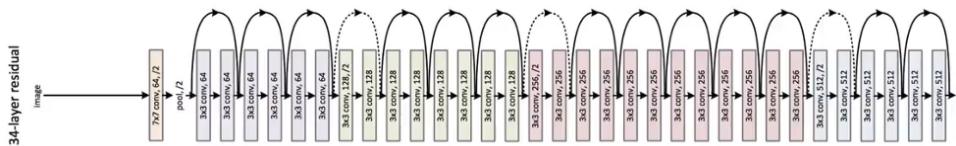


Figura 3.5: Arquitectura usada de ResNet34 [He et al., 2016]

ResNet34 se compone de varias capas organizadas en bloques residuales. Como **capa de entrada** tiene una capa convolucional inicial con 64 filtros de tamaño 7x7 y un stride de 2, seguida de una capa de normalización y una capa de pooling de 3x3 con un stride de 2. Tras ello tenemos los siguientes **bloques residuales**:

- 3 bloques con 64 filtros.
- 4 bloques con 128 filtros.
- 6 bloques con 256 filtros.
- 3 bloques con 512 filtros.

Cada bloque está compuesto por dos capas convolucionales de tamaño 3x3. Los bloques están organizados de manera que cada grupo de bloques de un determinado número de filtros tiene el mismo número de filtros a lo largo del grupo.

Como **capa final** después de todos los bloques residuales, se añade una capa de pooling global seguida de una capa completamente conectada (fully connected) que produce la salida final de la red. En este trabajo eliminamos esas dos capas finales para añadirle la representación latente.

Arquitectura del codificador Xception

A continuación, mostramos y comentamos la arquitectura de uno de los posibles codificadores que usaremos, **Xception**.

La arquitectura Inception, particularmente en sus versiones más recientes como Inception-v3, utiliza módulos Inception para capturar tanto características locales como globales en las imágenes. Sin embargo, estos módulos son complejos y pueden ser optimizados. Xception simplifica y mejora este diseño usando convoluciones separables en profundidad, lo que resulta en una red más eficiente y efectiva.

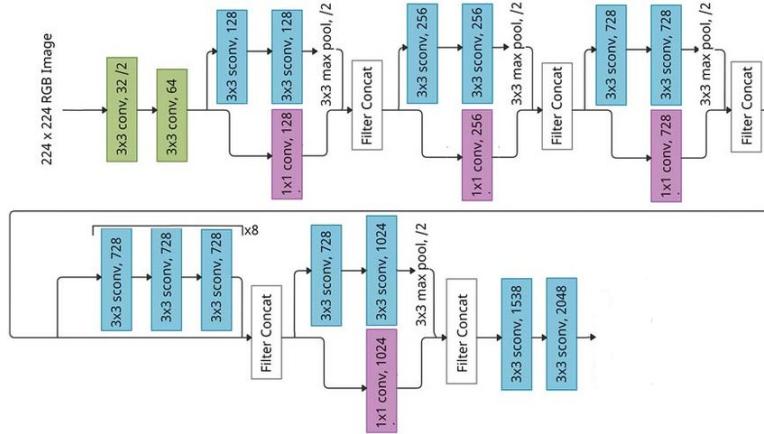


Figura 3.6: Arquitectura usada de Xception [Srinivasan et al., 2021]

La convolución separable en profundidad es una técnica que factoriza una convolución estándar en dos operaciones más simples y eficientes: una **convolución en profundidad** (depthwise convolution) y una **convolución puntual** (pointwise convolution). En una convolución estándar, los filtros de convolución mezclan canales de entrada y espaciales en una única operación. En cambio, la convolución separable en profundidad realiza estos dos pasos por separado.

Xception se compone de varias capas organizadas en bloques de convolución separables en profundidad. Como capa de entrada tiene una capa convolucional inicial con 32 filtros de tamaño 3x3 y un stride de 2, seguida de una capa convolucional con 64 filtros de tamaño 3x3.

- **Entrada al módulo:** Tres bloques con convoluciones separables en profundidad con 128, 256 y 728 filtros respectivamente.
- **Módulos intermedios:** Ocho bloques idénticos con convoluciones separables en profundidad con 728 filtros cada uno.
- **Salida del módulo:** Tres bloques con convoluciones separables en profundidad con 728, 1024 y 1536/2048 filtros respectivamente.

Cada bloque de convolución separable en profundidad consiste en una

convolución en profundidad seguida de una convolución puntual y una capa de activación ReLU.

Al igual que **ResNet34** en la arquitectura original se añade una capa de pooling global seguida de una capa completamente conectada (fully connected) que produce la salida final de la red. Sin embargo, nosotros eliminamos esas capas y le conectamos la representación latente.

Espacio de capas de la representación latente y bloque ConvBlock

A continuación, describiremos qué capas y dimensiones tiene la representación latente. Para todas las tareas (clasificación y segmentación) al igual que el codificador se compartirá la misma representación latente. Podemos ver la arquitectura de la representación latente en un diagrama.

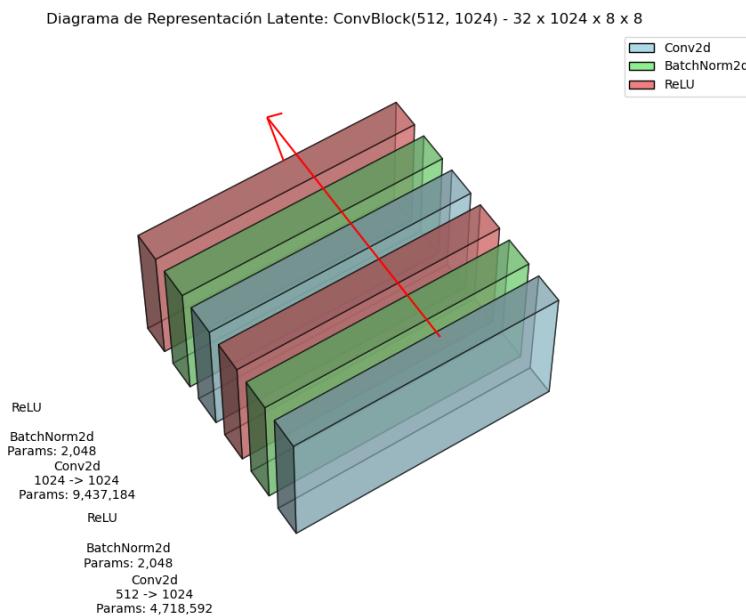


Figura 3.7: Arquitectura de la representación latente

El bloque **ConvBlock(512, 1024)** se utiliza como representación latente dentro de nuestro autoencoder representa una etapa de procesamiento que transforma una entrada de 512 canales (o características) en una salida de 1024 canales mediante operaciones convolucionales seguidas de normalización y activación no lineal. En detalle, el bloque consiste de dos capas convolucionales cada una seguida por capas de Batch Normalization y activación ReLU. La primera capa convolucional transforma los 512 canales de entrada en 1024 canales de salida, manteniendo el tamaño de la entrada

con un kernel de 3x3, stride de 1 y padding de 1. Luego, la segunda capa convolucional opera sobre los 1024 canales de salida, manteniendo este número de canales con el mismo tamaño de kernel y configuración de stride y padding. La capa de Batch Normalization se aplica después de cada capa convolucional para estabilizar y acelerar el entrenamiento, y la ReLU como función de activación introduce no linealidades que ayudan a capturar características más complejas en los datos de entrada. Este tipo de bloque que funciona como representación latente del conjunto de datos, es crucial en redes autoencoder para aprender representaciones más profundas y significativas de los datos, condensando y expandiendo la información a través de capas convolucionales apiladas.

Reducción de canales

Ambas arquitecturas que probaremos **ResNet34** y **Xception** están inicialmente pensadas para imágenes a color, es decir, imágenes en RGB que contenga tres canales o capas pertenecientes a cada tonalidad de colores primarios. Intuitivamente y apoyado por el estado del arte una simple forma de manejar las pruebas para construir una entrada a la red es **concatenarlas**. De esta forma, una entrada a la red estaría compuesta de una imagen de cuatro canales (uno por tipo resonancia que tenemos o prueba), en otras palabras cuatro imágenes en escala de grises concatenadas respecto un eje Z que representaría la profundidad del volumen de la entrada.

Sin embargo, las arquitecturas empleadas como codificador sólo aceptan imágenes de tres canales. Para lidiar con ello se plantea una reducción de dimensionalidad para transformar las imágenes de cuatro canales a tres canales. A nivel práctico se plantea llevarse a cabo mediante:

1. **Transformación mediante análisis de componentes principales PCA a las tres componentes principales.** Se plantea como la mejor solución de las dos que exploraremos ya que se reduce la información manteniendo la mayor variabilidad posible tras combinar linealmente las imágenes.

Podemos apreciar como la imagen de una resonancia es transformada con PCA dando la siguiente salida.

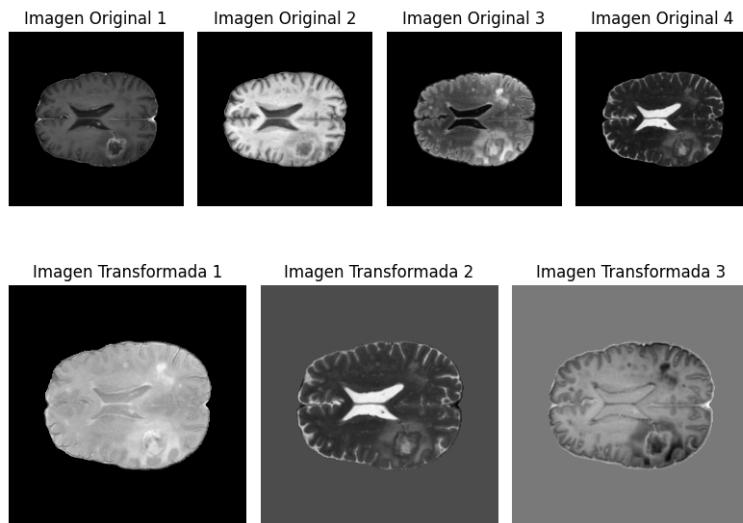


Figura 3.8: Ejemplo de transformación mediante PCA

Para aplicar PCA podemos optar por diferentes estrategias atendiendo si queremos incluirlo o no como una más del preprocesamiento de datos.

- a) **Mediante el preprocesamiento de todo el conjunto de datos aplicando PCA.** Se puede preprocesar los datos y subirlos a Kaggle de nuevo. Sin embargo, esta opción no se pudo aplicar en la práctica debido a las limitaciones de espacio en disco que nos daba la plataforma.
 - b) **Mediante la aplicación de PCA en el mismo entrenamiento antes de introducirlo a la red.** Esta opción elimina la sobrecarga en memoria en disco de Kaggle pero en la práctica es rápidamente descartada ya que aumenta $\approx 50\%$ el tiempo en entrenamiento. Esta implicación teniendo una limitación de tiempo en Kaggle muy ajustada no la hacen compatible con un entrenamiento completo.
2. **Eliminación de una prueba.** Una solución más agresiva es la propia eliminación de un canal de entre los originales. Las principales ventajas de esta solución es la gran facilidad para aplicarla y la mejora del tiempo de extracción de las imágenes puesto ya ahorraríamos traer la eliminada mejorando un 25% los tiempos de extracción de disco.
- Ante la imposibilidad de aplicar PCA se aplicará esta segunda solución. Se elige eliminar a las imágenes de las resonancias de tipo **T2W** por ajuste al problema. Se descarta la eliminación de la información con y sin agente de contraste que proporcionan las tipo T1. Entre la

elección de **T2F** y **T2W** la diferencia principal entre las dos es que en la de tipo **T2W** el líquido cefalorraquídeo contrasta. En toda la revisión médica del problema no se indica que sea un factor relevante en la segmentación de tumores cerebrales por lo que se conserva a la de tipo **T2F** que mantiene una imagen más nítida al no contrastar con el líquido cefalorraquídeo.

Bloque UpConv

Con el objetivo de describir la parte decodificadora de la arquitectura vamos a definir nuestro propio bloque **UpConv**. Definimos el bloque UpConv como una operación de convolución transpuesta seguido de la aplicación del bloque **ConvBlock** definido en el apartado de la representación latente.

La **operación de convolución transpuesta o deconvolución** se define como la inversa de la convolución estándar en una red neuronal convolucional. A continuación, se explica los detalles matemáticos de la convolución transpuesta.

1. **Entrada:** Sea $x \in \mathbb{R}^{C_{in} \times H_{in} \times W_{in}}$, donde C_{in} es el número de canales de entrada, y H_{in}, W_{in} son la altura y anchura de la entrada, respectivamente.
2. **Salida:** Sea $y \in \mathbb{R}^{C_{out} \times H_{out} \times W_{out}}$, donde C_{out} es el número de canales de salida, y H_{out}, W_{out} son la altura y anchura de la salida, respectivamente.

La operación de convolución transpuesta aplica un kernel de convolución de tamaño (k_H, k_W) sobre la entrada x , utilizando un paso (stride) s y un relleno (padding) p . En nuestro caso, siendo siempre constante $p = 0$ y $s = 2$. La operación se define de la siguiente manera:

$$H_{out} = (H_{in} - 1) \cdot s - 2p + k_H$$

$$W_{out} = (W_{in} - 1) \cdot s - 2p + k_W$$

Número de canales de salida: C_{out}

Sea $w \in \mathbb{R}^{C_{out} \times C_{in} \times k_H \times k_W}$ el kernel de la convolución transpuesta.

Para cada posición (h', w') en la salida:

$$y_c(h', w') = \sum_{c'=1}^{C_{in}} \sum_{h=0}^{k_H-1} \sum_{w=0}^{k_W-1} w_{c,c',h,w} \cdot x_{c'}(h' \cdot s + h - p, w' \cdot s + w - p)$$

donde $y_c(h', w')$ es el elemento en la posición (h', w') del canal c de la salida y , $x_{c'}$ es el elemento en la posición correspondiente en el canal c' de la entrada x , y $w_{c,c',h,w}$ es el peso del kernel en la posición (c, c', h, w) .

El principal objetivo de la deconvolución es aumentar las dimensiones espaciales de la imagen de entrada, permitiendo reconstruir información espacialmente más detallada a partir de las características aprendidas. De esta forma, el decoder puede reconstruir las imágenes o máscaras a partir de la reducción de las imágenes entrada a la red.

Decodificador

Construimos el decodificador como la secuencia de cuatro bloques **Up-Conv**. En este caso, para la primera tarea de reconstrucción con entradas y salidas: (1024, 512), (512, 256), (256, 128), (128, 64).

3.4.4. Arquitectura para clasificación

Tras la construcción de la arquitectura para reconstrucción ya tendríamos listos el codificador y la representación latente. Para resolver el problema de clasificación creamos la siguiente arquitectura.

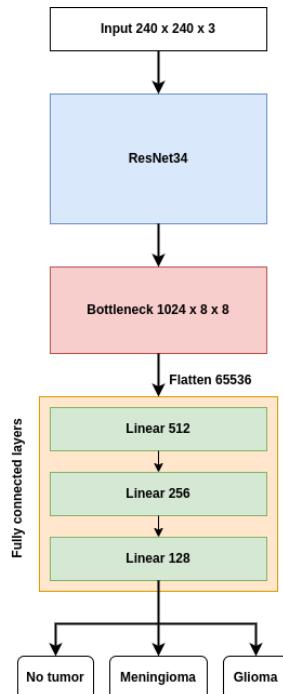


Figura 3.9: Arquitectura para la tarea de clasificación

Utilizamos el codificador seguido de la representación latente tras la representación latente aplazamos su salida con una capa **Flatten** para ahora solo quedará reducir esta salida a 3 neuronas mediante la aplicación de varias capas de neuronas densamente conectadas. En concreto, 3 capas con 512, 256 y 128 neuronas hasta la capa final de 3 neuronas de la que tomamos su máxima neurona como salida.

3.4.5. Arquitectura para segmentación

Tras construir la arquitectura autoencoder para el problema de la reconstrucción, adaptarla para el problema de segmentación es una tarea sencilla. Por un lado, observamos como la estandarización que presentan los datos es favorable para que una misma arquitectura (en el contexto que es una red con unas dimensiones y capas fijas) sea igualmente apta en ambos problemas (reconstrucción y segmentación). Si recordamos [Myronenko, 2019] utilizaba esta idea, su arquitectura de doble tarea comparte codificador y representación latente, y además ambos decodificadores son prácticamente idénticos. Por lo tanto, se mantendrá el mismo diseño de decodificador que en la tarea de la reconstrucción. La única diferencia que se introducirá en segmentación será el uso de **skips connections**.

Skips connections

Con la inclusión de **Skips connections** la arquitectura para la tarea de segmentación se convierte en una variante de una arquitectura ampliamente conocida, la **U-net** [Ronneberger et al., 2015]. Las similitudes que comparte la arquitectura propuesta con ella es que los decoder de ambas arquitecturas son iguales y se tiene el mismo número de Skips connections.

Sin embargo, a diferencia de la U-net que mantiene una simetría en dimensiones entre encoder y decoder, tanto ResNet34 como Xception son un encoder más profundos que el encoder de la U-net original rompiendo la simetría en la arquitectura propuesta. Por otro lado, ambos contienen conexiones residuales transformando la arquitectura en una variante llamada **Residual U-net o ResU-net**.

A continuación, mostramos en un diagrama la arquitectura para la tarea de segmentación.

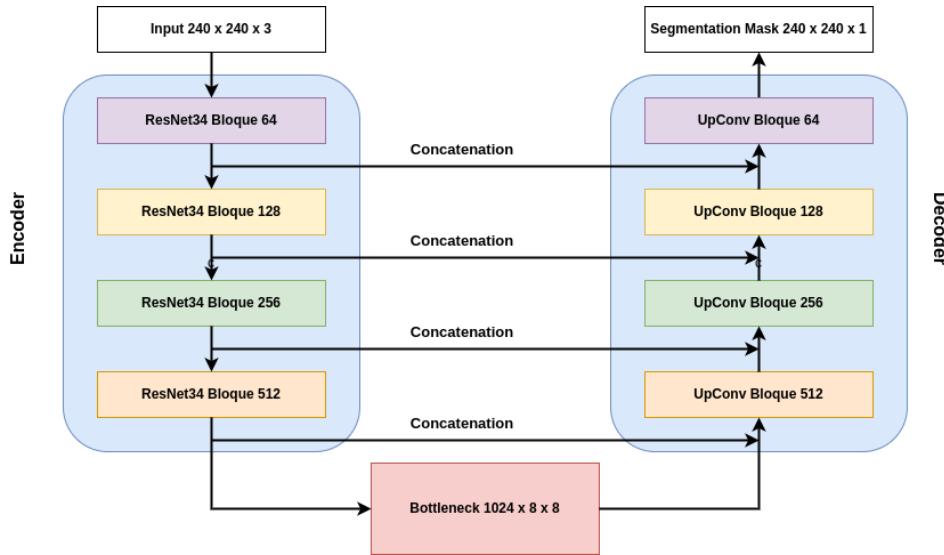


Figura 3.10: Arquitectura para la tarea de segmentación

Podemos ver como representamos a las skip connections como una simple concatenación de igual que forma que pasaba en las conexiones residuales pero de forma más general concatenando la salida de un módulo (grupo de capas) con la entrada de este.

3.5. Optimización de las arquitecturas

En la siguiente sección detallaremos las funciones de pérdida usadas en cada arquitectura y la metodología seguida para llevar a cabo el entrenamiento de la red.

3.5.1. Optimizador

En este trabajo usaremos al optimizador por defecto de **Fastai, Adam**. El optimizador **Adam** (Adaptive Moment Estimation) [Kingma and Ba, 2014] es uno de los métodos de optimización más populares utilizados en el entrenamiento de redes neuronales profundas. Combina las ventajas de otros dos métodos conocidos: AdaGrad y RMSProp, ofreciendo una optimización más eficiente.

Adam ajusta los parámetros de la red neuronal basándose en el promedio de los gradientes y el promedio de los cuadrados de los gradientes. Los pasos principales que sigue son los siguientes:

1. **Gradiente:** En cada iteración del entrenamiento, Adam calcula el gra-

diente de la función de pérdida con respecto a los parámetros del modelo. El gradiente indica la dirección y magnitud en la que los parámetros deben ajustarse para reducir la pérdida.

2. **Promedio de gradientes (primer momento):** Adam calcula un promedio móvil de los gradientes. Esto ayuda a suavizar los cambios bruscos en la dirección del gradiente, proporcionando una dirección más estable para la actualización de los parámetros.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

donde m_t es el promedio móvil de primer momento, g_t es el gradiente en el tiempo t , y β_1 es el coeficiente de decaimiento exponencial para los promedios de primer momento.

3. **Promedio de cuadrados de gradientes (segundo momento):** Adam también calcula un promedio móvil de los cuadrados de los gradientes. Esto permite ajustar la tasa de aprendizaje para cada parámetro individualmente, reduciendo la tasa de aprendizaje para los parámetros con gradientes grandes y aumentando la tasa para los parámetros con gradientes pequeños.

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

donde v_t es el promedio móvil de segundo momento, g_t es el gradiente en el tiempo t , y β_2 es el coeficiente de decaimiento exponencial para los promedios de segundo momento.

4. **Corrección de sesgo:** Al principio del entrenamiento, los promedios móviles pueden estar sesgados hacia cero. Adam corrige este sesgo para obtener estimaciones más precisas de los promedios.

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

5. **Actualización de parámetros:** Finalmente, Adam ajusta los parámetros del modelo utilizando los promedios móviles corregidos.

$$\theta_t = \theta_{t-1} - \frac{\alpha \hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

donde θ_t son los parámetros del modelo en el tiempo t , α es la tasa de aprendizaje, y ϵ es un pequeño valor para evitar la división por cero.

3.5.2. Funciones de pérdida

En este apartado detallamos las funciones de pérdida utilizadas en las diferentes tareas.

Función de pérdida para la reconstrucción de imágenes

Usamos el **error absoluto medio** (MAE) de los píxeles de salida reconstruidos y los verdaderos como función de pérdida para realizar la reconstrucción de las imágenes. El MAE mide la magnitud promedio de las diferencias absolutas entre los valores predichos por el modelo y los valores reales. Esta métrica es útil para evaluar la precisión de la reconstrucción de las imágenes, ya que considera todas las diferencias de manera uniforme, sin penalizar más las diferencias grandes que las pequeñas.

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

Donde n es el número total de píxeles en la imagen, y_i es el valor verdadero del píxel i y \hat{y}_i es el valor reconstruido o predicho del píxel i .

En el contexto de la reconstrucción de imágenes, cada y_i representa la intensidad del píxel en la imagen original, y cada \hat{y}_i representa la intensidad del píxel en la imagen reconstruida por el modelo. El MAE proporciona una medida directa de cuán cerca están las intensidades de los píxeles reconstruidos de las intensidades reales.

También usaremos la misma pérdida como métrica de bondad de ajuste del modelo. Esto significa que, además de utilizar el MAE para optimizar el modelo durante el entrenamiento, también lo emplearemos para evaluar el desempeño del modelo en la reconstrucción de imágenes. Utilizar el MAE como métrica de evaluación nos permite tener una interpretación clara y consistente de cómo de bien se está desempeñando el modelo en términos de error promedio de los píxeles reconstruidos.

Función de pérdida para clasificación

Focal Loss es una función de pérdida que se ha demostrado ser particularmente efectiva para problemas de clasificación, especialmente en contextos donde hay un desequilibrio en las clases o cuando algunas muestras son más difíciles de clasificar que otras. Tras el procesamiento de datos, no existe un desbalanceo de datos grande. Sin embargo, la naturaleza de los tumores hacen que existan imágenes muy variables entre sí (con distinto grado de dificultad de optimización) es por ello que se opta por una pérdida que

tenga algún mecanismo de control sobre esto. A continuación se explica por qué la Focal Loss puede ser mejor que la tradicional Cross Entropy Loss, especialmente en el contexto de datos con diferentes niveles de dificultad:

La Cross Entropy Loss es la función de pérdida estándar para problemas de clasificación y se define como:

$$\text{CE} = -y_i \log(p_i)$$

$$L_{\text{CE}} = - \sum_{i=1}^N y_i \log(p_i)$$

donde y_i es la etiqueta verdadera y p_i es la probabilidad predicha de la clase correcta. Aunque es efectiva, tiene algunas limitaciones:

1. **Tratamiento Igualitario de todos los ejemplos:** Cross Entropy Loss trata todos los ejemplos de entrenamiento por igual, sin importar si son fáciles o difíciles de clasificar.
2. **Sensibilidad al desbalanceo de clases:** No maneja bien los datos desbalanceados, donde algunas clases son mucho más frecuentes que otras.

Focal Loss, propuesta por [Lin et al., 2017], introduce un término adicional para enfocar el entrenamiento en ejemplos más difíciles. Se define como:

$$\begin{aligned} \text{FL} &= -\alpha(1 - p_t)^\gamma \log(p_t) \\ L_{\text{FL}} &= - \sum_{i=1}^N \alpha_t(1 - p_{t_i})^\gamma \log(p_{t_i}) \end{aligned}$$

donde p_t es la probabilidad predicha de la clase correcta, α es un factor de ponderación para balancear la importancia de las clases, y γ es el parámetro de focalización. Se puede desarrollar su principal ventaja:

1. **Enfoque en ejemplos difíciles:** El término $(1 - p_t)^\gamma$ reduce la pérdida para los ejemplos bien clasificados (donde p_t es alto) y mantiene alta la pérdida para los ejemplos mal clasificados (donde p_t es bajo).
 - a) Cuando un ejemplo es fácil de clasificar (p_t es alto), el término $(1 - p_t)^\gamma$ es pequeño, reduciendo su contribución a la pérdida total.
 - b) Cuando un ejemplo es difícil de clasificar (p_t es bajo), el término $(1 - p_t)^\gamma$ es grande, incrementando su contribución a la pérdida total.

2. **Parámetro de Focalización:** El parámetro γ ajusta el grado de focalización. Con $\gamma = 0$, Focal Loss se reduce a la Cross Entropy Loss. Valores más altos de γ incrementan el enfoque en los ejemplos difíciles.
3. **Reducción del impacto de ejemplos fáciles:** Dado que los ejemplos fáciles no dominan el gradiente debido al término de focalización, el modelo no se sesga hacia las clases dominantes o los ejemplos que ya son bien clasificados.

Esto implica los siguientes beneficios cuando existen datos de diferente dificultad.

1. **Mejora del rendimiento general:** En muchos problemas de clasificación, hay ejemplos que son intrínsecamente difíciles de clasificar debido a la superposición de clases, ruido en los datos, o características ambiguas. Focal Loss ayuda a mejorar el rendimiento general del modelo al hacer que el modelo preste más atención a estos ejemplos difíciles durante el entrenamiento.
2. **Mejor manejo de outliers:** Al reducir la contribución de ejemplos fáciles a la pérdida total, Focal Loss puede ayudar a mitigar el impacto de outliers o ejemplos atípicos que podrían desviar el modelo cuando se usa Cross Entropy Loss.
3. **Robustez en modelos de gran escala:** En problemas similares como la detección de objetos en imágenes, donde el desequilibrio entre fondo y objeto puede ser significativo y algunos objetos son más difíciles de detectar que otros, Focal Loss ha demostrado ser particularmente útil.

Focal Loss es una mejora sobre Cross Entropy Loss al proporcionar un mecanismo para enfocarse en ejemplos difíciles tales como imágenes donde los tumores sean pequeños. Con la gran en este problema de que realmente no queremos principalmente predecir muy bien la mayoría de ejemplos sino ser precisos en los ejemplos especialmente difíciles también para el personal médico. Por todo ello, se opta por utilizar Focal Loss para la primera tarea de clasificación.

Función de pérdida para segmentación

Como función de pérdida en segmentación se escoge el complemento negativo de su métrica más importante **similaridad Dice: Dice Loss**. Dice Loss se define como:

$$L_{dice} = 1 - Dice(A, B)$$

Por lo tanto, **Dice Loss** varía entre 0 y 1, donde 0 indica una superposición perfecta y 1 indica ninguna superposición (es decir, pérdida máxima).

Podemos enumerar las características que hacen de Dice Loss una de las pérdidas más acertadas para la segmentación de tumores cerebrales.

1. **Sensibilidad a pequeñas áreas:** En la segmentación de tumores cerebrales, es crucial detectar incluso pequeñas regiones de tumores. El Dice Loss está diseñado para ser sensible a estas pequeñas áreas de superposición, lo que permite que el modelo se entrene para identificar y segmentar con precisión los bordes y áreas menos visibles del tumor.
2. **Interpretabilidad clínica directa:** El coeficiente de similaridad Dice, en el que se basa el Dice Loss, proporciona una medida directa de cuán bien la predicción del modelo coincide con la realidad clínica. Esta métrica es intuitiva para los profesionales de la salud, ya que refleja la precisión de la segmentación en términos de superposición de áreas tumorales detectadas.
3. **Robustez ante desbalanceo en la distribución de clases:** Los tumores cerebrales pueden representar solo una pequeña fracción de la imagen total, lo que genera desequilibrios en la distribución de clases. Dice Loss maneja este desequilibrio al penalizar menos la falta de predicción en áreas dominadas por el tejido cerebral normal, centrándose en mejorar la detección precisa de los tumores.
4. **Optimización efectiva en el entrenamiento:** El uso de Dice Loss proporciona una superficie de optimización más suave y estable durante el entrenamiento de modelos de segmentación que otras pérdidas. Esto facilita la convergencia del modelo y reduce la necesidad de ajustes complejos de hiperparámetros, mejorando así la eficiencia del entrenamiento y la calidad de las segmentaciones obtenidas.

3.5.3. One-Cycle Policy

La **política 1cycle** fue propuesta por [Smith and Topin, 2019], y es una técnica de programación de tasas de aprendizaje para mejorar el rendimiento del entrenamiento de redes neuronales. El objetivo principal es encontrar tasas de aprendizaje que aceleren el entrenamiento y mejoren la generalización del modelo. También ajusta la tasa de aprendizaje durante el entrenamiento de manera específica en función del número de iteraciones. Esta utiliza un ciclo de entrenamiento único en lugar de tasas de aprendizaje fijas durante

todo el entrenamiento. Durante este ciclo, la tasa de aprendizaje varía de manera cíclica desde un valor inicial hasta un valor máximo y luego vuelve a descender.

En la primera mitad del ciclo, la tasa de aprendizaje aumenta gradualmente desde un valor inicial hasta un valor máximo. Este aumento permite una exploración más rápida del espacio de parámetros y puede ayudar a escapar de mínimos locales subóptimos. En la segunda mitad del ciclo, la tasa de aprendizaje disminuye gradualmente desde el valor máximo hasta el valor inicial. Esto permite una refinación más precisa de los pesos del modelo y una convergencia más estable hacia el mínimo global.

Además de ajustar la tasa de aprendizaje, la política 1cycle también propone variar el **momentum** (una técnica que ayuda a acelerar el entrenamiento). El momentum se incrementa en la fase de aumento y disminuye en la fase de descenso.

La política 1cycle también actúa como una forma de regularización. El uso de tasas de aprendizaje más altas en la fase de aumento y más bajas en la fase de descenso puede ayudar a prevenir el sobreajuste. También sugiere seleccionar los valores iniciales y máximos de la tasa de aprendizaje de manera específica. El valor máximo de la tasa de aprendizaje se elige típicamente como un valor que es varias veces mayor que el valor inicial.

La duración total del ciclo puede variar, pero en general, se sugiere que sea aproximadamente igual al número total de épocas de entrenamiento. La **política 1cycle** ha demostrado ser efectiva en mejorar la convergencia y la generalización en una variedad de tareas y arquitecturas de redes neuronales. Sin embargo, la selección precisa de los parámetros, como las tasas de aprendizaje inicial y máxima requiere ajustes específicos para cada objetivo y conjunto de datos.

En el uso conjunto de una política de **1cycle** y el optimizador **Adam** como se aplica en este trabajo, podemos entender como **1cycle** ajustaría la tasa de aprendizaje para una misma época (una tasa de aprendizaje global a una época) y Adam más tarde la refinaría ligeramente para cada ejemplo dentro de una época.

En este trabajo usaremos esta política como forma de regularización y como método general para evitar la búsqueda de hiperparámetros manual para el proceso de aprendizaje. Para ello, en todo el trabajo sólo usaremos las llamadas a las funciones `fit_one_cycle()` para entrenar toda las capas de la red y `fine_tune()` para cuando necesitemos diferenciar qué épocas entrenar toda la red (incluyendo la parte convolucional) o solo la parte densamente conectada. Ambas funciones son de la librería **Fastai** que implementan la política 1cycle.

3.6. Evaluación y métricas

Para la evaluación se distinguirán entre tres subconjuntos de datos: entrenamiento, validación y test. Se utilizará un conjuntos fijos basado en **hold out**, común a todas las tareas. Por motivos de eficiencia no podemos permitirnos usar validación cruzada o distintos conjuntos variables. Estos conjuntos serán separados mediante la utilización de archivos CSV con las rutas absolutas de cada ejemplo para cada uno de ellos.

Seguirán la siguiente distribución: un 49 % para entrenamiento, 21 % para validación y un 30 % para test del conjunto total de datos (glioblastomas y meningiomas juntos) haciendo un reparto aleatorio entre ellos. Se sigue la regla de partición: 70 % entrenamiento + validación y 30 % test. La finalidad de cada subconjunto es para entrenamiento ajustar los modelos a este subconjunto de datos, el objetivo del conjunto de validación es elegir el mejor modelo de los probados, y el conjunto de test dar el resultado final garantizador de la bondad de los modelos.

3.6.1. Métricas para clasificación

Las métricas que vamos a usar para el proceso de clasificación son **accuracy balanceado** y **accuracy** que podremos calcular a través de la construcción de la matriz de confusión.

1. **Accuracy.** Mide cuántas predicciones del modelo son correctas respecto el total de predicciones. Se calcula como:

$$\text{Accuracy} = \frac{\text{Nº de predicciones correctas}}{\text{Total de predicciones}} = \frac{TP + TN}{TP + TN + FP + FN}$$

2. **Accuracy balanceado.** Esta métrica se usa en problemas de clasificación para manejar conjuntos de datos desbalanceados, donde las clases no están igualmente representadas. Se define como el promedio de las tasas de acierto (recall) obtenidas en cada clase, lo que ayuda a proporcionar una evaluación más justa y equilibrada del desempeño del modelo cuando las clases tienen diferentes tamaños.

De forma general para un problema de clasificación de N clases se calcula como:

$$\text{Balanced Accuracy} = \frac{1}{N} \sum_{i=1}^N \frac{TP_i}{TP_i + FN_i}$$

De forma específica, para clasificación binaria $N = 2$ podemos expresarlo como:

$$\text{Balanced Accuracy} = \frac{1}{2} \left(\frac{TP}{TP + FN} + \frac{TN}{TN + FP} \right)$$

3.6.2. Métricas para segmentación

Siguiendo la definición de evaluación de **BraTS** 2023, la evaluación de los tumores en su forma original comprende la segmentación de tres zonas diferenciadas según los tipos de tejidos que hay en ella. En **BraTS** se estudian los siguientes tres conjuntos de tejidos tumorales.

1. **Enhancing Tumor ET**: Sólo incluye al tejido ET.
2. **Tumor Core TC**: Incluye al tejido ET y al tejido NCR.
3. **Whole Tumor WT**: Incluye a todos los tejidos enfermos: ET, NCR y ED. La segmentación de toda la lesión.

Sin embargo, en este trabajo hacemos una reducción del problema por falta de recursos a **la segmentación únicamente del conjunto WT Whole Tumor**. De esta forma, la misión que tendremos es la de segmentar tejido enfermo o lesionado en todo el volumen de la resonancia. En otras palabras, diferenciar con la segmentación el tejido enfermo del tejido sano o la no existencia de tejido.

Para ello, se utilizará tres métricas. Las dos primeras utilizadas en todas las competiciones de **BraTS** y la tercera adicionalmente utilizada en los trabajos que han ido conformando el estado del arte como [Zhou et al., 2021].

1. **Similaridad Dice** : Mide la similaridad de dos conjuntos a través de la intersección de ambos respecto el tamaño total de los dos conjuntos.

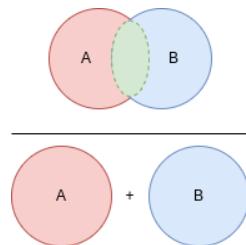


Figura 3.11: Interpretación del coeficiente de Similaridad Dice

Podemos expresarlo como:

$$Dice(A, B) = \frac{2 \times |A \cap B|}{|A| + |B|}$$

Donde A es la segmentación que proporcionará nuestro modelo y B la verdadera.

De forma más detallada, para una resonancia completa podemos expresarlo como:

$$Dice(A, B) = 2 \cdot \frac{\sum_{i=1}^N \sum_{j=1}^C A_{ij} B_{ij} + \epsilon}{\sum_{i=1}^N \sum_{j=1}^C A_{ij} + B_{ij} + \epsilon}$$

Donde N es el conjunto de todas las slices de la resonancia, C el conjunto de clases en nuestro caso $C = 2$, A_{ij} es el valor de la predicción en el pixel i para clase j . B_{ij} es el valor real en el pixel i para la clase j . ϵ es una pequeña constante para evitar dividir entre 0.

2. **Distancia Hausdorff** : Esta métrica tiene el objetivo de medir geométricamente la mayor distancia resultada entre A la predicción del modelo y la verdadera segmentación B , siendo A y B , conjuntos de puntos o píxeles.

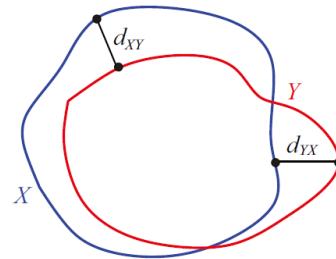


Figura 3.12: Distancias de Hausdorff entre dos conjuntos

Podemos enunciar la distancia máxima de Hausdorff:

$$H(A, B) = \max \left\{ \max_{a \in A} \min_{b \in B} d(a, b), \max_{b \in B} \min_{a \in A} d(a, b) \right\}$$

Donde a, b son puntos concretos en los conjuntos, $d(a, b)$ la distancia euclíadiana entre los dos puntos y A, B los conjuntos de puntos.

Obtendremos la distancia de Hausdorff media de todos los slices de cada resonancia:

$$\bar{H}(A, B) = \frac{1}{|N|} \sum_{s \in N} H(A, B)$$

De esta forma, a menor distancia de Hausdorff la segmentación salida y real son geométricamente más parecidas.

3. **Sensibilidad o Recall** : Mide la proporción de casos positivos que fueron correctamente identificados por el modelo. En otras palabras y para nuestro problema, mide cuánto la segmentación predicción coincide con la segmentación verdadera, cuantificado en porcentaje.

Podemos expresarlo como:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

Donde TP son los pixeles que son verdaderos positivos y FN los pixeles que son falsos negativos.

3.7. Desarrollo para la predicción de la evolución

Por falta de datos y recursos no pudimos entrar a resolver la predicción de la evolución de una lesión tumoral. Sin embargo, al menos teóricamente planteamos resolver esta tercera tarea. Es necesario tener en cuenta que la solución que aportaremos se sustenta en justo lo que no tuvimos, **más imágenes de una misma resonancia en diferentes momentos en el tiempo**. Se cree que la existencia de estos datos es una condición necesaria, ya que hasta el conocimiento que se tiene en este trabajo soluciones sin etiquetas no son viables.

Para construir una solución a este problema podemos valernos del **mismo encoder g_ϕ y representación latente z** utilizados en todas las arquitecturas con anterioridad. Posteriormente, podemos conectar un **Visual Transformer** que tenga como entrada la salida aplanada de la representación latente y tenga como output la segmentación en una instancia a corto plazo futura.

El objetivo del uso de una arquitectura transformadora es encontrar la correlación entre las **palabras** de entrada del decodificador (patches o píxeles de las imágenes) para poder minar (**si existen**) detalles en la imagen que lleven a poder predecir en el corto plazo cómo se extendería la lesión tumoral.

A continuación, podemos ver el esquema de este modelo hipótesis planteado.

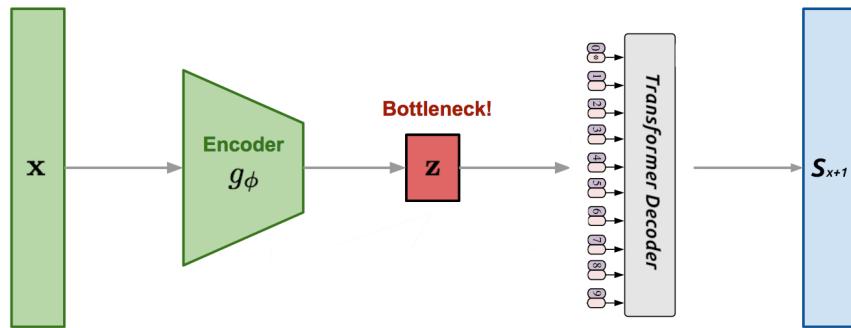


Figura 3.13: Arquitectura propuesta para predicción de la evolución de los tumores

A continuación, hacemos algunas consideraciones de cómo llevar a cabo la optimización de esta arquitectura y detalles para poder llevárselo a la práctica.

1. **Sigue siendo un problema de segmentación.** En la formulación del problema que estamos dando no estaríamos reconstruyendo la imagen futura sino su segmentación. Por lo tanto, pérdidas como **Dice Loss** seguirían siendo prometedoras.
2. **Discretización del tiempo.** El tiempo es una magnitud que de forma precisa es discretizada en intervalos muy pequeños, por ejemplo podría ser discretizada en horas o minutos. En este problema carece de sentido una discretización demasiado precisa porque justamente lo interesante es ver las diferencias con la segmentación original para adelantarse a ellas. Un buen intervalo de discretización antes de entrenar el modelo y con datos en este intervalo son necesarios.

Capítulo 4

Experimentación

En este capítulo se recoge el desarrollo de los experimentos llevados a cabo en este trabajo y todos los resultados que se han obtenido empíricamente de estos.

4.1. Bibliotecas y desarrollo de los experimentos

Para el desarrollo de los experimentos usaremos las bibliotecas de aprendizaje profundo **PyTorch** para la construcción de la estructura del proyecto y **Fastai** biblioteca extensión de PyTorch para el proceso de entrenamiento de los modelos. Construiremos nuestras propias clases en PyTorch para la construcción de las arquitectura y para la creación de la instancia de trabajo de nuestro dataset. Tenemos las siguientes componentes.

1. **Clase del Dataset:** **BraTS**. Clase que representa al conjunto de datos. Sus funciones principales son `len(self)` que devuelve la cantidad de datos en la clase y `getitem(self, index)` que devuelve un dato de la clase dado un índice. Esta clase hereda de la clase **Dataset** de PyTorch. La instancia de esta clase será la entrada a la clase **DataLoader** de PyTorch.
2. **Clases de las arquitecturas:** **SegNet** y **BinaryNet**. Implementan las arquitecturas para clasificación y segmentación. Aparte del constructor que es donde se define la arquitectura tiene la función `forward(self, x)` que define como se infiere por la red. PyTorch solo necesita la definición de la inferencia para calcular automáticamente la función `backward` internamente la función para la aplicación de back-propagation (necesario en el entrenamiento). Usa las instancias de los módulos siguientes.

3. **Clases de los módulos de submuestreo y sobremuestreo: DownConv y UpConv.** Análogas a las clases de la arquitecturas pero restringiéndolo a una parte. Usa la instancia de la clase siguiente (ya que sólo definimos uno de estos bloques por módulo).
4. **Clase de un bloque de convolución: ConvBlock.** Clase que define las capas de un bloque de en nuestra red. Dentro del constructor llama a las funciones de PyTorch de ReLU, convolución y batch normalization.

A continuación, podemos ver en detalle la estructura de todos nuestros experimentos a través de un diagrama de clases.

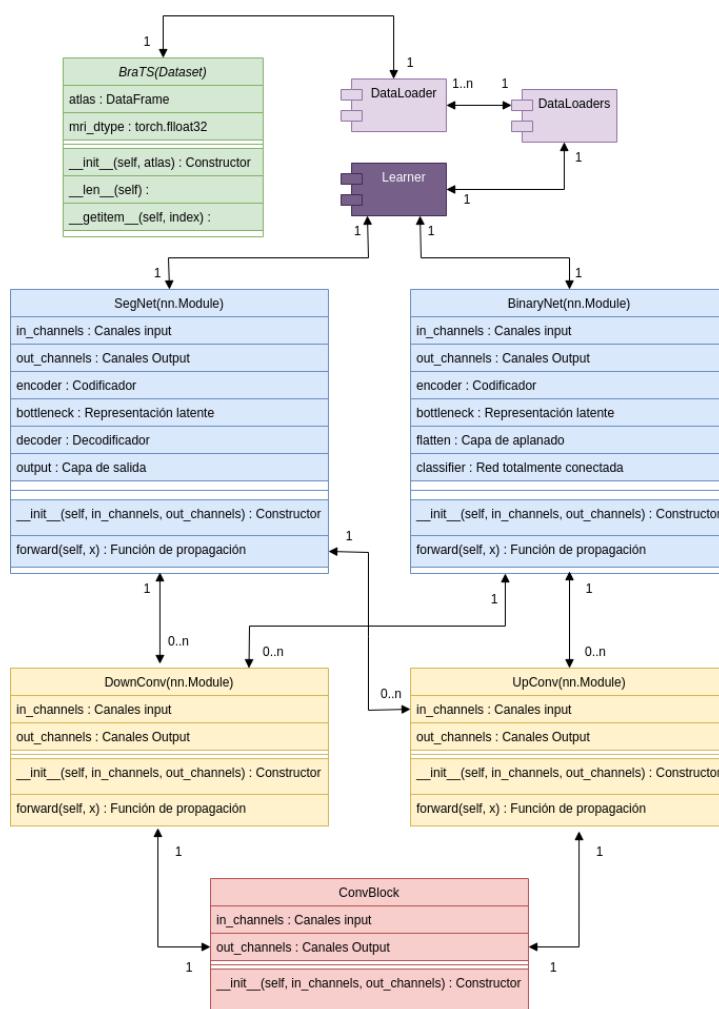


Figura 4.1: Diagrama de clases de PyTorch

4.2. Construcción del codificador y representación latente

Pasamos a comentar los experimentos llevados cabo en la tarea de la reconstrucción de imágenes.

4.2.1. Arquitecturas con conexiones residuales: ResNet34

Comenzamos ajustando **ResNet34** a las imágenes. Tras 7 épocas sumando un total de 8 horas 30 minutos de entrenamiento para toda la red obtenemos estos resultados. Podemos ver los resultados en forma tabular y en forma de gráfica. En la gráfica (ver Figura 4.2) el eje *Y* representa la pérdida y el eje *X* representa la cantidad de imágenes vistas (aunque haya visto la misma imagen más de una vez en distintas épocas).

epoch	train_loss	valid_loss	MAE	time
0	0.098790	0.111301	0.111301	1:09:46
1	0.079261	0.081548	0.081548	1:13:33
2	0.071293	0.075421	0.075421	1:14:20
3	0.064748	0.067580	0.067580	1:14:41
4	0.059911	0.062933	0.062933	1:15:55
5	0.057501	0.059307	0.059307	1:15:36
6	0.055986	0.058405	0.058405	1:15:37

Cuadro 4.1: Pérdida de entrenamiento y validación para la reconstrucción con ResNet34

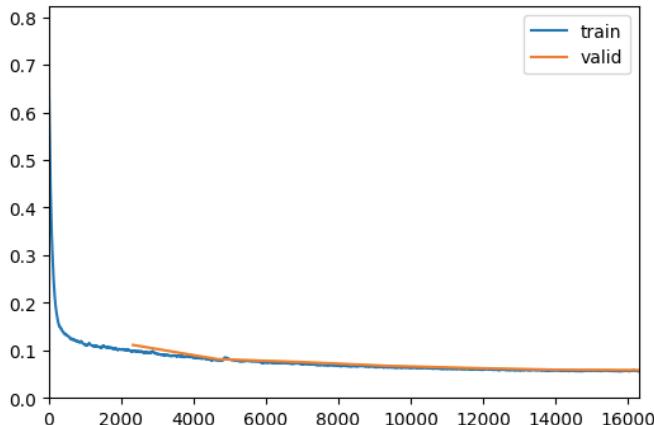


Figura 4.2: Curva de aprendizaje con ResNet34

Observamos una rápida convergencia al inicio y un entrenamiento perfecto con ambos errores muy cercanos en todas las épocas $E_{val} \approx E_{train}$. Indicando que el proceso de entrenamiento está realizando correctamente y que la arquitectura es válida para aprender a reconstruir las imágenes. Finalmente, obtenemos un $E_{val} \approx 0.058$ lo cual indica que nuestra red reconstruye las imágenes con una pérdida del 5.8 % de los detalles reales.

A continuación, observamos cómo la red reconstruye tres imágenes. En la siguiente salida vemos tres imágenes donde podemos apreciar la salida de la red como la imagen de la izquierda de cada pareja y la imagen real a la derecha.

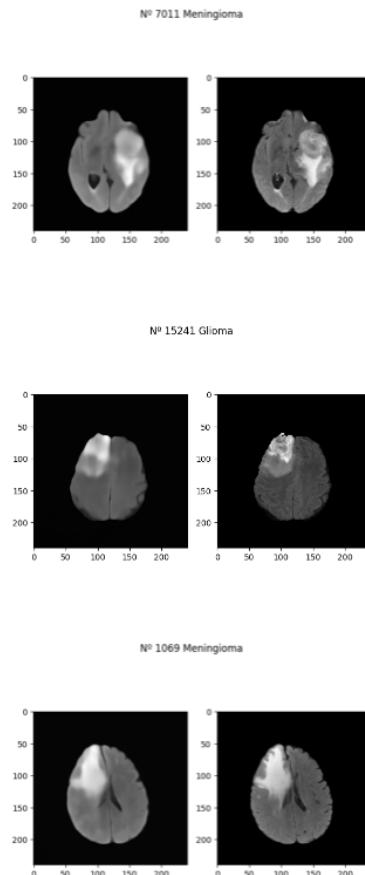


Figura 4.3: Reconstrucción de las imágenes con ResNet34

4.2.2. Arquitecturas con filtros con distinto tamaño: Xception

Dentro de las pruebas para elegir el mejor codificador seguimos con la prueba de bondad de **Xception**. Lo evaluamos en las mismas condiciones que **ResNet34**.

epoch	train_loss	valid_loss	MAE	time
0	0.102650	0.105854	0.105854	1:24:38
1	0.083989	0.089979	0.089979	1:28:34
2	0.074420	0.085525	0.085525	1:28:33
3	0.068684	0.071334	0.071334	1:30:09
4	0.063998	0.065802	0.065802	1:26:00
5	0.060919	0.063771	0.063771	1:27:03
6	0.060293	0.062978	0.062978	1:26:51

Cuadro 4.2: Pérdida de entrenamiento y validación para la reconstrucción de Xception

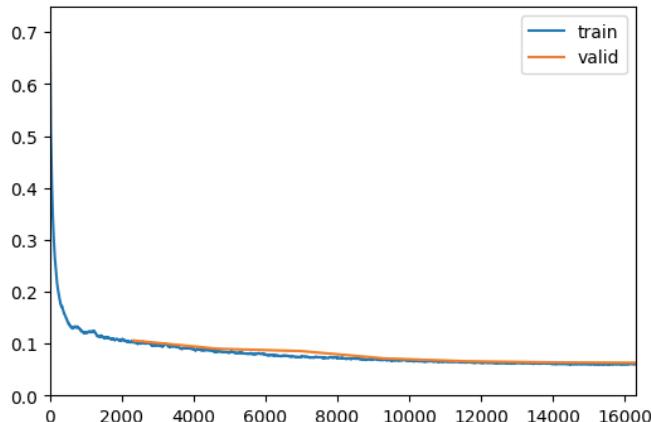


Figura 4.4: Curva de aprendizaje con Xception

En la Figura 4.4, observamos una más rápida convergencia que **ResNet34** motivado posiblemente por los filtros de distintos tamaños. Tenemos el mismo comportamiento de bondad entre la pérdida de entrenamiento y validación. Sin embargo, obtenemos una pérdida en validación de $E_{val} \approx 0.062$ la cual es superior a la de **ResNet34** indicando que a pesar de la rápida convergencia el resultado final es mejor con un mayor número de conexiones residuales.

A continuación observamos su reconstrucción con un resultado muy si-

milar en apariencia.

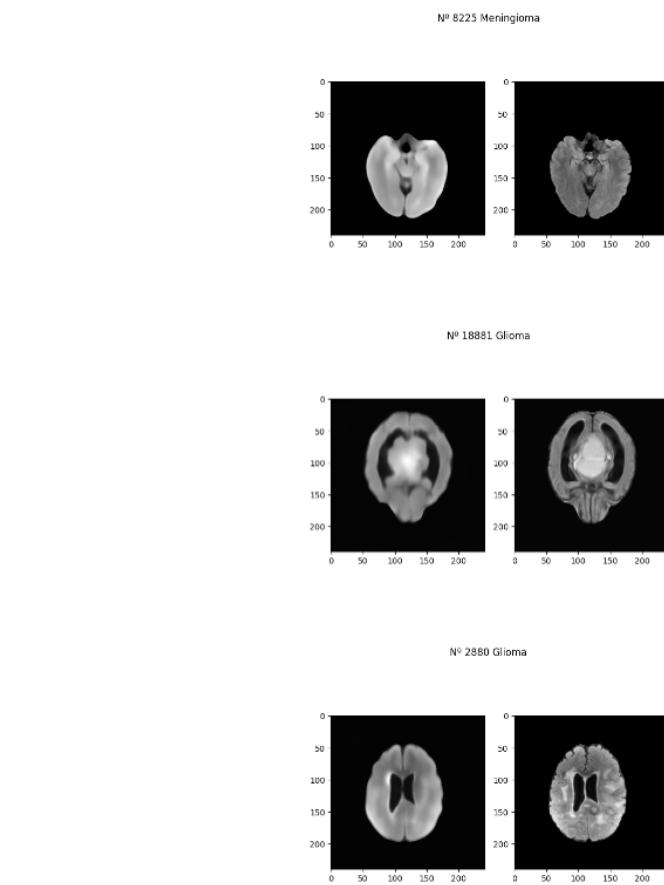


Figura 4.5: Reconstrucción de las imágenes con Xception

A continuación, ante los resultados de esta comparación mostramos la siguiente tabla.

Arquitectura	E_{train}	E_{val}
ResNet34	0.055986	0.058405
Xception	0.060293	0.062978

Cuadro 4.3: Pérdida de entrenamiento y validación para clasificación con la parte convolucional congelada

Eligiendo por tanto a **ResNet34** como codificador de las arquitecturas.

4.3. Clasificación

4.3.1. Entrenamiento en clasificación

A continuación, mostramos los experimentos realizados para la tarea de clasificación. Tomamos **ResNet34** como codificador le añadimos la representación latente y las capas densamente conectadas, entrenamos las capas fully-connected congelando las capas convolucionales (codificador y representación latente) durante 3 épocas tomando ≈ 3 horas con estos resultados.

epoch	train_loss	valid_loss	accuracy	balanced_accuracy	time
0	0.112794	0.178968	0.805254	0.781898	1:04:50
1	0.119699	0.174729	0.819330	0.773066	1:03:01
2	0.083459	0.141255	0.855540	0.794527	1:04:18

Cuadro 4.4: Pérdida de entrenamiento y validación para clasificación con la parte convolucional congelada

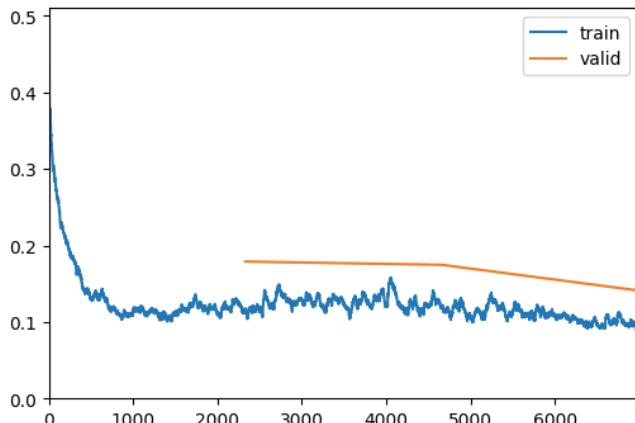


Figura 4.6: Curva de aprendizaje para clasificación parte convolucional congelada

En la figura 4.6, observamos una rápida convergencia al inicio del entrenamiento, estabilizándose durante la época número 1 y volviendo a converger ligeramente durante la época 2. En este caso y para esta primera fase, el entrenamiento no es tan óptimo, habiendo cierta distancia entre las pérdidas de validación y entrenamiento.

A continuación, tras estas 3 épocas ajustando las capas densamente conectadas pasamos a descongelar toda la red para que quede mejor ajustado.

epoch	train_loss	valid_loss	accuracy	balanced_accuracy	time
0	0.056434	0.209338	0.855477	0.840139	1:01:57
1	0.040761	0.190361	0.864976	0.812309	1:01:41
2	0.023256	0.260177	0.876575	0.832247	1:00:26
3	0.009723	0.330192	0.877014	0.834721	1:00:30

Cuadro 4.5: Pérdida de entrenamiento y validación para clasificación toda la red descongelada

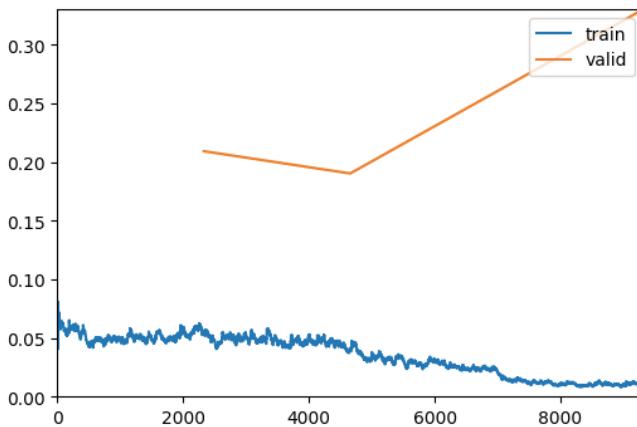


Figura 4.7: Curva de aprendizaje para clasificación toda la red descongelada tras ajustar capas densamente conectadas

En la figura 4.7, observamos como en las dos primeras épocas tenemos convergencia en validación, tras ellos se dispara validación indicándonos que hemos llegado al resultado tope en el entrenamiento. Este proceso de entrenamiento esta configurado con **EarlyStopping** por lo que PyTorch automáticamente guarda el mejor modelo (el que haya conseguido una pérdida en validación más baja en todas las épocas) en este caso tras descongelar obtiene como modelo final el correspondiente a la época número 1.

Observamos como tras haber pasado el proceso de ajuste inicial de las capas densamente conectadas al descongelar todo, obtenemos unas pérdida de validación y entrenamiento mucho más dispares. Una posible explicación es debido a que las diferentes partes de la arquitectura inducen complejidad en el proceso de entrenamiento: las capas densamente conectadas (que era lo único entrenable en la primera fase) hacen que esta diferencia sea muy pequeña porque estas capas tienen pocos parámetros entrenables en comparación a la capas previas. Al descongelar toda las capas de la red, siendo el codificador mucho mayor que las capas densas se observa como se induce un

ruido mayor.

A continuación, y a pesar de que la teoría indica que primero ajustemos las capas densamente conectadas primero congelando el resto de las capas en sus inicios, probamos a entrenar toda la red descongelada sin un ajuste previo.

epoch	train_loss	valid_loss	accuracy	balanced_accuracy	time
0	0.080457	0.178836	0.854348	0.796523	1:23:49
1	0.084583	0.175143	0.847827	0.781100	1:24:09
2	0.086810	0.159217	0.852185	0.800501	1:26:20
3	0.081304	0.370888	0.832309	0.747997	1:21:31

Cuadro 4.6: Pérdida de entrenamiento y validación para clasificación para todo el entrenamiento toda la red descongelada

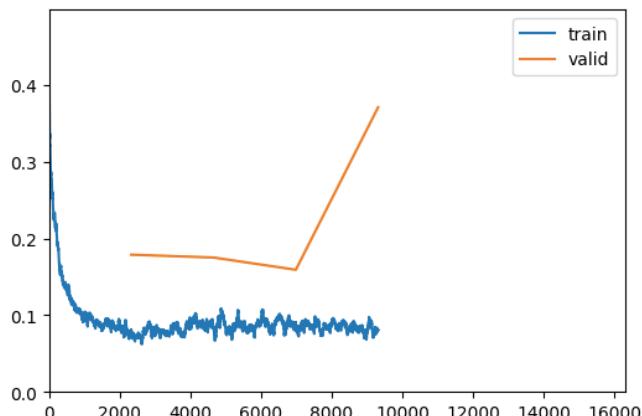


Figura 4.8: Curva de aprendizaje para clasificación todo el entrenamiento toda la red descongelada

En la figura 4.8, observamos como el mejor resultado que obtenemos es en la época número 2 siendo peor que la mejor con la estrategia anterior. Adicionalmente, observamos como se alcanza una convergencia rápida llevando a fuerte overfitting justo en la siguiente época. Podemos interpretar como esta estrategia no sólo da peores resultados sino tiene una peor estabilidad en el proceso de entrenamiento.

4.3.2. Validación en clasificación

Tras entrenar los modelos pasamos a obtener los resultados finales infiriendo con el modelo para los conjuntos de validación y test.

Antes de aplicar votación

En primer lugar calculamos la bondad sin aplicar el esquema de votación, la esperanza que se tiene es que tras la aplicación el esquema de votación los resultados sean iguales o mejores. No lo aplicaremos al conjunto test ya que la tarea real es clasificación binaria con el esquema por lo que no es relevante el resultado de test en estas condiciones.

La **matriz de confusión** es una herramienta en el aprendizaje automático y la estadística, utilizada para evaluar el rendimiento de modelos de clasificación. Para un problema de clasificación con N clases, la matriz de confusión tiene N filas y N columnas. Los elementos de la matriz se pueden denotar como M_{ij} , donde M_{ij} es el número de veces que una instancia de la clase i ha sido predicha como clase j . La diagonal principal (M_{ii}) contiene el número de instancias correctamente clasificadas para cada clase. Fuera de la diagonal principal (M_{ij} para $i \neq j$) se encuentran las instancias que fueron incorrectamente clasificadas, mostrando la confusión entre las clases i y j .

A continuación, vemos los resultados obtenidos de ajuste para las métricas en entrenamiento y validación sin votación. Construimos las siguientes matrices de confusión tras la inferencia de cada conjunto.

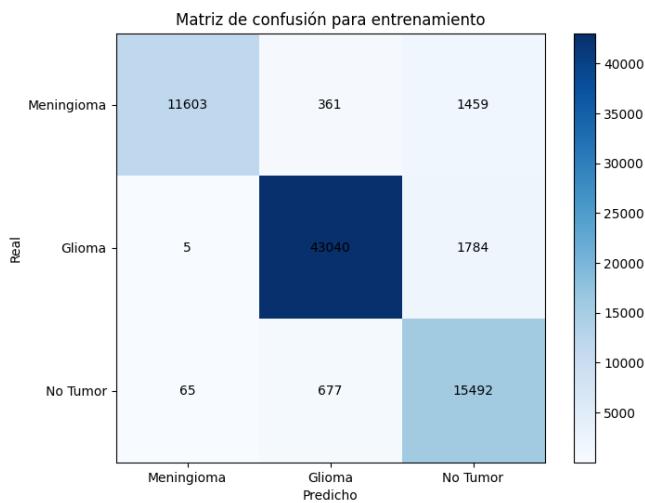


Figura 4.9: Matriz de confusión de entrenamiento sin votación

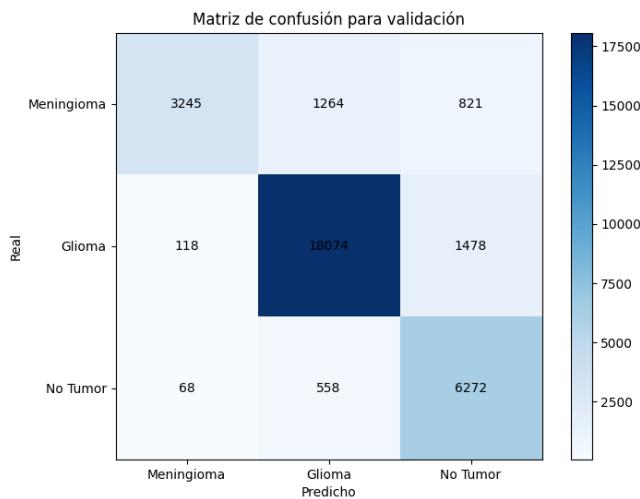


Figura 4.10: Matriz de confusión de validación sin votación

Conjunto	accuracy (%)	balanced_accuracy (%)
Entrenamiento	94.1586	92.6266
Validación	86.4976	81.2309

Cuadro 4.7: Resultados de validación y entrenamiento sin votación

En la Figura 4.9, observamos un ajuste alto en los datos de entrenamiento indicando un buen proceso de entrenamiento y en la Figura 4.10 resultados competitivos pero mejorables en validación. Observamos como la clase más conflictiva es **No Tumor** con las otras dos clases de tumores, indicando que la tarea más difícil es la distinción de que sea o no tumor y no la caracterización de este.

Tras aplicar votación

Aplicamos el esquema de votación, en primer lugar necesitamos explorar el parámetro *threshold* que definimos en la metodología. El parámetro *threshold* mide la tolerancia que tiene el modelo para predecir a meningioma, cuando existe un mayor número de predicciones por imagen a meningioma el modelo predice toda la resonancia a meningioma, y en caso contrario a glioblastoma.

Tras hacer inferencia y ver qué cantidad promedio de predicciones de cada clase obteníamos para toda la resonancia de unos cuantos ejemplos, observamos como las predicciones a meningiomas eran muchos menores que para los gliomas. Esto está motivado por la propia naturaleza de la cantidad

de imágenes de tumores de cada clase debido a que los meningiomas son más pequeños que los gliomas en promedio.

Como definimos en la metodología necesitamos un esquema que lidie con este desbalanceo. Por ello, descartamos valores altos de *threshold* y probamos dos valores pequeños: para *threshold* = 5 y *threshold* = 3.

A continuación mostramos las matrices de confusión asociadas a la aplicación de estos dos esquemas de votación.

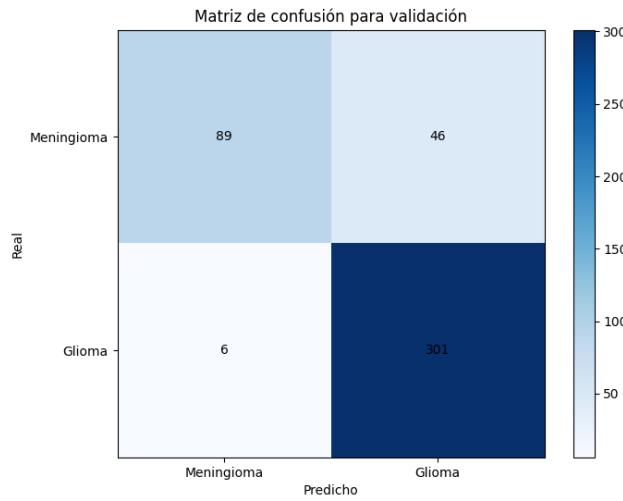


Figura 4.11: Matriz de confusión de validación con votación: *Meningiomas < 5*

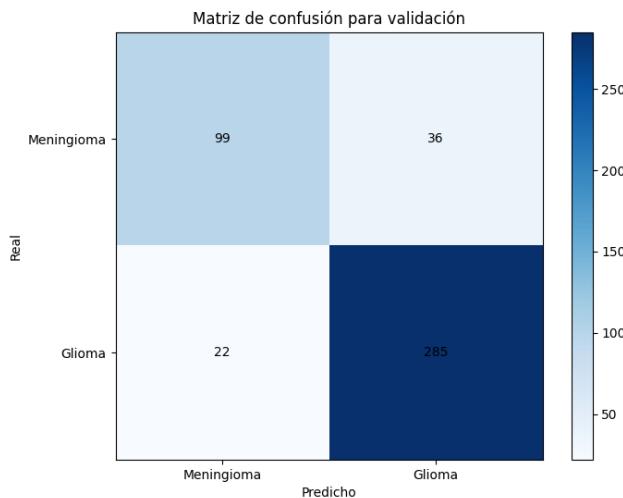


Figura 4.12: Matriz de confusión de validación con votación: *Meningiomas < 3*

Threshold	accuracy (%)	balanced_accuracy (%)
5	88.2352	81.9857
3	86.8779	83.0363

Cuadro 4.8: Resultados de validación con votación y de la búsqueda

Observamos como obtenemos un mejor accuracy balanceado con *threshold* = 3, así que optamos por su elección. Tras haber configurado *threshold* = 3 y haber visto su bondad en validación solo queda obtener el resultado final con test.

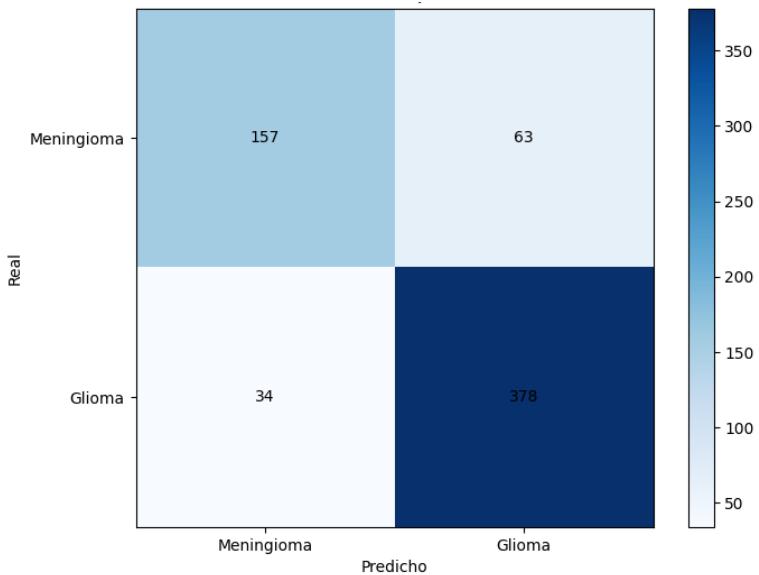


Figura 4.13: Matriz de confusión de Test con votación

Obtenemos resultados finales para test, obteniendo un accuracy del 84.6519 % y un accuracy balanceado del 81.5556 %. Con este resultado sabemos que la obtención de un modelo final debería tener un accuracy balanceado en clasificación $Acc_F \geq 81.5556\%$.

4.3.3. Comparativa de clasificación con el estado del arte

Para finalizar la experimentación de esta primera tarea ponemos en contexto estos resultados con el estado del arte. Aunque no sean del todo comparables debido a que no son formulaciones del problema idénticas (se tiene otro tipo de tumor en el estado del arte) y se utilizan conjuntos de datos distintos T1-CE image en el estado del arte frente a **BraTS** en el este tra-

bajo. Sin embargo, es importante hacer una observación de la bondad de los resultados teniendo en cuenta estas diferencias.

Autor	accuracy (%)	balanced_accuracy (%)	P_{test}
[Cheng et al., 2015]	91.2	-	61
[Cheng et al., 2016]	94.7	-	61
[Abiwinanda et al., 2019]	84.1	-	70
[Pashaei et al., 2018]	81.0	-	70
[Sultan et al., 2019]	96.1	-	75
[Díaz-Pernas et al., 2021]	97.3	-	46
Nuestro método	84.65	81.56	632

Cuadro 4.9: Comparativa resultados de test con la literatura

Los resultados del estado del arte no incluyen una validación con un esquema balanceado lo cual lo hace soluciones mucho menos generales y reales. Añadimos a la tabla 4.9, la entrada P_{test} que es la cantidad de pacientes que se usaron en el conjunto de test de cada trabajo.

Nuestro trabajo está validado en un conjunto de test ≈ 8 veces mayor que el mayor conjunto utilizado en la literatura y utiliza una política de aprendizaje balanceado lo cual puede hacer obtener peores resultados en un métrica más débil como puede ser accuracy sin balanceo. Aún así nuestro método supera a 2 de los 6 métodos presentados en la revisión histórica.

4.4. Segmentación

En esta sección recogemos los experimentos y resultados de estos para la tarea de segmentación.

4.4.1. Entrenamiento en segmentación

Entrenamos la arquitectura completamente descongelada con tamaño de lote de 32 ejemplos durante 7 épocas. El entrenamiento tuvo una duración de ≈ 8 horas. A continuación se presenta en la tabla 4.10 los resultados y en la figura 4.14 las curvas de aprendizaje obtenidas.

epoch	train_loss	valid_loss	Dice Loss	time
0	0.095941	0.319656	0.319656	1:34:08
1	0.073242	0.258797	0.258797	1:32:56
2	0.059904	0.284256	0.284256	1:34:48
3	0.050076	0.274452	0.274452	1:37:00
4	0.046783	0.266702	0.266702	1:40:18

Cuadro 4.10: Pérdida de entrenamiento y validación para segmentación toda la red descongelada

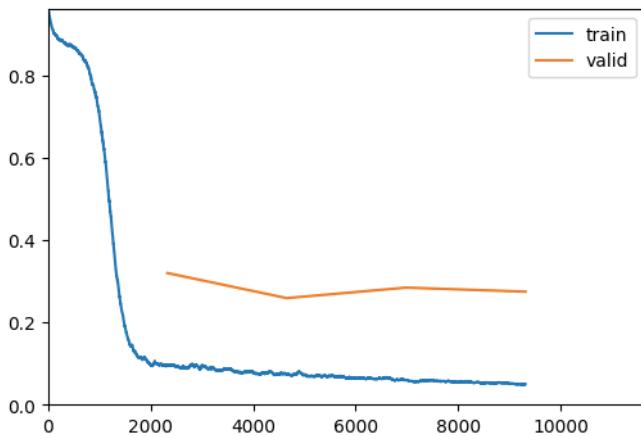
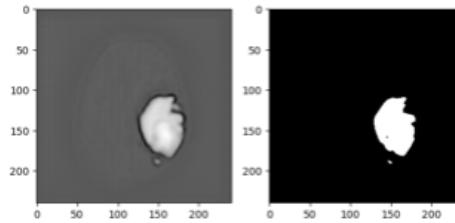


Figura 4.14: Curva de aprendizaje para la tarea de segmentación

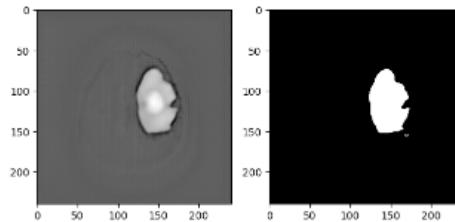
En la Figura 4.14, observamos como tenemos una muy rápida convergencia en la primera época y cómo se estabiliza en las sucesivas épocas refinándose la pérdida Dice muy ligeramente. En este tiempo de cómputo limitado observamos como no se produce overfitting. Sin embargo, obtenemos un error de validación que disminuye muy poco entre la primera y segunda época aumentando ligeramente en las sucesivas épocas. Podemos interpretar que existe un comportamiento estable en todo el proceso de entrenamiento.

A continuación, comparamos la salida del modelo con la imagen real. Esta salida es previa a aplicarle un proceso de umbralización con la cual obtendríamos una máscara binaria final. El proceso de umbralización que se aplicará consiste en determinar que píxeles formarán parte del tumor (usando la etiqueta 1) y en caso contrario, qué píxeles no (usando la etiqueta 0). El umbral U empleado será $U = 0.8$. Viendo en la práctica como este umbral o uno ligeramente más bajo como 0.5 no afecta significativamente a los resultados.

Nº 13743 Glioma



Nº 24744 Glioma



Nº 8092 Meningioma

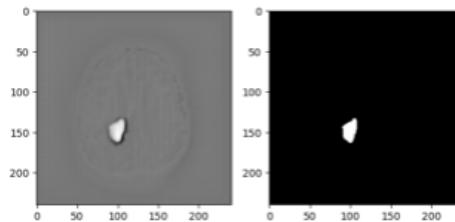


Figura 4.15: Comparación de la salida del modelo respecto la real

En la figura 4.15, observamos buenos resultados consiguiendo máscaras de segmentación que replican en detalle la segmentación original. En la imagen producida por la red se puede apreciar la sombra de un cerebro pudiendo ver un efecto residual del codificador y del uso de skip connections.

4.4.2. Validación en segmentación

A continuación, inferimos los resultados finales para segmentación en validación y test para nuestras tres métricas. En la tabla 4.11 se muestra los

resultados obtenidos.

Partición	Similaridad Dice	Distancia Hausdorff	Sensibilidad
Validación	0.777343	14.573561	0.720415
Test	0.772642	14.577154	0.709075

Cuadro 4.11: Resultados de hold-out en validación y test para segmentación

En la Figura 4.14, observamos que ambos conjuntos mantienen resultados muy similares empeorando ligeramente en test indicando robustez y coherencia en los resultados. Observamos que la peor métrica obtenida es la distancia Hausdorff que es demasiado alta, indicando que existe una distancia máxima media de ≈ 14 píxeles entre la segmentación original y la salida del modelo. Podría significar un problema de creación de artefactos por parte del modelo.

A continuación, para darle explicabilidad a los resultados se muestra el histograma de los resultados de las métricas para todos los ejemplos, primero del conjunto de validación y después para el conjunto de test. De esta forma, podríamos identificar ejemplos especialmente difíciles y caracterizarlos.

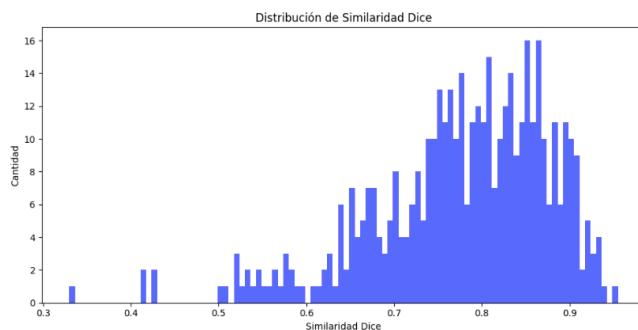


Figura 4.16: Distribución de similaridad Dice en el conjunto de validación

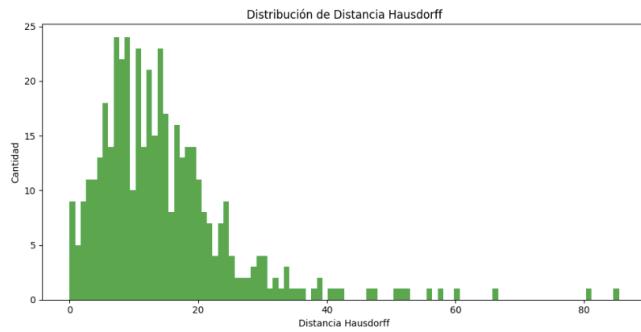


Figura 4.17: Distribución de distancia Hausdorff en el conjunto de validación

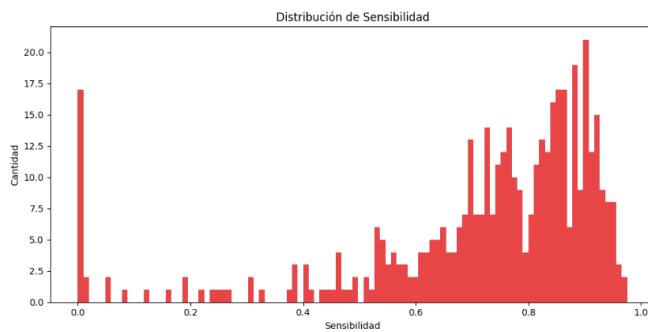


Figura 4.18: Distribución de sensibilidad en el conjunto de validación

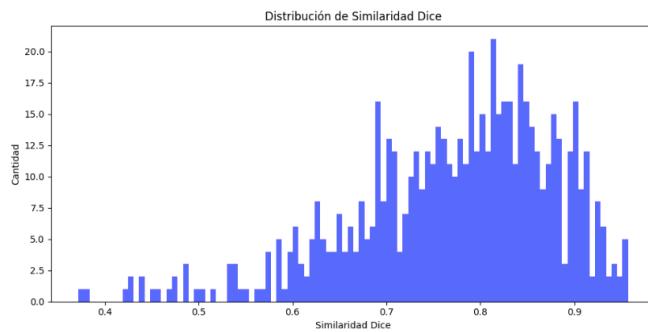


Figura 4.19: Distribución de similaridad Dice en el conjunto de test

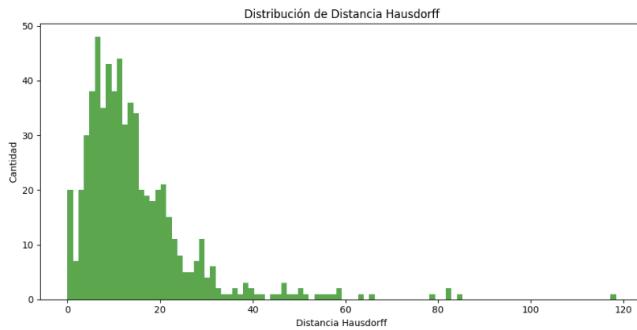


Figura 4.20: Distribución de distancia Hausdorff en el conjunto de test

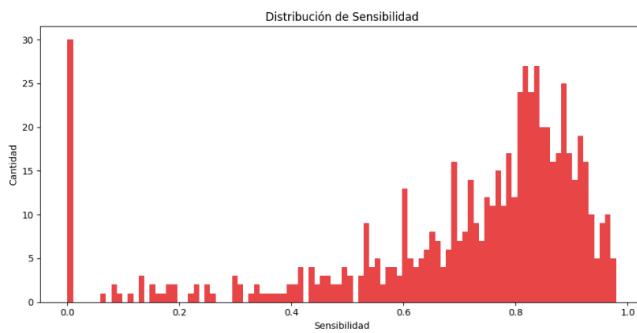


Figura 4.21: Distribución de sensibilidad en el conjunto de test

Los histogramas de estas métricas revelan una distribución en forma de campana de Gauss, lo cual es indicativo de una distribución normal. Este patrón indica robustez dentro del modelo de segmentación, observando que no existe un subconjunto de ejemplos que no puede segmentar bien. Nos sirve como prueba para demostrar la generalidad del modelo conseguida a pesar de poder obtener un modelo mejor.

La normalidad de las distribuciones también implica que las métricas son simétricas y centradas alrededor de un valor promedio consistente, lo que es indicativo de un rendimiento estable.

4.4.3. Comparativa de segmentación con el estado del arte

Para finalizar la experimentación de esta segunda tarea ponemos en contexto estos resultados con el estado del arte. Todos estos resultados han sido evaluados por conjuntos del mismo dataset **BraTS**, y respetando la misma proporción entre entrenamiento-test que la seguida en este trabajo.

Sin embargo, el resultado obtenido por nuestro método incluye la validación en un conjunto que incluye meningiomas y glioblastomas más diversos

(como los pediátricos) a diferencia de la literatura, que solo incluye glioblastomas de adultos. Nuestros resultados están validados en un conjunto de datos más diversos.

En la siguiente tabla se muestran las métricas y un apartado de recursos que indica la GPU que usaron para el entrenamiento. En nuestro método usamos una gráfica Tesla P100 limitados semanalmente 30 horas y en cada ejecución a 12 horas. Por lo que, no podemos entrenar más de 12 horas seguidas.

Autor	Dice (%)	Hausdorff	Recall (%)	Recursos
[Chen et al., 2021]	78.73	5.0	-	8 x Tesla V100
[Zhou et al., 2021]	87.1	6.5	86.8	Quadro P5000
[Myronenko, 2019]	90.4	4.4	-	8 x Tesla V100
[Hatamizadeh et al., 2021]	92.6	5.8	-	8 x Tesla V100
[Ferreira et al., 2024]	92.9	4.26	-	6 x RTX 6000
Nuestro método	77.3	14.5	70.9	Tesla P100

Cuadro 4.12: Comparativa de segmentación con el estado del arte

Como vemos nuestro método es peor que todo el estado del arte reciente que sea comparable. Sin embargo, es una buena aproximación para las capacidades de recursos en términos de capacidad (memoria del dispositivo GPU) y rendimiento de GPU que teníamos frente las del resto de autores que son inmensamente mayores.

Capítulo 5

Conclusiones y Trabajos Futuros

En este capítulo abordamos las conclusiones generales y específicas del trabajo. Adicionalmente, se expone qué mejoraríamos de este o cómo lo ampliaríamos en un trabajo futuro.

5.1. Conclusiones del trabajo

A continuación, comenzamos por describir las conclusiones del trabajo. Para ello, haremos tres apartados que ha sido clave para entender este trabajo: los resultados obtenidos en contexto con los recursos disponibles, el preprocesado aplicado (es decir, qué conclusión podemos extraer de la reducción de imágenes realizada para hacer el trabajo viable) y cómo la reconstrucción previa ha sido clave acelerando los tiempos de entrenamiento.

5.1.1. Resultados y recursos

En este trabajo desde el primer momento hemos estado en desventaja ante otros autores por la falta de recursos y capacidades, ya que el único hardware personal que nos podría permitir entrenar era un PC personal.

Sin embargo, un PC personal estaba descartado como posible hardware y la alternativa era usar el entorno de Kaggle que tiene sus limitaciones.

Los resultados son moderadamente peores que la parte más reciente del estado del arte. No obstante, interpretamos que esto se debe a la falta de un entrenamiento más potente.

5.1.2. Nuevo preprocesado en el problema

Para poder entrenar con los datos se tuvo que hacer un nuevo preprocesado que consistía en eliminar partes que podrían parecer redundantes. Se redujo las imágenes de entrenamiento a aquellas imágenes con solo contenían lesión tumoral más una parte balanceada sin lesión. Esta reducción supone una nueva forma de preprocesar este conjunto de datos, y ante los resultados podemos considerar que se han mantenido estables, validando en la práctica este preprocesado.

Podemos concluir que la eliminación de la prueba **T2W** tampoco ha influido significativamente en los resultados. Pudiendo significar la existencia de información redundante en el conjunto de datos.

Por otro lado, tras los resultados obtenidos no hemos apreciado efectos negativos tras haber hecho una reducción de dimensionalidad. La mayoría de los trabajos recientes usan información 3D, usando toda la resonancia como entrada a la red. Descartamos esta opción por recursos, pero en la práctica no vemos una diferencia sustancial que haga que la información espacial entre imágenes de una resonancia permita segmentar mejor.

5.1.3. El poder de la reconstrucción previa

Finalmente, uno de los aspectos clave para hacer posible este trabajo es el ahorro en tiempo de ejecución que hemos obtenido mediante el entrenamiento de una arquitectura encoder-decoder para obtener una reconstrucción de las imágenes. El aprendizaje derivado de esta arquitectura se ha usado en las tareas de clasificación y segmentación.

El preentrenamiento con el autoencoder ha permitido que el codificador y representación latente capten características relevantes y patrones inherentes a los datos antes de que se entrene para la tarea de clasificación o segmentación. Esto es especialmente útil cuando se dispone de una cantidad limitada de datos etiquetados como es en el caso de datos médicos. Esta técnica consiste en aprovechar al máximo las imágenes de entrada.

En los experimentos hemos visto una gran convergencia en entrenamiento motivado por este paso previo clave en el proceso.

5.2. Trabajos futuros

A continuación, detallamos los aspectos en los que este trabajo podría ampliarse o mejorar. En primer lugar, hacemos un listado de aspectos específicos de mejora.

1. **Mejora del hardware.** En un futuro necesitaríamos un hardware propio y suficiente con el podamos volver a entrenar el modelo más tiempo.
2. **Mejorar la exploración de modelos e hiperparámetros.** Consecuencia de la mejora anterior permitiría una investigación más guiada experimentalmente.
3. **Seguir investigando formas de abordar la tercera tarea.** La falta de datos y recursos impidieron poder resolver la predicción de la evolución del tumor. Una línea interesante es seguir investigando formas de aumento o creación de datos para llevarla a cabo.

5.2.1. Uso de transformers

En el estado del arte hemos visto que el uso de transformer es una tendencia al alza que puede mejorar los resultados rompiendo la localidad de las convoluciones creando codificadores más potentes. Sería interesante probar su bondad en un futuro.

5.2.2. Unificación de arquitecturas

En todo el trabajo hemos usado un codificador y representación latente común a las dos tareas y en la teoría también común a la tercera. Un trabajo a futuro sería investigar la bondad de los tres modelos para diferentes tareas optimizando esta parte común junto a los decoder o capas densamente conectadas de cada una en un mismo proceso de entrenamiento. Parte de la literatura ha indicado que esto bien ejecutado podría tener un efecto regularizador y reducir el ruido irreducible que veíamos en la optimización.

Por otro lado, esta puede ser una fuerte motivación a usar una arquitectura transformadora ya que en trabajos recientes han sido utilizadas como modelo multi-modal.

5.2.3. Exploración de otras técnicas de aprendizaje no supervisado

En la literatura se usan otras técnicas de aprendizaje no supervisado como las redes generativas adversarias, un trabajo futuro podría ser la investigación de la mejora de los modelos de este proyecto mediante ese tipo de aumento de datos.

Bibliografía

- [Abiwinanda et al., 2019] Abiwinanda, N., Hanif, M., Hesaputra, S. T., Handayani, A., and Mengko, T. R. (2019). Brain tumor classification using convolutional neural network. In *World Congress on Medical Physics and Biomedical Engineering 2018: June 3-8, 2018, Prague, Czech Republic (Vol. 1)*, pages 183–189. Springer.
- [Baid et al., 2021] Baid, U., Ghodasara, S., Mohan, S., Bilello, M., Calabrese, E., Colak, E., Farahani, K., Kalpathy-Cramer, J., Kitamura, F. C., Pati, S., et al. (2021). The rsna-asnr-miccai brats 2021 benchmark on brain tumor segmentation and radiogenomic classification. *arXiv preprint arXiv:2107.02314*.
- [Bakas et al., 2017] Bakas, S., Akbari, H., Sotiras, A., Bilello, M., Rozycski, M., Kirby, J. S., Freymann, J. B., Farahani, K., and Davatzikos, C. (2017). Advancing the cancer genome atlas glioma mri collections with expert segmentation labels and radiomic features. *Scientific data*, 4(1):1–13.
- [Brügger et al., 2019] Brügger, R., Baumgartner, C. F., and Konukoglu, E. (2019). A partially reversible u-net for memory-efficient volumetric image segmentation. In *Medical Image Computing and Computer Assisted Intervention–MICCAI 2019: 22nd International Conference, Shenzhen, China, October 13–17, 2019, Proceedings, Part III 22*, pages 429–437. Springer.
- [Bulten et al., 2022] Bulten, W., Kartasalo, K., Chen, P.-H. C., Ström, P., Pinckaers, H., Nagpal, K., Cai, Y., Steiner, D. F., van Boven, H., Vink, R., et al. (2022). Artificial intelligence for diagnosis and gleason grading of prostate cancer: the panda challenge. *Nature medicine*, 28(1):154–163.
- [cancer.org, 2024] cancer.org (2024). American cancer society, cancer statistics center. 17 de marzo de 2024.
- [Chang, 2016] Chang, P. D. (2016). Fully convolutional deep residual neural networks for brain tumor segmentation. In *Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries: Second International*

- Workshop, BrainLes 2016, with the Challenges on BRATS, ISLES and mTOP 2016, Held in Conjunction with MICCAI 2016, Athens, Greece, October 17, 2016, Revised Selected Papers 2*, pages 108–118. Springer.
- [Chen et al., 2021] Chen, J., Lu, Y., Yu, Q., Luo, X., Adeli, E., Wang, Y., Lu, L., Yuille, A. L., and Zhou, Y. (2021). Transunet: Transformers make strong encoders for medical image segmentation. *arXiv preprint arXiv:2102.04306*.
- [Cheng et al., 2015] Cheng, J., Huang, W., Cao, S., Yang, R., Yang, W., Yun, Z., Wang, Z., and Feng, Q. (2015). Enhanced performance of brain tumor classification via tumor region augmentation and partition. *PloS one*, 10(10):e0140381.
- [Cheng et al., 2016] Cheng, J., Yang, W., Huang, M., Huang, W., Jiang, J., Zhou, Y., Yang, R., Zhao, J., Feng, Y., Feng, Q., et al. (2016). Retrieval of brain tumors by adaptive spatial pooling and fisher vector representation. *PloS one*, 11(6):e0157112.
- [Cheng et al., 2022] Cheng, Q., Dong, Y., et al. (2022). Da vinci robot-assisted video image processing under artificial intelligence vision processing technology. *Computational and Mathematical Methods in Medicine*, 2022.
- [Díaz-Pernas et al., 2021] Díaz-Pernas, F. J., Martínez-Zarzuela, M., Antón-Rodríguez, M., and González-Ortega, D. (2021). A deep learning approach for brain tumor classification and segmentation using a multiscale convolutional neural network. In *Healthcare*, volume 9, page 153. MDPI.
- [Dominic LaBella, 2023] Dominic LaBella, e. a. (2023). The asnr-miccai brain tumor segmentation (brats) challenge 2023: Intracranial meningioma.
- [Ferreira et al., 2024] Ferreira, A., Solak, N., Li, J., Dammann, P., Klessiek, J., Alves, V., and Egger, J. (2024). How we won brats 2023 adult glioma challenge? just faking it! enhanced synthetic data augmentation and model ensemble for brain tumour segmentation. *arXiv preprint arXiv:2402.17317*.
- [Hatamizadeh et al., 2021] Hatamizadeh, A., Nath, V., Tang, Y., Yang, D., Roth, H. R., and Xu, D. (2021). Swin unetr: Swin transformers for semantic segmentation of brain tumors in mri images. In *International MICCAI Brainlesion Workshop*, pages 272–284. Springer.
- [Havaei et al., 2017] Havaei, M., Davy, A., Warde-Farley, D., Biard, A., Courville, A., Bengio, Y., Pal, C., Jodoin, P.-M., and Larochelle, H.

- (2017). Brain tumor segmentation with deep neural networks. *Medical image analysis*, 35:18–31.
- [He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.
- [Ioffe and Szegedy, 2015] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr.
- [Isensee et al., 2018] Isensee, F., Kickingereder, P., Wick, W., Bendszus, M., and Maier-Hein, K. H. (2018). Brain tumor segmentation and radiomics survival prediction: Contribution to the brats 2017 challenge. In *Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries: Third International Workshop, BrainLes 2017, Held in Conjunction with MICCAI 2017, Quebec City, QC, Canada, September 14, 2017, Revised Selected Papers 3*, pages 287–297. Springer.
- [Jiang et al., 2020] Jiang, Z., Ding, C., Liu, M., and Tao, D. (2020). Two-stage cascaded u-net: 1st place solution to brats challenge 2019 segmentation task. In *Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries: 5th International Workshop, BrainLes 2019, Held in Conjunction with MICCAI 2019, Shenzhen, China, October 17, 2019, Revised Selected Papers, Part I 5*, pages 231–241. Springer.
- [Kamnitsas et al., 2018] Kamnitsas, K., Bai, W., Ferrante, E., McDonagh, S., Sinclair, M., Pawlowski, N., Rajchl, M., Lee, M., Kainz, B., Rueckert, D., et al. (2018). Ensembles of multiple models and architectures for robust brain tumour segmentation. In *Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries: Third International Workshop, BrainLes 2017, Held in Conjunction with MICCAI 2017, Quebec City, QC, Canada, September 14, 2017, Revised Selected Papers 3*, pages 450–462. Springer.
- [Kamnitsas et al., 2017] Kamnitsas, K., Ledig, C., Newcombe, V. F., Simpson, J. P., Kane, A. D., Menon, D. K., Rueckert, D., and Glocker, B. (2017). Efficient multi-scale 3d cnn with fully connected crf for accurate brain lesion segmentation. *Medical image analysis*, 36:61–78.
- [Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.
- [Lin et al., 2017] Lin, T.-Y., Goyal, P., Girshick, R., He, K., and Dollár, P. (2017). Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988.

- [Liu et al., 2021] Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., and Guo, B. (2021). Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 10012–10022.
- [Liu et al., 2023] Liu, Z., Tong, L., Chen, L., Jiang, Z., Zhou, F., Zhang, Q., Zhang, X., Jin, Y., and Zhou, H. (2023). Deep learning based brain tumor segmentation: a survey. *Complex & intelligent systems*, 9(1):1001–1026.
- [Menze et al., 2014] Menze, B. H., Jakab, A., Bauer, S., Kalpathy-Cramer, J., Farahani, K., Kirby, J., Burren, Y., Porz, N., Slotboom, J., Wiest, R., et al. (2014). The multimodal brain tumor image segmentation benchmark (brats). *IEEE transactions on medical imaging*, 34(10):1993–2024.
- [Myronenko, 2019] Myronenko, A. (2019). 3d mri brain tumor segmentation using autoencoder regularization. In *Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries: 4th International Workshop, BrainLes 2018, Held in Conjunction with MICCAI 2018, Granada, Spain, September 16, 2018, Revised Selected Papers, Part II 4*, pages 311–320. Springer.
- [Pashaei et al., 2018] Pashaei, A., Sajedi, H., and Jazayeri, N. (2018). Brain tumor classification via convolutional neural network and extreme learning machines. In *2018 8th International conference on computer and knowledge engineering (ICCKE)*, pages 314–319. IEEE.
- [Ronneberger et al., 2015] Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5–9, 2015, proceedings, part III 18*, pages 234–241. Springer.
- [Smith and Topin, 2019] Smith, L. N. and Topin, N. (2019). Superconvergence: Very fast training of neural networks using large learning rates. In *Artificial intelligence and machine learning for multi-domain operations applications*, volume 11006, pages 369–386. SPIE.
- [Srinivasan et al., 2021] Srinivasan, K., Garg, L., Datta, D., Alaboudi, A. A., Jhanjhi, N., Agarwal, R., and Thomas, A. G. (2021). Performance comparison of deep cnn models for detecting driver’s distraction. *CMC-Computers, Materials & Continua*, 68(3):4109–4124.
- [Subedi et al., 2022] Subedi, B., Sathishkumar, V., Maheshwari, V., Kumar, M. S., Jayagopal, P., and Allayear, S. M. (2022). Feature learning-based generative adversarial network data augmentation for class-based few-shot learning. *Mathematical Problems in Engineering*, 2022(1):9710667.

- [Sultan et al., 2019] Sultan, H. H., Salem, N. M., and Al-Atabany, W. (2019). Multi-classification of brain tumor images using deep neural network. *IEEE access*, 7:69215–69225.
- [Wang et al., 2018] Wang, G., Li, W., Ourselin, S., and Vercauteren, T. (2018). Automatic brain tumor segmentation using cascaded anisotropic convolutional neural networks. In *Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries: Third International Workshop, BrainLes 2017, Held in Conjunction with MICCAI 2017, Quebec City, QC, Canada, September 14, 2017, Revised Selected Papers 3*, pages 178–190. Springer.
- [Wenxuan et al., 2021] Wenxuan, W., Chen, C., Meng, D., Hong, Y., Sen, Z., and Jiangyun, L. (2021). Transbts: Multimodal brain tumor segmentation using transformer. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, Springer, pages 109–119.
- [Zeiler and Fergus, 2014] Zeiler, M. D. and Fergus, R. (2014). Visualizing and understanding convolutional networks. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6–12, 2014, Proceedings, Part I 13*, pages 818–833. Springer.
- [Zhou et al., 2021] Zhou, T., Canu, S., Vera, P., and Ruan, S. (2021). Latent correlation representation learning for brain tumor segmentation with missing mri modalities. *IEEE Transactions on Image Processing*, 30:4263–4274.
- [Zhu and Yan, 1997] Zhu, Y. and Yan, Z. (1997). Computerized tumor boundary detection using a hopfield neural network. *IEEE transactions on medical imaging*, 16(1):55–67.

Apéndice A

Códigos y repositorio

En este apéndice incluimos las URL a los repositorios utilizados para el control de versiones del proyecto en GitHub y en Kaggle. Adicionalmente, hemos añadido un vídeo disponible en YouTube donde se prueba a la interfaz.

A.1. Repositorio de GitHub

En primer lugar, entramos a indicar cómo la estructura del repositorio de GitHub. En la raíz del repositorio tenemos tres carpetas:

- **inference**. Definida como la carpeta dedicada a la inferencia con los modelos. De ella cuelga la estructura de carpetas de toda la interfaz.
- **latex**. Carpeta dedicada a la memoria en latex.
- **train**. Carpeta dedicada a todo el proceso de entrenamiento y experimentación. En esta carpeta se encuentran los distintos notebooks que componen la construcción de los modelos y experimentos realizados.

Puede consultar este repositorio de GitHub en la siguiente dirección <https://github.com/jucls/TFGBraTS> o haciendo click en el siguiente hipervínculo: Ir al código.

A.2. Cuadernos en Kaggle

Adicionalmente, aunque los cuadernos de los experimentos están incluidos en el repositorio de GitHub también los haremos públicos en Kaggle. En la siguiente tabla incluimos los hipervínculos a cada uno.

Los links tienen la estructura común <https://www.kaggle.com/code/jaimecastillo>. Para buscar, a partir de la dirección de Kaggle, un cuaderno concreto solo es necesario añadirle la palabra de la columna Notebook de la siguiente tabla.

Nombre	Notebook	Botón
Preprocesado	preprocesado-encoder-y-tarea-1	Ir al código
Reconstrucción ResNet34	building-encoder-with-resnet34	Ir al código
Reconstrucción Xception	building-encoder-with-xception	Ir al código
Clasificación	task-1-classification-with-resnet34	Ir al código
Validación clasificación	task-1-test-postprocessing	Ir al código
Segmentación	task-2-segmentation	Ir al código
Validación segmentación	task-2-test-postprocessing	Ir al código

Cuadro A.1: Enlaces a los notebooks en Kaggle

A.3. Modelos en Kaggle

Los modelos obtenidos en este trabajo son muy pesados. No están en el repositorio de GitHub ya que tiene una capacidad limitada que estos sobrepasan. No obstante se puede descargar los checkpoints de los modelos obtenidos en este trabajo en Kaggle.

En la siguiente tabla recogemos los hipervínculos asociados a la descarga de cada checkpoint. La URL a buscar tiene esta estructura común a todos los modelos: <https://www.kaggle.com/datasets/jaimecastillo>. Para buscar un modelo solo es necesario añadirle la palabra de la columna Checkpoint de la siguiente tabla o hacer clic en el enlace.

Notebook	Checkpoint	Link
Reconstructor	encoder2	Ir al modelo
Clasificador	task1model	Ir al modelo
Segmentador	segmentation	Ir al modelo

Cuadro A.2: Enlaces a los checkpoints en Kaggle

A.4. Demostración de uso de la interfaz

A modo de demostración y tutorial del uso de la interfaz se ha creado un vídeo corto público en YouTube. Este puede ser consultado en el siguiente enlace: <https://youtu.be/zO6oS9FLhyA?si=ewN1tjjCp1G1tjbs> ó haciendo click en el siguiente hipervínculo: Ir al vídeo.

Apéndice B

Documentación de la interfaz

La interfaz tiene una finalidad simple y no es el objetivo del trabajo. Sin embargo, en este apéndice detallamos el código de la interfaz: su estructura y las dependencias necesarias para poder ser utilizada.

B.1. Estructura de la interfaz

A continuación, comentamos la implementación y diseño de la interfaz. Mostraremos su diagrama de clases y comentaremos sus funciones.

B.1.1. Diagrama de clases

La interfaz tiene una estructura de ficheros modular, separando un fichero principal **main** de otros ficheros que se encargan de la inferencia. Los programas que se ocupan de utilizar los modelos para inferir son: **model definition** y **predictions**.

- **Model definition.** Este fichero reúne el código necesario para definir los modelos.
- **Predictions.** Utiliza la instancia del **Learner** de **Model definition** para hacer la predicción final.

En el diagrama de clases (ver Figura B.1) mostraremos todos los ficheros representados como metaclasses que contienen a su vez sus propias clases. Dentro de la definición de la metaclass o fichero **model definition**, representaremos con un cuadro instancia todas las clases y métodos necesarios para las arquitectura y modelos que se definieron en la Figura 4.1.

A continuación, mostraremos el diagrama de clases de toda la interfaz.

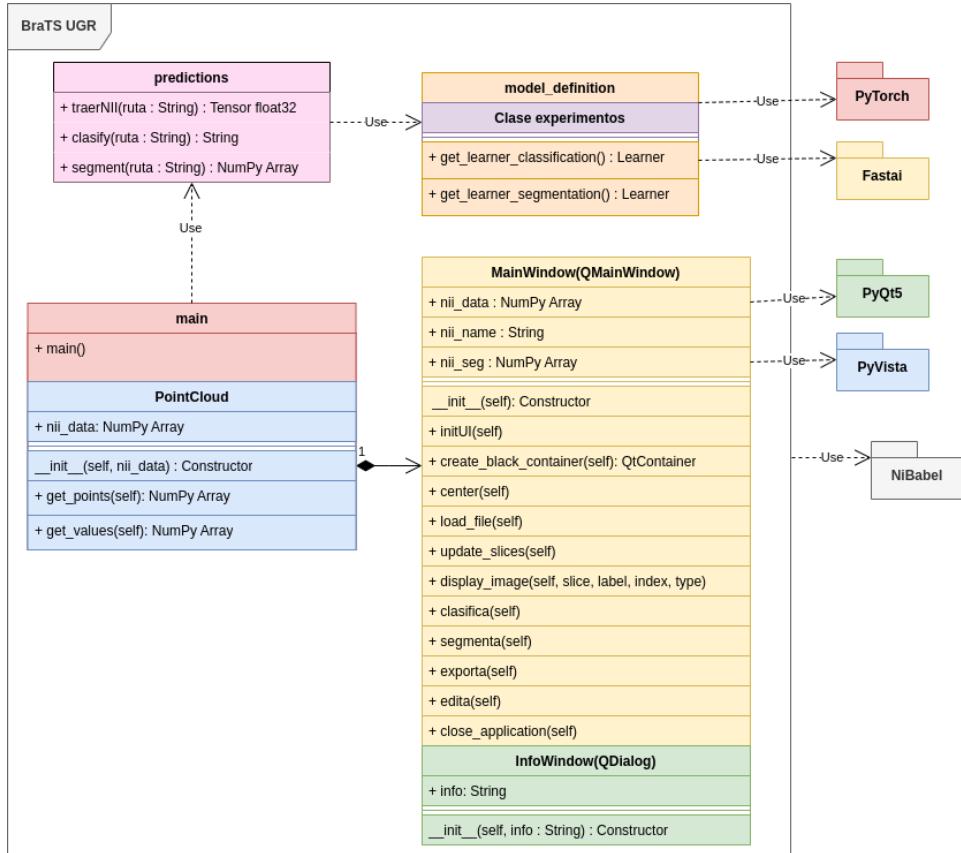


Figura B.1: Diagrama de clases de toda la interfaz

En la figura B.1, observamos el diagrama de clases de la interfaz. Las relaciones que se muestran en el diagrama tienen su origen en la metaclass principal (`main`). Así la metaclass principal instancia la ventana de la interfaz. Para inferir usa a la metaclass **predictions**. La metaclass **predictions** a su vez se relaciona con **model definition** ya que usa las funciones que devuelven sus objetos **Learner**. Además se observa como la metaclass de la definición del modelo utiliza a **PyTorch** y a **FastAI**. Y como la clase de la ventana principal usa **PyQt5** y **PyVista**. Finalmente, se observa como todas las clases usan a la librería **NiBabel**.

B.1.2. Documentación de funciones

En el siguiente apartado, creamos una lista con las funciones implicadas en cada clase. Comentaremos su función y lo que devuelve.

- **predictions**

- **traerNII(ruta : String) : Tensor float32**

Esta función recibe una cadena que representa la ruta de un archivo NII y devuelve un tensor de tipo float32.

- **clasify(ruta : String) : String**

Esta función toma como parámetro una cadena con la ruta de un archivo y devuelve una cadena con la clasificación resultante.

- **segment(ruta : String) : NumPy Array**

Esta función recibe una cadena que representa la ruta de un archivo y devuelve un arreglo de tipo NumPy con la segmentación resultante.

- **main**

- **main()**

Función principal que inicia la ejecución del programa.

- **PointCloud**

- **PointCloud(nii_data : NumPy Array)**

Constructor de la clase **PointCloud** que recibe como parámetro un arreglo de tipo NumPy.

- **get_points() : NumPy Array**

Esta función devuelve un arreglo de tipo NumPy con los puntos de la nube de puntos.

- **get_values() : NumPy Array**

Esta función devuelve un arreglo de tipo NumPy con los valores correspondientes a la nube de puntos.

- **model_definition**

- **get_learner_classification() : Learner**

Esta función devuelve un objeto de tipo **Learner** para la clasificación.

- **get_learner_segmentation() : Learner**

Esta función devuelve un objeto de tipo **Learner** para la segmentación.

- **MainWindow(QMainWindow)**

- **MainWindow()**

Constructor de la clase **MainWindow**.

- **initUI()**

Inicializa la interfaz de usuario.

- **create_black_container(self) : QtContainer**

Crea un contenedor de color negro.

- **center()**
Centra la ventana principal.
- **load_file()**
Carga un archivo.
- **update_slices()**
Actualiza las imágenes de las rebanadas.
- **display_image(self, slice, label, index, type)**
Muestra una imagen específica.
- **clasifica()**
Clasifica los datos cargados.
- **segmenta()**
Segmenta los datos cargados.
- **exporta()**
Exporta los datos.
- **edita()**
Edita los datos.
- **close_application()**
Cierra la aplicación.

- **InfoWindow(QDialog)**

- **InfoWindow(info : String)**
Constructor de la clase `InfoWindow` que recibe una cadena con información.

B.2. Dependencias

La interfaz gráfica desarrollada para este proyecto está construida utilizando una serie de bibliotecas:

- **PyTorch y FastAI:** Estas bibliotecas facilitan la inferencia de los modelos. Utilizamos a PyTorch para construir la definición del modelo, tras ello instanciamos el objeto Learner en FastAI al cual le cargamos los modelos.
- **PyVista:** Utilizada para la visualización de datos tridimensionales, PyVista ofrece una representación gráfica interactiva del cerebro como una nube de puntos, permitiendo una exploración detallada de las estructuras cerebrales. En la interfaz se utiliza para hacer la reconstrucción 3D a partir de las imágenes de la resonancia.

- **NiBabel:** Esta biblioteca es esencial para la manipulación y el procesamiento de imágenes médicas en formato NIfTI. Se utiliza para la lectura del formato **.nii** y el guardado también de las segmentaciones en este formato especializado.
- **PyQt5:** PyQt5 es la base sobre la cual se ha construido la interfaz gráfica de usuario (GUI). Dentro del desarrollo de interfaces gráficas es un estándar por su estética y facilidad de uso. Proporciona un marco robusto y flexible para el desarrollo de aplicaciones de escritorio interactivas en Python.

Apéndice C

Manual de uso de la interfaz

En este apéndice construiremos un breve manual del uso de la interfaz: para visualizar las resonancias, interactuar con ella, inferir con ella (clasificar o segmentar) y finalmente exportar una segmentación dada. Definiremos un flujo estándar del uso de la interfaz, aportando imágenes de su realización paso a paso.

C.1. Diagrama de flujo de la interfaz

A continuación, mostramos un diagrama de flujo correspondiente al uso estándar de la interfaz.

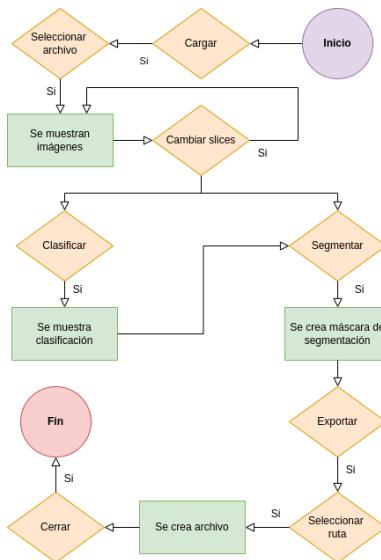


Figura C.1: Diagrama de flujo de la interfaz.

C.2. Tutorial sobre el flujo de la interfaz

A continuación, veremos un tutorial del uso de la interfaz. Detallamos paso por paso cómo utilizarla.

Al ejecutar la interfaz nos aparece la ventana principal que se muestra en la Figura C.2. Observamos tres bloques negros y uno blanco, los botones del uso de la interfaz y los selectores laterales para poder movernos en entre las imágenes.

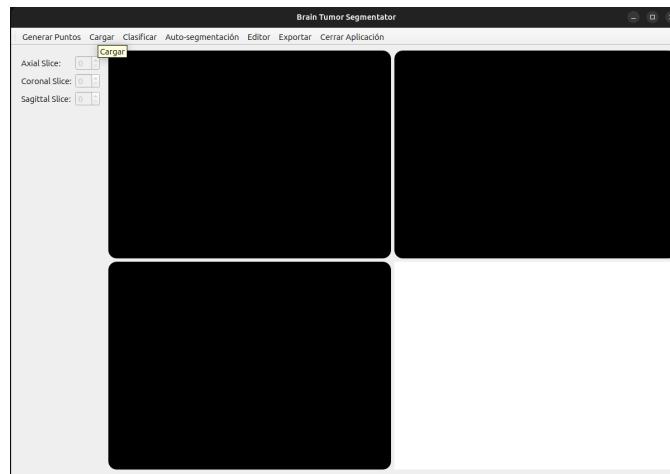


Figura C.2: Inicio de la interfaz.

Tras ello, debemos cargar alguna resonancia en nuestra carpeta de datos.

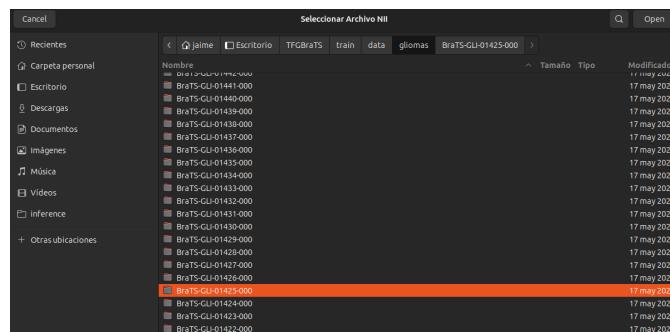


Figura C.3: Seleccionar archivo a cargar en la interfaz.

Por defecto, debemos escoger alguna prueba a visualizar (sólo se puede visualizar una prueba). Aunque selecciones sólo una prueba aquí cuando intentes inferir algo esto será indiferente ya que automáticamente seleccionará lo que necesite.

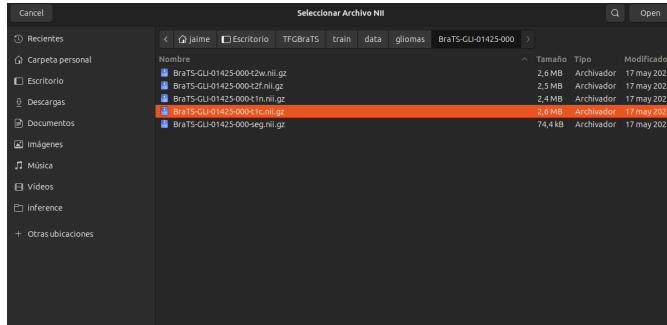


Figura C.4: Seleccionar prueba a cargar en la interfaz.

En la Figura C.5. observamos como al abrir la prueba que queremos se nos cargan las imágenes y una reconstrucción 3D del cerebro como una nube de puntos. Podemos hacer clic con el ratón o movernos por la reconstrucción como deseemos. Además observamos como ya los botones selectores de la derecha se habilitan, permitiendo poder recorrer el vector de imágenes de la resonancia para sus diferentes vistas.

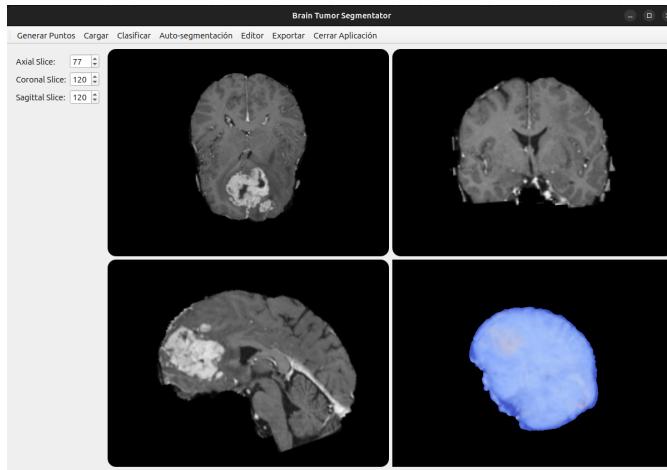


Figura C.5: Visualizado de imágenes y 3D en la interfaz.

Para segmentar la imagen solo debemos darle al botón **Auto-Segmentación** tras ello la interfaz entiende que quieras inferir la máscara de segmentación de esa resonancia. Debemos esperar unos segundos, ya que es un proceso costoso.

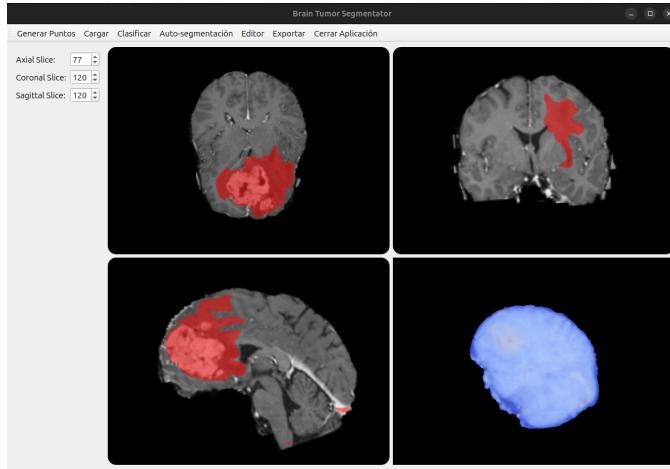


Figura C.6: Segmentar con la interfaz y visualizado de la máscara resultado.

En la Figura C.6, observamos como obtenemos la máscara de segmentación superponiéndose automáticamente a la imagen original. Esta máscara incluye la segmentación de todos los tejidos de la lesión tumoral.

Tras ello, podemos clasificar entre los dos tipos de tumores con la interfaz. Pulsamos el botón **Clasificar** y otra vez automáticamente entiende que quieras inferir la resonancia cargada.

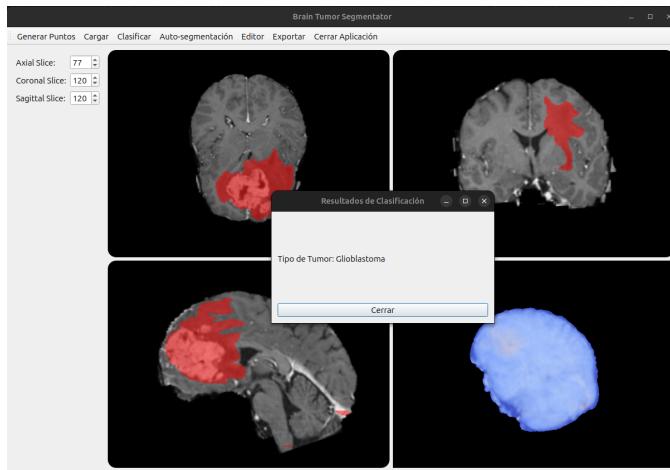


Figura C.7: Clasificar con la interfaz y resultados de clasificación.

En la figura C.7, observamos como tras pocos segundos aparece una pestaña emergente con los resultados del tipo de tumor.

Tras inferir ambos resultados, podemos exportar la máscara generada pulsando al botón **Exportar**.

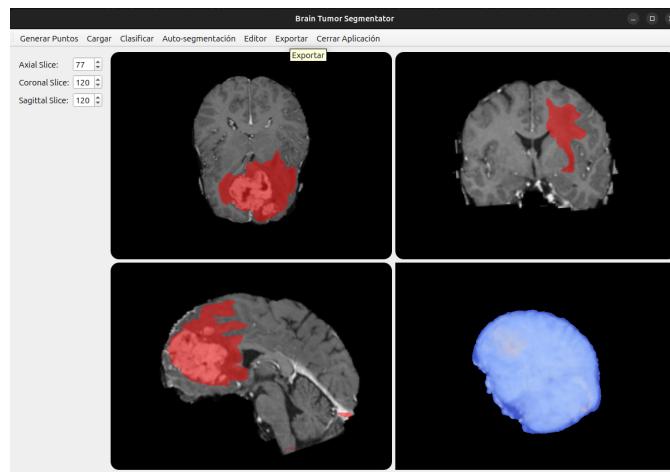


Figura C.8: Exportar segmentación con la interfaz.

Necesitamos guardarlo en alguna ruta que deseemos. Nos pregunta dónde guardararlo y con qué nombre.

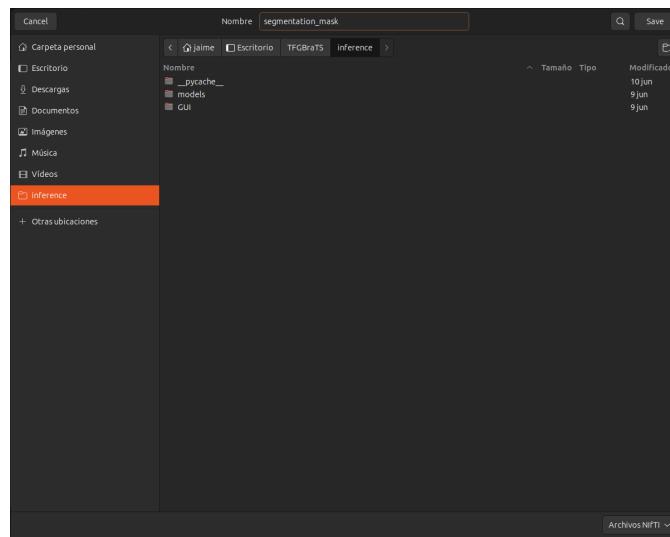


Figura C.9: Seleccionar ruta a exportar en la interfaz.

Si todo es correcto, nos dirá que fue correctamente guardado en una pestaña emergente. En caso contrario, también nos proporcionará el mensaje de error.

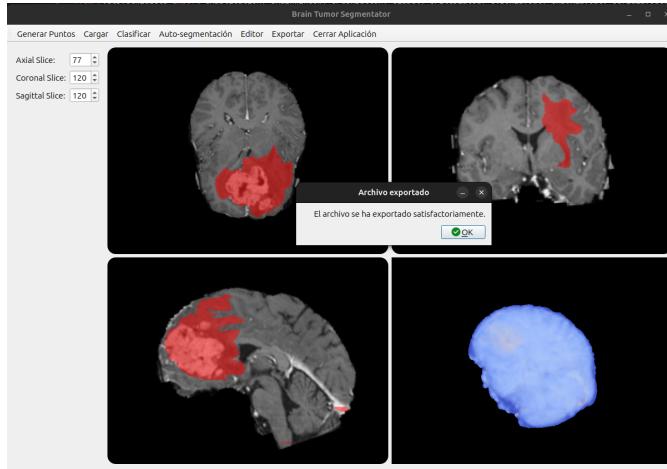


Figura C.10: Mensaje de exportación realizada correctamente.

Tras ello, podemos seguir infiriendo otras resonancias o cerrar la aplicación. Si cargamos otras resonancias el proceso es el mismo, la interfaz sobre escribirá los datos antiguos con los nuevos. Para cerrar correctamente la interfaz liberando explícitamente por el programa los recursos tan sólo pulsamos en **Cerrar Aplicación**.

