



ugr | Universidad
de **Granada**

TRABAJO FIN DE MÁSTER
MÁSTER UNIVERSITARIO EN DESARROLLO DE SOFTWARE

Técnicas de segmentación de datasets médicos orientada al desarrollo de herramientas de ayuda al diagnóstico clínico

Autor
Francisco de Asís Guerrero Aranda

Director
Alejandro J. León Salas



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE
TELECOMUNICACIÓN

—
Granada, Febrero de 2023



ugr | Universidad
de **Granada**

Técnicas de segmentación de datasets médicos orientada al desarrollo de herramientas de ayuda al diagnóstico clínico

Autor

Francisco de Asís Guerrero Aranda

Director

Alejandro J. León Salas

Técnicas de segmentación de datasets médicos orientada al desarrollo de herramientas de ayuda al diagnóstico clínico

Francisco de Asís Guerrero Aranda

Palabras clave: imagen médica, segmentación, dataset médicos, métricas, aprendizaje profundo, herramientas ayuda diagnóstico clínico, MONAI

Resumen

El diagnóstico clínico en la actualidad se apoya en un porcentaje muy alto en el análisis de imagen médica por parte de radiólogos y especialistas. La segmentación de información volumétrica es un proceso fundamental de cara a posibilitar el procesado posterior de la información en bruto obtenida para la comprensión clínica, el diagnóstico de enfermedades y la planificación quirúrgica. De esta forma, los especialistas pueden centrar su análisis en información relevante y de más alto nivel, mejorando los tiempos, la precisión, la calidad y eficiencia de la atención médica que reciben los pacientes. Este proyecto plantea como objetivo principal el estudio del estado del arte de las técnicas de segmentación de datasets médicos centrándose en las basadas en redes neuronales profundas de aprendizaje automático y su integración y aplicabilidad en el desarrollo de herramientas de visualización y análisis de más alto nivel que ayuden al especialista en el diagnóstico clínico.

Techniques for medical dataset segmentation aimed at developing clinical diagnostic aid tools

Guerrero Aranda, Francisco de Asís

Keywords: Medical imaging, segmentation, medical datasets, metrics, deep learning, clinical diagnosis aid tools, MONAI

Abstract

Nowadays, clinical diagnosis is highly supported by medical image analysis carried out by radiologists and specialists. Volumetric information segmentation is a fundamental process for enabling further processing of raw information obtained for clinical understanding, disease diagnosis, and surgical planning. In this way, specialists can focus their analysis on relevant and higher-level information, improving the time, precision, quality, and efficiency of the medical care received by patients. This project's main objective is to study the state of the art of medical dataset segmentation techniques, focusing on deep learning neural networks and their integration and applicability in the development of higher-level visualization and analysis tools to aid the specialist in clinical diagnosis.

Yo, **Francisco de Asís Guerrero Aranda**, alumno del **Máster en Desarrollo de Software** de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación** de la **Universidad de Granada**, con DNI 79.202.135W, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Máster en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Francisco de Asís Guerrero Aranda

Granada a 10 de Febrero de 2023 .

FIRMADO POR	FRANCISCO DE ASIS GUERRERO ARANDA	10/02/2023	PÁGINA 1/1
VERIFICACIÓN	Pk2jm9AU4M5ZUQQAETYY2BGC2GX59V	https://ws050.juntadeandalucia.es/verificarFirma	

Alejandro J. León Salas



Yo, **Francisco de Asís Guerrero Aranda**, alumno del **Máster en Desarrollo de Software** de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación** de la **Universidad de Granada**, con DNI 79.202.135-W, declaro explícitamente que el trabajo presentado es original, entendido en el sentido que no he utilizado ninguna fuente sin citarla debidamente.

Fdo: Francisco de Asís Guerrero Aranda

Granada a 10 de Febrero de 2023 .

FIRMADO POR	FRANCISCO DE ASIS GUERRERO ARANDA	10/02/2023	PÁGINA 1/1
VERIFICACIÓN	Pk2jm2UWEN7HBFW793Q4J5856R6285	https://ws050.juntadeandalucia.es/verificarFirma	

Agradecimientos

Quiero expresar mi profundo agradecimiento a todas las personas que han contribuido en la realización de este Trabajo de Fin de Master.

En primer lugar, quiero dar las gracias a mi tutor Alejandro cuya guía y apoyo han sido fundamental para la elaboración de este trabajo. Sin sus consejos, conocimiento y motivación, no hubiera sido posible llegar hasta aquí.

También dar las gracias especiales a mi mujer, por su comprensión y paciencia. Gracias por ser mi pilar y por estar ahí en los buenos y malos momentos.

Y por supuesto a mi hija Olivia, por ser mi luz y mi inspiración. Gracias por hacerme reír y por recordarme lo importante que es la vida.

Por último, también quiero agradecer a toda mi familia, especialmente a mi padre y mi madre, por su apoyo incondicional y haber puesto todos los medios que estaban a su alcance y muchos mas para que en el pasado pudiese hacer una carrera universitaria que me ha permitido tener una magnifica trayectoria profesional que ahora me permite de nuevo volver a estudiar y a apasionarme con la tecnología y la investigación.

Índice general

1. Introducción	1
1.1. Motivación	1
1.2. Objetivos	3
1.3. Estructura de Trabajo	4
2. Estado del arte	7
2.1. Modalidades de Imágenes Médicas	9
2.1.1. TC: Tomografía computarizada	9
2.1.2. SPECT: Tomografía computarizada por emisión de fotón único	11
2.1.3. PET: Tomografía por emisión de positrones	11
2.1.4. Resonancia Magnética	12
2.1.5. Otras modalidades	14
2.1.6. Multimodalidad	17
2.2. Métodos y Técnicas de Segmentación	18
2.2.1. Métodos de segmentación	18
2.2.2. Técnicas de segmentación	20
2.3. Segmentación basada en aprendizaje automático	23
2.3.1. Redes Neuronales Convolucionales (CNN)	23
2.3.2. Fully Convolutional Neural Networks (FCN)	25
2.3.3. U-Net	26
2.3.4. GAN	28
2.3.5. EfficientNet	29
2.3.6. SegResNet	29
2.3.7. TransUNet	31
2.3.8. DiNTS	32
2.3.9. Swin-UNETR	33
2.4. <i>Datasets</i> de Imágenes Médicas	36
2.4.1. Tipos de datasets	37
2.4.2. <i>Medical Segmentation Decathlon</i> (MSD)	38
2.4.3. <i>Brain tumor segmentation</i> (BRaTS)	39
2.4.4. <i>Beyond the Cranial Vault</i> (BTCV)	41
2.4.5. <i>Digital database for screening mammography</i> (DDSM)	41

2.4.6. <i>Ischemic stroke lesion segmentation</i> (ISLES)	42
2.4.7. <i>Digital retinal images for vessel extraction</i> (DRIVE) .	43
2.4.8. Otros	44
2.5. Preprocesamiento	44
2.6. Aumento de datos (<i>Data Augmentation</i>)	47
2.7. Métricas de evaluación	51
2.7.1. Cardinalidades básicas (Matriz de confusión)	52
2.7.2. Coeficiente de Similaridad de Dice o <i>F1-Score</i>	57
2.7.3. Índice de Jaccard o IoU	58
2.7.4. Distancia Hausdorff (HD)	60
2.7.5. Pautas de selección de métricas	63
2.8. Desafíos en la segmentación de imágenes médicas	64
3. Desarrollo del trabajo	67
3.1. MONAI	68
3.1.1. MONAI <i>Core</i>	69
3.1.2. MONAI <i>Label</i>	72
3.1.3. MONAI <i>Deploy App</i> SDK	74
3.1.4. MONAI Model Zoo	76
3.2. Entrenamiento desde cero: segmentación imágenes médicas .	77
3.2.1. Proceso	79
3.2.2. Resultados	84
3.3. Inferencias con modelos entrenados	87
3.4. Entrenamiento fino con Transfer Learning	90
3.5. Solución integral para segmentación a gran escala	92
3.5.1. <i>Autorunner</i>	94
3.5.2. Ejecutar con las API del módulo	98
3.6. Testing de herramientas de ayuda al diagnóstico clínico . .	103
3.6.1. Ejecución de MONAI Label Server con Google Colab	104
3.6.2. Ejecución en la Nube de AWS	104
4. Conclusiones y trabajo futuro	113
4.1. Reflexión crítica resultante de la experiencia del trabajo realizado	113
4.2. Cumplimiento de los objetivos del trabajo	114
4.2.1. Objetivo 1 - Revisión modalidades y técnicas de segmentación tradicionales	115
4.2.2. Objetivo 2 - Estudio de redes neuronales de aprendizaje profundo	115
4.2.3. Objetivo 3 - Recopilación de conjuntos de datos de imágenes médicas	116
4.2.4. Objetivo 4 - Investigación técnicas de preprocesamiento y aumento de datos	116
4.2.5. Objetivo 5 - Revisión de métricas de evaluación	117

4.2.6. Objetivo 6 - Búsqueda de comunidades colaboración clínica de imágenes médicas	117
4.2.7. Objetivo 7 - Desarrollo modelo de segmentación de imágenes	118
4.2.8. Objetivo 8 - Testear solución integral de segmentación	119
4.2.9. Objetivo 9 - Integrar y probar herramienta de ayuda al diagnóstico clínico. Casos de uso	119
4.3. Trabajo futuro	120
4.3.1. Uso de otras redes y arquitecturas de segmentación . .	120
4.3.2. Uso de otras herramientas de ayuda al diagnóstico clínico	121
4.3.3. Aplicación a VR/AR y gemelos digitales	121
4.3.4. Despliegues en entornos de producción reales	122
4.3.5. Centralización de Conjunto de Datos Médicos	122
4.3.6. Aprendizaje Federado	123
4.3.7. Patología Digital	123
4.3.8. Participación en <i>Challenges</i>	124
Bibliografía	131
Apéndice	131
A. Cuadernos de Jupyter Desarrollados	133
A.1. Cuaderno: Entrenamiento desde cero	133
A.2. Cuaderno: Inferencias con modelos entrenados	142
A.3. Cuaderno: Entrenamiento con Transfer Learning	147
A.4. Cuaderno: Solución integral Segmentación Autorunner	156
A.5. Cuaderno: Servidor de MONAI Label	162

Índice de figuras

2.1.	Imagen de Tomografía Computarizada. Fuente: MSD Task 09 Spleen. Imagen Spleen06	10
2.2.	Imagen de SPECT de perfusión cerebral. Fuente: Hospital Universitario Clínico San Cecilio	12
2.3.	Imagen de PET cerebral. Fuente: Wikipedia	13
2.4.	Imagen de Resonancia Magnética de secuencia T1. Fuente: Wikipedia	14
2.5.	Imágenes multimodales en la misma posición. (a) PET, (b) TC, (c) RM-T1, (d) RM-T2. Fuente: Guo <i>et al.</i>	17
2.6.	Convolución con kernel tamaño 3, padding 1 y stride 1.	24
2.7.	Max-pooling con kernel tamaño 2 y stride 1.	24
2.8.	Elementos de una Red Neuronal Convolucional (CNN). Fuente: Lundervold <i>et al.</i>	25
2.9.	Estructura de fully convolutional network (FCN). Fuente: Long <i>et al.</i> (modificada)	26
2.10.	Estructura de UNet. Fuente: Ronneberger <i>et al.</i>	27
2.11.	Estructura de una Red Generativa Adversarial (GAN). Fuente: web	28
2.12.	Comparación de tamaño del modelo vs. precisión de Efficient-Nets. Fuente: Tan <i>et al.</i>	30
2.13.	Visualización esquemática de la arquitectura de la red Se-gResNet. Fuente: Myronenko <i>et al.</i>	31
2.14.	(a) Esquema de la capa Transformer; (b) Arquitectura de TransUNet. Fuente: Chen <i>et al.</i>	32
2.15.	Espacio de búsqueda de DiNTS. Fuente: He <i>et al.</i>	33
2.16.	Modelo UNETR transformer encoder con conexión directa a decoder . Fuente: Hatamizadeh <i>et al.</i>	34
2.17.	Arquitectura de UNETR. Fuente: Hatamizadeh <i>et al.</i>	35
2.18.	Arquitectura Swin-UNETR. Fuente: Hatamizadeh <i>et al.</i>	36
2.19.	Resumen del mecanismo de ventana deslizante. Fuente: Hatamizadeh <i>et al.</i>	36
2.20.	Resumen de las 10 tareas del MSD. Fuente: Antonelli <i>et al.</i> .	39

2.21. Conjunto de datos BTCV con estructuras anatómicas etiquetadas. Fuente: Tang <i>et al.</i>	42
2.22. Matriz de confusión. Fuente: Reinke <i>et al.</i> (modificada)	54
2.23. Ejemplo esquemático de la matriz de confusión. Fuente: Reinke <i>et al.</i>	55
2.24. Sensibilidad. Fuente: Reinke <i>et al.</i> (modificada)	56
2.25. Especificidad. Fuente: Reinke <i>et al.</i> (modificada)	56
2.26. Precisión. Fuente: Reinke <i>et al.</i> (modificada)	57
2.27. DICE (DSC o F1-Score). Fuente: Reinke <i>et al.</i> (modificada)	58
2.28. DICE y Jaccard Index. Fuente: Reinke <i>et al.</i>	59
2.29. Problema tamaño pequeño estructura para DICE y JAC. Fuente: Reinke <i>et al.</i>	61
2.30. Distancia Hausdorff. Fuente: Reinke <i>et al.</i>	62
 3.1. Herramientas de MONAI para el ciclo de vida de la IA Médica.	69
3.2. Arquitectura de MONAI Core.	70
3.3. Importacion de librerías de MONAI Core.	71
3.4. Sistema de Active Learning de MONAI Label. Fuente: Díaz-Pinto <i>et al.</i>	72
3.5. Arquitectura de MONAI Label. Fuente: Díaz-Pinto <i>et al.</i>	73
3.6. Visor 3d Slicer integrado con MONAI Label. Fuente: Díaz-Pinto <i>et al.</i>	73
3.7. MONAI Label Apps de Radiología, Patología y Endoscopia. Visores OHIF, DSA, QuPath y CVAT.	74
3.8. MONAI Deploy SDK. Arquitectura de despliegue	75
3.9. MONAI Deploy SDK. Arquitectura de alto nivel	76
3.10. MONAI Model Zoo. Ejemplos de Modelos Preentrenados	77
3.11. Ruta de Google Drive donde se ha alojado el Dataset de BTCV	78
3.12. Imagen original, segmentacion Ground Truth y segmentacion obtenida con la inferencia	79
3.13. Imagen original de CT y etiqueta de especialista	81
3.14. Función de Pérdida y Métrica DICE para entrenamiento de 2.000 iteraciones	84
3.15. Imagen original de CT, etiqueta de especialista y resultado de inferencia con entrenamiento de 2.000 iteraciones	85
3.16. Función de Pérdida y Métrica DICE para entrenamiento de 10.000 iteraciones	85
3.17. Imagen original de CT, etiqueta de especialista y resultado de inferencia con entrenamiento de 10.000 iteraciones	86
3.18. Función de Pérdida y Métrica DICE para entrenamiento de 25.000 iteraciones	86
3.19. Imagen original de CT, etiqueta de especialista y resultado de inferencia con entrenamiento de 25.000 iteraciones	87

3.20. Imagen original de CT, etiqueta de especialista y comparativa entre las inferencias obtenidas para 2.000, 10.000 y 25.000 iteraciones	88
3.21. Resultado de inferencia usando el modelo anteriormente generado de 25.000 iteraciones	89
3.22. Resultado de inferencia usando el modelo Swin-UNETR	90
3.23. Función de Pérdida y Métrica DICE para entrenamiento de 2.000 iteraciones con red pre-entrenada	91
3.24. Imagen original de CT, etiqueta de especialista y comparativa red preentrenada y desde cero. 2000 iteraciones.	92
3.25. Solución integral para Segmentación basada en MONAI Auto3DSeg	93
3.26. Resultado de predicción para un entrenamiento de 2 épocas con Autorunner	96
3.27. Resultado de predicción para un entrenamiento de 200 épocas con Autorunner	97
3.28. Análisis de datos realizado por Autorunner	97
3.29. 4 Algoritmos generado por Autorunner y detalle del progreso	98
3.30. Módulos de la solución integral de auto3DSeg	98
3.31. Módulo de Analisis de Datos de la Solución Integral de Segmentación	99
3.32. Generación de Algoritmos del proceso de la Solución Integral de Segmentación	100
3.33. Ensamblado del proceso de la Solución Integral de Segmentación	102
3.34. Ensamblado del proceso de la Solución Integral de Segmentación	105
3.35. Imagen del escritorio remoto de la maquina EC2 en AWS con el 3D Slicer para ser instalado	106
3.36. Maquina en AWS - MONAI Label - Descarga de APP y <i>Dataset</i> para MONAI Label	107
3.37. Maquina en AWS - MONAI Label - Ejecución de Segmentación	107
3.38. Maquina en AWS - MONAI Label - Visualización 3D de la segmentación	108
3.39. MONAI Label - Caso de Uso probado - Pulmón y vías respiratorias	109
3.40. MONAI Label - Caso de Uso probado - Columna Vertebral y Vértebras	110
3.41. MONAI Label - Caso de Uso probado - Cuerpo Completo . .	111
3.42. MONAI Label - Caso de Uso probado - 24 órganos	111
3.43. MONAI Label - Caso de Uso probado - Cerebro Completo . .	112
3.44. MONAI Label - Paquetes de MONAI Model Zoo disponibles	112
4.1. Gemelo Digital y Aplicación de VR/AR	121

Índice de cuadros

2.1.	Conjunto de datos de <i>Medical Segmentation Decathlon</i>	40
2.2.	Resumen <i>datasets</i> de imágenes médicas y <i>urls</i>	44
2.3.	Resumen de métricas de evaluación	53

Capítulo 1

Introducción

La presente introducción tiene como objetivo dar una visión general del trabajo que se va a desarrollar. En primer lugar, se expondrá la motivación que ha llevado a llevar a cabo este trabajo, detallando los aspectos que han motivado a realizarlo. A continuación, se presentarán los objetivos perseguidos, tanto a nivel general como específicos, destacando el alcance y la relevancia del tema. Posteriormente, se describirá la estructura del trabajo, donde se describe la metodología seguida durante el proceso y se resume en qué consisten los principales capítulos de los que se compone esta memoria.

1.1. Motivación

Probablemente casi todos nos hemos hecho alguna vez en nuestra vida una radiografía, una resonancia magnética, una tomografía computarizada o una mamografía que han permitido a los profesionales médicos, sin necesidad de cirugías intrusivas, poder mirar nuestro cuerpo por dentro en detalle y establecer el diagnóstico clínico y en su caso el tratamiento.

La segmentación de imágenes médicas permite identificar y separar las estructuras en las imágenes proporcionando información precisa y rica en contenido sobre los órganos y tejidos del cuerpo humano así como de otras estructuras orgánicas relevantes.

La capacidad de identificar y separar objetos o estructuras relevantes dentro de una imagen médica permite a los profesionales de la salud obtener información valiosa sobre la condición de un paciente y llevar a cabo análisis cuantitativos y cualitativos de lesiones, enfermedades y estudios de áreas de interés.

Constituye la segmentación de imágenes médicas, por tanto, una tarea importante con un gran potencial para la comprensión clínica, el diagnóstico

de enfermedades, el seguimiento y tratamiento de éstas y la planificación quirúrgica. Juega además un rol crucial en el diagnóstico, ya que puede permitir revelar una enfermedad antes incluso que podamos llegar a sentir síntomas significativos los cuales detonarían un comienzo de tratamiento que pudiera llegar demasiado tarde.

La velocidad y la precisión de la segmentación son dos factores fundamentales para permitir un diagnóstico clínico más eficiente y preciso. La segmentación de las imágenes médica realizada manualmente por los expertos médicos puede tener una precisión muy grande, pero presenta otros inconvenientes como que el tiempo consumido para realizarla es muy alto, se necesita un alto grado de especialización del personal que la realiza o simplemente que a veces pueden ser indetectables a simple vista por el ojo humano.

Por lo tanto, nos enfrentamos al reto de incrementar la velocidad de diagnóstico, decrementar la carga de trabajo del especialista médico, detectar enfermedades en sus primeras etapas, mejorar la eficiencia y precisión del diagnóstico, reducir los errores humanos y, en última instancia, mejorar la atención médica que reciben los pacientes.

Los métodos tradicionales de segmentación de imágenes ya no se pueden comparar en prestaciones con la segmentación usando métodos basados en el aprendizaje profundo (*deep learning*), pero las ideas fundamentales aún siguen vigentes y merece la pena ser analizadas en detalle [1] [2] [3]. Estos métodos basados en el aprendizaje profundo han logrado un rendimiento superior en comparación con los métodos tradicionales en la tarea de segmentación de imágenes médicas [4] .

El aprendizaje profundo es un tipo de aprendizaje automático que imita el funcionamiento del cerebro humano en el procesamiento de datos y la creación de patrones para su uso en la toma de decisiones. Para la segmentación de imágenes médicas, una estructura llamada red neuronal artificial, se utiliza para entrenar y reconocer los datos para la predicción de enfermedades, así como para fines de diagnóstico y tratamiento.

Liu *et al.* [5] divide el proceso de segmentación de imágenes médicas en las siguientes etapas:

1. Obtención de los conjuntos de datos (*datasets*) de entrenamiento, de verificación y de tests.
2. Preprocesamiento y expansión de las imágenes (estandarización y *data augmentation*).
3. Uso de los métodos apropiados de segmentación.
4. Evaluación del rendimiento de la estimación para verificar la efectividad.

dad de la segmentación.

Por lo tanto, es importante llevar a cabo estudios en la segmentación de imágenes orientados al desarrollo de herramientas de ayuda al diagnóstico clínico, con el objetivo de mejorar la calidad y eficiencia de la atención médica. Estos estudios pueden ayudar a desarrollar nuevas técnicas de segmentación, evaluar la eficacia de las técnicas existentes, establecer protocolos para su uso clínico y desarrollar herramientas de ayuda al diagnóstico clínico, a la planificación de procedimientos quirúrgicos, a la cirugía guiada por imágenes, a planificar y administrar radioterapia de manera más precisa o a estudiar el desarrollo de enfermedades y el efecto de los tratamientos.

1.2. Objetivos

La segmentación de imágenes médicas es una tarea importante con un gran potencial para la comprensión clínica, el diagnóstico de enfermedades y la planificación quirúrgica. Según las estadísticas de las recientes conferencias de MICCAI, más del 60 % de los trabajos son aplicaciones de algoritmos de segmentación.

Este proyecto plantea como objetivo principal el estudio del estado del arte de las técnicas de segmentación de datasets médicos centrándose en las basadas en aprendizaje automático y su aplicabilidad en el desarrollo de herramientas de visualización y análisis de más alto nivel que ayuden al especialista en el diagnóstico clínico.

Para lograr este objetivo general se establecen los siguientes objetivos específicos:

1. Revisar la literatura existente sobre las distintas modalidades de imágenes médicas y los métodos y técnicas de segmentación tradicionales.
2. Estudiar las principales técnicas y redes de aprendizaje profundo que se usan para la segmentación de imágenes médicas para conocer las principales características y ventajas y desventajas de cada una de ellas.
3. Recopilación de los principales conjuntos de datos (datasets) de imágenes médicas que se utilizan comúnmente en el desarrollo y evaluación de técnicas de segmentación de imágenes médicas y la información más relevante de cada uno de ellos como tipo y cantidad de imágenes que dispone, modalidad con las que se han generado o donde está disponible para descarga.

4. Investigación acerca de las principales técnicas de preprocesamiento y aumento de datos para mejorar la calidad, la precisión, el número y la diversidad de las imágenes del *dataset* a través de la limpieza, normalización y la preparación de las mismas. Haciendo además un especial énfasis en las técnicas aplicables a las redes neuronales profundas.
5. Revisión de las métricas de evaluación para la segmentación y análisis de las más comúnmente usadas.
6. Búsqueda de comunidades y software de código abierto para la colaboración clínica en imágenes médicas.
7. Desarrollar y entrenar un modelo de segmentación de imágenes utilizando algunas o varias de las técnicas estudiadas. Evaluar la eficacia y precisión del modelo de segmentación desarrollado. Comprobar inferencias con nuevas imágenes.
8. Investigar y en su caso testear una solución integral de segmentación que utilice varios algoritmos de segmentación y construya una solución ensamblada del mejor modelo creado.
9. Integrar el modelo de segmentación en una herramienta de ayuda al diagnóstico clínico. Analizar posibles casos de uso de la herramienta usando algunos de los datasets recopilados en la fase de estudio.

1.3. Estructura de Trabajo

La estructura de trabajo ha tenido dos fases bien diferenciadas.

Una primera fase inicial que ha consistido en una revisión intensiva y exhaustiva de una gran cantidad de literatura relacionada con la segmentación de imágenes médicas en la que se ha estudiado el estado del arte de diversas materias, como los métodos y técnicas de segmentación para imágenes médicas, profundizando en aquellas basadas en aprendizaje automático y redes neuronales profundas. Así mismo se han identificado los conjuntos de datos para segmentación que se encuentran disponibles y se han estudiado técnicas para el preprocesamiento y aumento de estos conjuntos de datos finalizando esta fase con el análisis de las principales métricas de evaluación que se usan para la comparación de los algoritmos de segmentación de imágenes médicas y los desafíos a los que se enfrenta la segmentación en la actualidad.

La segunda fase ha sido más práctica donde hemos querido poner en práctica los conceptos aprendidos y adquiridos en la fase anterior para realizar de forma empírica un entrenamiento desde cero de un modelo de segmentación de imágenes médicas usando una o varias de las arquitecturas de

redes neuronales profundas y empleando conjunto de datos de los investigados en la fase anterior. En base al modelo de entrenamiento conseguido se ha experimentado acerca de hacer inferencias de resultados con nuevas imágenes y se ha desarrollado también un ejemplo de como reutilizar modelos de segmentación ya entrenados para que, mediante *transfer learning*, se realice un entrenamiento más fino para mejorar la precisión. También se ha querido en esta fase probar una solución integral que permite el análisis, entrenamiento y ensamblado de una solución usando distintos modelos de algoritmos de segmentación.

Por último y para cerrar el círculo, en esta segunda fase se han usado herramientas software para ayuda al diagnóstico integradas con estos modelos de segmentación y se han probado distintos casos de uso como una segmentación de las zonas del cerebro, segmentación completa del cuerpo, etc. que utilizan los modelos de segmentación basados en redes neuronales profundas que han sido entrenados anteriormente.

De todo ello se ha generado la presente memoria que consta de 4 capítulos principales y un apéndice donde se incluye el código fuente de los cuadernos *jupyter* que se han desarrollado para probar los distintos conceptos comentados anteriormente

La memoria por tanto consta de:

- Capítulo 1. Introducción que detalla la motivación y necesidad de este trabajo, los objetivos generales y específicos que se pretenden alcanzar y la metodología y estructura de la memoria.
- Capítulo 2. Estado del Arte que tiene como objetivo proporcionar una revisión exhaustiva de las técnicas de segmentación de datasets médicos existentes y su aplicabilidad en el desarrollo de herramientas de ayuda al diagnóstico clínico. Se abordan las diferentes modalidades de imágenes médicas utilizadas en la segmentación de datasets. A continuación, se describen las técnicas y métodos de segmentación más comunes, incluyendo aquellos basados en aprendizaje automático. Además, se presentan los diferentes datasets de imágenes médicas disponibles y se discuten las técnicas de preprocessamiento y *data augmentation* utilizadas en la segmentación de datasets. Por último, se analizan las métricas de evaluación utilizadas para medir el rendimiento de las técnicas de segmentación y se describen las limitaciones y desafíos actuales en este campo.
- Capítulo 3. Desarrollo del trabajo que describe el proceso de desarrollo y elaboración de diversos cuadernos de *jupyter* para entrenar distintas arquitecturas de segmentación de imágenes médicas basadas en redes neuronales profundas de aprendizaje automático usando distintos conjuntos de datos. Además se presenta una solución integral para

segmentación de imágenes médicas a gran escala donde se puede entrenar simultáneamente a varios modelos, optimizar los hiperparámetros y realizar un ensamblado de la mejor solución de segmentación de las obtenidas. Finalmente se evalúa una de herramientas de ayuda al diagnóstico clínico que integre las técnicas de segmentación y muestra la importancia y aplicabilidad real de estas herramientas en el mundo real a través de distintos casos de uso.

- Capítulo 4. Conclusiones y trabajo futuro donde se realiza una reflexión crítica de la experiencia obtenida durante el desarrollo del trabajo. Se evalúa el cumplimiento de los objetivos planteados y se identifican los aspectos que se han logrado y aquellos que requieren mejoras o ajustes. Además, se presentan las posibilidades y propuestas para el trabajo futuro, con el fin de continuar investigando y mejorando los resultados obtenidos en este trabajo.
- Apéndice A. Cuadernos de *Jupyter* Desarrollados donde se adjuntan el código completo de los cuadernos y las ejecuciones de los distintos cuadernos desarrollados para el trabajo, además se incluye el enlace al repositorio *Github* del autor de este trabajo donde se pueden descargar dichos cuadernos.

Capítulo 2

Estado del arte

La segmentación de imágenes médicas es una tecnología cada vez más utilizada en el campo de la medicina ya que proporciona información precisa y rica en contenido sobre los tejidos biológicos y tiene una gran variedad de aplicaciones en el contexto del análisis y el diagnóstico de enfermedades. La segmentación permite identificar y separar las estructuras en las imágenes de manera precisa, lo que facilita el diagnóstico y el tratamiento de enfermedades.

Algunas de las principales aplicaciones de la segmentación de imágenes médicas son:

- Ayuda al diagnóstico clínico: se utiliza para ayudar al diagnóstico de enfermedades, mejorar la precisión en la identificación y el análisis de estructuras anatómicas lo que permite realizar un diagnóstico preciso. Por ejemplo, la segmentación de órganos internos en imágenes de resonancia magnética puede ayudar a identificar tumores o anomalías en estos órganos. La segmentación de vasos sanguíneos en imágenes de tomografía computarizada puede ayudar a identificar obstrucciones o anomalías en estos vasos. La segmentación de áreas inflamadas en imágenes de radiografías puede ayudar a identificar enfermedades inflamatorias o infecciosas.
- Cirugía guiada por imágenes: se puede utilizar en la cirugía guiada por imágenes tanto en la planificación de los procedimientos quirúrgicos como en la navegación durante la cirugía. Por ejemplo, la segmentación de órganos internos en imágenes de resonancia magnética puede ayudar a guiar al cirujano durante una operación. La segmentación de vasos sanguíneos en imágenes de tomografía computarizada puede ayudar a identificar las estructuras vasculares que deben ser evitadas durante una operación. La segmentación de áreas inflamadas en imágenes de

radiografías puede ayudar a identificar la extensión de la inflamación y a guiar al cirujano en la elección de la mejor técnica quirúrgica.

- Radioterapia: puede ayudar a los médicos a planificar y administrar radioterapia de manera más precisa. Por ejemplo, la segmentación de órganos internos en imágenes de resonancia magnética puede ayudar a planificar el tratamiento de radioterapia de manera más precisa. La segmentación de vasos sanguíneos en imágenes de tomografía computarizada puede ayudar a evitar daños a estos vasos durante el tratamiento de radioterapia. La segmentación de áreas inflamadas en imágenes de radiografías puede ayudar a identificar las áreas que deben ser tratadas con radioterapia.
- Investigación biomédica: puede ayudar a los investigadores a estudiar el desarrollo de enfermedades y el efecto de tratamientos. Por ejemplo, la segmentación de órganos internos en imágenes de resonancia magnética puede ayudar a estudiar el desarrollo y la función de estos órganos en diferentes contextos. La segmentación de vasos sanguíneos en imágenes de tomografía computarizada puede ayudar a estudiar el flujo sanguíneo en diferentes condiciones. La segmentación de áreas inflamadas en imágenes de radiografías puede ayudar a estudiar los mecanismos de la inflamación en diferentes enfermedades.
- Realidad virtual y visualización: La segmentación de estructuras en imágenes médicas puede ayudar a crear modelos tridimensionales precisos y realistas de estructuras internas del cuerpo humano para la educación y la planificación de procedimientos quirúrgicos y terapias.

El presente capítulo tiene como objetivo proporcionar una revisión exhaustiva de las técnicas de segmentación de datasets médicos existentes y su aplicabilidad en el desarrollo de herramientas de ayuda al diagnóstico clínico. En primer lugar, se abordarán las diferentes modalidades de imágenes médicas utilizadas en la segmentación de datasets. A continuación, se describirán las técnicas y métodos de segmentación más comunes, incluyendo aquellos basados en aprendizaje automático. Además, se presentarán los diferentes datasets de imágenes médicas disponibles y se discutirán las técnicas de preprocesamiento y *data augmentation* utilizadas en la segmentación de datasets. Por último, se analizarán las métricas de evaluación utilizadas para medir el rendimiento de las técnicas de segmentación y se discutirán las limitaciones y desafíos actuales en este campo.

2.1. Modalidades de Imágenes Médicas

Las modalidades de imágenes médicas son diferentes tipos de tecnologías utilizadas para producir imágenes detalladas del cuerpo humano. Estas modalidades se utilizan comúnmente para diagnosticar problemas de salud y monitorizar el estado de salud de un paciente.

Las modalidades de imágenes médicas dependen en gran medida de los fundamentos físicos de los mecanismos de generación de imágenes, ya que cada modalidad utiliza una tecnología diferente que se basa en diferentes principios físicos.

Cada modalidad tiene sus propias ventajas y desventajas, y se utiliza en diferentes situaciones clínicas. Por ejemplo, algunas modalidades pueden producir imágenes más detalladas que otras, pero también pueden ser más costosas o requerir más tiempo para su obtención.

2.1.1. TC: Tomografía computarizada

La tomografía computarizada (TC) es una técnica de imagen médica que utiliza rayos X y un ordenador para producir imágenes en diferentes capas o cortes del cuerpo humano.

Los principios físicos de la tomografía computarizada se basan en la interacción de los rayos X con el cuerpo humano. Los rayos X son un tipo de radiación electromagnética que se utiliza en la medicina para producir imágenes del cuerpo. Los rayos X atraviesan el cuerpo humano y son absorbidos en diferentes cantidades por diferentes tejidos en función de su densidad y grosor.

La tomografía computarizada utiliza una máquina llamada tomógrafo, que consta de una fuente de rayos X, un detector y un ordenador. La fuente de rayos X emite rayos X que atraviesan el cuerpo del paciente y son detectados por el detector. El detector mide la cantidad de rayos X que atraviesan el cuerpo en diferentes puntos y envía esta información al ordenador. Toma múltiples imágenes del sujeto a distintos ángulos, moviendo mecánicamente la fuente de radiación y/o el receptor [6].

El ordenador utiliza esta información para producir una imagen detallada del cuerpo en diferentes capas o cortes. La imagen resultante muestra diferentes tejidos y órganos del cuerpo con diferentes tonos de gris, dependiendo de la cantidad de rayos X que han sido absorbidos por cada tejido.

Es una técnica muy útil en medicina ya que permite producir imágenes detalladas del cuerpo sin tener que recurrir a cirugías invasivas. También es una técnica segura, ya que los niveles de radiación utilizados en la TC son muy bajos y no presentan riesgos para la salud.

La TC es una técnica versátil que se puede utilizar en diferentes áreas de la medicina, como la neurología, la cardiología, la ortopedia y la oncología. Algunos de los diagnósticos clínicos en los que la TC es más adecuada incluyen el diagnóstico de tumores, el diagnóstico de enfermedades cardíacas y el diagnóstico de enfermedades neurológicas [7].

- Diagnóstico de tumores: la TC se puede utilizar para detectar tumores en diferentes órganos del cuerpo, como el cerebro, el corazón, los pulmones y el hígado.
- Diagnóstico de enfermedades cardíacas: la TC se puede utilizar para detectar enfermedades del corazón, como la arteriosclerosis, la cardiomiopatía y la enfermedad coronaria.
- Diagnóstico de enfermedades neurológicas: la TC se puede utilizar para detectar enfermedades del sistema nervioso, como el ictus, el trauma craneal y la esclerosis múltiple.
- Diagnóstico de enfermedades óseas: las imágenes de TC se pueden utilizar para detectar enfermedades óseas, como la osteoporosis y las fracturas.

La Figura 2.1 muestra una imagen de tomografía computarizada del bazo. Esta imagen proviene de uno de los conjuntos de datos médicos que analizaremos en este capítulo.

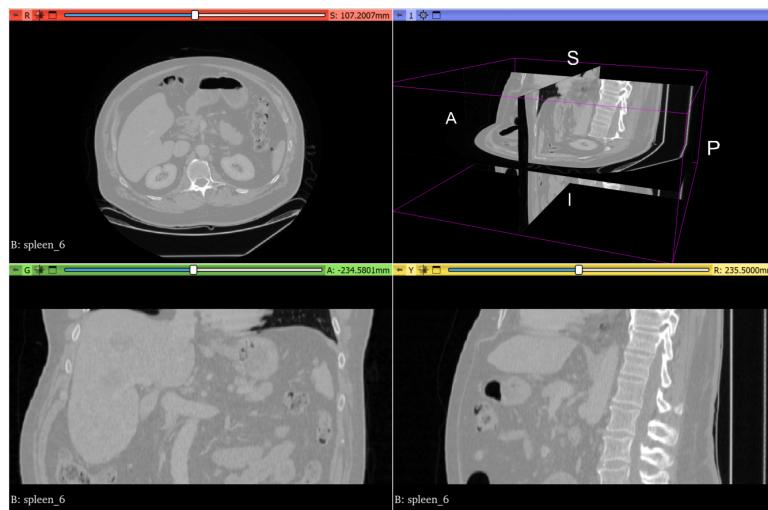


Figura 2.1: Imagen de Tomografía Computarizada. Fuente: MSD Task 09 Spleen. Imagen Spleen06

2.1.2. SPECT: Tomografía computarizada por emisión de fotón único

Se basa en la interacción de radiación gamma y un trácer radioactivo con el cuerpo humano, y se utiliza principalmente en el diagnóstico de enfermedades cardíacas y cerebro-vasculares.

La SPECT se diferencia de otras técnicas de TC en que utiliza un detector de fotón único en lugar de un detector de energía. Este detector de fotón único permite producir imágenes de alta calidad con una dosis de radiación más baja que en las técnicas de TC tradicionales.

Esta técnica muestra cómo fluye la sangre hacia tejidos y órganos. Se puede usar para ayudar a diagnosticar perfusiones miocárdicas, accidentes cerebro-vasculares, fracturas por estrés, infecciones, tumores en la espina y otros problemas cardíacos y trastornos óseos o cerebrales [8].

La tomografía computarizada de emisión de fotón único con perfusión (SPECT-P) utiliza un trácer radioactivo que permite visualizar el flujo sanguíneo en el cerebro. Esta técnica permite detectar áreas de baja perfusión (flujo sanguíneo insuficiente) en el cerebro, lo que es útil en el diagnóstico de enfermedades cerebro-vasculares, como el ictus [9]. La Figura 2.2 nos muestra un SPECT de perfusión cerebral de un paciente del Hospital Universitario San Cecilio de Granada.

La tomografía computarizada de emisión de fotón único con difusión (SPECT-D) utiliza un trácer radioactivo que permite visualizar el movimiento de agua en el cerebro. Esta técnica permite detectar áreas de alta difusión (movimiento de agua acelerado) en el cerebro. Se utiliza para producir imágenes detalladas de las fibras nerviosas en el cerebro.

La principal diferencia entre la SPECT-P y la SPECT-D es el trácer radioactivo utilizado en cada técnica, lo que permite detectar diferentes tipos de cambios en el cerebro.

La tomografía computarizada de doble emisión (DE-CT) utiliza un trácer radioactivo que emite dos tipos de radiación gamma, uno de baja energía y otro de alta energía. Se considera una técnica de imagen médica muy sensible, ya que permite detectar cambios en el metabolismo de los tejidos del corazón, lo que la hace especialmente útil en el diagnóstico de enfermedades cardíacas.

2.1.3. PET: Tomografía por emisión de positrones

Se basa en la interacción de radiación gamma y un trácer radioactivo con el cuerpo humano, y se utiliza principalmente en el diagnóstico de tumores y enfermedades neurodegenerativas, como el Alzheimer.

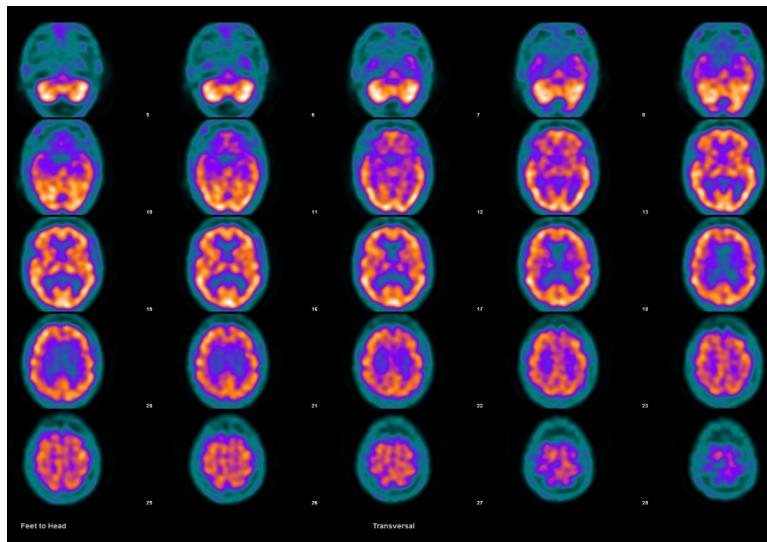


Figura 2.2: Imagen de SPECT de perfusión cerebral. Fuente: Hospital Universitario Clínico San Cecilio

Se diferencia de otras técnicas de tomografía computarizada es que utiliza un trácer radioactivo que emite positrones en lugar de rayos X. Estos positrones se detienen en el cuerpo y emiten radiación gamma, que es detectada por el equipo de PET para producir imágenes detalladas del cuerpo como la que se muestra en la Figura 2.3. La tomografía por emisión de positrones es en realidad una combinación de medicina nuclear y análisis bioquímicos.

La PET se considera una técnica de imagen médica muy sensible, ya que permite detectar cambios en el metabolismo de los tejidos del cuerpo, lo que la hace especialmente útil en el diagnóstico de tumores y enfermedades neurodegenerativas. También se utiliza en el seguimiento del tratamiento de estas enfermedades, ya que permite evaluar la eficacia del tratamiento y detectar posibles recidivas [10].

2.1.4. Resonancia Magnética

La resonancia magnética (RM) es una técnica de imagen médica que se utiliza para producir imágenes anatómicas detalladas de órganos y tejidos internos proporcionando un elevado detalle de las imágenes así como contraste entre los diferentes tejidos. La RM se basa en la interacción de campos magnéticos y ondas de radiofrecuencia con el cuerpo humano, y se utiliza principalmente en el diagnóstico de enfermedades neurológicas, cardiovasculares y musculo-esqueléticas.

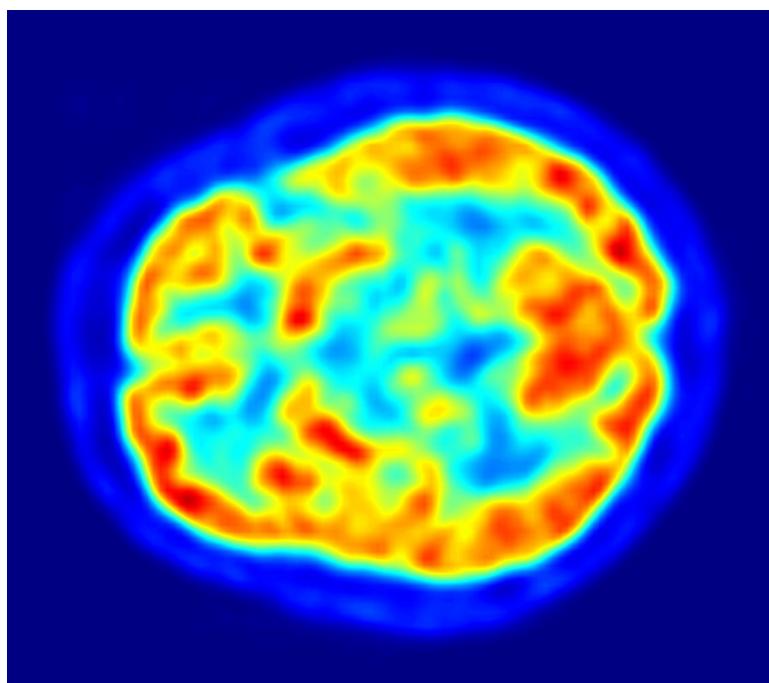


Figura 2.3: Imagen de PET cerebral. Fuente: Wikipedia

Está basada en una tecnología sofisticada que excita y detecta el cambio en la dirección del eje de rotación de los protones que se encuentra en el agua que forma los tejidos vivos [11].

La resonancia magnética se diferencia de la tomografía computarizada en el tipo de radiación, la TC utiliza radiación ionizante, mientras que la RM no utiliza radiación. La radiación ionizante puede ser perjudicial para el organismo, por lo que la RM es considerada una técnica más segura para el paciente. Además la RM es una técnica más sensible y específica que la TC, lo que significa que es capaz de detectar cambios en el metabolismo de los tejidos del cuerpo con mayor precisión. Esto hace que la RM sea especialmente útil en el diagnóstico de enfermedades tempranas. Otra diferencia entre ambas es que la RM permite obtener imágenes detalladas de diferentes órganos y tejidos del cuerpo, con una resolución superior a la de la TC, esto hace que la RM sea especialmente útil en el diagnóstico de enfermedades localizadas [12].

Existen diferentes tipos de resonancia magnética, como la resonancia magnética funcional, la resonancia magnética de espectroscopia, la resonancia magnética de dinamización y la resonancia magnética de difusión. La resonancia magnética funcional se utiliza para detectar cambios en el flujo sanguíneo y los patrones de actividad cerebral, mientras que la resonancia magnética de espectroscopia se utiliza para detectar moléculas específicas

en el cuerpo. La resonancia magnética de dinamización se usa para examinar el flujo sanguíneo en los vasos sanguíneos, mientras que la resonancia magnética de difusión se utiliza para detectar cambios estructurales en los tejidos.

Las secuencias de resonancia magnética son técnicas utilizadas para producir imágenes detalladas del interior del cuerpo humano. Cada secuencia utiliza un conjunto específico de parámetros de adquisición de datos para producir imágenes con diferentes características y propósitos. Las principales secuencias de resonancia magnética son la secuencia T1, la secuencia T2, la secuencia T2-FLAIR y la secuencia de difusión. Cada una tiene un propósito específico y se utiliza para visualizar diferentes estructuras y características del cuerpo. Así por ejemplo, la secuencia T1 produce imágenes en blanco y negro que muestran los detalles anatómicos de los tejidos como puede apreciarse en la Figura 2.4, mientras que la secuencia T2 muestra los detalles de los fluidos corporales; la secuencia T2-FLAIR produce imágenes con una mayor resolución para detectar lesiones en el sistema nervioso central y la secuencia de difusión se usa para detectar cambios estructurales en los tejidos.

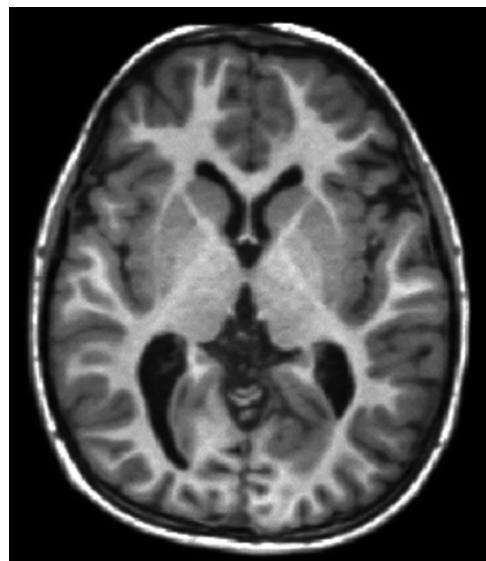


Figura 2.4: Imagen de Resonancia Magnética de secuencia T1. Fuente: Wikipedia

2.1.5. Otras modalidades

Además de las reseñadas anteriormente, existen otras modalidades que nos permiten la obtención de imágenes médicas con sus propias características y aplicaciones en el diagnóstico clínico.

Algunas de las modalidades de imágenes médicas más comunes, además de la tomografía computarizada y la resonancia magnética, son:

Radiografía

Se basa en el uso de radiación ionizante, que atraviesa el cuerpo y produce una imagen en una placa fotográfica o en un detector digital [13]. Se utiliza principalmente en el diagnóstico de enfermedades óseas y pulmonares, como fracturas, enfermedad de Paget y neumonía. También se utiliza en el diagnóstico de enfermedades gastrointestinales, como enfermedad de Crohn y obstrucción intestinal.

La radiografía y la tomografía computarizada tienen características similares, también existen diferencias importantes entre ellas como la resolución ya que la tomografía computarizada permite obtener imágenes detalladas de diferentes órganos y tejidos del cuerpo, con una resolución superior a la de la radiografía lo que hace que sea especialmente útil en el diagnóstico de enfermedades localizadas; la profundidad de campo y el campo de visión [14].

Ultrasonidos (Ecografía)

Se basa en el uso de ondas sonoras de alta frecuencia que no son audible para el oído humano. Cuando estas ondas sonoras entran en contacto con el cuerpo, rebotan en las estructuras internas y vuelven a la superficie donde son recogidas por un dispositivo llamado transductor. Estas ondas sonoras recogidas se procesan por una computadora que produce una imagen del área examinada. No utiliza radiación ionizante, por lo que es segura para el paciente y puede ser repetida tantas veces como sea necesario.

Los ultrasonidos se utilizan para examinar áreas como el abdomen, el corazón, la pelvis, los senos paranasales, las arterias y las venas. Esta técnica es muy útil para detectar anomalías en estas estructuras, como obstrucciones en los vasos sanguíneos, tumores o quistes. También se utilizan en el control de embarazos, ya que permite visualizar el feto en diferentes etapas del desarrollo y evaluar su salud y el diagnóstico de enfermedades ginecológicas, obstétricas y cardiovasculares, como fibromas uterinos, embarazo ectópico y enfermedad coronaria. También se utiliza en el diagnóstico de enfermedades hepáticas, renales y pancreáticas, como hepatitis, cálculos renales y pancreatitis.

Angiografía

Se utiliza para visualizar los vasos sanguíneos del cuerpo humano, ya sea en el cerebro, el corazón, los riñones o en cualquier otra parte del cuerpo. Esta técnica se realiza mediante la introducción de un contraste especial en los vasos sanguíneos, que permite que estos se visualicen en una radiografía o en una imagen de resonancia magnética.

El principal uso de la angiografía en el diagnóstico clínico es detectar obstrucciones o anormalidades en los vasos sanguíneos, como bloqueos o estrechamientos, que pueden ser causados por enfermedades como la aterosclerosis o el aneurisma. La angiografía también se utiliza para evaluar el flujo de sangre en diferentes partes del cuerpo y para planificar procedimientos médicos como angioplastias o cirugías cardíacas [15].

Mamografía

La mamografía es una técnica de imagen médica que se utiliza para examinar las mamas y detectar posibles anomalías o tumores. Esta técnica se realiza mediante la realización de una radiografía de las mamas, es decir, se utiliza un equipo de rayos X que emite radiación de baja intensidad para producir imágenes de las mamas, que permite visualizar cualquier anormalidad en su estructura.

La mamografía es una técnica de imagen muy sensible que puede detectar tumores en etapas muy tempranas, también se utiliza para evaluar la efectividad de tratamientos contra el cáncer de mama, como la quimioterapia o la radioterapia [16].

Tomografía de coherencia óptica (TCO)

Se basa en el uso de luz infrarroja de alta frecuencia que es enviada a través del tejido del cuerpo. Cuando la luz entra en contacto con el tejido, parte de ella es dispersada y reflejada hacia un detector, que convierte la información en una imagen.

El principal uso de la TCO en el diagnóstico clínico es visualizar estructuras internas del cuerpo con alta resolución y precisión. Esta técnica se utiliza para examinar áreas como el ojo, el corazón, el cerebro y los vasos sanguíneos. La TCO permite detectar anormalidades en estas estructuras, como obstrucciones en los vasos sanguíneos, tumores o enfermedades oculares como la degeneración macular. También se utiliza en el campo de la dermatología para evaluar lesiones cutáneas y determinar si son benignas o malignas. Esta técnica es muy útil para detectar anormalidades en etapas tempranas [17].

2.1.6. Multimodalidad

La multimodalidad en las imágenes médicas se refiere a la capacidad de utilizar varios tipos de tecnologías de imagen diferentes para obtener información sobre el cuerpo humano. Esto permite a los especialistas médicos obtener una imagen completa del área del cuerpo que están evaluando, lo que puede ser muy útil para el diagnóstico clínico.

Algunos ejemplos de modalidades de imagen médica que se pueden usar en conjunto para obtener una imagen multimodal incluyen la tomografía computarizada (CT), la resonancia magnética (RM), la ecografía y la radiografía. Cada una de estas modalidades proporciona información diferente sobre el cuerpo y, cuando se combinan, pueden ofrecer una visión mucho más completa del área en cuestión. La Figura 2.5 muestra esta visión más completa a través un conjunto de cuatro imágenes multimodales obtenidas mediante PET, TC, RM-T1 y RM-T2 del mismo individuo en la misma posición.

Además, la multimodalidad también se puede utilizar para comparar imágenes obtenidas en diferentes momentos, lo que permite a los médicos ver cualquier cambio que haya ocurrido en el área en cuestión y utilizar esta información para ayudar en el diagnóstico.

Como se deduce del artículo de Guo *et al.* [18], la aplicación de imágenes multimodales puede reducir la incertidumbre de la información y mejorar el diagnóstico clínico y la precisión de la segmentación.

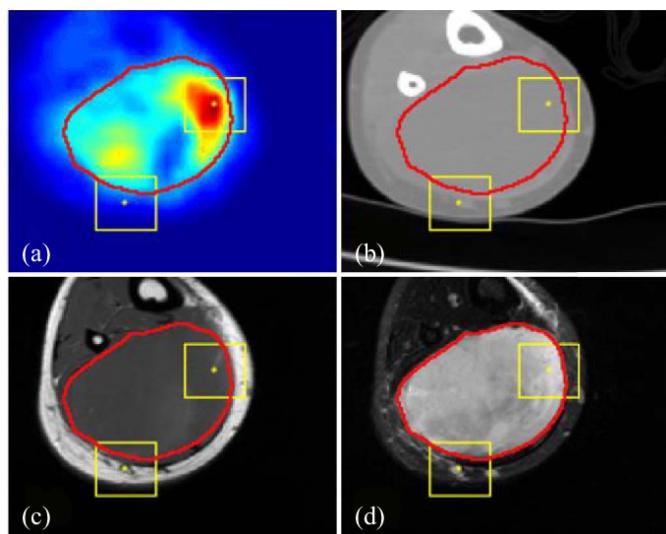


Figura 2.5: Imágenes multimodales en la misma posición. (a) PET, (b) TC, (c) RM-T1, (d) RM-T2. Fuente: Guo *et al.*

2.2. Métodos y Técnicas de Segmentación

La segmentación de imágenes médicas es una técnica que se utiliza en el contexto del análisis y el diagnóstico de enfermedades ya que puede mejorar la precisión y la eficiencia del análisis y el diagnóstico. Consiste en la identificación y la separación de estructuras específicas en las imágenes médicas como órganos internos, vasos sanguíneos o tumores.

Existen diferentes métodos y técnicas de segmentación de imágenes médicas cada uno con sus propias ventajas y desventajas que pueden ser útiles en diferentes contextos y para diferentes tipos de estructuras en las imágenes. La elección de la técnica de segmentación adecuada dependerá del tipo de imagen y de la estructura que se deseé segmentar.

2.2.1. Métodos de segmentación

Los métodos de segmentación de imágenes médicas son algoritmos que se utilizan para implementar las técnicas de segmentación de imágenes médicas. Algunos de los principales métodos de segmentación de imágenes médicas son:

Método de intensidad de la imagen

El método de intensidad de la imagen es un método de segmentación de imágenes médicas que se basa en la identificación de diferencias en la intensidad de la imagen. Se basa en el hecho de que las diferentes estructuras en una imagen médica tienden a tener diferentes niveles de intensidad, lo que permite identificarlas y segmentarlas mediante el uso de técnicas de procesamiento de imágenes.

En este método se analiza la intensidad de cada píxel en la imagen para detectar diferencias entre las estructuras y el fondo de la imagen. Una vez detectadas estas diferencias, se utilizan algoritmos de umbralización (*thresholding*) o de agrupamiento para separar las estructuras en diferentes regiones.

La umbralización o *thresholding* consiste en establecer un umbral de intensidad para separar la imagen en dos regiones: las estructuras de interés y el fondo. Una vez establecido el umbral, se utiliza un algoritmo para identificar todas las regiones que tienen un nivel de intensidad superior al umbral, que se consideran como las estructuras de interés.

Otra técnica es el análisis de la distribución de la intensidad de los píxeles en una imagen, lo que se conoce como histograma de intensidad. El histograma de intensidad se utiliza para identificar diferentes niveles de intensidad

en la imagen, lo que permite identificar los objetos en la imagen. A partir de este análisis, los objetos identificados se segmentan mediante el uso de umbrales en la intensidad.

Método de detección de bordes

El método de detección de bordes de la imagen es una técnica de procesamiento de imagen utilizada para identificar los límites entre los diferentes objetos en una imagen. Se basa en el principio de que los bordes de los objetos en una imagen se caracterizan por variaciones locales en la intensidad de la luz de la imagen.

En este método, se analiza la imagen para detectar cambios en la intensidad o en la dirección del gradiente que indican la presencia de un borde. Una vez detectados los bordes, se utilizan algoritmos de propagación de contornos o de extracción de características para separar las estructuras en diferentes regiones.

Existen varios algoritmos de detección de bordes, como el algoritmo de Sobel, el algoritmo de Canny, y el algoritmo basado en la Laplaciana, cada uno de ellos tiene sus propias ventajas y limitaciones y se utilizan en función del tipo de imagen y la estructura a segmentar.

Método de clústering

El método de clústering es un método de segmentación de imágenes médicas que se basa en la identificación de grupos de píxeles que comparten características similares. En este método, se analizan las características de cada píxel en la imagen, como el brillo, el contraste o la textura, para identificar los grupos de píxeles que comparten características similares. Una vez identificados estos grupos, se utilizan algoritmos de agrupamiento o de clasificación para separar las estructuras en diferentes regiones.

Existen varios algoritmos de clústering que se utilizan en la segmentación de imágenes médicas, como el algoritmo *K-means*, el algoritmo de agrupamiento jerárquico y el algoritmo de mezcla gaussiana, entre otros. Cada uno de ellos tiene sus propias ventajas y limitaciones y se utilizan en función del tipo de imagen y la estructura a segmentar.

Una vez realizado el clúster, se puede utilizar la información de los clústeres para segmentar las estructuras de interés. Por ejemplo, se puede utilizar un algoritmo de propagación de regiones para seguir los límites de los clústeres y segmentar las estructuras completas.

Método de deformación

Es un método de segmentación de imágenes médicas que se basa en el uso de algoritmos de optimización para estimar los parámetros que definen la segmentación. Estos parámetros pueden incluir el grosor de la superficie de segmentación, el contorno de la superficie de segmentación, y el valor de los píxeles en la imagen de entrada.

El algoritmo de optimización intenta minimizar una función de costo definida por el usuario que refleja los parámetros de la segmentación deseada. El algoritmo se basa en una aproximación iterativa para encontrar la solución óptima. Esta aproximación requiere que el usuario proporcione un conjunto de condiciones de frontera para establecer los límites de la segmentación. Estas condiciones establecen los límites de la deformación aceptable para la segmentación deseada, lo que evita que la solución óptima se aleje demasiado de los parámetros iniciales.

La idea es que el modelo inicial se ajuste lo más posible a la estructura que se desea segmentar, y luego se aplique un proceso de deformación controlado para adaptar el modelo a las características de la imagen y segmentar la estructura de interés.

Ejemplos de métodos de deformación utilizados en la segmentación de imágenes médicas incluyen el método de deformación de *snakes*, el método de deformación de mallas y el método de deformación basado en el modelo de active *shape*.

2.2.2. Técnicas de segmentación

Las técnicas de segmentación de imágenes médicas son algoritmos que se utilizan para identificar y separar estructuras específicas en las imágenes médicas. Algunas de las principales técnicas de segmentación de imágenes médicas son:

Segmentación por región

La segmentación por región se basa en la identificación de regiones homogéneas en las imágenes según el principio de que los objetos en una imagen tienen características comunes que los separan de otros objetos.

Esta técnica utiliza algoritmos que analizan las características de la imagen, como el brillo o el contraste, para identificar las regiones que comparten características similares. Una vez identificadas estas regiones, se utilizan algoritmos de agrupamiento para separarlas en diferentes grupos.

Por ejemplo, en una imagen de rayos X, la unión entre los huesos se

puede destacar mediante la segmentación por región. Esta técnica se utiliza para identificar y diferenciar áreas de interés en una imagen, como tumores, lesiones o órganos.

Segmentación por contorno

La segmentación por contorno se basa en la identificación de los contornos de las estructuras en las imágenes.

Esta técnica utiliza algoritmos que analizan las características de la imagen, como la intensidad o la dirección del gradiente, para detectar los bordes de las estructuras en las imágenes. Una vez detectados los bordes, se utilizan algoritmos de propagación de contornos para separar las estructuras en diferentes regiones.

Los algoritmos de segmentación por contorno comúnmente usados incluyen la transformada de Hough, el algoritmo de búsqueda en profundidad y la detección de bordes. Estos algoritmos se utilizan para detectar los límites entre el interior y el exterior de una región de interés y para definir sus contornos.

Esta técnica de segmentación es útil para la detección de lesiones, la segmentación de órganos y la detección de cáncer.

Segmentación por características de la imagen

La segmentación por características de la imagen se basa en la identificación de características específicas de las estructuras en las imágenes.

Esta técnica utiliza algoritmos que analizan las propiedades de la imagen, como la forma, el color, el tamaño o la textura, para identificar las estructuras en las imágenes. Una vez identificadas las estructuras, se utilizan algoritmos de clustering o de clasificación para separarlas en diferentes grupos.

Segmentación semiautomática

La segmentación semiautomática combina técnicas automáticas y técnicas manuales. En esta técnica, el usuario proporciona una serie de puntos de inicio en la imagen para que el algoritmo pueda iniciar la segmentación. A partir de estos puntos de inicio, el algoritmo utiliza técnicas de segmentación automática para identificar y separar las estructuras en la imagen. Sin embargo, el usuario puede intervenir en el proceso de segmentación para corregir errores o mejorar la precisión de la segmentación.

La segmentación livewire es un ejemplo representativo de este tipo de técnica.

Segmentación basada en modelos

La segmentación basada en modelos utiliza modelos matemáticos o físicos para representar las estructuras en las imágenes. En esta técnica, el usuario proporciona un modelo de la estructura que se desea segmentar, que puede ser un modelo geométrico, un modelo de intensidad o un modelo físico. A partir de este modelo, el algoritmo utiliza técnicas de optimización o de ajuste de modelos para encontrar la mejor representación de la estructura en la imagen. La ventaja de esta técnica es que permite una mayor precisión en la segmentación, ya que el modelo proporcionado por el usuario refleja la forma y la estructura de la estructura en la imagen de manera más precisa.

Un ejemplo es la segmentación basada en el modelo de etiquetado de regiones que utiliza un algoritmo para dividir la imagen en regiones conectadas de intensidad similar. Otra técnica es la segmentación basada en el modelo de máxima verosimilitud que utiliza un algoritmo para encontrar la mejor división de la imagen en regiones basándose en un modelo de distribución de probabilidad.

En general son menos flexibles que las de basada en aprendizaje automático ya que requieren un modelo matemático previamente construido. Sin embargo, son más eficientes y escalables en comparación con los algoritmos tradicionales y son útiles para segmentar estructuras con formas bien definidas y patrones de intensidad conocidos.

Segmentación basada en aprendizaje automático

La segmentación basada en aprendizaje automático utiliza algoritmos de aprendizaje automático para identificar y separar las estructuras en las imágenes. En esta técnica, el usuario proporciona un conjunto de imágenes etiquetadas, es decir, imágenes que ya han sido segmentadas por un experto. A partir de este conjunto de imágenes etiquetadas, el algoritmo entrena un modelo de aprendizaje automático que puede ser utilizado para segmentar nuevas imágenes. La ventaja de esta técnica es que permite una mayor precisión y una mayor rapidez en la segmentación, ya que el modelo aprende a segmentar las imágenes a partir de un conjunto de imágenes ya segmentadas lo que permite automatizar procesos clínicos y para mejorar la precisión y la velocidad de la detección de patologías.

2.3. Segmentación basada en aprendizaje automático

Los métodos tradicionales de segmentación como los basados en detección de bordes, en regiones o en umbrales se basan en cálculos matemáticos simples que le confieren una gran velocidad de segmentación, sin embargo, la precisión de la segmentación a nivel de detalle ha sido ampliamente superada por técnicas que hacen uso del aprendizaje profundo.

Esto se puede lograr mediante la creación de un algoritmo de aprendizaje automático que use un gran conjunto de datos de imágenes previamente etiquetadas para entrenar a una red neuronal y luego aplicar lo que ha aprendido a nuevas imágenes no etiquetadas para identificar y segmentar las estructuras relevantes.

A continuación se hace un repaso de las principales redes de aprendizaje profundo, desde los inicios a las últimas novedades.

2.3.1. Redes Neuronales Convolucionales (CNN)

LeCun *et al.* [19] en 1998 introdujo el algoritmo de backpropagation basada en el gradiente, posteriormente en 2012 con AlexNet y su enorme éxito en la clasificación de imágenes del *dataset* de ImageNet fue cuando las CNN acapararon una gran atención en la investigación de visión artificial.

CNN se compone de una capa de entrada, una capa de salida y varias capas ocultas. Cada una de las capas ocultas realiza una operación específica, como convolución, agrupación y activación. La capa de entrada está conectada a la imagen de entrada, y el número de neuronas en esta capa se corresponde con el número de píxeles de la imagen de entrada siguiendo una correspondencia uno a uno [20].

La capa convolucional media realiza la extracción de características en los datos de entrada a través de una operación de convolución para obtener un mapa de características. Cada capa convolucional está compuesta por un conjunto de *kernels* o filtros que se aplican a la imagen de entrada. Los filtros buscan patrones específicos en la imagen, como bordes, texturas o patrones de intensidad. A medida que la imagen se procesa a través de las capas de la red, los patrones se vuelven cada vez más complejos y abstractos.

Como podemos ver en la Figura 2.6 tenemos una convolución de 3x3, es decir, el filtro o kernel es una matriz de dimensión 3x3, con un *stride* de 1. Este elemento llamado *stride* se define como el tamaño de desplazamiento del kernel al deslizarse por la imagen de entrada. Un stride con valor 1 significa que deslizamiento del kernel a través de la imagen es píxel a píxel. Un *stride* con valor 2 significa que el deslizamiento del kernel es moviendo el kernel

dos píxeles en cada paso que se realiza durante la reducción de la imagen. Por otro lado hay otro elemento en dichas capas convolucionales que es el *padding* que agrega los píxeles necesarios en los bordes de la imagen para poder aplicar el kernel en estos antes de aplicar el filtro. Un *padding* de 1 para una imagen original de 5x5x1 hace que tenga un tamaño de entrada de 7x7x1 y mantiene el tamaño de salida en 5x5x1 como la imagen original.

0	0	0	0	0	0	0	0
0	2	4	9	1	4	0	
0	2	1	4	4	6	0	
0	1	1	2	9	2	0	
0	7	3	5	1	3	0	
0	2	3	4	8	5	0	
0	0	0	0	0	0	0	

X =

1	2	3	
-4	7	4	
2	-5	1	

21	59	37	-19	2
30	51	66	20	43
-14	31	49	101	-19
59	15	53	-2	21
49	57	64	76	10

Entrada (7x7x1)
Original (5x5x1) + pad 1 Kernel o Filtro (3x3x1) Salida (5x5x1)

Figura 2.6: Convolución con kernel tamaño 3, padding 1 y stride 1.

La arquitectura común en las redes neuronales convolucionales emplea parejas de capas convolucionales (extracción de características) seguidas de capas de *pooling*. Este tipo de capas *pooling* comprimen el mapa de características obtenido a la salida de la anterior capa. Se suele usar *Max-pooling* que consiste en seleccionar el valor máximo dentro de la ventana y se reemplaza con ese valor. La Figura 2.7 muestra como se aplica el primer paso de realizar *Max-pooling* con kernel de tamaño 2 y un *stride* de tamaño 1 a la salida del ejemplo anterior de la Figura 2.6.

21	59	37	-19	2
30	51	66	20	43
-14	31	49	101	-19
59	15	53	-2	21
49	57	64	76	10

Mapa de Características (5x5x1)

59			

Max Pooling Kernel = 2 Max Pooling Salida (4x4x1)

Figura 2.7: Max-pooling con kernel tamaño 2 y stride 1.

Tras extraer las características más importantes y reducir la dimensionalidad de los datos mediante un cierto número de parejas convolucionales *pooling*, finalmente se emplea un conjunto de capas densas que se ocuparán

de realizar la predicción. Puesto que las capas convolucionales trabajan con objetos tridimensionales y las densas con unidimensionales, será necesario aplicar una capa de conversión llamada “flatten” entre ambas.

Las peculiaridades vienen en la última capa, donde entra en juego el objetivo que se trate de lograr con el modelo. Para problemas de clasificación y segmentación, dicha capa deberá tener un número de neuronas igual al número de clases a predecir, a modo de probabilidades o distribución de probabilidades. Esta última capa se implementa a través de una convolución de filtro y *stride* unidad. La Figura 2.8 [21] muestra los principales elementos de una CNN.

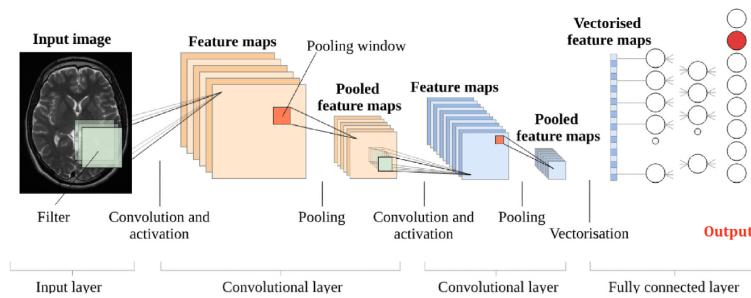


Figura 2.8: Elementos de una Red Neuronal Convolucional (CNN). Fuente: Lundervold *et al.*

Existen arquitecturas CCN como ResNet, VGG, GoogleNet o SegNet o las que describimos a continuación que se utilizan para el filtrado, la convolución, el aprendizaje, la clasificación o la detección de patrones.

2.3.2. Fully Convolutional Neural Networks (FCN)

Este tipo de redes son las que mayor éxito han alcanzado en la segmentación semántica. Long *et al.* [22] propusieron una evolución de las CNN donde las entradas son de tamaño arbitrario y produce una salida con inferencia y aprendizaje eficientes conforme a la estructura que muestra la Figura 2.9.

Esta arquitectura se basa en la idea de reemplazar completamente las capas densas por capas convolucionales, lo que permite una inferencia más rápida y una mejor generalización de los modelos. Esto se logra al convertir las entradas de la imagen en mapas de características a través de varias capas convolucionales y luego utilizando una combinación de capas decodificadoras para producir un mapa de segmentación. Las capas decodificadoras combinan los mapas de características de entrada para generar la salida deseada. Estas capas incluyen capas de suavizado, capas de convolución transpuesta, capas de agregación, etc.

FCN utiliza la capa de deconvolución para aumentar la resolución (*up-*

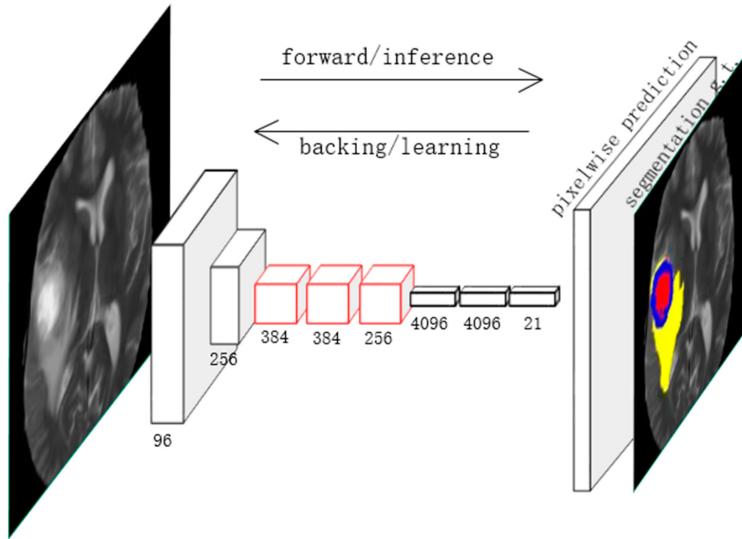


Figura 2.9: Estructura de fully convolutional network (FCN). Fuente: Long *et al.* (modificada)

sampling) del mapa de características de la última capa de convolución y restaurarlo al mismo tamaño de la imagen de entrada. De esta manera, se puede generar una predicción para cada píxel manteniendo al mismo tiempo la información espacial de la imagen de entrada original. Finalmente, se realiza una clasificación píxel por píxel en el mapa de características aumentado de resolución para completar la segmentación de imagen final. Según el grado de ampliación de aumento de resolución, se divide en FCN-32s, FCN-16s y FCN-8s.

Se tiene una capa de salida con tantos nodos como píxeles en la imagen. Cada nodo de la capa de salida se corresponde con un píxel de la imagen y su valor representa la probabilidad de que ese píxel pertenezca a una determinada clase o etiqueta.

Además se adaptan las redes como AlexNet, VGG y GooglNet a redes FCNN.

2.3.3. U-Net

Ronneberger *et al.* [23] diseñó la que se ha convertido en el estándar de facto y ha conseguido un gran éxito y difusión en la segmentación de imágenes médicas.

La arquitectura consta de dos partes: un camino de contracción (o *encoder*) que captura el contexto y un camino de expansión (o *decoder*) que permite una localización precisa. El *encoder* se encarga de analizar la imagen

y extraer características y el *decoder* se encarga de recuperar la resolución original de la imagen y generar la salida de segmentación.

El *encoder* está compuesto por una sucesión de bloques convolucionales, cada uno de los cuales seguido de una capa de *max pooling*. Esto permite que la red aprenda características de la imagen a niveles de abstracción cada vez más altos. El *decoder* usa estas características para generar la etiqueta de segmentación deseada y utiliza capas de *upsampling* para aumentar el tamaño de la imagen y recuperar la resolución original. La arquitectura también incluye una capa de conexión entre el *encoder* y el *decoder*, que asegura que los patrones aprendidos por el *encoder* sean transferidos correctamente al *decoder* lo cual ayuda a la red a mantener información espacial importante durante el proceso de segmentación y a la vez a mantener un alto nivel de precisión en la localización de la estructura de interés. La Figura 2.10 muestra una estructura típica asociada a U-Net.

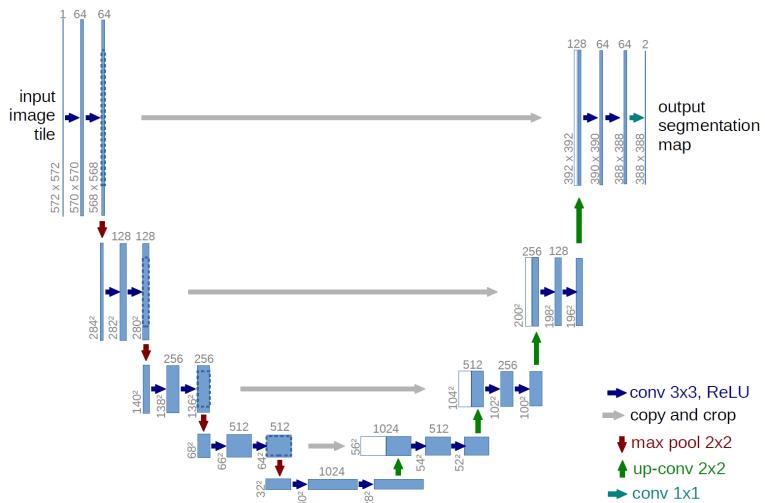


Figura 2.10: Estructura de UNet. Fuente: Ronneberger *et al.*

En la actualidad existen múltiples variantes como V-net, Res-UNet, H-DenseUNet, pero todas ellas se basan en la idea central de U-Net donde los operadores de *pooling* son sustituidos por operadores de *upsampling* y con ello las capas pueden aumentar la resolución de la salida con la finalidad de localizar las características de mayor resolución del camino que seguimos hasta abajo, para que luego se puedan llegar combinar con el camino ascendente de salida de tal forma que un camino contrae y el otro expande.

2.3.4. GAN

Una GAN (*Generative Adversarial Network*) es una red neuronal artificial que consiste en dos redes neuronales enfrentadas, una que genera datos (la generadora) y otra que evalúa la calidad de esos datos (la adversaria o discriminador). El generador se encarga de generar imágenes sintéticas a partir de un ruido aleatorio, mientras que el discriminador se encarga de determinar si una imagen es real o sintética.

La arquitectura de GAN para la segmentación de imágenes médicas se basa en la idea de que el generador y el discriminador pueden entrenarse por separado para mejorar el desempeño de la segmentación. El generador aprenderá a generar una imagen objetivo a partir de una imagen de entrada. El discriminador, por otro lado, aprenderá a distinguir entre una imagen de segmentación real y una imagen de segmentación generada por el generador. Una vez que el generador y el discriminador hayan sido entrenados, se pueden usar para mejorar el rendimiento de la segmentación. La Figura 2.11 [24] muestra la estructura de una red generativa adversaria GAN.

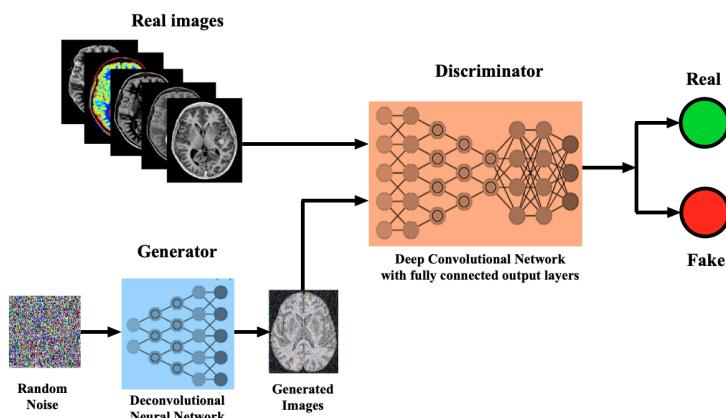


Figura 2.11: Estructura de una Red Generativa Adversarial (GAN). Fuente: web

Las imágenes son enviadas aleatoriamente al discriminador, el cual debe decidir cuáles son reales y cuáles han sido generadas. Si su juicio es incorrecto (es decir, una imagen falsa es confundida con una real), el discriminador aprenderá de su error y mejorará en ejecuciones posteriores. Si su juicio es correcto, esta información se enviará de vuelta al generador para ayudarlo a desarrollar mejores imágenes ‘falsas’. A través de la iteración, el rendimiento de ambos componentes mejora, al igual que el GAN en su conjunto.

Este tipo de red propuestas en 2014 por Goodfellow *et al.* [25] han evolucionado y son usadas para segmentación de imágenes médicas en redes como *Segmentation Adversarial Network* (SegAN), *Structure Correcting Adversa-*

rial Network (SCAN), *Projective Adversarial Network* (PAN), *Distributed Asynchronized Discriminator GAN* (AsynDGAN), *Deep-supGAN* o *Conditional GANs* (cGAN) entre otras.

2.3.5. EfficientNet

EfficientNet es una arquitectura de red neuronal desarrollada para mejorar el rendimiento de las redes CNN en tareas de visión artificial. Se diseñó para aumentar la precisión y eficiencia de los modelos de redes neuronales convolucionales tradicionales. La arquitectura se basa en tres principios: escalabilidad, profundidad y anchura.

En cuanto a la escalabilidad, EfficientNet busca encontrar los parámetros óptimos para la arquitectura basada en el tamaño de la imagen de entrada y el número de canales. Esto significa que EfficientNet puede escalar su arquitectura para adaptarse a diferentes tamaños de imagen sin tener que entrenar un modelo diferente para cada tamaño de imagen lo cual permite que la red sea eficiente en el uso de recursos, ya que se ajusta automáticamente al tamaño de la imagen.

Referente a la profundidad, EfficientNet utiliza capas de convolución y de *pooling* de manera eficiente para extraer características de la imagen. Estas capas se ajustan automáticamente en función de la complejidad de la imagen de entrada.

Por último, EfficientNet utiliza una técnica llamada ‘anchura eficiente’ para aumentar el número de canales en las capas de la red. Esto permite que la red aprenda características más complejas de la imagen de entrada.

Se trata de un conjunto de 8 arquitecturas donde la primera EfficientNet-B0 cuenta con 4 millones de parámetros entrenables hasta la EfficientNet-B7 con 66 millones de parámetros. Todas ellas logran resultados parecidos e incluso mejores frente a otras arquitecturas populares como ResNet, Inception o DenseNet a la vez que reducen el número de parámetros en un factor de entre 4 y 8 veces como se comprueba con la Figura 2.12 del artículo de Tan *et al.* [26].

2.3.6. SegResNet

SegResNet es el nombre de la arquitectura del modelo, basado en el modelo ganador utilizado por Myronenko [27] en su artículo de 2018, que fue el ganador de la competición del reto de BraTS 2018. La arquitectura se basa en el marco *encoder-decoder* (codificador-decodificador) de las redes neuronales convolucionales como UNet, con un codificador proporcionalmente más grande, responsable de extraer características profundas de la imagen,

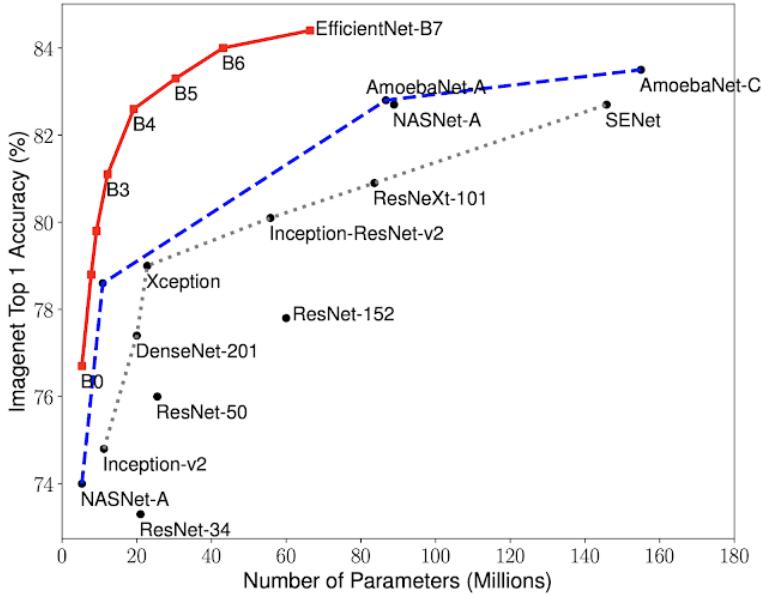


Figura 2.12: Comparación de tamaño del modelo vs. precisión de EfficientNets. Fuente: Tan *et al.*

y una estructura de decodificador correspondiente para construir las máscaras densas de segmentación. Myronenko [27] también incluyó una rama de *autoencoder variacional* (VAE) a la red para reconstruir las imágenes de entrada conjuntamente con la segmentación con el fin de regular el codificador compartido. En el momento de la inferencia, solo se utiliza la parte principal de la segmentación codificador-decodificador.

Tal y como muestra la Figura 2.13, la entrada es un recorte de resonancia magnética 3D de cuatro canales, seguido de una primera convolución 3x3x3 3D con 32 filtros. Cada bloque verde es un bloque similar a ResNet con la normalización GroupNorm. La salida del decodificador de segmentación tiene tres canales (con el mismo tamaño espacial que la entrada) seguido de una función sigmoid para mapas de segmentación de las tres subregiones del tumor (WT, TC, ET). La rama VAE reconstruye la imagen de entrada en sí misma, y solo se utiliza durante el entrenamiento para regular el codificador compartido.

La rama del *autoencoder* se utiliza para proporcionar orientación y regularización adicionales a la parte del codificador, ya que el tamaño del conjunto de datos de entrenamiento es limitado. Se sigue el enfoque del *autoencoder variacional* (VAE) para agrupar/agrupar mejor las características del punto final del codificador.

Lo que distingue a SegResNet de otras redes neuronales de segmenta-

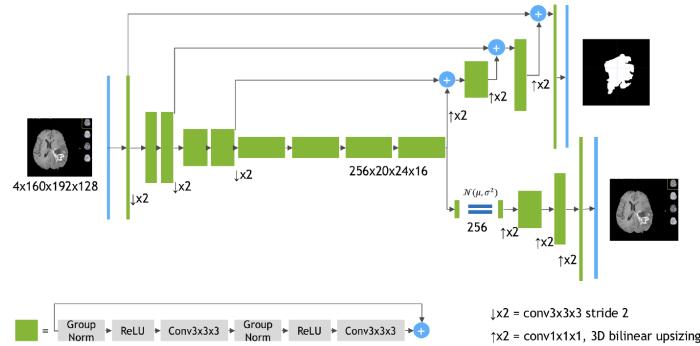


Figura 2.13: Visualización esquemática de la arquitectura de la red SegResNet. Fuente: Myronenko *et al.*

ción es que utiliza una variante de la arquitectura de red conocida como '*Residual Network*' (ResNet) [28] [29]. Esta arquitectura es conocida por su capacidad para entrenar redes neuronales profundas de forma eficiente y reducir el problema del '*vanishing gradient*' (desvanecimiento del gradiente), que puede dificultar el aprendizaje en redes neuronales profundas.

2.3.7. TransUNet

Chen *et al.* [30] proponen una arquitectura que combina *Transformers* y U-Net y que podemos ver de forma resumida en la Figura 2.14.

La arquitectura de TransUNET establece mecanismos de atención automática desde la perspectiva de la predicción de secuencia a secuencia. Para compensar la pérdida de resolución de características traída por los *transformers*, TransUNet emplea una arquitectura híbrida CNN-*Transformer* para aprovechar tanto la información espacial de alta resolución detallada de las características CNN y el contexto global codificado por los *transformers*.

Inspirado en el diseño arquitectónico en forma de U, la característica de atención automática codificada por los *transformers* se amplía entonces para ser combinada con diferentes características CNN de alta resolución omitidas del camino de codificación, para permitir una localización precisa.

Esta arquitectura permite conservar las ventajas de los *transformers* y también beneficiar la segmentación de imágenes médicas. Los resultados empíricos sugieren que la arquitectura basada en *transformers* presenta una mejor manera de aprovechar la atención automática en comparación con los métodos de atención automática basados en CNN anteriores. Además, se observa que una incorporación más intensiva de características de bajo nivel generalmente conduce a una mejor precisión de segmentación.

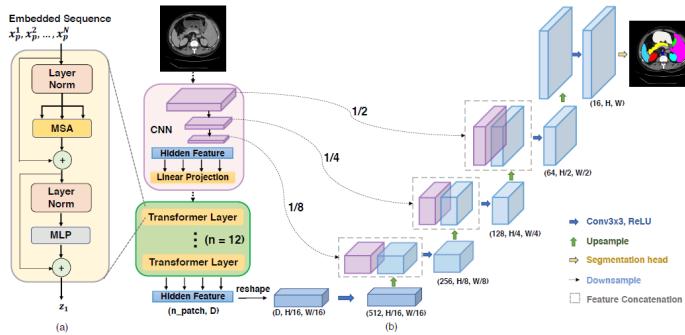


Figura 2.14: (a) Esquema de la capa Transformer; (b) Arquitectura de TransUNet. Fuente: Chen *et al.*

2.3.8. DiNTS

Las redes diseñadas manualmente, como U-Net, han sido ampliamente utilizadas en diferentes tareas. Sin embargo, la diversidad de tareas de segmentación de imágenes médicas puede ser extremadamente alta ya que las características y apariencias de las imágenes pueden ser completamente distintas para diferentes modalidades y la presentación de las enfermedades puede variar considerablemente. Esto hace que la aplicación directa de incluso una red exitosa como U-Net a una nueva tarea sea menos probable que sea óptima.

Los algoritmos de búsqueda de arquitectura neural (NAS) intentan descubrir automáticamente las arquitecturas óptimas dentro de un espacio de búsqueda. El espacio de búsqueda de NAS para la segmentación suele contener dos niveles: el nivel de topología de red y el nivel de celda. La topología de red controla las conexiones entre las celdas y decide el flujo de los mapas de características a través de diferentes escalas espaciales. El nivel de celda decide las operaciones específicas en los mapas de características. Un espacio de búsqueda más flexible tiene más potencial para contener arquitecturas de mejor rendimiento.

DiNTS (*Differentiable Neural Network Topology Search*) es una arquitectura de red neuronal para la segmentación de imágenes médicas 3D que utiliza la búsqueda de topología de redes neuronales diferenciables (DNAS) para encontrar automáticamente la mejor arquitectura de red para una tarea de segmentación específica. La Figura 2.15 muestra el espacio de búsqueda DiNTS definido por He *et al.* en su artículo [31].

Esta arquitectura usa una búsqueda diferenciable de topología de red neuronal para optimizar los parámetros de la red neuronal y así mejorar la precisión de la segmentación. El proceso de búsqueda diferenciable de

la topología de la red neuronal se lleva a cabo mediante un proceso de optimización de búsqueda dirigida para encontrar la mejor combinación de capas, tamaños de kernel, etc. para la tarea de segmentación específica. Esto se logra mediante la identificación de una serie de configuraciones de red neuronales viables, selección de la mejor opción y posterior mejora de los parámetros de la red neuronal.

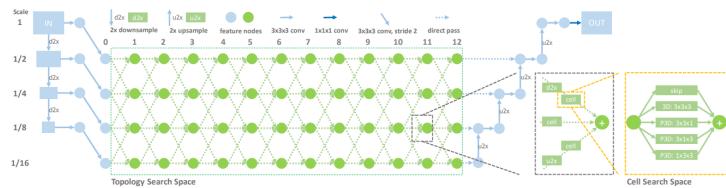


Figura 2.15: Espacio de búsqueda de DiNTS. Fuente: He *et al.*

El espacio de búsqueda contiene $L = 12$ capas. Las aristas azules son el tallo que contiene operaciones predefinidas. Las operaciones de celdas se definen en las aristas mientras que los nodos son mapas de características. Las aristas que se seleccionan en el espacio de búsqueda de topología para que las características fluyan desde la entrada hasta la salida forman una topología de red candidata. Cada arista en el espacio de búsqueda incluye una celda que contiene 5 operaciones para seleccionar. Una arista de reducción/ampliación de tamaño también contiene una operación de 2x de reducción/ampliación de tamaño.

Al utilizar DNAS, DiNTS es capaz de superar a las arquitecturas de segmentación de imágenes médicas diseñadas manualmente en términos de precisión y eficiencia. El espacio de búsqueda de topología de DiNTS es altamente flexible y logra el mejor rendimiento en el desafío MSD (Medical Segmentation Decathlon) de referencia utilizando arquitecturas no-UNet.

2.3.9. Swin-UNETR

Aunque los enfoques basados en FCNN tienen grandes capacidades de aprendizaje de representación, su rendimiento en el aprendizaje de dependencias a larga distancia está limitado a sus campos receptivos localizados. Como resultado, esta carencia en la captura de información a múltiples escalas conduce a una segmentación sub-óptima de estructuras con formas y escalas variables (por ejemplo, lesiones cerebrales de diferentes tamaños).

UNet TRansformers (UNETR) es una arquitectura que utiliza un transformador como *encoder* (codificador) para aprender representaciones de secuencia del volumen de entrada y capturar efectivamente la información global multi-escala, al mismo tiempo que sigue el diseño de red ‘en U’ exitoso para el *encoder* y el *decoder* (codificador y decodificador). El codificador

transformador está conectado directamente a un decodificador a través de conexiones directas en diferentes resoluciones para calcular la salida final de segmentación semántica. La Figura 2.16 muestra el modelo UNETR definido por Hatamizadeh *et al.* [32].

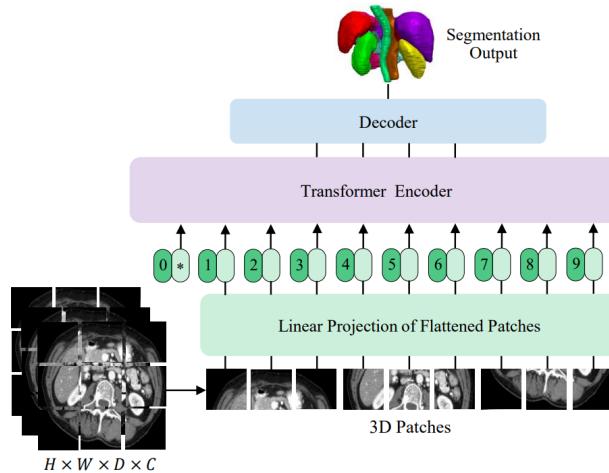


Figura 2.16: Modelo UNETR transformer encoder con conexión directa a decoder . Fuente: Hatamizadeh *et al.*

UNETR aprovecha la potencialidad de los transformes para la segmentación de imágenes médicas volumétricas. Reformula la tarea de segmentación 3D como un problema de predicción de secuencia a secuencia y utiliza un *transformer* como codificador para aprender información contextual desde los parches (piezas) de entrada embebidos. Las representaciones extraídas del codificador del transformador se combinan con el decodificador basado en CNN a través de conexiones directas a múltiples resoluciones para predecir las salidas de segmentación. En lugar de utilizar transformadores en el decodificador se utiliza un decodificador basado en CNN. Esto se debe a que los *transformers* son incapaces de capturar adecuadamente información localizada, a pesar de su gran capacidad para aprender información global.

Como puede observarse en la Figura 2.17, en la arquitectura de UNETR un volumen de entrada 3D (por ejemplo, $C = 4$ canales para imágenes MRI) se divide en una secuencia de parches no solapantes y uniformes y se proyecta en un espacio de incrustación utilizando una capa lineal. La secuencia se agrega con una incrustación de posición y se utiliza como entrada para un modelo transformador. Se extraen las representaciones codificadas de diferentes capas en el transformador y se combinan con un decodificador mediante conexiones directas para predecir la segmentación final. Se proporcionan tamaños de salida para una resolución de parche $P = 16$ y un tamaño de incrustación $K = 768$.

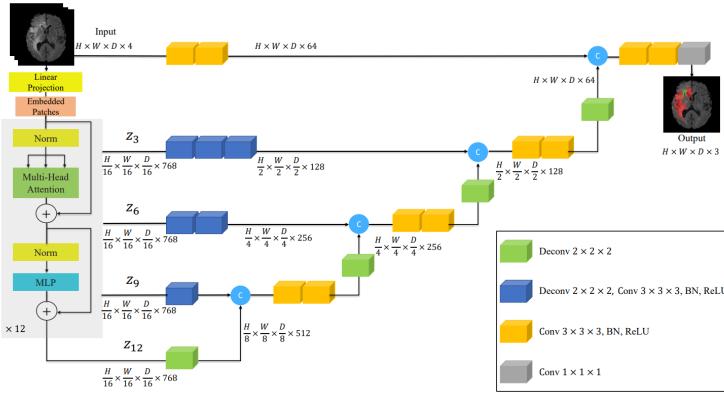


Figura 2.17: Arquitectura de UNETR. Fuente: Hatamizadeh *et al.*

En visión artificial (*Computer Vision*), los *Vision Transformers* (ViT) han demostrado un rendimiento de última generación en varios *benchmark*. Específicamente, el módulo de auto-atención en los modelos basados en ViT permite modelar información de largo alcance mediante la interacción entre las incrustaciones de token y, por lo tanto, llevando a representaciones contextuales locales y globales más efectivas. Además, los ViT han logrado el éxito en el aprendizaje efectivo de tareas pretexto para el preentrenamiento autorregulado en varias aplicaciones. En el análisis de imágenes médicas, UNETR es la primera metodología que utiliza un ViT como su codificador sin depender de un extracto de características basado en CNN.

Recientemente, Hatamizadeh *et al.* ha propuesto los *Swin transformers* [33] como un transformador de visión jerárquico que calcula la auto-atención en un esquema de partición de ventana deslizante eficiente. Como resultado, los transformadores *Swin* son adecuados para varias tareas posteriores en las que se pueden aprovechar las características multiescala extraídas para un procesamiento posterior.

El *encoder* de transformador de ventanas deslizantes (*Swin*) extrae características en cinco resoluciones diferentes utilizando ventanas desplazadas para calcular la auto-atención y está conectado a un decodificador basado en FCNN en cada resolución a través de conexiones directas (*skip connections*).

Como puede observarse en la Figura 2.18, la entrada del modelo Swin-UNETR son imágenes MRI multi-modales 3D con 4 canales. El Swin UNETR crea parches no superpuestos de los datos de entrada y utiliza una capa de partición de parches para crear ventanas con un tamaño deseado para calcular la auto-atención. Las representaciones de características codificadas en el transformador *Swin* se alimentan a un decodificador CNN a través de una conexión de salto en varias resoluciones. La salida final de la segmentación consta de 3 canales de salida correspondientes a las subregiones ET,WT y

TC.

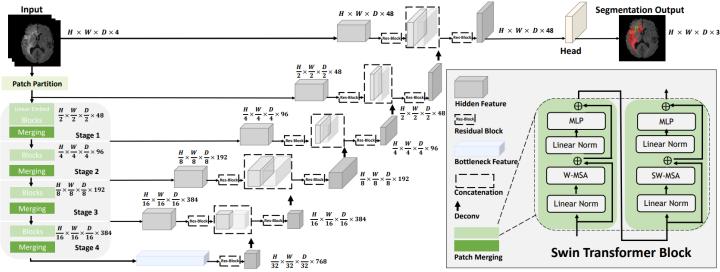


Figura 2.18: Arquitectura Swin-UNETR. Fuente: Hatamizadeh *et al.*

Para el mecanismo de atención, primero se utiliza una capa de partición de parches para crear una secuencia de tokens 3D y se proyecta en un espacio de incrustación con dimensión C. La auto-atención se calcula en ventanas no superpuestas que se crean en la etapa de partición para modelar la interacción de tokens de manera eficiente. Específicamente, se utiliza ventanas de tamaño M x M x M para particionar de manera equitativa un token 3D en una capa dada l en el codificador del transformador. Posteriormente, en la capa l+1, las regiones de ventana particionadas se desplazan por voxel M/2, M/2, M/2. En la Figura 2.19 se muestra el mecanismo de ventana deslizante. Hay que tener en cuenta que se ilustran 8 × 8 × 8 tokens 3D y un tamaño de ventana de 4 × 4 × 4.

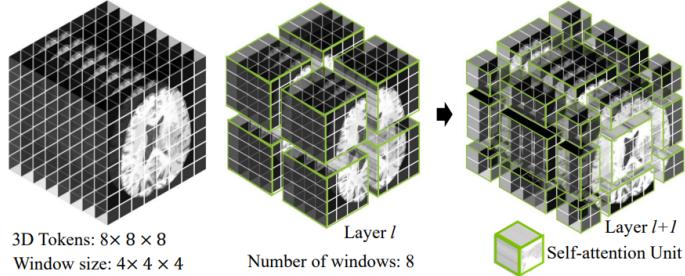


Figura 2.19: Resumen del mecanismo de ventana deslizante. Fuente: Hatamizadeh *et al.*

2.4. Datasets de Imágenes Médicas

La segmentación semántica se refiere al proceso de transformar imágenes médicas brutas en información relevante clínicamente y estructurada espacialmente, como el trazado de los límites del tumor, y es un prerrequisito esencial para una serie de aplicaciones clínicas, como la planificación de

radioterapia y la monitorización de la respuesta al tratamiento.

Como ya hemos visto, la segmentación de imágenes médicas es una técnica que permite identificar y marcar diferentes regiones o estructuras en una imagen médica, como órganos, tejidos y lesiones. Los datasets son conjuntos de datos utilizados en el desarrollo y evaluación de técnicas de segmentación de imágenes médicas. Son esenciales para entrenar y evaluar modelos de segmentación de imágenes médicas ya que proporcionan datos reales y precisos para su desarrollo.

Los *datasets* de imágenes médicas se utilizan en el desarrollo y evaluación de técnicas de segmentación de imágenes médicas por varias razones: Primero, proporcionan un conjunto de datos reales y precisos que se pueden utilizar para entrenar y evaluar modelos de segmentación de imágenes médicas, esto permite que los modelos sean más precisos y fiables en su capacidad para identificar y marcar diferentes regiones o estructuras en una imagen médica. Además, los datasets también pueden incluir imágenes etiquetadas y clasificadas, lo que facilita la comparación y el análisis de diferentes técnicas de segmentación de imágenes médicas. Por último, también pueden contener imágenes de diferentes modalidades médicas, como Tomografía Computarizada y Resonancia Magnética, lo que permite la comparación y el análisis de diferentes tipos de imágenes médicas.

2.4.1. Tipos de datasets

Los datasets de imágenes médicas utilizados en técnicas de segmentación de imágenes médicas pueden ser de varios tipos, cada uno con sus propias características y ventajas. Algunos ejemplos de tipos de datasets de imágenes médicas incluyen:

1. *Datasets* de imágenes médicas etiquetadas: Incluyen imágenes médicas con regiones etiquetadas y clasificadas por expertos en el campo, como radiólogos o patólogos. Son útiles para identificar y marcar diferentes estructuras como órganos, tejidos y lesiones en una imagen médica con el objetivo de producir resultados precisos y confiables.
2. *Datasets* de imágenes médicas de alta resolución: Contienen imágenes médicas con alta resolución y detalle, como imágenes histológicas microscópicas.
3. *Datasets* de imágenes médicas de diferentes modalidades: estos datasets contienen imágenes médicas de diferentes modalidades, como TC y RM, para permitir la comparación y el análisis de diferentes tipos de imágenes médicas. Estos datasets son útiles para entrenar y evaluar modelos de segmentación de imágenes médicas que pueden manejar diferentes modalidades de imágenes médicas.

En general, los tipos de datasets de imágenes médicas utilizados en técnicas de segmentación de imágenes médicas varían según las necesidades y objetivos del desarrollo y evaluación de estas técnicas.

En las siguientes secciones se presentan ejemplos de datasets de imágenes médicas que se utilizan comúnmente en el desarrollo y la evaluación de técnicas de segmentación de imágenes médicas. Se describe información sobre el tipo de imágenes que se incluyen en cada *dataset*, así como el número de imágenes y la resolución de las mismas. Cuando se dispone de información se detalla la disponibilidad y accesibilidad de cada *dataset*, así como cualquier otra información relevante, como si el *dataset* está etiquetado o no y si está disponible una versión de entrenamiento y prueba.

2.4.2. *Medical Segmentation Decathlon* (MSD)

El objetivo del desafío MSD era encontrar un solo algoritmo o sistema de aprendizaje que pudiera generalizar y funcionar con precisión en múltiples tareas de segmentación médica diferentes, sin necesidad de ninguna intervención humana.

Los *challenges* (desafíos o retos) internacionales se han convertido en el estándar de facto para la evaluación comparativa de los algoritmos de análisis de imágenes en una tarea específica. La segmentación es hasta ahora la tarea de procesamiento de imágenes médicas más ampliamente investigada, pero los diversos desafíos de segmentación suelen organizarse de manera aislada, de tal manera que el desarrollo de algoritmos se ve impulsado por la necesidad de abordar un único problema clínico específico.

El *Decathlon Challenge* creó diez conjuntos de datos organizadas por órganos y patologías como tumores cerebrales, tumores de pulmón, cardíacas, etc. que pueden verse en la Figura 2.20 [34]. Cada conjunto de datos tenía entre uno y tres objetivos de interés (ROI) (17 objetivos en total). Todos los conjuntos de datos han sido liberados con una licencia de copyright permisiva (CC-BY-SA 4.0), lo que permite el intercambio de datos, la redistribución y el uso comercial, y promoviendo el conjunto de datos como una plataforma de prueba estándar para todos los usuarios. Este conjunto de datos puede evaluar objetivamente los métodos generales de segmentación a través de puntos de referencia integrales [35].

Las imágenes (2.633 en total) se adquirieron de múltiples instituciones, anatomías y modalidades provenientes de aplicaciones clínicas reales. Todas las imágenes se anonimizaron y reformatearon al formato *Neuroimaging Informatics Technology Initiative* (NIfTI). Todas las imágenes se transpusieron (sin remuestreo) al marco de coordenadas derecho-anterior-superior más aproximado, asegurando que la dirección de la matriz de datos x-y-z fuera consistente. Finalmente, las modalidades no cuantitativas (por ejem-

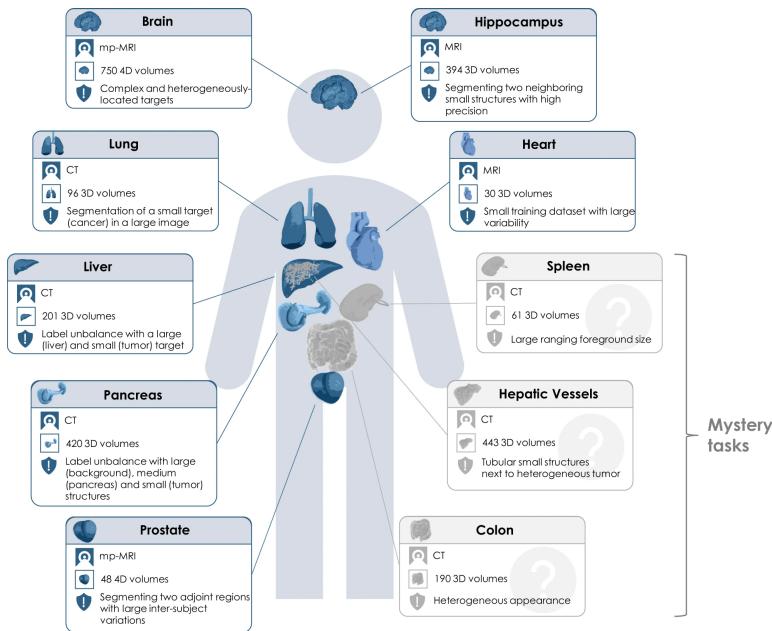


Figura 2.20: Resumen de las 10 tareas del MSD. Fuente: Antonelli *et al.*

plo, MRI) fueron escaladas robustamente min-max al mismo rango. Para cada tarea de segmentación, se proporcionó una etiqueta de nivel de píxel según la definición de cada tarea específica. Para 8 de 10 conjuntos de datos, dos tercios de los datos se liberaron como conjuntos de entrenamiento (imágenes y etiquetas) y un tercio como conjunto de prueba (imágenes sin etiquetas). Como las dos tareas restantes (tumor cerebral y hígado) consistían en datos de dos desafíos conocidos, se preservó la división original de entrenamiento/prueba.

La Tabla 2.1 resume los diez conjuntos de datos del MSD. Las abreviaturas utilizadas son: mp-MRI:imagen de resonancia magnética multiparamétrica. FLAIR:recuperación de inversión atenuada por líquido. T1w:imagen ponderada en T1. T1 nw Gd:post-contraste con Gadolinio (Gd) ponderada en T1. T2w:imagen ponderada en T2, CT:tomografía computarizada. PZ:zona periférica. TZ:zona de transición.

2.4.3. *Brain tumor segmentation (BRaTS)*

BRaTS, (*Brain tumor segmentation*), es un conjunto de datos de imagen médica específico para la segmentación de tumores cerebrales. Se utiliza para la competición de MICCAI [36] que se organiza todos los años desde 2012 y a la que se le añaden nuevos datos anualmente.

Los datos de entrenamiento y validación de BraTS describen un total de

Tarea	Modalidad	Protocol	Objetivo	# Muestras	(Train/ n/Test)	
Cerebro	mp-MRI	FLAIR, T1w, T1 ^ Gd, T2w	Edema, tumor real- zado y no realizado	750 (484/266)	4D	vol
Corazón	MRI	-	Aurícula izquierda	30	3D vol (20/10)	
Hipocampo	MRI	T1w	Anterior y posterior del hipocampo	394 (263/131)	3D	vol
Hígado	CT	Fase portal- venosa	Hígado y tumor hepático	210	3D vol (131/70)	
Pulmón	CT	-	Cáncer de pulmón y pulmón	96	3D vol (64/32)	
Páncreas	CT	Fase portal- venosa	Páncreas y masa tumoral pancreáti- ca	420 (282/139)	3D	vol
Próstata	mp-MRI	T2, ADC	Próstata PZ y TZ	48	4D vol (32/16)	
Colon	CT	Fase portal- venosa	Primarios de cáncer de colon	190	3D vol (126/64)	
Vasos Hepáticos	CT	Fase portal- venosa	Vasos hepáticos y tumor hepático	443 (303/140)	3D	vol
Bazo	CT	Fase portal- venosa	Bazo	61	3D vol (41/20)	

Cuadro 2.1: Conjunto de datos de *Medical Segmentation Decathlon*

5880 resonancias magnéticas de 1470 pacientes con gliomas cerebrales difusos. Estos datos están disponibles para la descarga online en formato NIfTI files (.nii.gz) y son idénticos a los de la última competición realizada en 2021 [37]. Todos los datos del desafío son escaneos de resonancia magnética multi-paramétricos (mpMRI) rutinarios adquiridos clínicamente de pacientes con tumores cerebrales de varias instituciones. El *dataset* incluye imágenes T1, T1 con contraste, T2 y FLAIR, que son diferentes tipos de escaneos MRI que se utilizan para visualizar el tejido cerebral de diferentes maneras.

Además, el *dataset* también incluye máscaras de segmentación de tumores, que son imágenes binarias que muestran las áreas del cerebro donde se encuentra el tumor. Estas máscaras se han creado manualmente por radiólogos expertos, de uno a cuatro evaluadores, siguiendo el mismo protocolo de anotación y sus anotaciones han sido aprobadas por neuroradiólogos con experiencia. Estas máscaras o etiquetas son utilizadas como '*ground truth*' o referencia para evaluar la precisión de los algoritmos de segmentación automática de tumores. Existen cinco tipos de etiquetas: tejido cerebral sano,

área necrótica, área de edema, realce del tumor y área sin realce.

2.4.4. *Beyond the Cranial Vault* (BTCV)

Beyond the Cranial Vault (BTCV) es un *dataset* de imágenes médicas de código abierto para la investigación en el campo de la medicina. Consiste de 50 tomografías computarizadas del abdomen seleccionadas al azar de una combinación de un ensayo de quimioterapia contra el cáncer colorrectal en curso y un estudio retrospectivo de hernia ventral todo ello bajo la supervisión de la Junta de Revisión Institucional (IRB). Los 50 escaneos se capturaron durante la fase de contraste venoso portal con tamaños de volumen variables (512 x 512 x 85 - 512 x 512 x 198) y campo de visión (aprox. 280 x 280 x 280 mm³ - 500 x 500 x 650 mm³). Los datos de registro estándar fueron generados por NiftyReg. Los datos se separan en 30 de entrenamiento y 20 de prueba [38].

Todos los escaneos se adquirieron para la atención clínica de rutina: el conjunto de imágenes abdominales se adquirió de escáneres de tomografía computarizada en el Centro Médico de la Universidad de Vanderbilt (VUMC); las imágenes del cuello uterino se adquirieron de escáneres CT en el Instituto del Cáncer del Centro Médico Erasmus (EMC) en Rotterdam. Evaluadores capacitados etiquetaron manualmente todos los conjuntos de datos y un radiólogo u oncólogo radioterapeuta revisó la precisión de la etiqueta. La división de escaneos en cohortes de entrenamiento y prueba se realizó de forma pseudoaleatoria de modo que los datos de todos los escáneres se incluyeron en ambas cohortes de prueba de entrenamiento.

Las imágenes vienen con etiquetas clínicas, lo que permite a los científicos realizar análisis estadísticos y explorar diferentes patrones en los datos. En concreto y como puede observarse en la Figura 2.21 [39], se etiquetaron manualmente trece órganos abdominales y un radiólogo los verificó volumétricamente usando el software MIPAV, estos órganos incluye: bazo, riñón derecho, riñón izquierdo, vesícula biliar, esófago, hígado, estómago, aorta, vena cava inferior, vena porta y vena esplénica, páncreas, glándula suprarrenal derecha, glándula suprarrenal izquierda.

2.4.5. *Digital database for screening mammography* (DDSM)

Digital Database for Screening Mammography (DDSM) es un conjunto de datos que contiene imágenes digitalizadas de mamografías. Esta base de datos fue desarrollada por el Laboratorio de Imagen Digital de la Universidad de South Florida para apoyar la investigación en detección de cáncer de mama. El *dataset* de DDSM contiene cerca de 2.620 imágenes digitalizadas de mamografías, además de información clínica asociada. Contiene casos

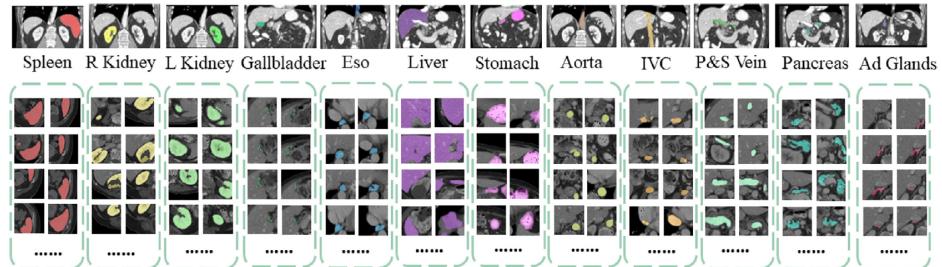


Figura 2.21: Conjunto de datos BTCV con estructuras anatómicas etiquetadas. Fuente: Tang *et al.*

normales, benignos y malignos con información patológica verificada [40].

Las imágenes están divididas en dos categorías principales. Primero, hay un subconjunto de imágenes crudas sin procesar, que contienen los resultados originales de la mamografía. Estas imágenes muestran detalles finos de los tejidos mamarios, como las estructuras de los tejidos normales y anormales. El segundo subconjunto de imágenes contiene imágenes procesadas con un software de análisis de cáncer de mama, que se utiliza para detectar nódulos anormales o otras anomalías en las mamografías. Estas imágenes procesadas también contienen información clínica, incluyendo la edad de la paciente, el diagnóstico y el resultado de la biopsia.

El DDSM es ampliamente utilizado en la investigación en detección de cáncer de mama debido a su gran cantidad de imágenes de alta calidad y a la información clínica asociada a cada paciente. Los investigadores utilizan el DDSM para desarrollar y evaluar algoritmos de detección de cáncer de mama, así como para analizar patrones en las imágenes de mamografía relacionados con el cáncer de mama.

2.4.6. *Ischemic stroke lesion segmentation (ISLES)*

El *Ischemic Stroke Lesion Segmentation* (ISLES) es un conjunto de datos de imágenes médicas que se utiliza para la investigación en segmentación de lesiones cerebro-vasculares isquémicas. El conjunto de datos incluye imágenes de resonancia magnética (MRI) y tomografía computarizada (CT) del cerebro obtenidas de pacientes con accidente cerebrovascular isquémico, junto con máscaras de lesión correspondientes.

La segmentación del infarto en el ictus isquémico es crucial en estadios agudos para guiar la toma de decisiones de tratamiento (reperfundir o no, y tipo de tratamiento) y en estadios subagudos y crónicos para evaluar el resultado de la enfermedad de los pacientes, para su evolución clínica, seguimiento y definir estrategias terapéuticas y de rehabilitación óptimas.

para maximizar las ventanas críticas para la recuperación.

El reto (*challenge*) ISLES es un esfuerzo continuo para desarrollar e identificar métodos de referencia para la segmentación de lesiones de accidentes cerebrovasculares isquémicos agudos y subagudos. La edición de 2022 del reto ISLES proporciona un conjunto de datos que comprende 400 casos de resonancia magnética de múltiples proveedores con alta variabilidad en el tamaño, la cantidad y la ubicación de las lesiones por accidente cerebrovascular. Se divide en un conjunto de datos de entrenamiento de 250 y un conjunto de datos de prueba de 150. Todos los datos de entrenamiento están disponibles públicamente.

La edición de ISLES'22 se diferencia de las ediciones anteriores por centrarse en la delimitación no solo de lesiones de infarto grandes, sino también de múltiples infartos embólicos y/o corticales (generalmente vistos después de la recanalización mecánica), además de por evaluar tanto e imágenes de resonancia magnética post-intervención y por incluir casi 3 veces más datos que ISLES'15 (una edición de desafío anterior con objetivos similares) [41].

2.4.7. *Digital retinal images for vessel extraction (DRIVE)*

El *dataset* de *Digital retinal images for vessel extraction* (DRIVE) es un conjunto de imágenes oftalmológicas digitales que se utilizan para la extracción de vasos retinianos. Consiste en 40 imágenes de retina seleccionadas al azar, de las cuales 33 no muestran ningún signo de retinopatía diabética y 7 muestran signos de retinopatía diabética temprana leve. La población de cribado consistió en 400 sujetos diabéticos entre 25-90 años de edad.

El conjunto de 40 imágenes se ha dividido en un conjunto de entrenamiento y otro de prueba, ambos con 20 imágenes. Para las imágenes de entrenamiento, está disponible una única segmentación manual para cada imagen realizada por un experto oftalmólogo. Para los casos de prueba no hay anotaciones disponibles. La resolución de cada imagen es de 584*565 píxeles con ocho bits por canal de color (3 canales).

La base de datos DRIVE se ha establecido para permitir estudios comparativos sobre la segmentación de los vasos sanguíneos en las imágenes de la retina que se utilizan para el diagnóstico, la detección, el tratamiento y la evaluación de diversas enfermedades cardiovasculares y oftalmológicas, como la diabetes, la hipertensión y la arteriosclerosis y neovascularización coroidea [42].

2.4.8. Otros

Existen otros datasets libres y competiciones/retos donde se proporcionan datasets de imágenes médicas entre los que destacamos: *Segmentation in Chest Radiographs* (SCR) .[43], *Liver tumor segmentation* (LiTS) .[44], *Lung image database consortium image collection* (LIDC-IDRI) .[45], *Open Access Series of Imaging Studies* (OASIS) .[46], *Sunnybrook cardiac data* (SCD) .[47] o SIIM.ISIC reto de clasificación de melanoma [48].

A continuación, la tabla 2.2 resume los principales conjuntos de datos, la modalidad empleada para obtenerlos, la zona objetivo del cuerpo que estudian y la URL donde se pueden descargar.

Datasets	Modalidad	Zona	URL
MSD	MR, CT	Varias	http://medicaldecathlon.com/
BTCV	CT	Varias	https://www.synapse.org/#!Synapse:syn3193805/wiki/217753
BRATS	MR	Cerebro	http://braintumorsegmentation.org/
DDSM	Mamografía	Pecho	http://www.eng.usf.edu/cvprg/Mammography/Database.html
ISLES	MR	Cerebro	http://www.isles-challenge.org/
DRIVE	Fundoscopia	Ojo	https://drive.grand-challenge.org/
SCD	MR	Cardiaco	http://www.cardiacatlas.org/studies/
LiTS	CT	Higado	https://competitions.codalab.org/competitions/17094
SIIM.ISIC	Multiples	Piel	https://www.kaggle.com/c/siim-isic-melanoma-classification

Cuadro 2.2: Resumen *datasets* de imágenes médicas y *urls*

2.5. Preprocesamiento

El preprocesamiento de imágenes médicas consiste en un conjunto de técnicas utilizadas para preparar y mejorar las imágenes médicas antes de su análisis y diagnóstico. El objetivo es mejorar la calidad y la precisión de las imágenes para ayudar a los especialistas a detectar y analizar con mayor precisión las estructuras presentes en las imágenes. El preprocesamiento de imágenes médicas es un paso crucial en el análisis de imágenes médicas y puede ser realizado tanto manualmente como mediante el uso de herramientas y algoritmos de procesamiento de imágenes.

Técnicas de preprocesamiento

El preprocesamiento de imágenes médicas se utiliza en diferentes modalidades de imagen médica, como la tomografía computarizada, la resonancia magnética o la radiografía, para mejorar la calidad y la precisión de las imágenes médicas. Las técnicas de preprocesamiento pueden ayudar a corregir distorsiones, niveles de intensidad, eliminar ruido y mejorar el contraste

de las imágenes entre otras. Todo ello con el fin de facilitar el análisis y el diagnóstico de enfermedades.

Por ejemplo, en el contexto de la tomografía computarizada, las imágenes pueden presentar distorsión geométrica debido a la rotación del paciente durante la adquisición de la imagen. La corrección de distorsión puede ayudar a eliminar esta distorsión y a mejorar la precisión y la calidad de las imágenes. En el contexto de la resonancia magnética, las imágenes pueden presentar ruido debido a la presencia de artefactos electromagnéticos o a la variabilidad en la señal de resonancia magnética; la eliminación de ruido puede ayudar a mejorar la claridad y la nitidez de las imágenes, lo que puede facilitar la identificación de estructuras específicas en las imágenes médicas y el diagnóstico de enfermedades. Otro ejemplo en el contexto de la radiografía es que las imágenes pueden presentar un bajo contraste debido a la absorción de la radiación por parte de los tejidos. El realce de la imagen puede ayudar a mejorar el contraste de las imágenes y a facilitar la identificación de estructuras específicas en las imágenes médicas [49].

Algunas de las técnicas de preprocesamiento más comunes son:

- Corrección de distorsión: se utiliza para corregir cualquier distorsión geométrica presente en las imágenes, como la distorsión de perspectiva o la deformación del campo de visión. La corrección de distorsión ayuda a mejorar la precisión y la calidad de las imágenes médicas, lo que puede facilitar el análisis y el diagnóstico de enfermedades.
- Eliminación de ruido: se utiliza para eliminar cualquier ruido presente en las imágenes médicas, como el ruido aleatorio o el ruido de alta frecuencia. La eliminación de ruido puede ayudar a mejorar la claridad y la nitidez de las imágenes, lo que puede facilitar la identificación de estructuras específicas en las imágenes médicas.
- Realce de la imagen: se utiliza para mejorar la visibilidad de ciertas estructuras en las imágenes médicas, como órganos, tejidos o vasos sanguíneos. El realce de la imagen se puede realizar mediante diferentes técnicas, como la equalización de histograma, la filtración de mediana o la técnica de la máscara de enfoque. El realce de la imagen puede ayudar a mejorar la claridad y la contraste de las imágenes.
- La interpolación de la imagen: se utiliza para aumentar la resolución de las imágenes médicas, lo que puede mejorar la precisión del análisis y el diagnóstico.
- La normalización de la imagen: se utiliza para ajustar el brillo y el contraste de las imágenes médicas, lo que puede mejorar la claridad y la nitidez de las mismas.

- El registrado de imágenes: se utiliza para alinear diferentes imágenes médicas tomadas en diferentes momentos o desde diferentes ángulos, lo que puede facilitar la comparación y el análisis de las imágenes.

Preprocesamiento orientado a redes neuronales profundas

Para nuestro caso de análisis de segmentación de imágenes con redes neuronales profundas, el preprocesamiento de imágenes médicas es una etapa fundamental en el entrenamiento de algoritmos y modelos. Consiste en una serie de pasos que se llevan a cabo antes de entrenar el algoritmo para limpiar, normalizar y preparar las imágenes para su uso en el entrenamiento del modelo lo que aumenta la precisión y el rendimiento del algoritmo de segmentación de imágenes.

Algunos de los pasos comunes del preprocesamiento de imágenes médicas para el entrenamiento de algoritmos de segmentación de imágenes con redes neuronales profundas incluyen [50]:

- Limpieza de imágenes: El primer paso en el preprocesamiento de imágenes médicas es la limpieza de imágenes. En esta etapa se eliminan ruido, artefactos y otros elementos indeseables de las imágenes que pueda interferir en el procesamiento posterior de las imágenes. Esto puede incluir el uso de filtros para eliminar el ruido aleatorio o morfológico para eliminar objetos no deseados y destacar los detalles importantes en las imágenes.
- Corrección de errores: En esta etapa se corrigen errores comunes en las imágenes médicas, como el desenfoque o la distorsión. Esto se puede realizar mediante el uso de técnicas como la deconvolución o el realce de detalles.
- Normalización de imágenes: Se lleva a cabo la normalización de las imágenes médicas para garantizar que estén en un formato adecuado para su procesamiento posterior. Esto puede incluir la corrección de intensidad y la estandarización de tamaño para asegurar que todas las imágenes estén en el mismo rango de valores y tamaño.
- Registrado y alineación: Se realiza el registrado y alineación de las imágenes médicas para combinar diferentes modalidades y vistas de un mismo paciente. Esto es importante para obtener una visión más completa del paciente y para facilitar la comparación entre diferentes exámenes médicos.
- Preprocesamiento de etiquetas: El preprocesamiento de las etiquetas o máscaras de segmentación se ajustan y refinen las máscaras de seg-

mentación para asegurarse de que sean precisas y coherentes con las imágenes.

- Aumento de datos (*Data Augmentation*): En esta etapa se generan nuevas imágenes a partir de las imágenes existentes mediante técnicas como el corte, el giro, el cambio de escala y otros para aumentar el tamaño y la diversidad del *dataset* de entrenamiento. Veremos con mas detalle en el siguiente apartado esta etapa.

2.6. Aumento de datos (*Data Augmentation*)

El Aumento de datos (*Data Augmentation*) es una técnica utilizada en el aprendizaje automático para aumentar el tamaño y diversidad del conjunto de datos (*dataset*) utilizado para entrenar un modelo. Se necesita una gran cantidad de datos etiquetados para obtener una alta precisión de los resultados. Estos datos etiquetados no están siempre disponibles y la realización por parte de un experto es un proceso muy costoso en tiempo y dinero. A través de un proceso de aumento de datos se crean nuevas imágenes a partir de los datasets de entrenamiento aplicando transformaciones aleatorias como rotaciones, traslaciones, escalado, añadiendo ruido gaussiano u otros métodos que son descritos por Pérez *et al.* [51].

El objetivo del *data augmentation* en el preprocesamiento de imágenes médicas es generar un *dataset* más grande y variado para entrenar una red neuronal profunda, lo que puede mejorar el rendimiento del modelo en términos de precisión y generalización. Además también ayuda a reducir el *overfitting* (sobreajuste) en el entrenamiento del modelo, lo que puede mejorar su capacidad de generalización a datos desconocidos.

Sin embargo, es importante tener en cuenta que el *data augmentation* también puede introducir distorsiones en las imágenes que pueden afectar negativamente al rendimiento del modelo, por lo que es importante ajustar adecuadamente las transformaciones utilizadas para evitar este problema.

Las transformaciones más comunes que se suelen usar son:

Rotación

La rotación consiste en girar una imagen un cierto ángulo aleatorio alrededor de un punto central para generar una nueva imagen. La rotación se puede realizar mediante el uso de una matriz de rotación que contiene los coeficientes de rotación en función del ángulo de rotación.

Por ejemplo, si queremos rotar una imagen un ángulo aleatorio de 30 grados en sentido antihorario alrededor del centro de la imagen, podríamos

utilizar la siguiente matriz de rotación:

$$\begin{bmatrix} \cos(30) & -\sin(30) & 0 \\ \sin(30) & \cos(30) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

La matriz de rotación se multiplica por cada píxel de la imagen para calcular su nueva posición en la imagen rotada.

La rotación permite simular diferentes posiciones del paciente durante un examen médico y con ello mejorar la capacidad de la red neuronal para generalizar a diferentes orientaciones de las imágenes. Por ejemplo, si el *dataset* original contiene imágenes de un paciente en una orientación determinada, la rotación puede generar nuevas imágenes del mismo paciente en diferentes orientaciones, lo que permite entrenar una red neuronal que sea capaz de generalizar a diferentes orientaciones de las imágenes.

Además, la rotación también puede ser útil para aumentar la diversidad del *dataset* y reducir el *overfitting* (sobreajuste) en el entrenamiento de la red neuronal. Al rotar las imágenes de manera aleatoria, se generan nuevas imágenes que no son exactamente iguales a las originales, lo que puede ayudar a evitar que el modelo se sobreajuste a patrones específicos en el *dataset* y mejorar su capacidad de generalización a datos desconocidos.

Sin embargo, es importante tener en cuenta que la rotación también puede introducir distorsiones en las imágenes que pueden afectar negativamente al rendimiento del modelo. Por ejemplo, si se rota una imagen en exceso, es posible que se corten partes importantes de la imagen y que se pierda información valiosa el entrenamiento de la red neuronal. Por lo tanto, es importante ajustar adecuadamente el ángulo de rotación utilizado para evitar este problema. Además, también es importante utilizar un algoritmo de interpolación adecuado para suavizar los bordes de la imagen rotada y evitar que se produzcan artefactos visuales en la imagen resultante.

Traslación

Consiste en mover una imagen en cualquiera de los ejes (horizontal, vertical o diagonal) una distancia aleatoria para generar una nueva imagen. La traslación se puede realizar mediante el uso de una matriz de traslación que contiene los desplazamientos en cada eje. La matriz de traslación se multiplica por cada píxel de la imagen para calcular su nueva posición en la imagen trasladada.

Por ejemplo, si queremos trasladar una imagen en el eje horizontal una distancia aleatoria de 4 píxeles y en el eje vertical una distancia aleatoria de 2 píxeles, podríamos utilizar la siguiente matriz de traslación:

$$\begin{bmatrix} 1 & 0 & 4 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix}$$

Al igual que ocurría con la rotación, la traslación genera nuevas imágenes que no son exactamente iguales a las originales, lo que puede ayudar a evitar que el modelo se sobreajuste a patrones específicos en el *dataset* y mejorar su capacidad de generalización a datos desconocidos. Igualmente, es importante tener en cuenta que la traslación también puede introducir distorsiones que pueden afectar negativamente. Por ejemplo, si se traslada una imagen en exceso, es posible que se corten partes importantes de la imagen y que se pierda información de interés.

Escalado

En esta transformación se cambia el tamaño de las imágenes médicas mediante el uso de un factor de escalado aleatorio para generar nuevas imágenes. El escalado se puede realizar mediante el uso de una matriz de escalado que contiene los factores de escalado en cada eje.

Por ejemplo, si queremos escalar una imagen un factor aleatorio de 1.5 en el eje horizontal y un factor aleatorio de 0.5 en el eje vertical, podríamos utilizar la siguiente matriz de escalado:

$$\begin{bmatrix} 1.5 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

El escalado permite simular diferentes tamaños de las estructuras y objetos en las imágenes médicas y mejorar la capacidad de la red neuronal para generalizar a diferentes tamaños de las imágenes. Por ejemplo, si el *dataset* original contiene imágenes de un paciente con un tamaño determinado, el escalado puede generar nuevas imágenes del mismo paciente en diferentes tamaños, lo que permite entrenar una red neuronal que sea capaz de generalizar a diferentes tamaños de las imágenes.

Por contra, si se escala una imagen en exceso, es posible que se pierda resolución y detalle en la imagen y que se reduzca la cantidad de información valiosa para el entrenamiento de la red neuronal.

Reflejo

Esta transformación consiste en reflejar una imagen en un eje horizontal o vertical aleatorio para generar una nueva imagen. El reflejo se puede realizar

mediante el uso de una matriz de reflejo que contiene los coeficientes de reflejo en función del eje de reflejo.

El reflejo puede ser útil para simular diferentes orientaciones de las estructuras y objetos en las imágenes médicas y para mejorar la capacidad de la red neuronal para generalizar a diferentes orientaciones de las imágenes así como aumentar la diversidad del *dataset* y reducir el *overfitting* en el entrenamiento de la red neuronal.

Debemos tener en cuenta los problemas que puede introducir el reflejo como por ejemplo si se refleja una imagen en un eje incorrecto es posible que se altere la orientación de las estructuras y objetos en la imagen de manera no deseada.

Cambio de contraste

Esta transformación consiste en modificar el contraste de una imagen mediante el uso de un factor de contraste aleatorio para generar una nueva imagen. El cambio de contraste se puede realizar mediante el uso de una fórmula que mapea los valores de los píxeles en la imagen original a nuevos valores en la imagen con el contraste modificado.

Por ejemplo, si queremos aumentar el contraste de una imagen mediante un factor aleatorio de 1.5, podríamos utilizar la siguiente fórmula:

```
nuevo_contraste = contraste_pixel * factor_aleatorio
```

(Ej. Factor_aleatorio=1.5)

El cambio de contraste permite simular diferentes condiciones de iluminación y ajustes de contraste en las imágenes médicas así como generalizar a diferentes niveles de contraste, aumentar la diversidad del *dataset* y reducir el *overfitting*.

Por otro lado, hay que tener en cuenta que si se modifica el contraste en exceso, es posible que se pierda información valiosa en la imagen y que se dificulte la detección de las estructuras y objetos en la imagen por parte de la red neuronal.

Warp

Warp consiste en deformar una imagen mediante el uso de una matriz de deformación aleatoria para generar una nueva imagen. El *warp* se puede realizar mediante el uso de una matriz de transformación que contiene los coeficientes de deformación en cada eje.

El *warp* puede ser útil para simular diferentes deformaciones y movimientos de las estructuras y objetos de las imágenes médicas mejorando la generalización, reduciendo el *overfitting* y aumentando el *dataset*.

Como parte negativa, si se deforma una imagen en exceso, es posible que se pierda resolución y detalle en la imagen y que se reduzca la cantidad de información para el entrenamiento.

Superposición

La superposición consiste en superponer una imagen de manera aleatoria sobre otra imagen pudiéndose realizar mediante el uso de una fórmula que combine los valores de los píxeles en ambas imágenes para calcular los valores de los píxeles en la imagen resultante.

Como en transformaciones anteriores, al aumentar la cantidad de imágenes del *dataset* introduce mejoras en la capacidad de generalización, diversidad y reducir el sobreajuste.

Ruido

En esta transformación se añade ruido aleatorio a las imágenes médicas lo cual puede ser útil para simular diferentes condiciones de adquisición de las imágenes médicas y para mejorar la capacidad de la red neuronal para generalizar, tener mayor diversidad y reducir el sobreajuste.

2.7. Métricas de evaluación

Las métricas de evaluación son herramientas que se utilizan para medir el desempeño y la precisión de los algoritmos de segmentación de imágenes médicas. Debido a la variedad de arquitecturas de segmentación existentes resulta difícil comparar el rendimiento de diferentes algoritmos, porque la mayoría de los algoritmos se evalúan con diferentes conjuntos de datos y se documentan usando diferentes métricas. Las métricas de evaluación permiten comparar diferentes algoritmos de segmentación de imágenes y determinar cuál es el mejor en función de ciertos criterios.

La mayoría de los investigadores eligen las métricas de evaluación arbitrariamente o de acuerdo con su popularidad. Una métrica mal definida puede conducir a conclusiones imprecisas, como seleccionar modelos subóptimos al comparar el rendimiento de los clasificadores [52].

Para tratar de solventar este problema, por un lado se organizan las competiciones o retos donde el *dataset* es igual para todas las arquitecturas

propuestas. Por otro lado, evaluar la calidad de un algoritmo requiere un indicador objetivo correcto. Se suele usar las segmentaciones manuales de los expertos médicos como estándar de oro (*GT, Ground Truth*).

La evaluación de segmentación es la tarea de comparar dos segmentaciones midiendo la distancia o similitud entre ellas, donde una es la segmentación a evaluar y la otra es la segmentación *ground truth* correspondiente.

Existen diferentes aspectos de calidad en la segmentación de imágenes médicas 3D según los cuales se pueden definir tipos de errores de segmentación. Se espera que las métricas indiquen algunos o todos estos errores, según los datos y la tarea de segmentación. Sobre la base de cuatro tipos básicos de errores (regiones añadidas, fondo añadido, agujeros interiores y agujeros de borde), Shi *et al.* [53] describieron cuatro tipos de errores de segmentación de imágenes que son la cantidad (número de objetos segmentados), el área de los objetos segmentados, el contorno (grado de coincidencia de los límites) y el contenido (existencia de agujeros interiores y agujeros de contorno en la región segmentada).

Fenster *et al.* [54] clasificaron los requisitos de la evaluación de la segmentación médica en exactitud, precisión y eficiente. La exactitud es el grado en que los resultados de la segmentación concuerdan con la segmentación real, la precisión se considera como medida de repetibilidad y la eficiencia se relaciona principalmente con el tiempo. Bajo la primera categoría (exactitud), mencionaron dos aspectos de calidad como la delineación del límite (contorno) y el tamaño (volumen del objeto segmentado). La alineación, que denota la posición general del objeto segmentado, es otro aspecto cualitativo, que puede ser más importante que el tamaño y el contorno cuando los objetos segmentados son muy pequeños.

Taha *et al.* [55] proporciona una descripción general de 20 métricas de evaluación para la segmentación de volumen, seleccionadas en base a una revisión de la literatura. Identifican los casos en los que se han utilizado definiciones inconsistentes de las métricas en la literatura y se sugieren definiciones unificadas.

A continuación, la tabla 2.3 presenta las 20 métricas definidas en el artículo de Taha *et al.* de las cuales describiremos en algo más de profundidad aquellas más usadas.

2.7.1. Cardinalidades básicas (Matriz de confusión)

La matriz de confusión es una herramienta que permite visualizar el desempeño de un algoritmo de aprendizaje supervisado. Como se muestra en la Figura 2.22 [56], cada columna de la matriz representa el número de predicciones de cada clase, mientras que cada fila representa a las instancias

Métrica	Simb.	Categoría
Dice (=F1-Medida)	DICE	Superposición
Índice de Jaccard	JAC	Superposición
Tasa de verdaderos positivos (Sensibilidad, Recall)	TPR	Superposición
Tasa de verdaderos negativos (Especificidad)	TNR	Superposición
Tasa de falsos positivos (=1-Especificidad, Fallout)	FPR	Superposición
Tasa de falsos negativos (=1-Sensibilidad)	FNR	Superposición
F-Medida (F1-Medida=Dice)	FMS	Superposición
Error de consistencia global	GCE	Superposición
Similitud volumétrica	VS	Volumen
Índice de Rand	RI	Contar parejas
Índice de Rand ajustado	ARI	Contar parejas
Información mutua	MI	Información
Variación de información	VOI	Información
Correlación entre clases	ICC	Probabilística
Distancia probabilística	PBD	Probabilística
Cohens kappa	KAP	Probabilística
Área bajo la curva ROC	AUC	Probabilística
Distancia de Hausdorff	HD	Distancia espacial
Distancia promedio	AVD	Distancia espacial
Distancia de Mahalanobis	MHD	Distancia espacial

Cuadro 2.3: Resumen de métricas de evaluación

en la clase real. La matriz representa las predicciones del modelo en relación a los resultados reales de las etiquetas de clase. En términos prácticos nos permite ver qué tipos de aciertos y errores está teniendo nuestro modelo a la hora de pasar por el proceso de aprendizaje con los datos.

La matriz de confusión se utiliza comúnmente en tareas de clasificación, pero también se usan en tareas de segmentación y detección de objetos. Para las tareas de segmentación, a menudo se las denomina métricas basadas en superposición. Cada una de las métricas cubre propiedades específicas.

Está formada por cuatro elementos básicos:

- Verdadero positivo (TP o *True Positive*): Se refiere a la cantidad de casos correctamente clasificados como positivos. Es decir, el valor real

		VALORES PREDICCIÓN	
		Positive	Negative
VALORES REALES	Positive	True positive (TP)	False negative (FN)
	Negative	False positive (FP)	True negative (TN)

Figura 2.22: Matriz de confusión. Fuente: Reinke *et al.* (modificada)

es positivo y la prueba predijo también que era positivo. Por ejemplo, una persona está enferma y la prueba así lo demuestra.

- Falso positivo (FP o *False Positive*): Se refiere a los casos incorrectamente clasificados como positivos. Es decir, el valor real es negativo, y la prueba predijo que el resultado es positivo. Por ejemplo, una persona no está enferma, pero la prueba nos dice de manera incorrecta que sílo está. Esto es lo que en estadística se conoce como error tipo I.
- Verdadero negativo (TN o *True Negative*): Se refiere a la cantidad de casos correctamente clasificados como negativos. Es decir, el valor real es negativo y la prueba predijo también que el resultado era negativo. Por ejemplo, una persona no está enferma y la prueba así lo demuestra.
- Falso negativo (FN o *False Negative*): Se refiere a los casos incorrectamente clasificados como negativos. Es decir, el valor real es positivo, y la prueba predijo que el resultado es negativo. Por ejemplo, una persona está enferma, pero la prueba dice de manera incorrecta que no lo está. Esto es lo que en estadística se conoce como error tipo II.

A partir de estas 4 opciones surgen las métricas de la matriz de confusión que veremos a continuación. Para ayudar a la comprensión visual de las métricas, definimos en la Figura 2.23 un ejemplo esquemático de espacio de muestras y su clasificación que será utilizado posteriormente:

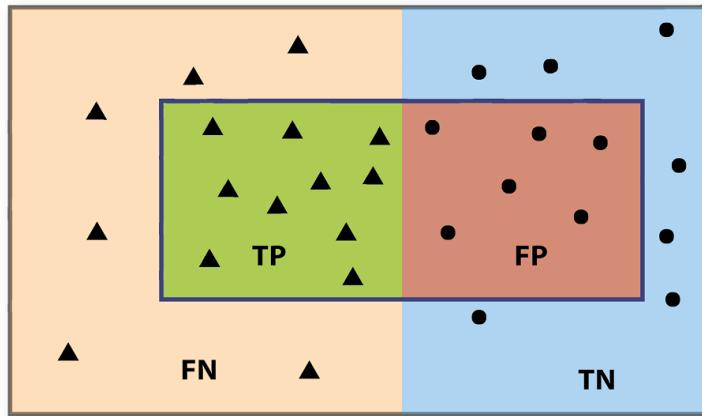


Figura 2.23: Ejemplo esquemático de la matriz de confusión. Fuente: Reinke *et al.*

Sensibilidad (*Recall o Sensitivity*)

La sensibilidad, también conocida como *recall* o Tasa de verdaderos positivos (TPR), es una métrica que mide la capacidad de un modelo de aprendizaje automático para identificar correctamente las instancias positivas en un conjunto de datos.

Se calcula como la relación entre el número de verdaderos positivos (aquellos correctamente identificados como positivos) y el número total de casos positivos reales en los datos. En tareas de segmentación 3D mide la porción de véxeles positivos en el *ground truth* que también se identifican como positivos por la segmentación que se está evaluando.

Matemáticamente, se define como

$$\text{Sensibilidad} = \frac{TP}{TP + FN} \quad (2.1)$$

La sensibilidad es especialmente importante en tareas en las que es deseable minimizar el número de falsos negativos, como en la detección de enfermedades.

A modo visual teniendo en cuenta la Figura 2.23 del ejemplo esquemático, podemos definir visualmente la sensibilidad como muestra la Figura 2.24.

Especificidad (*Specificity*)

En analogía con la Sensibilidad para los positivos, la Especificidad (también conocida como Selectividad o Tasa de verdaderos negativos TNR) se

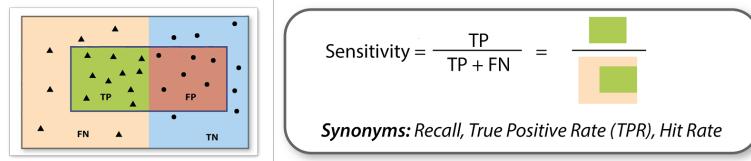


Figura 2.24: Sensibilidad. Fuente: Reinke *et al.* (modificada)

centra en los casos negativos calculando la fracción de negativos que se detectaron correctamente como tal.

Se calcula como la relación entre el número de verdaderos negativos (aquellos correctamente identificados como negativos) y el número total de casos negativos reales en los datos. En tareas de segmentación 3D mide la porción de vértices negativos (fondo) en la segmentación del *ground truth* que también son identificados como negativos por la segmentación que se está evaluando.

Matemáticamente, se define como

$$\text{Sensibilidad} = \frac{TN}{TN + FP} \quad (2.2)$$

La especificidad es especialmente importante en tareas en las que es deseable minimizar el número de falsos positivos.

A modo visual teniendo en cuenta la Figura 2.23 del ejemplo esquemático, podemos definir visualmente la especificidad como muestra la Figura 2.25.



Figura 2.25: Especificidad. Fuente: Reinke *et al.* (modificada)

La especificidad y la sensibilidad son métricas complementarias que se utilizan juntas para evaluar la precisión de un modelo de aprendizaje automático en la tarea de clasificación binaria. Sin embargo, ni la sensibilidad ni la especificidad son comunes como medidas de evaluación de la segmentación de imágenes médicas debido a su sensibilidad al tamaño de los segmentos, es decir, penalizan más los errores en los segmentos pequeños que en los grandes.

Precisión

La presición o valor predictivo positivo (PPV) es una métrica que mide la capacidad de un modelo de aprendizaje automático para identificar correctamente las instancias positivas en un conjunto de datos.

En contraposición con la sensibilidad, la presición divide el TP por el número total de casos positivos pronosticados, con el objetivo de representar la probabilidad de una predicción positiva correspondiente a un positivo real. Un valor de 1 implicaría que todos los casos predichos positivos son en realidad positivos, pero aún podría ser el caso que se perdieron los casos positivos.

Matemáticamente, se define como

$$\text{Presicion} = \frac{TP}{TP + FP} \quad (2.3)$$

A modo visual teniendo en cuenta la Figura 2.23 del ejemplo esquemático, podemos definir visualmente la presición como muestra la Figura 2.26.

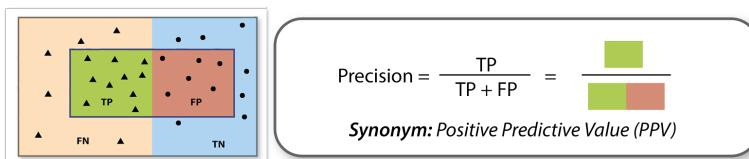


Figura 2.26: Precisión. Fuente: Reinke *et al.* (modificada)

La presición es especialmente importante en tareas en las que es deseable minimizar el número de falsos positivos. No se usa comúnmente en la validación de imágenes médicas, pero se usa para calcular la medida F (*F-score*).

El informe de una sola métrica como Sensibilidad, Precisión o Especificidad puede ser muy engañoso porque, por ejemplo, los clasificadores no informativos pueden alcanzar valores altos sobre clases desequilibradas. Es por ello que se suelen usar otras métricas como las definidas a continuación.

2.7.2. Coeficiente de Similaridad de Dice o *F1-Score*

El Coeficiente de Similaridad de Dice (DICE o DSC), también llamado índice de superposición, es la métrica más utilizada para validar segmentaciones de volumen médico. Se utiliza para medir la similitud entre dos regiones de una imagen; una región segmentada y una región de referencia. Esta métrica se basa en el cálculo de la cantidad de similitud entre dos segmentaciones.

Además de la comparación directa entre segmentaciones automáticas y reales, es común usar DICE para medir la reproducibilidad (repetibilidad).

Esta métrica se calcula como la relación entre la área común entre las dos regiones y el área total de la región de referencia. Matemáticamente se define como

$$DICE = 2 \frac{|A \cap B|}{|A| + |B|} \quad (2.4)$$

El DSC es idéntico al *F1 Score*. Relacionándolos con las métricas de la matriz de confusión se demuestra que

$$DICE = \frac{2TP}{2TP + FP + FN} \quad (2.5)$$

El área de intersección de la región de referencia y la región segmentada representa la cantidad de píxeles correctamente identificados como pertenecientes a la estructura de interés. Por lo tanto, este valor es equivalente a los verdaderos positivos. Mientras que el área de referencia y la área segmentada representan la cantidad total de píxeles identificados como pertenecientes a la estructura de interés, incluyendo tanto los verdaderos positivos como los falsos positivos.

El resultado de esta fórmula es un valor entre 0 y 1, donde un valor cercano a 1 indica una segmentación precisa y un valor bajo indica una precisión de segmentación pobre.

El DICE (DSC o *F1-Score*), supera el problema de representar la media armónica de Precisión y Sensibilidad y, por lo tanto, penalizar valores extremos de cualquiera de las métricas a la vez que es relativamente robusto frente a conjuntos de datos desequilibrados.

Visualmente lo podemos ver representado en la Figura 2.27

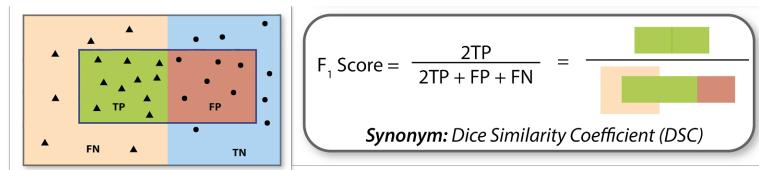


Figura 2.27: DICE (DSC o F1-Score). Fuente: Reinke *et al.* (modificada)

2.7.3. Índice de Jaccard o IoU

El Índice Jaccard (JAC) es una métrica ampliamente utilizada en la segmentación de imágenes médicas para evaluar la similitud entre dos conjuntos

de datos, se utiliza para comparar la segmentación realizada por un algoritmo con la segmentación manual realizada por un experto, con el objetivo de evaluar la precisión del algoritmo.

Se calcula como la relación entre el área de la intersección entre las dos regiones y el área total de la unión de ambas regiones. En otras palabras, se mide la proporción de superposición entre la región segmentada y la región objetivo. El índice de Jaccard también es conocido como *Intersection over Union* (IoU). La Figura 2.28 muestra visualmente como se calcula del DICE y el índice de Jaccard. Matemáticamente se define como

$$JAC = \frac{|A \cap B|}{|A \cup B|} \quad (2.6)$$

Relacionándolo con las métricas de la matriz de confusión se demuestra que

$$JAC = \frac{TP}{TP + FP + FN} \quad (2.7)$$

Se observa que JAC siempre es mayor que DICE excepto en los extremos 0,1 donde son iguales. Además, las dos métricas están relacionadas según la fórmula matemática

$$JAC = \frac{DICE}{2 - DICE} \quad (2.8)$$

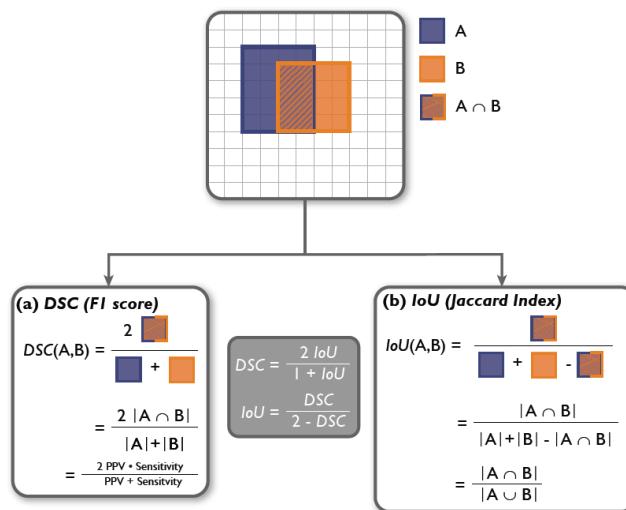


Figura 2.28: DICE y Jaccard Index. Fuente: Reinke *et al.*

El Índice de Jaccard tiene una mayor precisión y es capaz de identificar los cambios en la estructura de los conjuntos. Sin embargo, el Índice de Jaccard es más propenso a los errores porque es más sensible a la presencia de elementos únicos en los conjuntos.

En cualquier caso, tanto el DSC como JAC miden los mismos aspectos y proporcionan la misma clasificación del sistema. Por lo tanto, se encuentran con problemas comunes como por ejemplo el tamaño pequeño de las estructuras en relación con el tamaño del píxel.

La segmentación de pequeñas estructuras, como el cerebro, imágenes de lesiones o células a bajo aumento, es esencial para muchas aplicaciones de procesamiento de imágenes. En estos casos, el DSC o JAC pueden no ser métricas apropiadas. De hecho, una diferencia de un solo píxel entre dos predicciones puede tener un gran impacto en los valores de la métrica. Dado que los contornos correctos (por ejemplo, de patologías) a menudo se desconocen y teniendo en cuenta la variabilidad entre observadores potencialmente alta relacionada con la generación de anotaciones de referencia, es por lo general, no es deseable que unos pocos píxeles influyan tanto en las métricas. Este problema es particularmente amplificado en casos de gran variabilidad de tamaños de estructuras.

La Figura 2.29 muestra el efecto del tamaño de la estructura en el Coeficiente DICE (DSC). Las predicciones de dos algoritmos (Predicción 1 y Predicción 2) difieren en un solo píxel. En el caso de la estructura pequeña (fila de abajo), esto tiene un efecto sustancial en el valor métrico correspondiente de DSC (similar efecto para la Intersección sobre la Unión (IoU)). Los efectos son considerablemente menores para otro tipos de métricas basadas en límites como Distancia Hausdorff (HD) que veremos a continuación

2.7.4. Distancia Hausdorff (HD)

Las métricas basadas en la distancia espacial se utilizan ampliamente en la evaluación de la segmentación de imágenes como medidas de disimilitud. Se recomiendan cuando la precisión general de la segmentación, por ejemplo, la delineación de los límites (contorno) de la segmentación es importante. Existen varias métricas basadas en distancia espacial como la distancia de Hausdorff, la distancia promedio y la distancia de Mahalanobis. Nos centraremos en la Distancia Hausdorff.

Las medidas basadas en la distancia tienen en cuenta la posición espacial de los vértices. Todas las distancias están en vértice, lo que significa que no se tiene en cuenta el tamaño del vértice.

Las métricas basadas en la distancia operan exclusivamente en los verdaderos positivos (TP) y calcular una o varias distancias entre la referencia

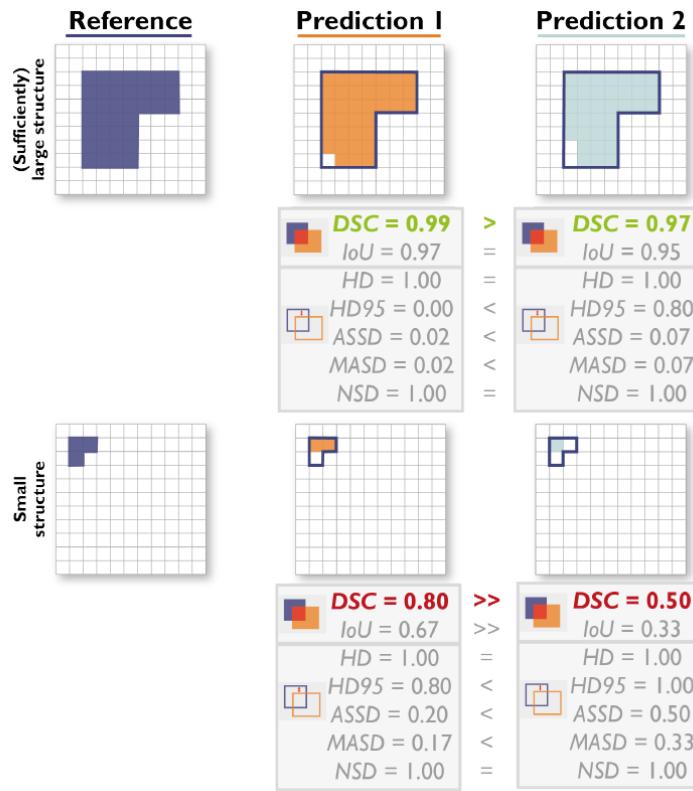


Figura 2.29: Problema tamaño pequeño estructura para DICE y JAC. Fuente: Reinke *et al.*

y la predicción. Suelen ser métricas basadas en límites que se centran en la evaluación de la precisión de los límites de los objetos. Las más usadas son la HD y la HD95 que calcula el percentil 95 en lugar del máximo. El HD calcula el máximo de todas las distancias más cortas para todos los puntos de un objeto límite con el otro, por lo que también se conoce como la Distancia Superficial Simétrica Máxima. El HD95 calcula el percentil del 95 en lugar del máximo, por lo que ignora valores atípicos.

La Figura 2.30 muestra como se calcula la distancia Hausdorff y la HD95 y muestra como la HD95 ignora valores atípicos.

Matemáticamente, la distancia de Hausdorff (HD) entre dos conjuntos de puntos finitos A y B se define por

$$HD(A, B) = \max(h(A, B), h(B, A)) \quad (2.9)$$

donde $h(A, B)$ se denomina distancia de Hausdorff dirigida y está dada por

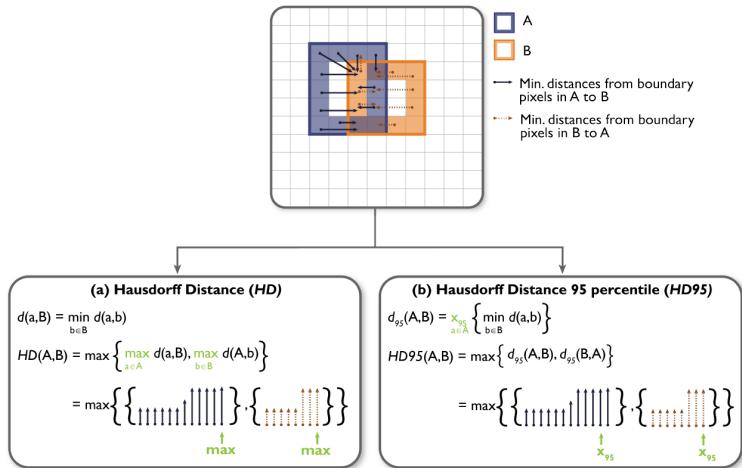


Figura 2.30: Distancia Hausdorff. Fuente: Reinke *et al.*

$$h(A, B) = \max_{a \in A} \min_{b \in B} \|a - b\| \quad (2.10)$$

donde $\|a - b\|$ es alguna norma, por ejemplo la distancia euclídea.

El HD es generalmente sensible a los valores atípicos. Debido a que el ruido y los valores atípicos son comunes en las segmentaciones médicas, no se recomienda usar el HD directamente. Sin embargo, el método de cuantiles es una forma de manejar los valores atípicos. De acuerdo con el método del cuantil de Hausdorff, el HD se define como el q -ésimo cuantil de las distancias en lugar del máximo, por lo que se excluyen posibles valores atípicos, donde q se selecciona en función de la aplicación y la naturaleza de los conjuntos de puntos medidos.

Mientras que las métricas basadas en solapamiento y las métricas basadas en distancia son las métricas estándar utilizadas por la comunidad de visión por computadora en general, las aplicaciones biomédicas a menudo tienen requisitos específicos del dominio. En la imagen médica, por ejemplo, el volumen real de un objeto puede ser de interés especial (por ejemplo, el volumen del tumor). En este caso, se pueden calcular métricas de volumen, como el Error de Volumen Absoluto o Relativo y la Diferencia de Volumen Relativa Simétrica. Sin embargo, son menos comunes que las métricas de solapamiento, ya que la ubicación de los objetos no se considera en absoluto. Si el centro de la estructura o la línea central es de interés especial (por ejemplo, en células o vasos), entran en juego las métricas de conectividad, que miden el acuerdo de la línea central entre dos objetos. Esto es de interés especial si hay objetos lineales o tubulares en un conjunto de datos, por ejemplo, en el análisis vascular cerebral. En estos casos, la sobresegmentación

o subsegmentación no son de interés especial, sin embargo, la conectividad o la topología de la red son de importancia específica.

2.7.5. Pautas de selección de métricas

Además de las métricas vistas en detalle en los anteriores apartados, existen muchas más métricas como las recogidas en la Tabla 2.3 y muchas otras. Basándonos en el trabajo de Taha *et al.* [55] pretendemos dar unas pequeñas pautas acerca de que métricas sería más conveniente usar para cada casuística y del objetivo de segmentación de imágenes médicas.

- *Outliers* (Valores Atípicos): En la segmentación los valores atípicos son regiones erróneamente segmentadas relativamente pequeñas fuera (normalmente lejos) de la segmentación. Las métricas sensibles a los valores atípicos los penalizan demasiado. Cuando los valores atípicos no son perjudiciales, se deben evitar las métricas sensibles a ellos como HD.
- Segmento pequeño: Cuando el tamaño de un segmento es significativamente más pequeño que el fondo de manera que es comparable en magnitud con la expectativa del error de alineación, entonces todas las métricas basadas en las cuatro cardinalidades de solapamiento (TP, TN, FP, FN), por ejemplo, las métricas basadas en solapamiento, así como las métricas basadas en volumen (VS), no son adecuadas. Los segmentos pequeños son aquellos con al menos una dimensión significativamente más pequeña que la dimensión correspondiente de la malla en la que se define la imagen (por ejemplo, menos del 5 % de la dimensión correspondiente de la malla). En este caso, se recomiendan las métricas basadas en distancia.
- Borde complejo: Mientras algunos segmentos tienen formas casi redondas o bordes suaves, hay otros que tienen un borde complejo con forma no regular, que se denotan por esta propiedad. Las métricas sensibles a las posiciones de los puntos (por ejemplo, HD y AVD) son más adecuadas para evaluar esta segmentación que otras. Las métricas basadas en volumen deben evitarse en este caso.
- Densidades bajas: Algunos algoritmos producen segmentaciones que tienen una buena calidad en términos de contorno y alineación, pero los segmentos no son sólidos, sino que tienen una densidad más baja, por ejemplo, debido a numerosos agujeros pequeños. Todas las métricas basadas en las cuatro cardinalidades son sensibles a la densidad de los segmentos. Penalizan la baja densidad y, por lo tanto, deben evitarse en casos donde la baja densidad no es perjudicial. En estos casos, las métricas basadas en distancia (HD, AVD y MHD) son buenas opciones.

- Calidad baja de segmentación: Esta propiedad describe segmentaciones que en general tienen una baja calidad, es decir, se puede asumir que los segmentos en general tienen una superposición baja con los segmentos correspondientes en la segmentación de verdad absoluta. Cuando la superposición es baja, las métricas basadas en distancia son más capaces de diferenciar entre las calidades de segmentación que las métricas basadas en volumen. Se debe evitar la similitud volumétrica VS.
- El contorno es importante: Dependiendo de la tarea individual, el contorno puede ser relevante, es decir, los algoritmos de segmentación deben proporcionar segmentos con una delimitación de límite lo más exacta posible. Las métricas sensibles a las posiciones de los puntos (por ejemplo, HD y AVD) son más adecuadas para evaluar esta segmentación que otras. Las métricas basadas en volumen deben evitarse en este caso.
- La alineación es importante: Cuando el requisito es la ubicación (alineación general) del segmento en lugar de la delimitación de límite. En este caso, las métricas basadas en volumen no son una buena opción.
- El *recall* es importante: En algunos casos es un requisito importante que la región segmentada incluya al menos todo el segmento verdadero, independientemente de incluir partes de la región falsa. Obviamente, en este caso la delimitación de límite es de menos interés y los algoritmos deben maximizar el *recall*. Las métricas que recompensan el *recall* son la información mutua MI y la tasa de verdaderos positivos TPR.
- El volumen es importante: A veces la magnitud de la región segmentada es de más importancia que el límite y la alineación. Aquí, los algoritmos deben segmentar la región para tener un volumen lo más cercano posible al del verdadero segmento. Se recomienda la similitud volumétrica VS.
- Solo forma general y alineación: El límite exacto y la alta superposición no son requisitos siempre. Dependiendo del objetivo, a veces la forma general y la alineación (ubicación) son suficientes, por ejemplo, cuando el requisito es identificar lesiones y dar una estimación del tamaño. Para este caso, la distancia Mahalanobis MHD es una buena opción.

2.8. Desafíos en la segmentación de imágenes médicas

A pesar de los avances en la tecnología de segmentación de imágenes médicas, todavía existen algunos desafíos que limitan la precisión y la eficien-

cia de estas técnicas. Algunos de los principales desafíos en la segmentación de imágenes médicas son:

- Variabilidad en la calidad de las imágenes: uno de los principales desafíos en la segmentación de imágenes médicas es la variabilidad en la calidad de las imágenes. La calidad de las imágenes puede variar según el equipo utilizado para adquirir las imágenes, según el protocolo de adquisición utilizado o según el estado del paciente en el momento de la adquisición de las imágenes. Esta variabilidad en la calidad de las imágenes puede dificultar la identificación y la separación de las estructuras en las imágenes.
- Dificultad en la identificación de las estructuras en las imágenes: otro desafío en la segmentación de imágenes médicas es la dificultad en la identificación de las estructuras en las imágenes. Las estructuras en las imágenes pueden ser muy pequeñas, muy delgadas o muy parecidas a otras estructuras en la imagen. Esta dificultad en la identificación de las estructuras puede dificultar la segmentación de las imágenes.
- Dificultad en la separación de las estructuras en las imágenes: además de la dificultad en la identificación de las estructuras, otro desafío en la segmentación de imágenes médicas es la dificultad en la separación de las estructuras en las imágenes. Las estructuras en las imágenes pueden estar superpuestas o fusionadas entre sí, lo que dificulta la separación de estas estructuras en diferentes regiones.
- Necesidad de una gran cantidad de imágenes etiquetadas: en el caso de la segmentación basada en aprendizaje automático, un desafío es la necesidad de una gran cantidad de imágenes etiquetadas. Para entrenar un modelo de aprendizaje automático para la segmentación de imágenes médicas, es necesario contar con un conjunto de imágenes previamente etiquetadas, en las que se haya identificado y separado las estructuras en las imágenes. Esta tarea de etiquetado de imágenes puede ser muy laboriosa y requerir un gran esfuerzo humano.
- Necesidad de validación clínica: otro desafío en la segmentación de imágenes médicas es la necesidad de validación clínica. Para poder utilizar la segmentación de imágenes médicas en el contexto del diagnóstico y el tratamiento de enfermedades, es necesario validar que los resultados de la segmentación son precisos y confiables. Esta validación clínica puede requerir la realización de estudios clínicos y la comparación de los resultados de la segmentación con los resultados de otros métodos de diagnóstico.

Capítulo 3

Desarrollo del trabajo

El presente capítulo describe el trabajo que se ha realizado a partir de todos los conocimientos adquiridos durante la fase del análisis y revisión sistemática realizada para la elaboración del capítulo anterior del estado del arte.

El trabajo ha consistido en la elaboración de diversos cuadernos de *jupyter* para entrenar distintas arquitecturas de segmentación de imágenes médicas basadas en redes neuronales profundas de aprendizaje automático usando distintos datasets para comprobar de acuerdo a ciertas métricas cuales de ellas se muestran más adecuadas dependiendo de cada conjunto de datos.

Para ello, por un lado se ha decidido entrenar a un modelo desde cero y se ha probado con distintas arquitecturas como Unet, UNETR o Swin-UNETR. Tal y como se verá en el trabajo desarrollado, se ha ido probando el entrenamiento con distintas épocas (*epochs*) guardando los checkpoint y se va haciendo inferencia de nuevas imágenes no utilizadas durante el entrenamiento y se va viendo como el resultado de la segmentación automática va mejorando las métricas de evaluación acercándose al *ground truth* de la etiqueta realizada por el especialista.

Debido a la cantidad de tiempo y de recursos que consume un entrenamiento desde cero, posteriormente se ha implementado unos cuadernos *jupyter* que contienen ya los modelos pre-entrenados de ciertas arquitecturas y a través de *transfer learning* se hace un entrenamiento fino del modelo pre-entrenado permitiendo mejorar las métricas de evaluación del modelo pre-entrenado.

Posteriormente se presenta una solución integral para segmentación de imágenes médicas a gran escala donde se puede entrenar simultáneamente a varios modelos, optimizar los hiperparámetros y realizar un ensamblado de la mejor solución de segmentación de las obtenidas.

Todo este trabajo se ha realizado usando los cuadernos jupyter de Google Colab puesto que los entrenamientos y las inferencias hacen uso intensivo de GPU de al menos 16GB de RAM y el autor de este TFM no dispone de un equipo lo suficientemente potente. Para algunas de las pruebas de los cuadernos de jupyter no es suficiente con el entorno de ejecución de la versión GPU estándar de Google Colab y se ha procedido a realizarlos con Colab PRO que proporciona entornos configurable con más memoria RAM del sistema e incluso GPU Premium que permite tener GPU Nvidia A100-SXM4 de 40 GB con CUDA 11.6 y una memoria RAM del sistema de 80GB lo cual acelera considerablemente los procesos de entrenamiento.

Toda la programación se ha realizado en lenguaje Python utilizando las librerías más comunes como por ejemplo Numpy para el manejo de los array n-dimensionales de datos, matplotlib para la visualización de gráficos y visualizaciones complejas o MONAI que veremos en detalle en el siguiente apartado para el análisis y procesamiento de imágenes médicas para redes neuronales de aprendizaje profundo.

Por último, se ha realizado un testing de herramientas de ayuda al diagnóstico clínico que integre las técnicas de segmentación y visibilice la gran importancia y aplicación real de este tipo de herramientas en el mundo real y sus ventajas que aporta.

3.1. MONAI

MONAI (*Medical Open Networks for Artificial Intelligence*) es un conjunto de marcos colaborativos, de código abierto y disponibles gratuitamente, construidos para acelerar la investigación y la colaboración clínica en Imágenes Médicas [57].

El objetivo es acelerar el ritmo de innovación y traducción clínica al construir un robusto marco de software que aporte valor a cada nivel de la investigación de aprendizaje profundo y despliegue en imágenes médicas.

MONAI es un proyecto de código abierto construido en la plataforma PyTorch y liberado bajo la licencia Apache 2.0. Tiene como objetivo capturar las mejores prácticas de desarrollo de IA (Inteligencia Artificial) para investigadores en atención médica, con un enfoque inmediato en imágenes médicas. Cuando se trata de IA médica, es importante tener herramientas que cubran el flujo de trabajo de extremo a extremo. MONAI proporciona esas herramientas para todo el flujo de desarrollo del modelo de IA médica, desde la investigación hasta la producción clínica.

Es una iniciativa iniciada inicialmente por NVIDIA y el King's College de Londres para establecer una comunidad inclusiva de investigadores de IA para desarrollar y intercambiar mejores prácticas para la IA en imágenes de

atención médica en toda la academia y los investigadores empresariales. Esta colaboración se ha expandido para incluir líderes académicos e industriales en todo el campo de imágenes médicas. MONAI ha sido utilizado en una amplia variedad de aplicaciones de aprendizaje automático en el ámbito médico, incluyendo la segmentación de imágenes médicas.

Proporciona herramientas y bibliotecas especializadas que facilitan el desarrollo de aplicaciones de aprendizaje profundo en el campo de la imagen médica, desde la preparación y preprocesamiento de datos hasta la implementación y evaluación de modelos de aprendizaje profundo. También ofrece una gran cantidad de funciones preconstruidas que pueden ser utilizadas para implementar técnicas de segmentación de imágenes médicas utilizando redes neuronales profundas. Además de herramientas software de alta calidad para anotar, construir, entrenar, desplegar y optimizar flujos de trabajo de IA en atención médica como se ve reflejado esquemáticamente en la Figura 3.1. Estas herramientas proporcionan un software fácil de usar que facilita la reproducibilidad y la fácil integración.

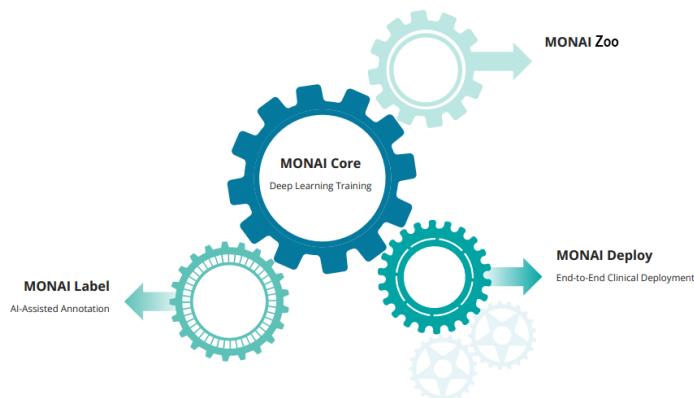


Figura 3.1: Herramientas de MONAI para el ciclo de vida de la IA Médica.

El paquete de bibliotecas, herramientas y SDKs dentro de MONAI brindan una base sólida y común que abarca el ciclo de vida completo de la IA médica, desde la anotación hasta el despliegue.

En las siguientes secciones describimos las principales herramientas de MONAI, algunas de las cuales hemos hecho uso en el desarrollo de nuestro trabajo.

3.1.1. MONAI *Core*

Un marco específico del dominio para entrenar modelos de IA para imágenes de atención médica.

MONAI *Core* es la biblioteca principal del Proyecto MONAI y ofrece capacidades específicas para entrenar modelos de IA para imágenes médicas. Estas capacidades incluyen desde transformaciones de imágenes específicas para la atención médica a algoritmos de segmentación 3D basados en transformadores de última generación como UNETR.

La arquitectura general de MONAI *core* se muestra en la Figura 3.2 extraída de [58].

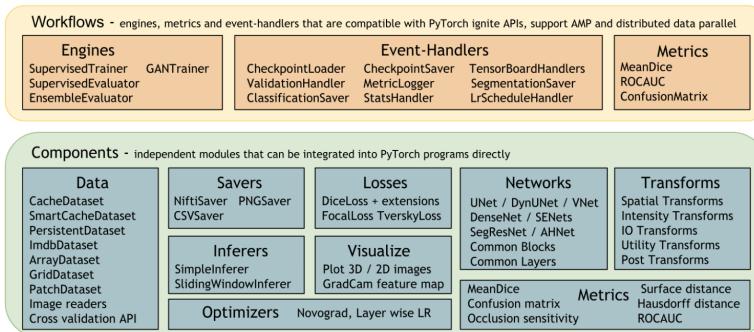


Figura 3.2: Arquitectura de MONAI Core.

Todos los componentes principales son módulos independientes, que se pueden integrar fácilmente en cualquier programa existente de PyTorch.

Las imágenes médicas requieren métodos especializados para entrada/salida, preprocessamiento y aumento de datos. A menudo siguen formatos específicos, se manejan con protocolos específicos y las matrices de datos a menudo son de muchas dimensiones. Los módulos de MONAI Core de '*monai.transforms*' y '*monai.data*' incluyen un conjunto de APIs específicas que se pueden usar en funcionalidades de aprendizaje profundo como transformaciones básicas de imágenes, *pipelines* de preprocessamiento complejos, operaciones sincronizadas en diferentes modalidades y entradas de supervisión del modelo, todo ello de una forma determinista y reproducible con transformaciones inversas y asegurando un uso óptimo de los recursos GPU hardware.

MONAI Core extiende las APIs '*Dataset*' y '*DataLoader*' siguiendo el patrón de diseño de PyTorch como mejoras importantes en términos de usabilidad específica del dominio y rendimiento del pipeline. Los métodos basados en datos requieren muchas (potencialmente miles) épocas de lectura y preprocessamiento de datos de entrenamiento. MONAI proporciona conjuntos de datos basados en caché con múltiples hilos para acelerar el proceso. La caché puede ser persistente y dinámica (SmartCacheDataset) y reutilizarse en diferentes experimentos. Además, MONAI proporciona varias clases de conjuntos de datos listas para usar, como MedNISTDataset, DecathlonDataset o TciaDataset, que incluyen la descarga de datos y funcionalidades de

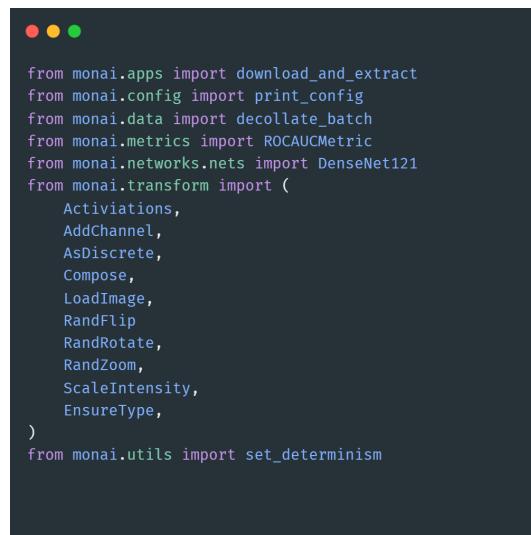
apoyo a la generación de divisiones de los datos de entrenamiento/evaluación con transformaciones.

MONAI implementa arquitecturas de redes de referencia con el doble objetivo de la flexibilidad y la legibilidad del código. Implementa arquitecturas de redes neuronales profundas como SegResNet, UNET, UNETR que han demostrado ser particularmente efectivas para tareas de análisis de imágenes médicas. Las capas y bloques de la red están implementados de manera general para ser compatibles con entradas espaciales 1D, 2D y 3D. Se puede integrar fácilmente las capas, bloques y arquitectura de redes como parte de *pipelines* personalizados. Así mismo, MONAI proporciona varias utilidades para aprovechar los pesos existentes del modelo.

Proporciona también funciones de pérdida comúnmente utilizadas para varias aplicaciones como DiceLoss, GeneralizedDiceLoss, TverskyLoss, DiceFocalLoss así como optimizaciones numéricas y utilidades relevantes como Novograd y LearningRateFinder.

Para ejecutar inferencias de modelos y evaluar la calidad del modelo, MONAI proporciona implementaciones de referencia para los enfoques relevantes ampliamente utilizados. Actualmente, varias métricas de evaluación y patrones de inferencia populares están incluidos como la inferencia con ventana deslizante, métricas para tareas médicas como DICE, ROCAUC, matrices de confusión, distancia de Hausdorff, distancia de superficie y sensibilidad a la oclusión.

La Figura 3.3 [57] muestra un ejemplo de como se importan distintos componentes de MONAI *Core*.



```
● ● ●
from monai.apps import download_and_extract
from monai.config import print_config
from monai.data import decollate_batch
from monai.metrics import ROCAUCMetric
from monai.networks.nets import DenseNet121
from monai.transforms import (
    Activations,
    AddChannel,
    AsDiscrete,
    Compose,
    LoadImage,
    RandFlip,
    RandRotate,
    RandZoom,
    ScaleIntensity,
    EnsureType,
)
from monai.utils import set_determinism
```

Figura 3.3: Importacion de librerías de MONAI Core.

3.1.2. MONAI Label

MONAI *Label* es una herramienta inteligente de etiquetado y aprendizaje de imágenes que utiliza la asistencia de IA para reducir el tiempo y el esfuerzo de anotar nuevos conjuntos de datos. Al utilizar las interacciones del usuario, MONAI *Label* entrena un modelo de IA para una tarea específica y aprende y actualiza continuamente ese modelo a medida que recibe imágenes anotadas adicionales con un modelo de *Active Learning* como el que puede verse en la Figura 3.4 [59].

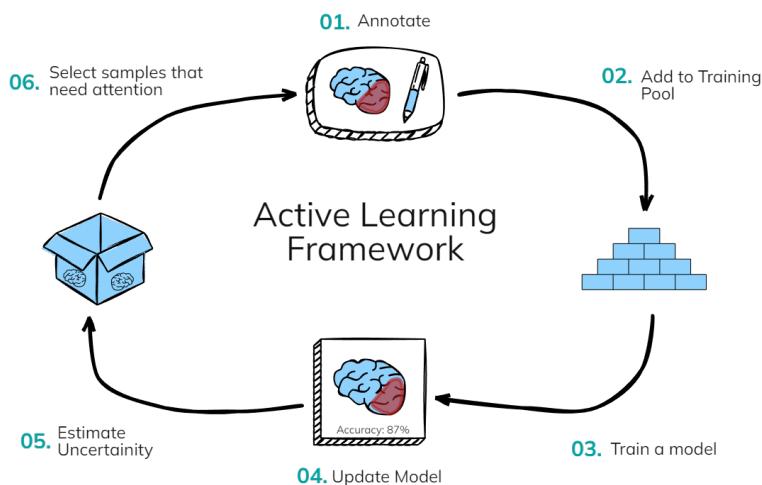


Figura 3.4: Sistema de Active Learning de MONAI Label. Fuente: Díaz-Pinto *et al.*

MONAI *Label* permite a investigadores y desarrolladores mejorar continuamente sus aplicaciones, pudiendo interactuar con ellas como lo haría un usuario. Los usuarios finales (clínicos, tecnólogos y anotadores en general) se benefician de que la IA aprenda continuamente y mejore su capacidad de comprender lo que el usuario final está tratando de anotar. Tiene un sistema cliente-servidor como muestra la Figura 3.5. Es un ecosistema de código abierto fácil de instalar que puede ejecutarse localmente en una máquina con una o múltiples GPUs. Tanto el servidor como el cliente funcionan en la misma o en diferentes máquinas. Comparte los mismos principios con MONAI.

En la parte cliente, ofrece soporte de anotación a través de visores como 3DSlicer y OHIF para radiodiagnóstico, QuPath, Digital Slide Archive y CVAT para patología y CVAT para endoscopia.

3DSlicer es un paquete de software gratuito, de código abierto y multiplataforma ampliamente utilizado para la investigación en imágenes médicas, biomédicas y relacionadas. La Figura 3.6 muestra una imagen en 3DS-

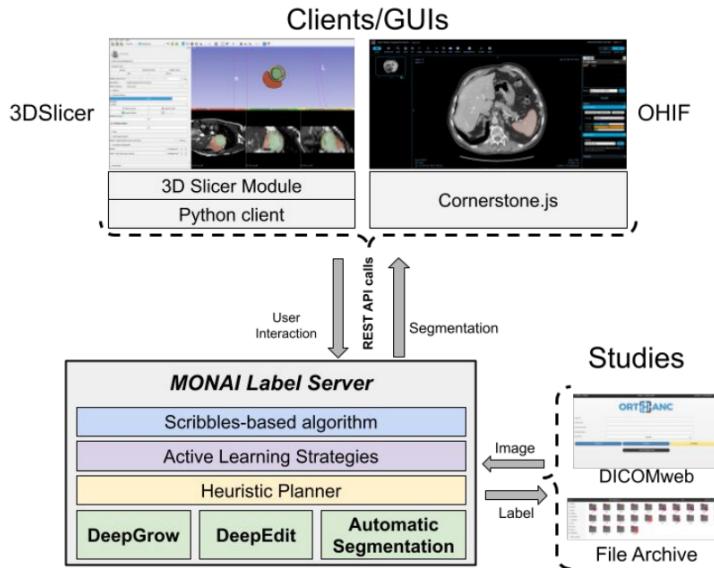


Figura 3.5: Arquitectura de MONAI Label. Fuente: Díaz-Pinto *et al.*

licer que ha sido segmentada mediante MONAI Label. 3DSlicer cuenta con una comunidad de apoyo muy importante, una interfaz gráfica sencilla y fácil de personalizar con muchas herramientas de anotación manual y un registro de imágenes eficiente.

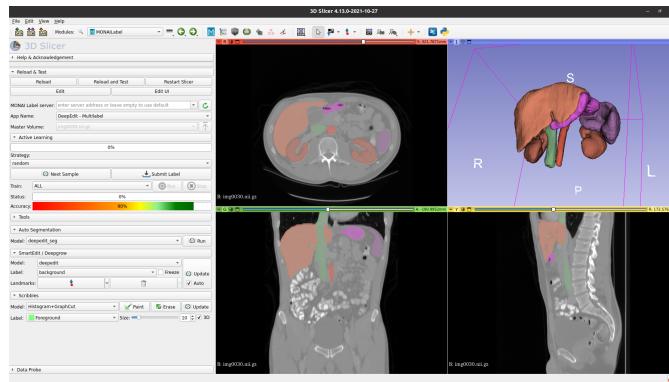


Figura 3.6: Visor 3d Slicer integrado con MONAI Label. Fuente: Díaz-Pinto *et al.*

MONAI Label tiene integración tanto para imágenes de Radiología como de Patología. Al habilitar un flujo de trabajo que se integra directamente en un visor de radiólogo o patólogo y permitiendo el aprendizaje continuo, MONAI pretende acelerar la adopción del Aprendizaje Profundo en la Ima-

gen Médica. La Figura 3.7 muestra un resumen de las distintas aplicaciones de MONAI Label.



Figura 3.7: MONAI Label Apps de Radiología, Patología y Endoscopia. Visores OHIF, DSA, QuPath y CVAT.

Aplicación de Radiología: Esta aplicación tiene modelos de ejemplo para realizar tanto segmentación interactiva como automatizada sobre imágenes de radiología (3D). Incluye segmentación automática con los últimos modelos de aprendizaje profundo (por ejemplo, UNet, UNETR) para múltiples órganos abdominales. Las herramientas interactivas incluyen *DeepEdit* y *Deepgrow* para mejorar activamente los modelos entrenados y el despliegue.

Aplicación de Patología: Esta aplicación tiene modelos de ejemplo para realizar tanto segmentación interactiva como automatizada sobre imágenes de patología (WSI). Incluye segmentación de múltiples etiquetas de núcleos para células neoplásicas, inflamatorias, de tejido conectivo/suave, muertas y epiteliales. La aplicación proporciona herramientas interactivas, incluyendo DeepEdits para segmentación interactiva de núcleos.

Aplicación de Endoscopia: La aplicación de Endoscopia permite a los usuarios usar modelos interactivos, automatizados de segmentación y clasificación sobre imágenes 2D para el uso de endoscopia. Se integra con el visor de CVAT .

Aplicación *Bundle* de MONAI: La aplicación *Bundle* permite a los usuarios personalizar modelos para inferencia, entrenamiento o procesamiento previo y posterior de cualquier anatomía objetivo. La especificación de integración de MONAILabel de la aplicación *Bundle* enlaza el Model-Zoo archivado para etiquetado personalizado (por ejemplo, el modelo transformador de terceros para etiquetar la corteza renal, medula y sistema pelvicaliceal).

3.1.3. MONAI Deploy App SDK

Permite a los desarrolladores convertir un modelo de IA en una aplicación de IA.

MONAI *Deploy* pretende convertirse en el estándar de facto para desarrollar, empaquetar, probar, implementar y ejecutar aplicaciones de IA médica en producción clínica. MONAI *Deploy* crea un conjunto de pasos intermedios donde los investigadores y especialistas médicos pueden tener

confianza en las técnicas y enfoques utilizados con la IA, lo que permite un flujo de trabajo iterativo hasta que la infraestructura de inferencia de IA esté lista para moverse a entornos clínicos. La arquitectura de despliegue responde al esquema que muestra la Figura 3.8.

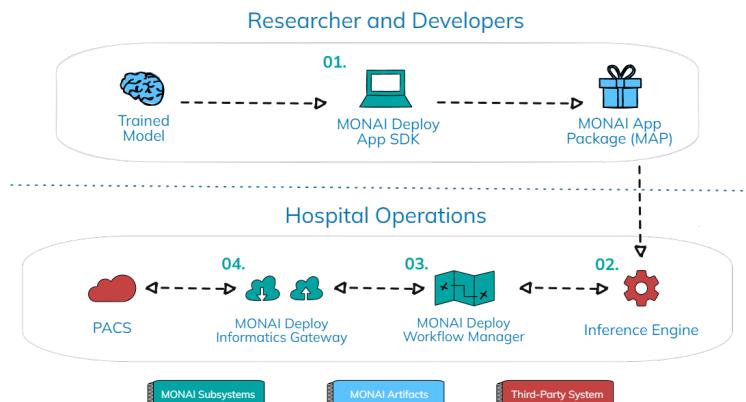


Figura 3.8: MONAI Deploy SDK. Arquitectura de despliegue

El SDK de aplicación MONAI *Deploy* ofrece un marco y herramientas asociadas para diseñar, verificar y analizar el rendimiento de las aplicaciones impulsadas por IA en el dominio de la atención médica.

Contiene elementos como un mecanismo para empaquetar una aplicación en una instancia de 'Paquete de Aplicación MONAI' (MAP), un mecanismo para ejecutar localmente un MAP a través de *App Runner*, un marco de trabajo para el desarrollo de aplicaciones basado en Python, un conjunto de aplicaciones de muestra, o un conjunto de funcionalidades específicas del dominio DICOM para acelerar aún más el desarrollo de la aplicación de inferencia de IA de imágenes médicas.

El SDK de la aplicación MONAI Deploy permite el diseño de aplicaciones de inferencia IA aptas para uso, reproducibles y listas para producción en el ámbito de la atención médica. El SDK fue diseñado con los cuatro objetivos principales de diseño como usabilidad (proporciona una API para estructurar el código como un flujo de trabajo), composición (proporciona una colección de operadores como bloques de construcción), portabilidad (permite empaquetar la aplicación, los modelos asociados y los metadatos relevantes en un paquete autocontenido que es fácil de portar) y listo para producción (diseñado para ejecutar una aplicación en varios entornos, desde el desarrollo hasta la producción).

El concepto central en el SDK de aplicaciones MONAI *Deploy* es una aplicación de inferencia en el dominio de la atención médica. Una aplicación representa la lógica de negocios de lo que necesita ser calculado con algunos datos específicos del dominio de la atención médica como entrada. La

preocupación central del diseño del SDK de aplicaciones MONAI *Deploy* es cómo proporcionar un marco de trabajo donde los desarrolladores puedan crear fácilmente una aplicación de inferencia en el dominio de la atención médica utilizando bloques de construcción y código personalizado.

Se recomienda al usuario estructurar el código de la aplicación de una manera que permita la reproducibilidad y la depurabilidad. Por otro lado, el SDK está diseñado para minimizar los problemas relacionados con la preparación para la producción durante el tiempo de desarrollo.

Como se observa en la Figura 3.9, los conceptos principales en tiempo de ejecución (*run-time*) en el SDK de la aplicación MONAI Deploy son el paquete de aplicación MONAI (MAP) y el ejecutor de aplicación MONAI (MAR). Una decisión clave de diseño del SDK es hacer que el marco sea independiente del tiempo de ejecución. El mismo código debería ser ejecutable en diferentes entornos, como en una estación de trabajo durante el desarrollo o en un orquestador (*orchestrator*) de flujos de trabajo listo para producción durante la producción.

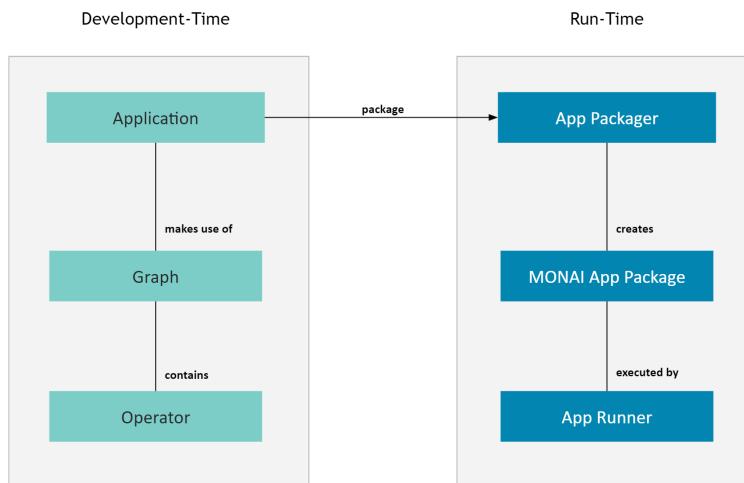


Figura 3.9: MONAI Deploy SDK. Arquitectura de alto nivel

3.1.4. MONAI Model Zoo

MONAI *Model Zoo* aloja una colección de modelos de imágenes médicas en el formato MONAI *Bundle* [60].

MONAI *Bundle* es un paquete de modelo autocontenido con pesos pre-entrenados y todos los metadatos asociados abstraídos a través de configuraciones basadas en JSON y YAML de modelos de aprendizaje profundo.. Al enfocarse en la facilidad de uso y flexibilidad, puede anular directamente

o personalizar estas configuraciones o utilizar un modelo de programación híbrido que admite la abstracción de configuración a código de Python.

Actualmente en MONAI *Model Zoo* hay 20 modelos preentrenados de los socios de MONAI, incluyendo el King's College London, la Universidad Charité, la Universidad de Warwick, la Universidad Vanderbilt y la Clínica Mayo. La Figura 3.10 muestra ejemplos de algunos de estos modelos disponibles.

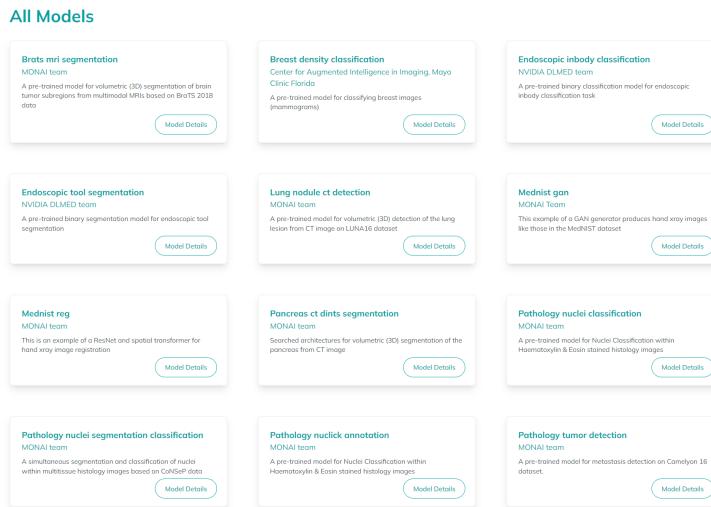


Figura 3.10: MONAI Model Zoo. Ejemplos de Modelos Preentrenados

3.2. Entrenamiento desde cero: segmentación imágenes médicas

En la presente sección vamos a describir como se entrena desde cero un modelo de segmentación de imágenes médicas. Para ello vamos a usar las librerías de MONAI *Core* que hemos comentado en la sección anterior.

En cuanto al entorno de trabajo, hemos realizado un cuaderno de jupyter que hemos ejecutado sobre Google Colab. El cuaderno de jupyter realizado se puede ejecutar sobre cualquier equipo o servidor que posea una GPU de al menos 16GB y tenga al menos 32 GB de memoria RAM. Como en nuestro caso no disponíamos de dicho equipo, la solución ha sido realizarlo con Google Colab en su versión PRO. Para una óptima ejecución debemos elegir la configuración de Google Colab PRO con acelerador por hardware GPU y clase de GPU premium. También se podría llegar a ejecutar con clase de GPU Estándar y Alta capacidad de RAM aunque con el consiguiente aumento en el tiempo del entrenamiento. Si se selecciona una configuración

783.2. Entrenamiento desde cero: segmentación imágenes médicas

inferior a esta como por ejemplo la de la versión gratuita de Google Colab, el cuaderno falla por falta de memoria RAM o bien por alcanzar el pico en GPU.

Hemos seleccionado el conjunto de datos de BTCV (*Beyond the Cranial Vault*) que describimos en el apartado 2.4.4 que contiene imágenes médicas provenientes de 50 Tomografía computarizada (CT) de la zona del abdomen para la segmentación de múltiples órganos, en concreto 13 órganos abdominales entre los que se encuentran los riñones, páncreas, esófago, estómago, etc. [32].

El dataset ha sido descargado de <https://www.synapse.org/#!Synapse:syn3193805/wiki/217752>. y se ha subido a la aplicación de google drive de la cuenta de gmail del correo del alumno que la universidad de granada nos proporciona.

En concreto a la ruta \MyDrive\TFM_SegmentacionImagenesMedicas\Datasets\BTCV que muestra la Figura 3.11.

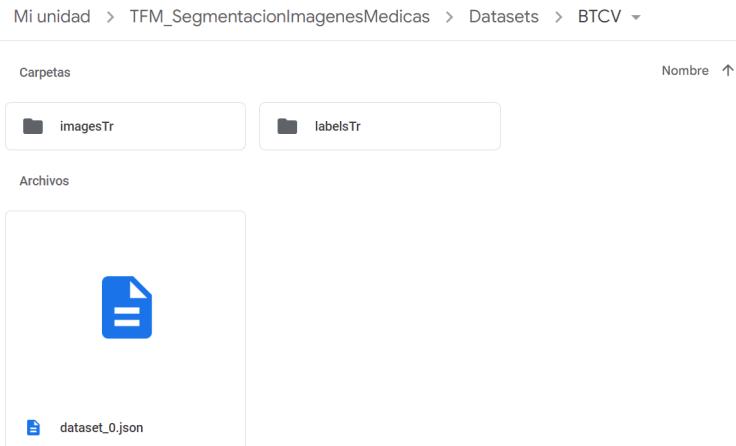


Figura 3.11: Ruta de Google Drive donde se ha alojado el Dataset de BTCV

Es importante señalar que las imágenes del *dataset* de BTCV van a la carpeta \imagesTr y las etiquetas van a la carpeta \labelsTr.

Además también se encuentra el fichero *dataset_0.json* que contiene la descripción de las imágenes y sus correspondientes etiquetas de *ground truth* que van a ser usadas para el entrenamiento y las que van a ser usadas para validación. En nuestro caso se han usado 24 imágenes (y correspondientes etiquetas) para entrenamiento y 6 para validación. Las de validación se corresponden con la numeración desde la 35 a la 40 ambas incluidas.

El objetivo pues es que una vez que tengamos el modelo entrenado de

acuerdo a los parámetros que definamos como el tipo de modelo de arquitectura de red neuronal, la función de pérdida, el optimizador, las épocas de entrenamiento, etc. poderle pasar nuevas imágenes que no han sido usadas en el entrenamiento (en este caso las imágenes de validación) y que por inferencia nos de un resultado lo más parecido posible al etiquetado de *ground truth* como puede verse en la Figura 3.12.

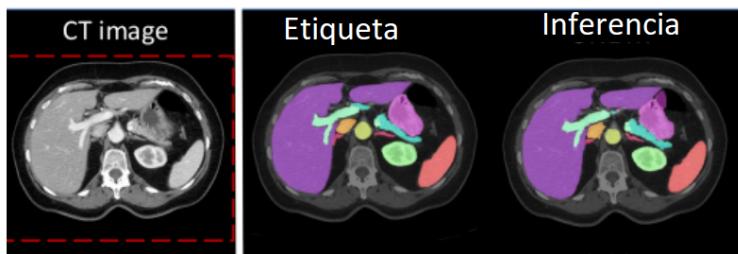


Figura 3.12: Imagen original, segmentacion Ground Truth y segmentacion obtenida con la inferencia

En nuestro caso vamos a usar para la métrica de evaluación el Mean DICE de la librería de MONAI. Esta métrica calcula la puntuación promedio de Dice entre dos tensores. Puede admitir tanto tareas de multi-clases como de multi-etiquetas.

A continuación vamos a describir brevemente el proceso completo que se ha realizado adjuntando algunas pequeñas partes de código sólo en aquellos pasos que consideramos más importantes ser destacados. El cuaderno completo de jupyter con todo el procedimiento y todo el código se encuentra disponible en el Apéndice A.1 de la presente memoria así como en el repositorio de Github del autor indicado en dicho apéndice.

3.2.1. Proceso

En primer lugar hacemos un *setup* del entorno verificando si está conectada la GPU del entorno de ejecución ya que nos será luego imprescindible para ejecutar el entrenamiento, en caso de que esté conectada mostramos la información con las características de la GPU por pantalla.

Instalamos las librerías de MONAI en su ultima versión estable que en el momento de la realización del cuaderno es la 1.1.0 así como otras librerías que nos son necesarias y que no estén instaladas por defecto en el entorno de Google Colab.

Importamos aquellas librerías que nos van a hacer falta tanto de MONAI como otras como de numpy o matplotlib y finalizamos viendo las versiones instaladas de cada una de ellas.

80B.2. Entrenamiento desde cero: segmentación imágenes médicas

A continuación procedemos al *setup* del directorio de trabajo. En nuestro caso establecemos la conexión con la cuenta de Google Drive y le especificamos la carpeta donde vamos a almacenar los resultados del modelo que obtengamos.

Seguimos con el *setup* de las transformaciones para entrenamiento (*training*) y validación. Aquí se establece una composición de distintas transformaciones que se realizan para las imágenes de entrenamiento y de validación, se usa la librería de transformación de MONAI *Core* y se realizan distintas transformaciones como recortes, cambios de intensidad, rotaciones, etc. En el siguiente código se pueden ver en resumen algunas de ellas:

```
1 train_transforms = Compose(
2     [
3         LoadImaged(keys=["image", "label"]),
4         EnsureChannelFirstd(keys=["image", "label"]),
5         Orientationd(keys=["image", "label"], axcodes="RAS"),
6         Spacingd(
7             keys=["image", "label"],
8             pixdim=(1.5, 1.5, 2.0),
9             mode=("bilinear", "nearest"),
10            ),
11            ScaleIntensityRanged(
12                keys=["image"],
13                a_min=-175,
14                a_max=250,
15                b_min=0.0,
16                b_max=1.0,
17                clip=True,
18                ),
19                CropForegroundd(keys=["image", "label"], source_key="image"),
20                RandCropByPosNegLabeld([...]),
21                RandFlipd([...]),
22                RandFlipd([...]),
23                RandFlipd([...]),
24                RandRotate90d([...]),
25                RandShiftIntensityd([...]),
26            ]
27        )
```

Listing 3.1: Transformaciones para entrenamiento

Seguidamente se procede a la carga de los datos del *dataset* que habíamos descargado previamente, estos datasets son almacenados en caché para hacer más eficiente el posterior entrenamiento:

```
1 # Cacheamos los datos de entrenamiento y validacion
2 train_ds = CacheDataset(
3     data=datalist,
4     transform=train_transforms,
```

```

5     cache_num=24,
6     cache_rate=1.0,
7     num_workers=8,
8 )
9 train_loader = DataLoader(
10     train_ds, batch_size=1, shuffle=True, num_workers=8,
11     pin_memory=True
11 )

```

Listing 3.2: Cacheado del dataset

Procedemos a continuación a visualizar alguna imagen de prueba de validación y a comprobar la forma de la imagen y de los datos. Como resultado obtenemos la imagen original de la tomografía y el etiquetado *ground truth* que ha realizado el especialista que podemos ver en la Figura 3.13.

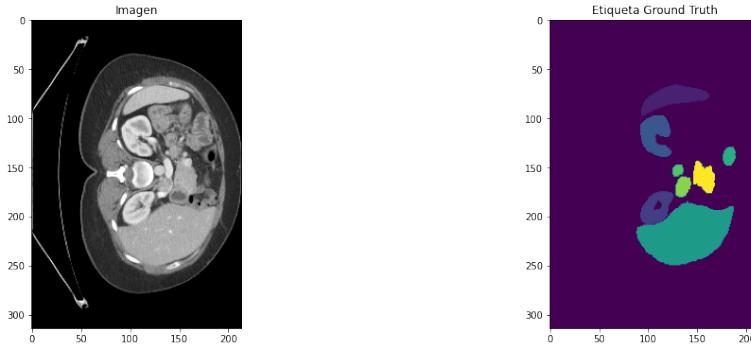


Figura 3.13: Imagen original de CT y etiqueta de especialista

A continuación pasamos a definir ya el modelo de arquitectura de red neuronal de segmentación que vamos a usar, así como la función de pérdida y el optimizador. Para ello se utilizan las librerías de MONAI *Core*, en este primer caso del cuaderno de jupyter vamos a usar una red basada en el modelo de UNETR que vimos en la Sección 2.3.9, para la función de pérdida vamos a usar la DiceCELoss que calcula la pérdida promedio de Dice entre dos tensores y que Milletari *et al.* definió en su artículo[61] y como optimizador vamos a usar AdamW de PyTorch que implementa el algoritmo de AdamW [62].

```

1 # Definimos el modelo de arquitectura de segmentacion, en este
2 # caso una UNETR
3 model = UNETR(
4     in_channels=1,
5     out_channels=14,
6     img_size=(96, 96, 96),
7     feature_size=16,
8     hidden_size=768,
8     mlp_dim=3072,

```

8.2.2. Entrenamiento desde cero: segmentación imágenes médicas

```
9     num_heads=12,
10    pos_embed="perceptron",
11    norm_name="instance",
12    res_block=True,
13    dropout_rate=0.0,
14 ).to(device)
15
16 # Definimos la función de perdida
17 loss_function = DiceCELoss(to_onehot_y=True, softmax=True)
18 torch.backends.cudnn.benchmark = True
19
20 #Definimos el optimizador
21 optimizer = torch.optim.AdamW(model.parameters(), lr=1e-4,
22                               weight_decay=1e-5)
```

Listing 3.3: Modelo de Red. Función de Pérdida y Optimizador

Si quisiésemos utilizar otro modelo de red de segmentación, por ejemplo Swin-UNETR, simplemente tendríamos que cambiar la variable *model* y definir la red de segmentación que queremos utilizar para el entrenamiento, definiendo correctamente los parámetros de entrada de la red.

```
1 model = SwinUNETR(
2     img_size=(96, 96, 96),
3     in_channels=1,
4     out_channels=14,
5     feature_size=48,
6     use_checkpoint=True,
7 ).to(device)
```

Listing 3.4: Modelo de Red con Swin-UNETR

Por último, ejecutamos un proceso de entrenamiento de red neuronal típico de PyTorch una vez que previamente hemos definido modelo, función de pérdida y optimizador. Se define el número de iteraciones o épocas se va a entrenar el modelo, se establece cada cuantas iteraciones se va a realizar la evaluación para comprobar la métrica de evaluación que en este caso se selecciona la DICE y se guarda el modelo de entrenamiento que va resultando que da mejores valores de DICE así como se informa de cual es la mejor métrica conseguida y en que iteración se ha conseguido esa mejor métrica DICE.

Este modelo resultado se puede usar para ejecutar inferencias posteriores de imágenes nuevas que el modelo no ha utilizado para su entrenamiento y que por tanto no conoce.

```
1 def train(global_step, train_loader, dice_val_best,
2           global_step_best):
3     model.train()
```

```
3     epoch_loss = 0
4     step = 0
5     epoch_iterator = tqdm(
6         train_loader, desc="Training (X / X Steps) (loss=X.X)",
7         dynamic_ncols=True
8     )
9     for step, batch in enumerate(epoch_iterator):
10        step += 1
11        x, y = (batch["image"].cuda(), batch["label"].cuda())
12        logit_map = model(x)
13        loss = loss_function(logit_map, y)
14        loss.backward()
15        epoch_loss += loss.item()
16        optimizer.step()
17        optimizer.zero_grad()
18        epoch_iterator.set_description(
19             "Training (%d / %d Steps) (Perdida loss=%2.5f)" % (
20                 global_step, max_iterations, loss)
21         )
22         if (
23             global_step % eval_num == 0 and global_step != 0
24         ) or global_step == max_iterations:
25             epoch_iterator_val = tqdm(
26                 val_loader, desc="Validar (X / X Steps) (dice=X
27                 .X)", dynamic_ncols=True
28             )
29             dice_val = validation(epoch_iterator_val)
30             epoch_loss /= step
31             epoch_loss_values.append(epoch_loss)
32             metric_values.append(dice_val)
33             # Guardamos el modelo que tiene mejor metrica DICE
34             if dice_val > dice_val_best:
35                 dice_val_best = dice_val
36                 global_step_best = global_step
37                 torch.save(
38                     model.state_dict(), os.path.join(root_dir,
39                     "mejor_modelo_metrica.pth")
40                 )
41                 print(
42                     "Se ha salvado el modelo ! Mejor Avg. Dice:
43                     {} Actual Avg. Dice: {}".format(
44                         dice_val_best, dice_val
45                     )
46                 )
47             else:
48                 print(
49                     "No se ha salvado el modelo ! Mejor Avg.
Dice: {} Actual Avg. Dice: {}".format(
50                         dice_val_best, dice_val
51                     )
52                 )
53             global_step += 1
54     return global_step, dice_val_best, global_step_best
```

8.3.2. Entrenamiento desde cero: segmentación imágenes médicas

Listing 3.5: Entrenamiento del modelo

Por tanto, el proceso concluye aquí con el guardado del mejor modelo obtenido. En el siguiente apartado vamos a ver resultados obtenidos y como estos varían significativamente en función del número de iteraciones que realicemos en nuestro entrenamiento.

3.2.2. Resultados

Pretendemos en este apartado mostrar algunos de los resultados del modelo aplicando distintos números de iteraciones en su entrenamiento.

Comenzamos con un entrenamiento de 2.000 iteraciones.

Con este entrenamiento, podemos ver en la Figura 3.14 que alcanzamos un DICE de apenas 0.1323 con una función de pérdida en valores aún muy altos.

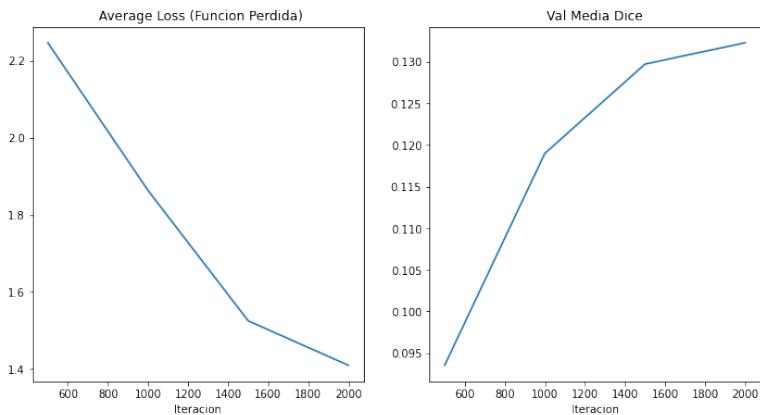


Figura 3.14: Función de Pérdida y Métrica DICE para entrenamiento de 2.000 iteraciones

El tener un DICE tan bajo y lejos de 1 va a implicar que la inferencia sobre imágenes nuevas que se haga no va a tener mucha calidad de segmentación como demuestra por ejemplo la Figura 3.15 que se corresponde a una de las imágenes de validación en la que podemos ver a la izquierda la imagen original, en el centro la etiqueta *ground truth* de la segmentación del especialista y a la derecha la segmentación obtenida por el modelo entrenado desde cero con 2000 iteraciones. Vemos que no es capaz aun de segmentar por multietiquetas y este es el motivo que solo vemos una imagen en amarillo muy distinta a los distintos colores de órganos que podemos ver en la imagen etiquetada del *ground truth*. Por tanto, podemos considerar que el resultado obtenido es bastante pobre y poco preciso.

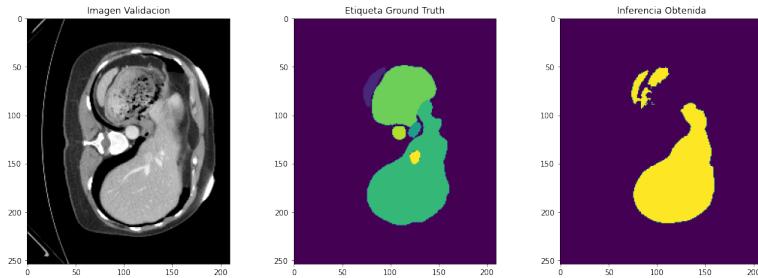


Figura 3.15: Imagen original de CT, etiqueta de especialista y resultado de inferencia con entrenamiento de 2.000 iteraciones

Entrenamos ahora al modelo con un entrenamiento de 10.000 iteraciones.

Usando la configuración Premium de Google Colab (GPU de 40GB y 80 GB de RAM, tarda aproximadamente 60 minutos solo en la parte del entrenamiento.

En la Figura 3.16 ya podemos ver que el DICE ha mejorado considerablemente con respecto a la anterior simulación de 2.000 iteraciones. Ahora tenemos un DICE de 0.5451 con las 10.000 iteraciones.

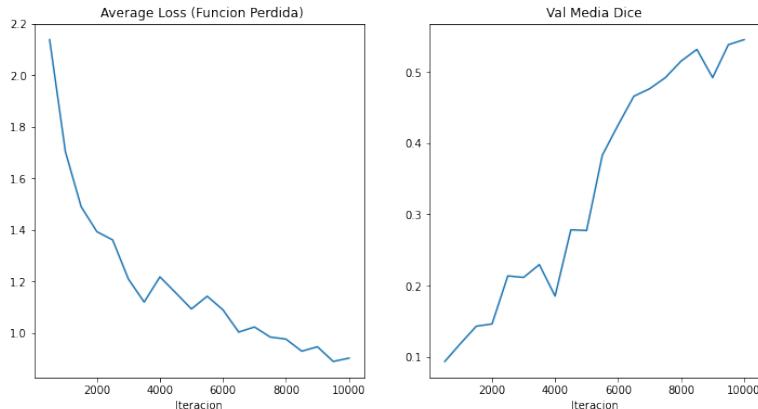


Figura 3.16: Función de Pérdida y Métrica DICE para entrenamiento de 10.000 iteraciones

Este DICE va a hacer que el resultado de la inferencia ya tenga más calidad como puede observarse en la Figura 3.17 donde la imagen de la derecha que es el resultado de la inferencia de la imagen original usando el mejor modelo entrenado con 10.000 iteraciones se parece más a la imagen del centro de las etiquetas de *ground truth* generadas por el especialista.

Por último, entrenamos al modelo con un entrenamiento de 25.000 iteraciones.

863.2. Entrenamiento desde cero: segmentación imágenes médicas

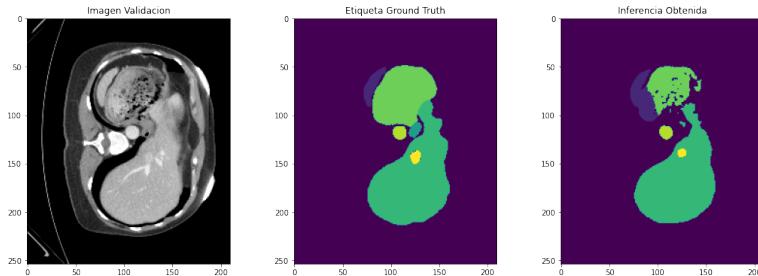


Figura 3.17: Imagen original de CT, etiqueta de especialista y resultado de inferencia con entrenamiento de 10.000 iteraciones

Usando la configuración Premium de Google Colab (GPU de 40GB y 80 GB de RAM, tarda aproximadamente 3 horas solo en la parte del entrenamiento.

Ya con este número de iteraciones podemos ver como la función de pérdida comienza a estabilizarse al igual que los DICE que se consiguen ya empiezan a variar relativamente poco desde las 20.000 iteraciones. Se consigue un mejor DICE de 0.7839 como puede verse en la Figura 3.18.

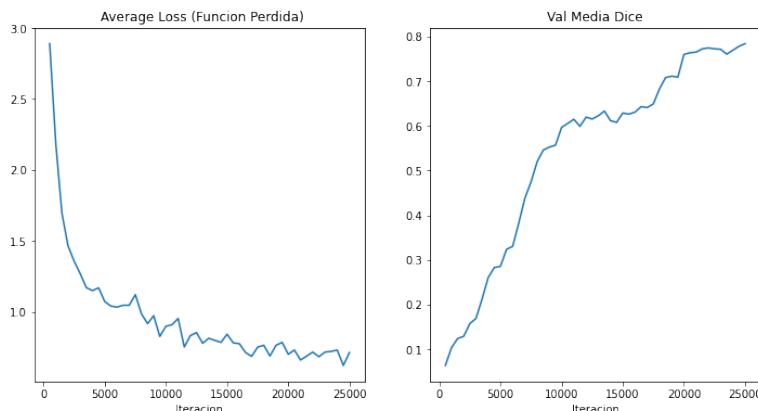


Figura 3.18: Función de Pérdida y Métrica DICE para entrenamiento de 25.000 iteraciones

Por tanto, en este caso la inferencia que obtenemos usando el mejor modelo conseguido con un entrenamiento de 25.000 iteraciones y que se muestra en la Figura 3.19, va ser un resultado bastante aceptable en comparación con el *ground truth* del especialista de mayor precisión que la imagen obtenida con el modelo de 10.000 y 2.000 iteraciones. En esta caso ya podemos ver como ambas segmentaciones se asemejan mucho y el modelo es capaz de distinguir segmentaciones multiclase para los 13 órganos distintos que se definieron como objetivo.

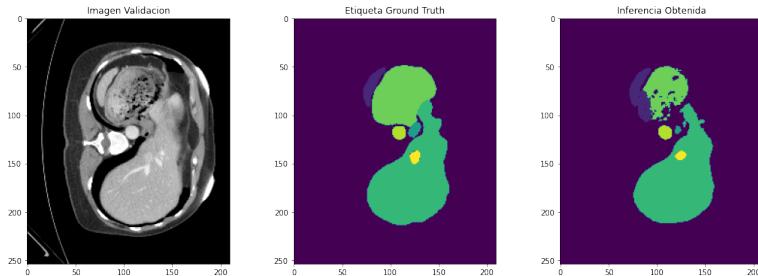


Figura 3.19: Imagen original de CT, etiqueta de especialista y resultado de inferencia con entrenamiento de 25.000 iteraciones

Si ponemos las 3 inferencias obtenidas en una misma imagen como muestra la Figura 3.20 podemos ver más claramente la diferencia que se obtiene en la salida de los modelos y como la de 25.000 iteraciones es más parecida a la imagen de etiqueta del especialista.

3.3. Inferencias con modelos entrenados

En la anterior sección vimos como se entrena un modelo desde cero. Una vez que tenemos un modelo entrenado, el archivo resultante (normalmente .pth o .pt) ya contiene todos los pesos del entrenamiento y puede ser usado para aplicar inferencia en nuevas imágenes y obtener las segmentaciones de las imágenes.

Para ello, hemos creado un cuaderno de jupyter en Google Colab y hemos probado a cargar directamente los modelos generados en nuestra sección anterior de entrenamiento desde cero y obtenemos con éxito el resultado de la inferencia de los datos de entrada obteniendo una segmentación como resultado con una calidad que depende del modelo seleccionado.

En nuestro caso hemos probado con los modelos generados de 2.000, 10.000 y 25.000 iteraciones que habíamos generado en el entrenamiento anterior y como cabe esperar, ante la misma imagen entrada usada en la sección anterior obtenemos como resultado la misma segmentación que antes sin necesidad de volver a entrenar de nuevo el modelo desde cero. La Figura 3.21 muestra el resultado usando el modelo de 25.000 iteraciones que si se compara con la anterior Figura 3.19 se pueden ver que son idénticas.

En el apéndice A.2 se ha incluido el código del cuaderno de Jupyter que puede ser ejecutado en Google Colab para cargar el modelo entrenado y ejecutar la inferencia. El cuaderno también se encuentra disponible en el repositorio de Github del autor así como se encuentran en el repositorio de Github los modelos generados previamente por si se quieren usar para

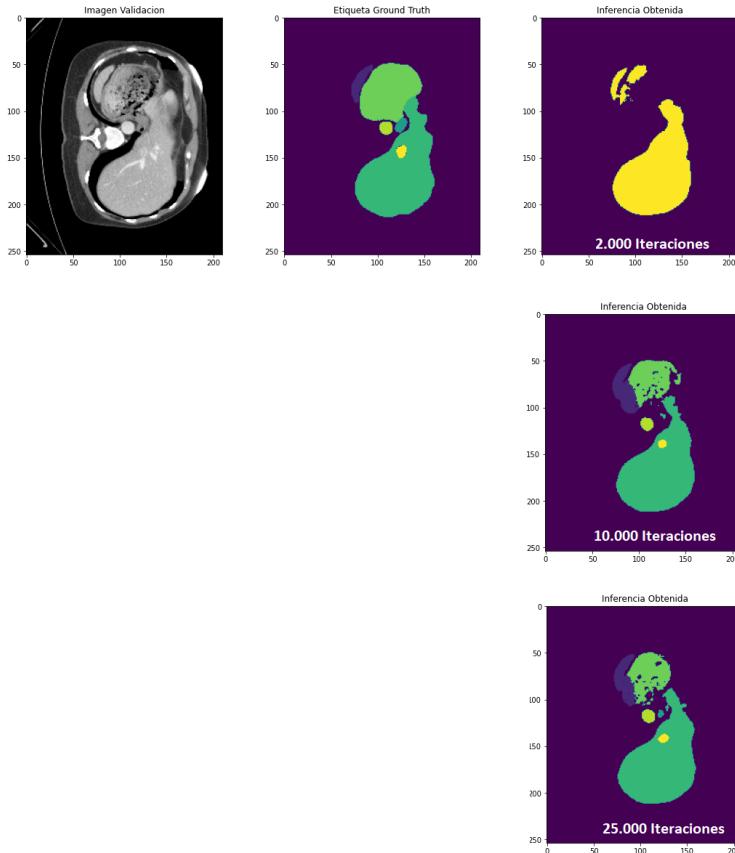


Figura 3.20: Imagen original de CT, etiqueta de especialista y comparativa entre las inferencias obtenidas para 2.000, 10.000 y 25.000 iteraciones

pruebas.

En este caso, para ejecutar la inferencia con Google Colab hemos comprobado que no es necesario tener la versión Google Colab PRO, con la estándar gratuita parece que funciona aunque no podemos asegurar que esto sea así en el futuro si las condiciones de Google Colab cambiaran.

El proceso del cuaderno se resumen en un *setup* del entorno, la instalación de las librerías de MONAI en la última versión estable que en el momento de la realización de la presente memoria es la 1.1.0, la importación de las librerías que van a hacer falta, tanto de MONAI como otras como matplotlib, el setup de los directorios de trabajo donde por un lado se establece la conexión con Google Drive y por otro se le indica la ruta donde están los modelos entrenados, la definición del modelo entrenado que se va a usar puesto que es necesario para las inferencias, la carga y almacenamiento en caché de los datos originales que se van a inferir y por último la ejecución

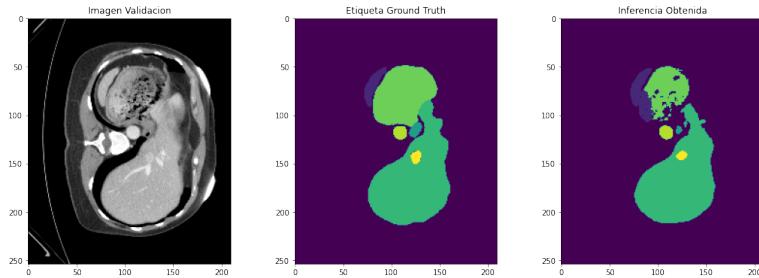


Figura 3.21: Resultado de inferencia usando el modelo anteriormente generado de 25.000 iteraciones

de la inferencia donde se le indica el modelo entrenado que se va a usar y se muestra por pantalla el resultado de la misma.

Además de poder hacer inferencias con modelos que hemos entrenado nosotros, podemos usar modelos que hayan entrenado terceras personas. Por ejemplo, dentro de MONAI Model Zoo, <https://monai.io/model-zoo.html>, encontramos un modelo llamado 'Swin unetr btcv segmentation'. Según se puede ver en la descripción, este modelo se ha entrenado con la red de Swin-Unetr usando un optimizador de Adam y consiguiendo un DICE de 0.8269 que es superior a los obtenidos por nosotros anteriormente entrenando el modelo con UNETR.

Podemos pues descargar este modelo y probarlo en nuestro cuaderno de Jupyter, aunque tenemos que hacer algunos pequeños cambios como en este caso definir ahora el modelo con Swin-UNETR que quedaría de esta forma:

```

1 # Modelo de SwinUNETR para el modelo preentrenado de model.pt
2 model = SwinUNETR(
3     img_size=(96, 96, 96),
4     in_channels=1,
5     out_channels=14,
6     feature_size=48,
7     use_checkpoint=True,
8 ).to(device)

```

Listing 3.6: Uso del modelo de Swin-UNETR

Además de indicarle la ruta y el nombre del modelo descargado, que en nuestro caso hemos alojado en Google Drive en la misma carpeta de resultados de los modelos entrenados por nosotros.

El resultado es el que se puede ver en la Figura 3.22 y que es resultado de la inferencia de la imagen de entrada con la red descargada previamente entrenada de Swin-UNETR a simple vista nos puede parecer muy similar al que obtuvimos con UNETR y 25.000 iteraciones, pero es ligeramente

superior y más preciso.

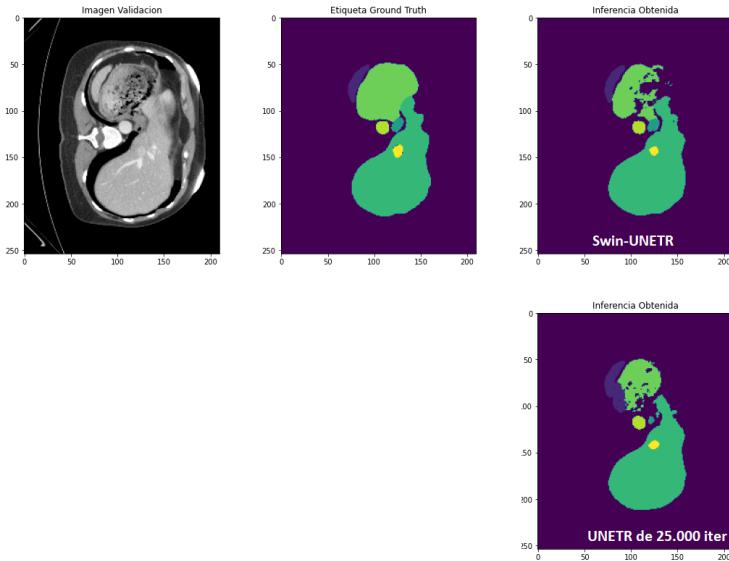


Figura 3.22: Resultado de inferencia usando el modelo Swin-UNETR

3.4. Entrenamiento fino con Transfer Learning

Hasta ahora hemos visto como entrenar una red desde cero y como utilizar una red preentrenada para inferir el resultado de nuevas imágenes según el modelo preentrenado.

En esta sección lo que vamos a tratar de ver es como utilizar una red preentrenada con otros datos para que a través de un *transfer learning* (transferencia de aprendizaje) la entrenemos de forma mas fina o precisa con nuestros datos de entrenamiento y observar como se consigue para el mismo número de iteraciones una métrica de validación superior que si entramos la red completamente desde cero con lo que conseguimos un modelo que es capaz de segmentar de forma más precisa (o también lo que es lo mismo, como podemos alcanzar el mismo valor de métrica de evaluación para un menor número de iteraciones con el consiguiente ahorro de tiempo de entrenamiento).

Para este caso, se ha creado un cuaderno Jupyter que está disponible en el repositorio Github del autor y también se incluye como Apéndice A.3. El proceso que se sigue es muy similar al ya descrito para el entrenamiento desde cero, con la gran diferencia que antes de entrenar el modelo con los datos del *dataset* procedemos a cargar los pesos del modelo preentrenado a

través de la siguiente instrucción:

```

1 # Cargamos los pesos del preentrenado Swin-UNETR
2 weight = torch.load(os.path.join(root_dir, "model_swinvit.pt"))
3 model.load_from(weights=weight)
4 print("Usando pesos del backbone Swin UNETR preentrenados auto-
supervisados")

```

Listing 3.7: Carga de pesos de modelo preentrenado

El modelo preentrenado proviene de pesos de preentrenamiento auto-supervisado del codificador Swin UNETR (Transformador Swin 3D) de un grupo de 5050 escaneos de tomografías computarizadas (TC) de conjuntos de datos disponibles públicamente. El codificador se preentrena usando tareas previas de reconstrucción, predicción de rotación y aprendizaje contrastivo como se describe en el artículo de Tang *et al.* [63].

Las 5050 imágenes de CT provienen de los *datasets* de *Head and Neck Squamous Cell Carcinoma* (HNSCC), *Lung Nodule Analysis* 2016 (LUNA 16), TCIA CT *Colonography*, TCIA Covid 19 y TCIA LIDC. Es decir, datasets con imágenes distintas a las de BTCV con las que vamos a hacer el entrenamiento fino a partir de los pesos del modelo preentrenado con esos datasets.

Para ejecutar el cuaderno de Jupyter se necesita la versión de Google Colab PRO además de seleccionar una GPU Premium que tiene 40GB de GPU ya que el proceso requiere una media de 20 GB de GPU y con la versión estándar de Google Colab se produciría un error por desbordamiento de memoria.

Por ejemplo, para el caso de un entrenamiento de 2000 iteraciones, utilizando la red preentrenada obtenemos un coeficiente DICE de 0.5575 como puede observarse en la Figura 3.23, mientras que para la red desde cero, para 2000 iteraciones obteníamos un DICE de 0.1323 tal y como señala la Figura 3.14.

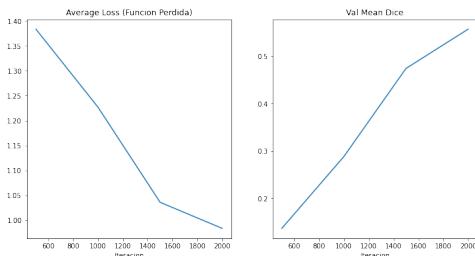


Figura 3.23: Función de Pérdida y Métrica DICE para entrenamiento de 2.000 iteraciones con red pre-entrenada

El resultado pues que obtenemos de segmentación para el entrenamiento con la red preentrenada como puede verse en la Figura 3.24 es de mayor calidad y precisión al obtenido con el mismo número de iteraciones de la red desde cero que se muestra en la Figura 3.15.

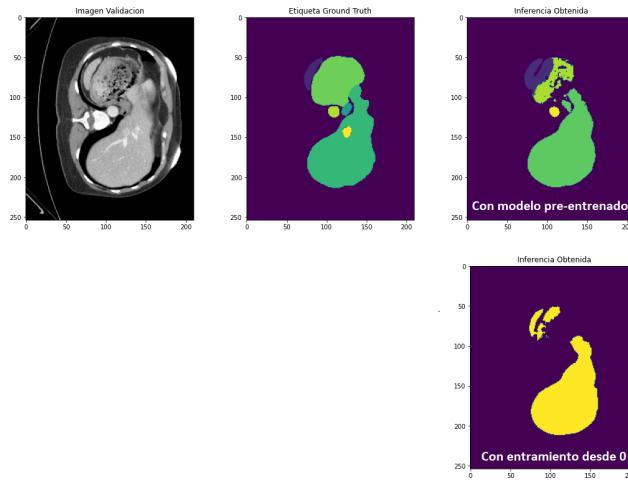


Figura 3.24: Imagen original de CT, etiqueta de especialista y comparativa red preentrenada y desde cero. 2000 iteraciones.

De igual forma ocurre cuando el número de iteraciones es muy alto, para un entrenamiento de 25.000 iteraciones obtenemos un DICE de 0.8430 bastante superior al DICE de 0.7839 que se obtenía con la red con entrenamiento desde cero para el mismo número de iteraciones.

Se comprueba, por tanto, que para el mismo número de iteraciones respecto a entrenamiento desde cero, se obtiene una métrica de evaluación superior. Obteniendo por tanto mayor precisión en la segmentación cuando estamos considerando un alto número de iteraciones. Se obtiene en total mejor DICE si se usa redes preentrenadas que con un entrenamiento desde cero para el mismo número de iteraciones.

3.5. Solución integral para segmentación a gran escala

Como hemos visto, la segmentación de imágenes médicas 3D es una tarea importante con un gran potencial para la comprensión clínica, el diagnóstico de enfermedades y la planificación quirúrgica. Según las estadísticas de las recientes conferencias MICCAI, más del 60 % de los artículos son aplicaciones de algoritmos de segmentación y más de la mitad de ellos utilizan conjuntos de datos 3D.

Hemos visto en los apartados anteriores como podemos entrenar desde cero nuestros propios modelos de segmentación o incluso también como hacer uso de modelos preentrenados para tomar sus pesos y reentrenarlos con los datos objetivos para conseguir una mejor métrica de valoración. Existen una gran variedad de modelo de arquitecturas de redes de segmentación y algunos de esto se adaptan mejor que otros según el tipo de *dataset*.

Si tuviésemos que hacer un entrenamiento por cada modelo para ir comprobando la métrica de evaluación que obtenemos sería un proceso tedioso es por lo que se hace muy recomendable disponer de una solución integral para segmentación a gran escala.

Por ello, proponemos una solución basada en el componente Auto3DSeg de MONAI que aprovecha los últimos avances en MONAI y GPUs para desarrollar y desplegar eficientemente algoritmos con un rendimiento de última generación de una manera simple y con un mínimo de entrada por parte del usuario.

Esta solución integral analiza primero la información global, como la intensidad, el tamaño de datos y el espacio de datos del conjunto de datos, y luego genera carpetas de algoritmos en formato MONAI Bundle basadas en las estadísticas de datos y plantillas de algoritmos. A continuación, todos los algoritmos inician el entrenamiento de modelos para obtener puntos de control con la mejor métrica de validación. Finalmente, el módulo de ensamble selecciona los algoritmos a través del ranking de puntos de control entrenados y crea predicciones de ensamble. Con esto ya tenemos nuestro mejor modelo creado a partir de un conjunto de algoritmos que podemos definir. La Figura 3.25 muestra la arquitectura descrita para la solución integral basada en Auto3DSeg [64].

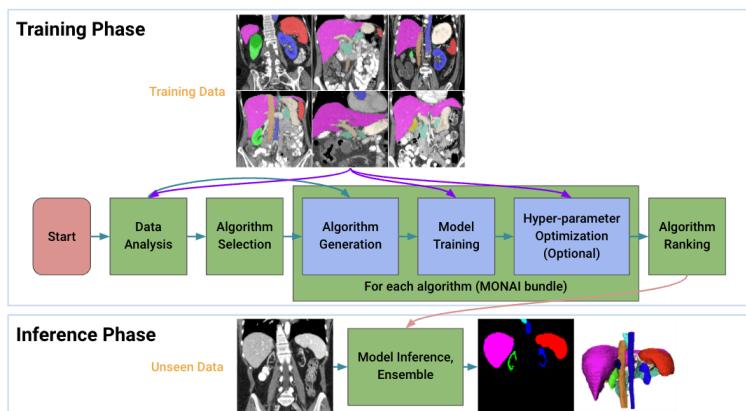


Figura 3.25: Solución integral para Segmentación basada en MONAI Auto3DSeg

La solución permite a cualquier usuario entrenar modelos que puedan segmentar rápidamente regiones de interés en datos de modalidades de imágenes 3D como CT y MRI. Con solo unas líneas de código usando la opción *AutoRunner* permite obtener un modelo de segmentación altamente preciso. Al centrarse en la precisión e incluir modelos de última generación como Swin UNETR, DiNTS y SegResNet, se pueden utilizar los últimos y mejores algoritmos para maximizar su productividad.

Se ha probado en grandes conjuntos de datos de imágenes médicas 3D en varias modalidades diferentes y ha sido pieza fundamental en los últimos *Challenge* consiguiendo el primer puesto en MICCAI 2022 *challenge* HEKTOR (Head and neck tumor segmentation and outcome prediction in PET/CT images o los segundos puestos en los *challenge* de MICCAI 2022 de ISLES (*Ischemic Stroke Lesion Segmentacion Challenge*) y INSTANCE (*Intracranial Hemorrhage Segmentation Challenge on NOnContrast head CT*).

Entre las principales características de la solución integral de Auto3DSeg cabe destacar que posee un marco unificado, es decir, es una solución autónoma para la segmentación de imágenes médicas 3D con un mínimo de entrada por parte del usuario. Implementa un diseño modular flexible, es decir, los componentes se pueden utilizar de forma independiente para satisfacer las diferentes necesidades de los usuarios. Se pueden usar tanto los algoritmos preexistentes como UNET, UNETR, Swin-UNETR, DiNTS, etc. como permite también la introducción de algoritmos propios en lo que se ha llamado BYOA (*Bring Your Own Algorithm*). Y por último pero no menos importante, consigue una alta precisión y eficiencia logrando un rendimiento de última generación en la mayoría de las aplicaciones de la segmentación de imágenes médicas 3D.

Como hemos comentado, el marco de trabajo permite ejecutar con unas pocas líneas de código usando el *AutoRunner* que internamente desencadena todos los procesos o también ir llamando a los distintos componentes o bloques funcionales a través de API. En las siguientes secciones vamos a ver cada una de estas opciones donde las explicaremos y pondremos ejemplos con cuadernos Jupyter ejecutados sobre Google Colab.

3.5.1. *Autorunner*

Autorunner realiza el análisis de datos, generación de algoritmos, entrenamiento y ensamblado de modelos de la solución integral de segmentación de un solo paso simplemente configurando una mínima entrada de código. Ejecutar con entrada mínima usando *Autorunner*.

Entre estos datos a proporcionar se encuentran la modalidad, el *datalist* y la ruta donde están los datos. Estos datos se pueden pasar en un fichero

YAML simple o como variable input de *AutoRunner* como se muestra a continuación

```

1 runner = AutoRunner(
2     work_dir=work_dir,
3     input={
4         "modality": "MRI",
5         "datalist": datalist_file,
6         "dataroot": dataroot_dir,
7     },
8 )

```

Listing 3.8: Paso de parámetros a AutoRunner

En la modalidad, se incluye de que tipo son los datos. Actualmente Auto3DSeg es compatible con CT y MRI (MRI monomodal o multimodal). En el *datalist* se incluye el archivo JSON que contiene la estructura de los archivos de entrenamiento, pruebas y etiquetas y como estos se dividen en los distintos grupos de entrenamiento. El *dataroot* contiene la ruta a la raíz de donde se encuentran físicamente los datos.

Para probar y validar *AutoRunner*, se ha creado un cuaderno de Jupyter que se ejecuta sobre Google Colab el cual se adjunta como Apéndice A.4 de este trabajo, además de estar disponible para su descarga y ejecución desde el repositorio de github del autor. La ruta viene en el apéndice.

Para ejecutarlo en principio es posible con la versión normal de Google Colab, aunque no podemos asegurar que funcione siempre por lo que aconsejamos el uso de la versión de Google Colab PRO con el entorno de ejecución de GPU y el tipo estándar de GPU.

El cuaderno proporciona una demostración simple de cómo utilizar Auto3DSeg *AutoRunner* para procesar un conjunto de datos simulados y generar resultados en muy pocos minutos. En este caso se trabaja con datos simulados y muy pocas épocas de entrenamiento para ver el resultado del proceso y los pasos que realiza sin importarnos demasiado la calidad de precisión del resultado obtenido por la inferencia.

El proceso que sigue el cuaderno es el siguiente: Se realiza un *setup* del entorno y de las librerías de *imports* que van a ser necesarias. Se crea el conjunto de datos simulados y el *datalist* JSON que proporciona la información de los conjuntos de datos simulados. En él se listan 12 imágenes de entrenamiento y 2 de pruebas. Los datos de entrenamiento se dividen en 3 grupos. Cada grupo utilizará 8 imágenes para entrenar y 4 para validar. Cada conjunto de validación solo tiene 4 imágenes en un grupo de entrenamiento. A continuación se procede a la generación de dichas imágenes y etiquetas y si almacenamiento en el directorio de trabajo definido. Inspeccionamos dichas imágenes y etiquetas creadas y ya pasamos a la ejecución del *AutoRunner*.

AutoRunner automáticamente configurará cuatro redes neuronales diferentes y realizará un entrenamiento multigrupo para lograr el mejor rendimiento. *Autorunner* realiza el análisis de datos, generación de algoritmos, entrenamiento y ensamblado de modelos. Por último inspeccionamos los resultados de inferir las predicciones de una imagen de prueba.

Como podemos ver en la Figura 3.26, el resultado de la predicción para un entrenamiento de 2 épocas es de muy baja calidad de precisión.

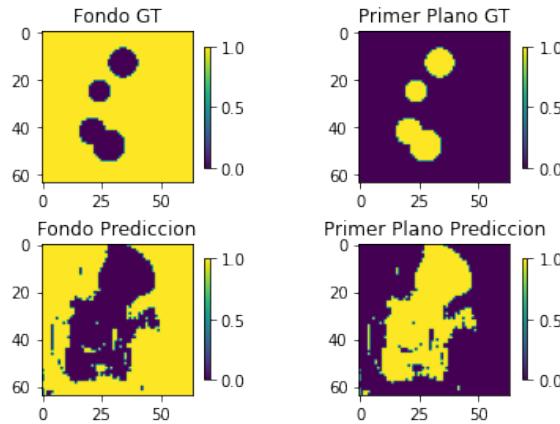


Figura 3.26: Resultado de predicción para un entrenamiento de 2 épocas con Autorunner

Si volvemos a ejecutar el cuaderno, pero esta vez con un entrenamiento de 200 épocas en lugar de sólo 2 épocas, al cabo de casi 1 hora que tarda el entrenamiento, como vemos en la Figura 3.27 la imagen obtenida como resultado de la predicción es prácticamente idéntica al *Ground Truth* por lo que la métrica de evaluación DICE está muy cercana a 1.

Centrándonos en el proceso que ha ejecutado el *Autorunner*, podemos ver como inicialmente realiza el análisis de datos y guarda la información en el fichero 'datastats.yaml' que mostramos resumidamente en la Figura 3.28. 'datastats.yaml' es un resumen del conjunto de datos del analizador de datos. El informe de resumen incluye información como el tamaño de los datos, el espaciado, la distribución de intensidad, etc., para una mejor comprensión del conjunto de datos.

La carpeta 'algorithm_templates' contiene las plantillas de algoritmo utilizadas para generar carpetas de paquetes de algoritmo reales con información de las estadísticas de datos.

Podemos ver también en la Figura 3.29 las carpetas de los 4 algoritmos que ha generado *Autorunner*, que son segresnet2d, segresnet, swinunetr y dints. Son carpetas autónomas, que se pueden utilizar para el entrenamiento de modelos, la inferencia y la validación a través de la ejecución de comandos

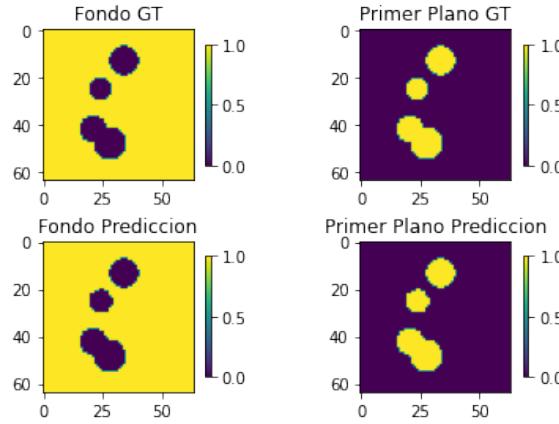


Figura 3.27: Resultado de predicción para un entrenamiento de 200 épocas con Autorunner

```

1 stats_summary
2 image_stats
3 shape:
4   mean: [64, 64, 64]
5   median: [64.0, 64.0, 64.0]
6   min: [64, 64, 64]
7   max: [64.0, 64.0, 64.0]
8   percentile_00_5: [64.0, 64.0, 64.0]
9   percentile_05_5: [64.0, 64.0, 64.0]
10 percentile_50_5: [64.0, 64.0, 64.0]
11 percentile_95_5: [64.0, 64.0, 64.0]
12 percentile_99_5: [64.0, 64.0, 64.0]
13 percentile_99_5: [64.0, 64.0, 64.0]
14 percentile_99_5: [64, 64, 64]
15 percentile_99_5: [64, 64, 64]
16 percentile_99_5: [64, 64, 64]
17 percentile_99_5: [64, 64, 64]
18 percentile_99_5: [64, 64, 64]
19 chanevals:
20   mean: 1.0
21   median: 1.0
22   min: 1.0
23   max: 1.0
24   percentile: [1, 1, 1, 1]
25   percentile_00_5: 1
26   percentile_05_5: 1
27   percentile_50_5: 1
28   percentile_95_5: 1
29   percentile_99_5: 1
30   crossplot_shown: 1
31   mean: [64.0, 56.0, 57.0]
32   median: [64.0, 56.0, 57.0]
33   min: [64, 56, 57]

```

Figura 3.28: Análisis de datos realizado por Autorunner

de cada carpeta de paquete y entrando en una cualquiera de ellas podemos ver la estructura de 'model_foldx' que es donde se guardan los puntos de control después del entrenamiento junto con el historial de entrenamiento y los archivos de eventos de *tensorboard* como el fichero progress.yaml donde va almacenando la mejor métrica DICE conseguida (estamos viendo una captura del entrenamiento de 200 épocas, por eso podemos ver como mejora el DICE con el entrenamiento) y va guardando el modelo en 'best_metric_model.pt' para luego ensamblarlo con los otros modelos y conseguir el mejor resultado posible que se almacenaran en la carpeta de 'ensemble_output'. Por defecto, se selecciona el mejor modelo/algoritmo de cada pliegue para el ensamble.



Figura 3.29: 4 Algoritmos generados por Autorunner y detalle del progreso

3.5.2. Ejecutar con las API del módulo

El proceso de *AutoRunner* también puede ser ejecutado a través de un API en la que se van llamando a los distintos módulos o pasos de los que se compone la solución integral de segmentación y que se representan en la Figura 3.30.

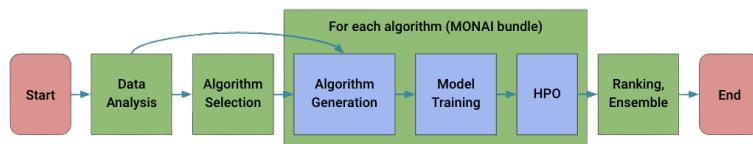


Figura 3.30: Módulos de la solución integral de auto3DSeg

Cada módulo de la solución integral de segmentación basada en Auto3DSeg en diferentes pasos se puede usar individualmente para diferentes propósitos. Y se pueden personalizar las funciones/métodos en los componentes.

Vamos a describir brevemente en qué consisten cada uno de los pasos y las principales funcionalidades que se podrían usar:

Paso 1: Analizador de datos (*Data Analyzer*)

Este módulo proporciona un informe de análisis exhaustivo utilizando la clase *DataAnalyzer*. El *DataAnalyzer* analiza automáticamente el conjunto de datos de imágenes médicas dado y informa las estadísticas. El módulo espera las rutas de archivo a los datos de imágenes y utiliza la transformación LoadImage para leer los archivos, que admite formatos nii, nii.gz, png, jpg, bmp, npz, npy y dcm.

La Figura 3.31 muestra de forma esquemática la entrada y salida de este módulo.

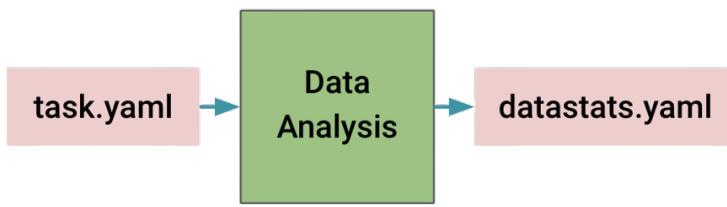


Figura 3.31: Módulo de Análisis de Datos de la Solución Integral de Segmentación

El informe generado por *DataAnalyzer* incluye información como el tamaño de los datos, el espaciado, la distribución de intensidad, etc., para una mejor comprensión del conjunto de datos.

Se puede usar tanto para datos simulados como para datos reales. Para ejecutar *DataAnalyzer* se hace mediante el siguiente código o incluso se puede hacer en CLI con un script de shell.

```

1 # En datalist_file debe estar la descripción de los archivos,
  # normalmente en formato JSON
2 # En dataroot la ruta donde se encuentran físicamente los
  # archivos a analizar
3
4 analyser = DataAnalyzer(datalist_file, dataroot)
5 datastat = analyser.get_all_case_stats()
  
```

Listing 3.9: Ejecución de DataAnalyzer dentro de un cuaderno Jupyter

Paso 2: Generación de algoritmos

El módulo de generación de algoritmos es para crear carpetas de algoritmos autónomas para un entrenamiento posterior del modelo, inferencia y validación con varias arquitecturas de redes neuronales y recetas de entrenamiento. Este módulo toma archivos de configuración '.yaml' de entrada, los resúmenes de conjunto de datos (por ejemplo, 'data_stats.yaml') del análisis de datos de *DataAnalyzer* y plantillas de algoritmos tal y como refleja la Figura 3.32.

El proceso crea diferentes carpetas de algoritmos con validación cruzada. En el diseño por defecto, las carpetas de algoritmos generadas siguen los diseños del paquete MONAI *Bundle*. Se ejecuta dentro del módulo el entrenamiento del modelo, la inferencia y la validación dentro de esas carpetas autónomas.

Los archivos de configuración de entrada y los resúmenes de conjunto de datos son críticos para la generación de algoritmos. Por ejemplo, la

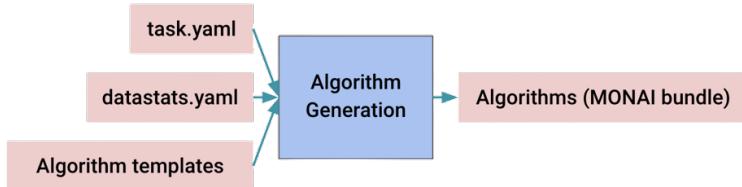


Figura 3.32: Generación de Algoritmos del proceso de la Solución Integral de Segmentación

modalidad de datos es importante para las estrategias de normalización de intensidad, la forma promedio de la imagen determina el recorte de la región de interés (ROI) de la imagen y los canales de entrada/salida deciden las primeras/últimas capas de la red.

Paso 3: Entrenamiento, validación e inferencia de modelos

Se incluyen cuatro algoritmos predeterminados (DiNTS, 2D/3D SegResNet, SwinUNETR) como algoritmos de línea base de Auto3DSeg. Cada algoritmo incluye arquitecturas de red avanzadas y recetas de entrenamiento de modelos bien afinadas. Los algoritmos están formulados en el formato de paquete MONAI *Bundle*, con una estructura de carpetas independiente, proporcionando las funciones necesarias para el entrenamiento de modelos, verificación e inferencia.

Dentro de la estructura de carpetas de cada algoritmo, hay una carpeta *configs* que proporciona la configuración para diferentes operaciones. Esta configuración incluye los hiperparámetros, la red, la función de pérdida, el optimizador, *data augmentation*, etc. Esta configuración puede ser modificada directamente sobre los ficheros .YAML que se encuentran dentro de la carpeta o bien usando la función *'fill_template_config'* en el algoritmo como muestra el siguiente código.

```

1 class DintsAlgo(BundleAlgo):
2     def fill_template_config(self, data_stats_file, output_path
3         , **kwargs):
4         ...
5             patch_size = [128, 128, 96]
6             max_shape = data_stats["stats_summary#image_stats#
7             shape#max"]
8             patch_size = [
9                 max(32, shape_k // 32 * 32) if shape_k < p_k
10                else p_k for p_k, shape_k in zip(patch_size, max_shape)
11            ]
12
13             input_channels = data_stats["stats_summary#
```

```

11     image_stats#channels#max"]
12         output_classes = len(data_stats["stats_summary#
13         label_stats#labels"])
14
15         hyper_parameters.update({"data_file_base_dir": os.
16             path.abspath(data_src_cfg["dataroot"])})
17
18     ...

```

Listing 3.10: Modificación de los datos de entrenamiento por defecto

Se podrían incluir también nuevos algoritmos y nuestros propios algoritmos siguiendo la estructura de MONAI *Bundle*.

Paso 4: Optimización de hiperparámetros (HPO)

Se pueden cambiar los algoritmos de Auto3DSeg en Optimización de Hiperparámetros (HPO) proporcionando un conjunto de parámetros de reemplazo. Por ejemplo, para reducir las épocas/iteraciones de entrenamiento, se pueden modificar num_epochs o num_iterations, en función de los algoritmos elegidos. Los algoritmos BundleAlgo en Auto3DSeg siguen los formatos de paquete MONAI Bundle. Por lo tanto, se pueden proporcionar claves para modificar el valor en la configuración del paquete.

```

1 override_param = {
2     "num_epochs": <num_epoch>,
3     "num_epochs_per_validation": <num_epoch_val>,
4 }

```

Listing 3.11: Optimización de Hiperparámetros

Paso 5: Ensamblado del modelo conjunto

Para lograr predicciones robustas para los datos no vistos (aquellos que no han sido utilizados para el entrenamiento), la solución integral de segmentación Auto3dSeg proporciona un módulo de ensamblaje de modelos para resumir las predicciones de varios modelos entrenados conforme a lo señalado en la Figura 3.33.

El módulo primero clasifica los puntos de control de diferentes algoritmos en función de la precisión de validación en cada pliegue de la validación cruzada N-pliegues, selecciona los mejores M algoritmos de cada pliegue y crea predicciones de ensamblaje utilizando MN puntos de control.

El algoritmo de ensamblaje por defecto promedia los mapas de probabilidad (de las activaciones de *softmax/sigmoid*) de las predicciones candidatas y genera las salidas finales.

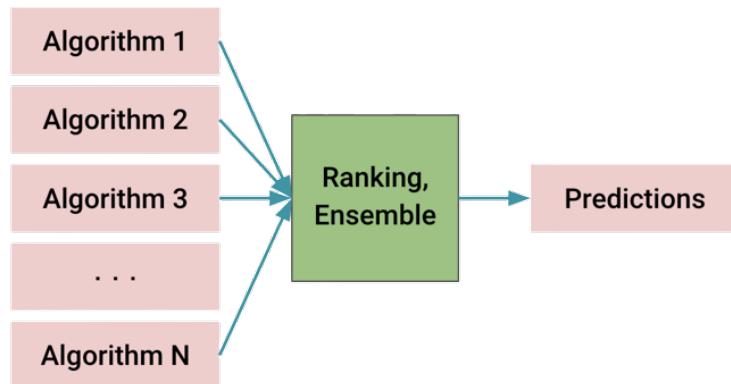


Figura 3.33: Ensamblado del proceso de la Solución Integral de Segmentación

El componente esencial para el ensamblaje de modelos es la función 'infer()' en la clase 'InferClass'. La función infer() toma los nombres de archivo de imágenes como entrada y genera mapas de probabilidad de múltiples canales como salida. Y el 'infer.py' de diferentes algoritmos se encuentra dentro de sus plantillas de paquete. En general, el módulo de ensamblaje funcionará para cualquier algoritmo siempre y cuando la función 'infer()' se proporcione con la configuración adecuada.

Podemos definir nuestro propio ensamblador a partir de la herencia de la clase AlgoEnsemble y luego ejecutarlo.

```

1 class MyAlgoEnsemble(AlgoEnsemble):
2     """
3         Randomly select N models to do ensemble
4     """
5
6     def __init__(self, n_models=3):
7
8         super().__init__()
9         self.n_models = n_models
10
11    def collect_algos(self):
12
13        collect_algos defines the method to collect the target
14        algos from the self.algos list
15        """
16
17        n = len(self.algos)
18        if self.n_models > n:
19            raise ValueError(f"Number of loaded Algo is {n},
20                             but {self.n_models} algos are requested.")
21
22        indexes = list(range(n))
23        random.shuffle(indexes)
24        indexes = indexes[0 : self.n_models]
  
```

```

22     self.algo_ensemble = []
23     for idx in indexes:
24         self.algo_ensemble.append(deepcopy(self.algos[idx]))
25     )
26
27 ...
28 builder = AlgoEnsembleBuilder(history, input_cfg)
29 builder.set_ensemble_method(MyAlgoEnsemble())
30 ensemble = builder.get_ensemble()
31 preds = ensemble()
32
33 print("The ensemble randomly picks the following models:")
34 for algo in ensemble.get_algo_ensemble():
35     print(algo[AlgoEnsembleKeys.ID])

```

Listing 3.12: Definición de Ensamblador

3.6. Testing de herramientas de ayuda al diagnóstico clínico

Como parte del trabajo realizado, por último se ha realizado un *testing* de herramientas de ayuda al diagnóstico clínico que integre las técnicas de segmentación y visibilice la gran importancia y aplicación real de este tipo de herramientas en el mundo real y sus ventajas que aporta.

Se ha seleccionado MONAI *Label* y su integración con el visor de 3D Slicer. En la sección 3.1.2 explicamos en que consiste MONAI *Label* y sus principales integraciones. Ahora vamos a ver como lo hemos implementado y los principales resultados de las pruebas realizadas.

Lo primero que hay que destacar es que necesitamos un equipo que tenga una GPU/CUDA de al menos 8GB. En la página oficial de MONAI *Label* <https://docs.monai.io/projects/label/en/latest/installation.html> explican varias formas en las que se puede llevar a cabo la instalación para Windows o para Ubuntu instalándolo a través de PyPI o de Dockerhub, pero en cualquiera de los casos se hace necesario disponer de suficiente GPU.

Como en el caso del autor no se dispone de un equipo que tenga una GPU mínima para poder ejecutar el servidor de MONAI *Label*, se ha recurrido a dos alternativas: Por un lado, probar la instalación de MONAI *Label server* en Google Colab y disponer del cliente 3D Slicer en el propio equipo y por otro lado hacer una instalación completa en un servidor de AWS en la nube. Los dos siguientes apartados describirán ambas formas y los problemas encontrados así como ventajas e inconvenientes.

3.6.1. Ejecución de MONAI Label Server con Google Colab

Al no disponer de GPU suficiente en el equipo del autor, en primer lugar se pensó en desplegar el servidor de MONAI sobre Google Colab y el visor de 3D Slicer en el equipo del autor.

Respecto a la instalación de 3D Slicer y del *plugin* de MONAI *Label*, no han surgido ningún problema siguiendo las instrucciones que se detallan en la documentación oficial de la propia web de slicer (<https://download.slicer.org/>) y para descargar e instalar el *plugin* desde las instrucciones de la web de monai *label* (<https://docs.monai.io/projects/label/en/latest/installation.html#guide-of-visualization-tools>)

Respecto a la ejecución de MONAI *Label server* en Google Colab no se ha encontrado ninguna documentación al respecto, ni oficial ni de nadie que lo haya intentado previamente.

El autor ha generado un cuaderno de Jupyter que se puede ejecutar en Google Colab para ejecutar dicho servidor de MONAI *Label* con la App de *Radiology* y el *Dataset* de MSD de la Task09_Spleen. Dicho cuaderno instala todo lo necesario y descarga la App y el *dataset*. En el Apéndice A.5 se encuentra el cuaderno Jupyter generado así como el enlace al repositorio de github del autor desde donde se puede descargar dicho cuaderno.

Es importante comentar que el cuaderno genera un proxy para poder conectarse al servidor, aunque este proxy solo es accesible desde un navegador de Chrome que tenga iniciada la sesión con el mismo usuario que el cuaderno de Google Colab.

Por lo tanto, cuando intentamos acceder desde el cliente de 3D Slicer de nuestro equipo al servidor desplegado en Google Colab, no es posible y nos da error.

El servidor desplegado es solo accesible mediante Google Chrome por lo que nos vale para hacer pruebas de su API y conocerlo más en profundidad su funcionamiento sin necesidad de hacer otros gastos, pero no podemos conectarlo con el cliente de 3D Slicer para hacer pruebas de testing.

La Figura 3.34 muestra como desde un navegador de chrome con la misma sesión iniciada que lo ha hecho el cuaderno de Google Colab puede acceder al API del servidor de MONAI Label que se está ejecutando sobre Google Colab.

3.6.2. Ejecución en la Nube de AWS

Otra solución alternativa que hemos realizado para poder probar MONAI *Label* y su integración con el visor de 3D Slicer ha consistido en realizar un despliegue completo tanto del servidor como del cliente del visor en un equipo

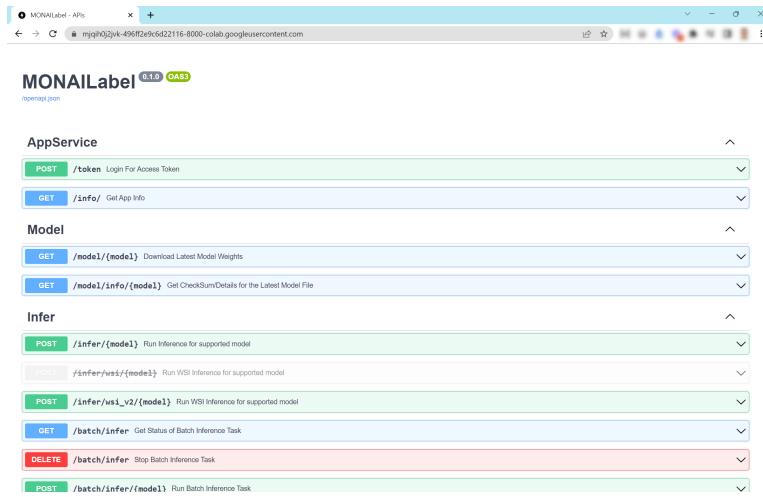


Figura 3.34: Ensamblado del proceso de la Solución Integral de Segmentación

en la nube de AWS (Amazon Web Services).

El principal inconveniente de esta opción es el coste que supone puesto que necesitamos una instancia de AWS del tipo EC2 Windows 'g' *instance* que consume muchos recursos de vCPU y GPU y que tiene un coste cercano a los 2€ por cada hora que la instancia está en funcionamiento.

Lo primero que hay que tener en cuenta es que normalmente las instancias EC2 de tipo 'g' no están disponibles para todos los usuarios, suelen estar restringidas por el coste que supone levantar 'por error' una instancia de este tipo, por lo que lo primero que hay que hacer es solicitar al servicio de soporte de AWS que se habilite este tipo de instancia para la cuenta de login que se haya usado a través del incremento de cuota de GPU.

Una vez que tenemos habilitada la posibilidad de levantar instancias de ese tipo 'g' podemos iniciarla. Para optimizar el proceso, se hace uso de *Stack template* de *CloudFormation* que ha sido diseñado para instalar automáticamente los drivers de NVIDIA, git, MONAI Label server, TotalSegmentator, lungmask, el instalador de 3D Slicer, Firefox y S3bucket con lo que cuando la máquina inicie ya tengamos listos todos estos componentes.

El fichero YAML que contiene el *Stack template* se puede descargar desde el repositorio de github del autor https://github.com/franguerrero/TFM_SegmentacionImagenesMedicas/tree/main/monailabel_on_aws o bien desde la URL https://projectweek.na-mic.org/PW38_2023_GranCanaria/Projects/SlicerCloud/HowToSetupAWSEC2Server.html que además contiene las instrucciones acerca de como desplegar el Stack en AWS.

De forma resumida el proceso consiste en lo siguiente: se hace *log in* en AWS y se selecciona la región. Se busca el servicio de CloudFormation y se

106 3.6. Testing de herramientas de ayuda al diagnóstico clínico

le da a *Create Stack*, aquí se sube el fichero YAML con el *stack* de MONAI comentado anteriormente y se selecciona una instancia de tipo g4 y se pone la IP del equipo personal con el que te conectarás a la instancia de EC2 de AWS. Se aceptan las condiciones y en unos 20-25 minutos la instancia está lista.

Una vez que la instancia está levantada, se entra de forma remota desde la DCVwebConsole en la pestaña de output que lo que hace es realizar una conexión de escritorio remoto. En el escritorio remoto lo primero que debemos hacer es instalar el *slicer-setup* que veremos descargado como se muestra en la Figura 3.35.

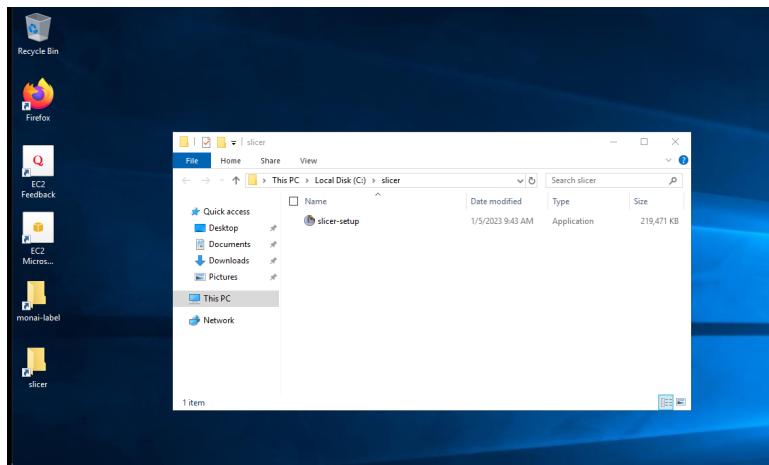


Figura 3.35: Imagen del escritorio remoto de la máquina EC2 en AWS con el 3D Slicer para ser instalado

Respecto a la instalación de 3D Slicer y del *plugin* de MONAI Label, no han surgido ningún problema siguiendo las instrucciones que se detallan en la documentación oficial de slicer y para descargar e instalar el *plugin* desde las instrucciones de la web de monai label (<https://docs.monai.io/projects/label/en/latest/installation.html#guide-of-visualization-tools>)

Vamos a comentar un poco en detalle un caso de uso de MONAI Label y luego listaremos y pondremos imágenes de los distintos casos de uso que se han probado.

En primer lugar hemos probado el caso de uso de *Radiology* con el *dataset* de MSD de Task09_Spleen. Para ello, una vez que hemos instalado el slicer abrimos una ventana de comandos de windows e introducimos las órdenes para descargar la app de *radiology* y el dataset de Task09_Spleen tal y como se muestra en la Figura 3.36 y ha continuación se pone las instrucciones para arrancar el servidor con esa APP y ese *dataset*.

Una vez que está arrancado el servidor de MONAI Label, en el 3D Slicer

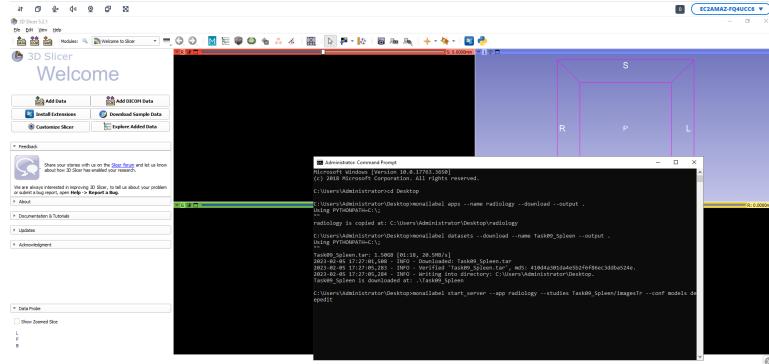


Figura 3.36: Maquina en AWS - MONAI Label - Descarga de APP y *Dataset* para MONAI Label

pasamos a conectar el *plugin* de MONAI Label con el servidor, para ello en primer lugar rellenamos la dirección donde está el MONAI Label server que en nuestro caso es en la misma máquina de AWS por lo que ponemos 'http://127.0.01:8000' y le damos a refrescar. Automáticamente se nos rellena el *App Name* y todos aquellos campos que tengan opciones como Model.

Para visualizar cualquier imagen de prueba le damos a *Next*. Inicialmente veremos la imagen tal cual, como se muestra a la izquierda en la Figura 3.37, si queremos lanzar una segmentación automática, seleccionaremos el modelo y le damos a '*Run*' y cuando el servidor de MONAI Label server termina de hacer la inferencia veremos el resultado en pantalla como se puede apreciar en la misma Figura 3.37 a la derecha que es la misma imagen pero ya con la segmentación superpuesta realizada.

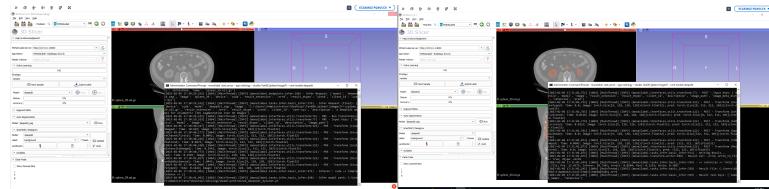


Figura 3.37: Maquina en AWS - MONAI Label - Ejecución de Segmentación

Si además queremos ver como sería la segmentación en 3D, en el submenu de '*Segment Editor*' podemos pulsar sobre '*Show 3D*' y tal y como muestra la Figura 3.38 podemos ver la visualización 3D de los elementos segmentados automáticamente.

Este *plugin* de MONAI Label permitiría ahora realizar ediciones de la segmentación por si queremos editar algún defecto que puedan observar los especialistas para dejar una segmentación más precisa y al volver a ejecutar el modelo iniciaría un proceso de *Active Learning* donde aprende a refinar el

108 3.6. Testing de herramientas de ayuda al diagnóstico clínico

modelo de segmentación a partir de las correcciones manuales que le hacen los usuarios.

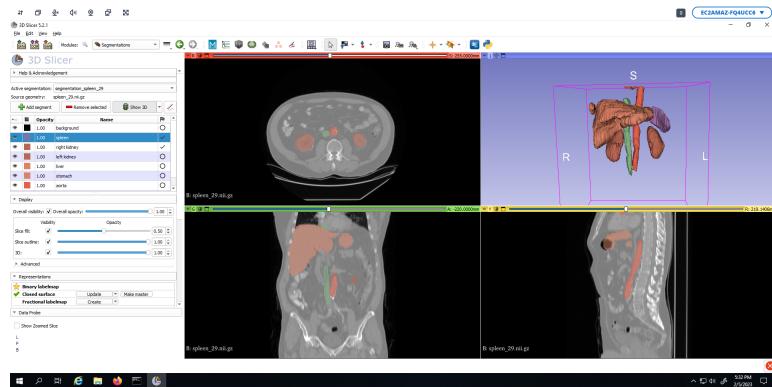


Figura 3.38: Maquina en AWS - MONAI Label - Visualización 3D de la segmentación

Un esquema parecido de carga de la APP en MONAI Label server, descarga de *dataset* y visualización en 3D Slicer se repite para otros casos de usos distintos usando otros modelos y otros conjuntos de datos distintos. Las posibilidades son muy grandes e incluso podríamos poner nuestro propio modelo y nuestro propio *dataset*.

A modo resumen vamos a ir viendo los casos de uso más destacados y significativos que hemos podido probar.

Pulmón y vías respiratorias

Este caso de uso es referente a la segmentación de los pulmones y la vía respiratoria (*Lung and Airway*).

El *dataset* que se usa es el de MSD Task06_Lung que contiene 87 volúmenes anotados y consta de 3 segmentos correspondientes al pulmón derecho, pulmón izquierdo y vías respiratorias. Se utiliza la arquitectura UNet. Los datos se pueden descargar de la siguiente manera:

```
1 monailabel datasets --download --name Task06_Lung --output datasets
```

Listing 3.13: Descarga dataset para Pulmon y Vias

En la Figura 3.39 se muestra una captura de este caso de uso con la imagen 3D.

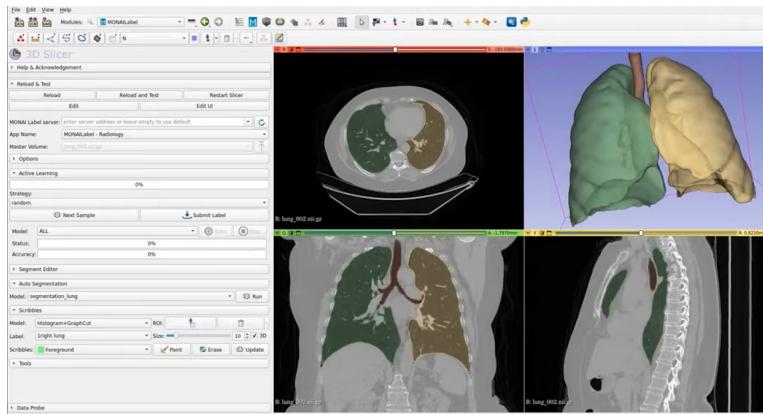


Figura 3.39: MONAI Label - Caso de Uso probado - Pulmón y vías respiratorias

Columna Vertebral y Vértebras

Este es un ejemplo de un enfoque multietapa para la segmentación de varias estructuras en una imagen de TC. El modelo tiene tres etapas que se pueden usar juntas o de forma independiente:

1. Etapa 1: Localización de la columna vertebral: Como su nombre indica, esta etapa localiza la columna vertebral como una sola etiqueta.
2. Etapa 2: Localización de vértebras: Esta imagen utiliza la salida de la primera etapa, recorta el volumen alrededor de la columna vertebral y segmenta aproximadamente las vértebras.
3. Etapa 3: Segmentación de vértebras: Por último, esta etapa toma la salida de la segunda etapa, calcula los centroides y luego segmenta cada vértebra a la vez tal y como muestra la Figura 3.40

El modelo está preentrenado con el *dataset* de VerSe que está disponible en <https://github.com/anjany/verse>. Este *dataset* contiene 60 volúmenes 3D de CT. Utiliza una arquitectura de red basada en el modelo SegResNet.

Para iniciar el MONAI Label server con este modelo se hace de la siguiente forma:

```
1 monailabel start_server --app workspace/radiology --studies
   workspace/images --conf models localization_spine,
   localization_vertebra,segmentation_vertebra
```

Listing 3.14: Inicio de MONAI Label server para Columna y Vértebras

110 3.6. Testing de herramientas de ayuda al diagnóstico clínico

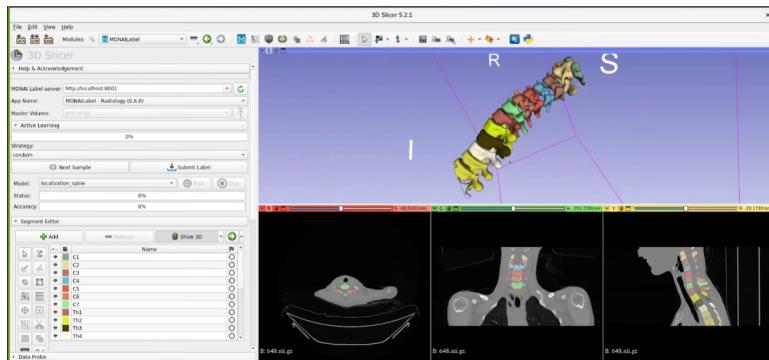


Figura 3.40: MONAI Label - Caso de Uso probado - Columna Vertebral y Vértebras

Cuerpo Completo

Este caso de uso es muy novedoso y de muy reciente aparición. Se trata de un modelo basado en *TotalSegmentator* (<https://github.com/wasserth/TotalSegmentator>) que es capaz de segmentar el cuerpo entero incluyendo 104 estructuras anatómicas (todos los órganos abdominales, huesos, vasos más grandes, músculos, etc.).

Entre sus principales características se encuentra que es muy robusto, es decir puede segmentar cualquier imagen de TC de todo el cuerpo, abdominal o de tórax, independientemente de la resolución de la imagen y el campo de visión y es muy rápido con un tiempo de cálculo a resolución completa es de 1 a 2 minutos en una GPU compatible con CUDA; y aproximadamente 1 minuto en modo de baja resolución en CPU.

El entrenamiento de la red se ha realizado a partir de 1204 volúmenes 3D del dataset <https://zenodo.org/record/6802614#.Y81vvBXP1EY> utilizando una arquitectura de red con el modelo SegResNet. Los resultados que se obtienen son espectaculares como puede verse en la Figura 3.41.

24 órganos

Este caso de uso es una versión simplificada del anterior, pero entrenado para obtener mayor precisión de ciertos órganos. En este caso solo se tiene en cuenta la segmentación de 24 órganos. Se usa el mismo dataset que anteriormente, <https://zenodo.org/record/6802614#.Y81vvBXP1EY> pero en este caso solo se tienen en cuenta 400 volúmenes. Se utiliza una arquitectura SegResNet.

La Figura 3.42 muestra un ejemplo de los resultados que podemos obtener.

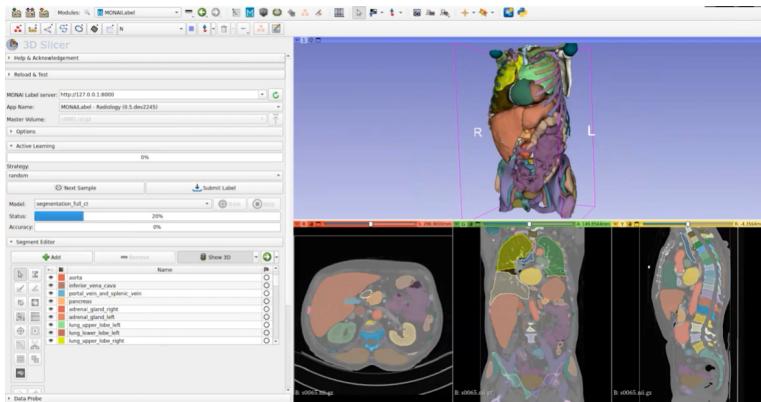


Figura 3.41: MONAI Label - Caso de Uso probado - Cuerpo Completo

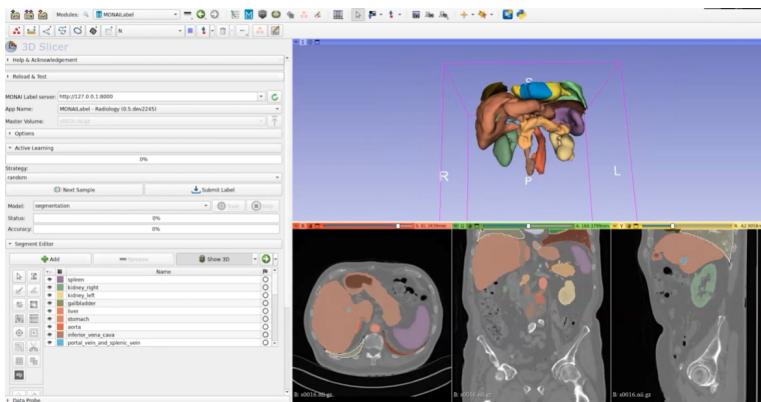


Figura 3.42: MONAI Label - Caso de Uso probado - 24 órganos

Cerebro Completo

La segmentación detallada del cerebro completo es una técnica cuantitativa esencial en el análisis de imágenes médicas, que brinda una forma no invasiva de medir regiones cerebrales a partir de una imagen de resonancia magnética (MRI) estructural adquirida clínicamente.

Para este último caso de uso del Cerebro Completo se ha hecho uso del MONAI Model Zoo y se ha cogido una red preentrenada, en concreto wholeBrainSeg_UNEST_segmentation que consiste en un modelo preentrenado con UNEST para la segmentación volumétrica (3D) del cerebro completo con imágenes T1w MR con el objetivo de entrenar e inferir la segmentación del cerebro completo con 133 estructuras. Se proporciona una línea de entrenamiento para apoyar el aprendizaje activo en MONAI Label y el entrenamiento con bundle.

Los datos de entrenamiento provienen de la Universidad Vanderbilt y

112 3.6. Testing de herramientas de ayuda al diagnóstico clínico

del Centro Médico Vanderbilt con los conjuntos de datos OASIS y CANDI publicados. Los datos de entrenamiento y prueba son volúmenes 3D MRI T1 ponderados (T1w) provenientes de 3 sitios diferentes. Hay un total de 133 clases en la tarea de segmentación del cerebro completo.

Los resultados de la segmentación de las 133 estructuras cerebrales pueden verse en la Figura 3.43

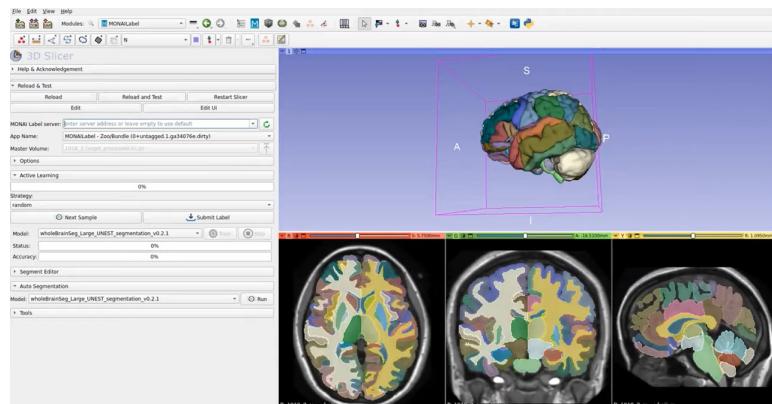


Figura 3.43: MONAI Label - Caso de Uso probado - Cerebro Completo

Se pueden usar muchos de los paquetes que tenemos en MONAI Model Zoo, aunque no todos ya que tienen que cumplir ciertas restricciones. En concreto tenemos disponible los siguientes paquetes del MONAI Model Zoo que podríamos usar en MONAI Label Server y están definidos en la Figura 3.44:

Bundle	Model	Objects	Modality	Note
spleen_ct_segmentation	UNet	Spleen	CT	A model for (3D) segmentation of the spleen
swin_unetr_btcv_segmentation	SwinUNETR	Multi-Organ	CT	A model for (3D) multi-organ segmentation
prostate_mri_anatomy	UNet	Prostate	MRI	A model for (3D) prostate segmentation from MRI image
pancreas_ct_dints_segmentation	DINTS	Pancreas/Tumor	CT	An automl method for (3D) pancreas/tumor segmentation
renalStructures_UNEST_segmentation	UNesT	Kidney Substructure	CT	A pre-trained for inference (3D) kidney cortex/medulla/pelvis segmentation
wholeBrainSeg_UNEST_segmentation	UNesT	Whole Brain	MRI T1	A pre-trained for inference (3D) 133 whole brain structures segmentation
spleen_dedeepit_annotation	DeepEdit	Spleen	CT	An interactive method for 3D spleen Segmentation
lung_nodule_ct_detection	RetinaNet	Lung Nodule	CT	The detection model for 3D CT images

Figura 3.44: MONAI Label - Paquetes de MONAI Model Zoo disponibles

Capítulo 4

Conclusiones y trabajo futuro

El presente capítulo realiza una reflexión crítica de la experiencia obtenida durante el desarrollo del trabajo. Se evalúa el cumplimiento de los objetivos planteados y se identifican los aspectos que se han logrado y aquellos que requieren mejoras o ajustes. Además, se presentan las posibilidades y propuestas para el trabajo futuro, con el fin de continuar investigando y mejorando los resultados obtenidos en este trabajo.

4.1. Reflexión crítica resultante de la experiencia del trabajo realizado

Hemos visto la importancia de llevar a cabo estudios en la segmentación de imágenes orientados al desarrollo de herramientas de ayuda al diagnóstico clínico con el objetivo de mejorar la calidad y eficiencia de la atención médica. Estos estudios pueden ayudar a desarrollar nuevas técnicas de segmentación, evaluar la eficacia de las técnicas existentes, establecer protocolos para su uso clínico y desarrollar herramientas de ayuda al diagnóstico clínico, a la planificación de procedimientos quirúrgicos, a la cirugía guiada por imágenes, a planificar y administrar radioterapia de manera más precisa o a estudiar el desarrollo de enfermedades y el efecto de los tratamientos.

Para conseguir todo esto es necesario todo el conocimiento que hemos compilado y resumido en el capítulo del estado del arte y hemos visto como empezar a ponerlo en práctica en el apartado del trabajo desarrollado con el desarrollo de cuadernos jupyter que contienen el entrenamiento de un modelo desde cero, con la solución íntegra de segmentación a gran escala y con la integración de herramientas de ayuda al diagnóstico como 3dSlicer

con estos modelos para obtener segmentaciones que faciliten todo lo expuesto anteriormente.

Por un lado hemos observado que el campo de la segmentación de imágenes médicas es muy amplio, hay muchas modalidades de imágenes, muchos posibles objetivos a estudiar (radiologías, patologías, endoscopia ,etc. muchos algoritmos de segmentación, muchos datasets, etc. por lo que es muy importante tener la visión general que ofrece esta memoria.

Por otro lado hemos descubierto una comunidad abierta de una red de imágenes médicas llamada MONAI y formada por empresas y universidades que está avanzando mucho en el campo de la imagen médica y comparten los avances y conocimiento con la comunidad. Tratan de estandarizar y marcar las mejores prácticas para la inteligencia artificial (IA) aplicada a las imágenes médicas. Es altamente recomendable seguirlos y ver todos los avances que van publicando.

También hemos comprobado lo costoso computacionalmente que resulta todo el tema de entrenamiento o inferencias. Se necesitan equipos muy potentes con GPU muy buenas para poder investigar en este tema. Afortunadamente existen recursos en la nube a unos precios relativamente asequibles como Google Colab o instancias de AWS que nos pueden servir para momentos específicos, pero que para una unos trabajos de investigación más profundos se requiere tener de alguna forma acceso a ese hardware sin el coste que suponen las plataformas en la nube.

Por último destacar que dentro de la IA se están produciendo unos avances exponenciales en el último año, que cada vez hay una comunidad más creciente interesada en la IA, en la Visión por Computador y en las Imágenes Médicas por lo que es un ámbito en pleno crecimiento y expansión que va a seguir así los próximos años por la repercusión además a nivel mundial y los beneficios que podemos obtener los seres humanos de este tipo de tecnología aplicada en nuestra salud.

4.2. Cumplimiento de los objetivos del trabajo

La sección 1.2 de la memoria define los objetivos generales del trabajo y los objetivos específicos. En la presente sección analizaremos cada uno de los objetivos específicos e identificaremos el grado de cumplimiento y aquellos aspectos más destacables.

4.2.1. Objetivo 1 - Revisión modalidades y técnicas de segmentación tradicionales

Para cumplir este objetivo se ha revisado numerosa literatura a partir de la cual se ha elaborado la sección 2.1 y 2.2 de la presente memoria.

Para el caso de las modalidades, se han seleccionado las más importantes como la TC, SPECT, PET y RM y descrito las principales características físicas de cada una de ellas así como para que casos resulta más adecuado usar una modalidad u otra. Se mencionan brevemente otras modalidades que existen además de las comentadas anteriormente y se describe el concepto de multimodalidad.

Referente a los métodos y técnicas de segmentación, se definen los principales métodos como los de intensidad de la imagen, detección de bordes, clustering y deformación y se explican las técnicas como la segmentación por región, por contorno, por característica de la imagen, semiautomática y basada en modelos. Se explica en qué consisten y cuándo son útiles aplicar.

Consideramos que hemos cumplido el objetivo que nos habíamos puesto ya que aunque siempre se puede seguir profundizando más, la idea era tener un conocimiento mínimo de base que nos sirviera para entender todo lo demás y conocer la base del funcionamiento de la segmentación aplicada a imágenes médicas.

4.2.2. Objetivo 2 - Estudio de redes neuronales de aprendizaje profundo

Para cumplir este objetivo se ha revisado una gran cantidad de artículos con información acerca de las principales redes de aprendizaje profundo desde los inicios a las últimas novedades.

Además no nos hemos conformado con simplemente leer los artículos y resumirlos sino que hemos hecho un estudio a conciencia puesto que necesitábamos comprender el funcionamiento de todas estas redes para poder seguir avanzando y poder en un futuro diseñar y probar redes de segmentación de imágenes médicas más precisas y eficientes.

El resultado de este objetivo por un lado se plasma en la sección 2.3 de la presente memoria y también es conocimiento que ha sido empleado en la parte de desarrollo para programar los entrenamientos con las redes neuronales profundas.

Hemos, por tanto, estudiado, explicado su funcionamiento y documentado las principales redes como las redes neuronales convolucionales, las *fully convolutional neural networks*, las redes U-Net, las redes generativas adversarias (GAN), la EfficientNet, y las redes más actuales como la SegResNet,

TransUNet que utiliza *Transformers*, DiNTS que utiliza una topología de redes neuronales diferenciables y Swin-UNETR que utiliza también *transformers* como *encoder*.

Consideramos que hemos cumplido con el objetivo marcado puesto que hemos hecho un repaso exhaustivo de las redes más importantes hasta las más actuales, comprendiendo su funcionamiento y posteriormente aplicándolo en el desarrollo del trabajo.

4.2.3. Objetivo 3 - Recopilación de conjuntos de datos de imágenes médicas

Para cumplir con este objetivo se ha hecho una intensa búsqueda en internet acerca de los datasets que se encontraban disponibles, bien a través de páginas especializadas o bien mirando en retos y competiciones.

El resultado de este objetivo se plasma en la sección 2.4 de la memoria donde se recogen los principales datasets que hemos encontrado entre los que se incluyen los de *Medical Segmentation Decathlon* (MSD), *Brain tumor segmentation* (BRaTS), *Beyond the Cranial Vault* (BTCV), *Ischemic stroke lesion segmentation* y otros.

Para cada uno de ellos se proporciona información básica como el número de imágenes de la que está compuesto, en qué modalidad, cuál es el objetivo del *dataset*, en qué URL está disponible, etc. Además toda esta información se ha compilado en dos cuadros resumen que son el Cuadro 2.1 centrado en los datasets de MSD y el Cuadro 2.2 con el resumen de todos los datasets.

Consideramos que hemos cumplido suficientemente con este objetivo porque si bien sabemos que existen algunos otros datasets disponibles en abierto, los que hemos descrito son los que hemos considerado más interesantes y que luego hemos usado alguno de ellos en el desarrollo del trabajo. Por tanto, el listado de *dataset* no pretende ser un listado completo y podría ser en el futuro ampliado incluyendo más datasets de los disponibles u otros que se liberen.

4.2.4. Objetivo 4 - Investigación técnicas de preprocesamiento y aumento de datos

Para cumplir con este objetivo se ha analizado y revisado bastante documentación y publicaciones y como resultado se ha elaborado la Sección 2.5 y 2.6 de la presente memoria.

En cuanto al preprocesamiento se han estudiado las técnicas de preprocesamiento como corrección de la distorsión, realce de la imagen, etc. y se ha enfatizado en el preprocesamiento orientado a redes neuronales profundas

con los principales pasos como la limpieza, corrección, normalización, etc. de imágenes.

Para el aumento de datos se han explicado transformaciones más comunes como la rotación, traslación, escalado, reflejo, cambio de contraste, *warp*, superposición y ruido.

Considerado que hemos cumplido suficientemente con el objetivo de entender las técnicas puesto que son necesarias para el ciclo de vida del entrenamiento de un modelo de segmentación de imágenes médicas al ser muy habitual tener que preprocesar las imágenes y aplicarles *data augmentation* para tener un *dataset* más amplio con el cual entrenar el modelo para que no generalice y sea más preciso y eficiente.

4.2.5. Objetivo 5 - Revisión de métricas de evaluación

Para cumplir con este objetivo se han estudiado varios artículos específicos sobre métricas además de ver de forma general las que se repiten cuando se publica un nuevo modelo de segmentación al compararlo con los anteriores.

Como resultado se ha generado la sección 2.7 de la presente memoria donde se recoge un listado de todas las métricas de evaluación relevantes y luego se explica en detalle y de una forma muy gráfica y visual para que sea fácilmente entendible las más importantes entre las que se encuentran las cardinalidades básicas, el coeficiente de similaridad de Dice o *F1-Score*, el índice de Jaccard o IoU, la distancia Hausdorff. Además se dan unas pautas acerca de que métrica sería más conveniente usar para cada casuística y dependiendo también del objetivo de la segmentación de imágenes médicas.

Consideramos que hemos cumplido satisfactoriamente con el objetivo marcado puesto que hemos comprendido en qué consisten las principales métricas y cómo se obtienen y además lo hemos reflejado de forma gráfica para que si tenemos que recurrir de nuevo a ello por si se olvida sea muy rápido volver a acordarse de en qué consistía la métrica y cómo se calcula.

4.2.6. Objetivo 6 - Búsqueda de comunidades colaboración clínica de imágenes médicas

Para cumplir con este objetivo se ha realizado una búsqueda intensa por internet hasta dar con la principal comunidad de innovación abierta en imágenes médicas que es MONAI. Además de estudiar exhaustivamente su documentación de la web, hemos visto cientos de horas de vídeos que la comunidad tiene publicado en Youtube con seminarios, *bootcamps*, conferencias, etc.

Como resultado se ha generado la sección 3.1 de la presente memoria que contiene una descripción de MONAI y de sus principales herramientas como *MONAI Core*, *MONAI Label*, *MONAI Deploy App SDK* y *MONAI Model Zoo* que han sido nuestra base tecnológica para el trabajo de desarrollo que hemos realizado a continuación.

Consideramos que hemos cumplido con el objetivo que nos habíamos marcado puesto que ahora tenemos un conocimiento muy amplio de la comunidad de MONAI, de las iniciativas y de todas las herramientas de software y librerías que pone al alcance de todo el mundo y que permiten ahorrar horas y horas de desarrollo.

4.2.7. Objetivo 7 - Desarrollo modelo de segmentación de imágenes

Para cumplir con este objetivo hemos desarrollado 3 *notebooks* de Jupyter que hemos desplegado y ejecutado en Google Colab. La explicación de los cuadernos se encuentra en esta memoria en las secciones 3.2, 3.3 y 3.4 y el código fuente de los cuadernos y la ejecución de los mismos se encuentra en los apéndices A.1, A.2 y A.3 además de en el repositorio de github del autor de este trabajo.

Se ha desarrollado un entrenamiento desde cero un modelo de segmentación de imágenes médicas usando las librerías de MONAI y el *dataset* de BTCV. Se explica como se puede reproducir sin necesidad de tener un hardware potente con una GPU de 16GB sino usando recursos de la nube a un precio relativamente asequible. Se explica detalladamente el proceso de entrenamiento hasta obtener el modelo entrenado y luego se detallan los resultados mostrando las diferencias que se producen al entrenar el modelo con distinto número de iteraciones las métricas de precisión que vamos obteniendo así como visualmente el resultado de segmentación que vamos obteniendo comparado con la etiqueta de *ground truth*.

Se desarrolla posteriormente otro cuaderno que directamente puede cargar un modelo entrenado sin necesidad de tener que entrenarlo desde cero y que es capaz de ejecutar inferencias de imágenes nuevas.

Por último para el cumplimiento de este objetivo se desarrolla un cuaderno que incluye como se hace un *transfer learning* de un modelo pre-entrenado con otros datos para ser reentrenado con nuestro *dataset* y conseguir una mayor precisión que si solamente se entrenara el modelo desde cero con nuestro *dataset*.

Consideramos que hemos cumplido sobradamente con el objetivo que nos habíamos marcado, teniendo disponible unos cuadernos jupyter reproducibles y fácilmente entendible que marcan todo el ciclo de entrenamiento

de una red neuronal profunda para segmentación de imágenes médicas incluyendo incluso el *transfer learning*.

4.2.8. Objetivo 8 - Testear solución integral de segmentación

Para cumplir con este objetivo hemos desarrollado un cuaderno de Jupyter que hemos desplegado y ejecutado en Google Colab. La explicación del cuaderno se encuentra en esta memoria en la sección 3.5 y el código fuente del cuaderno y la ejecución de los mismos se encuentra en el apéndice A.4 además de en el repositorio de github del autor de este trabajo.

Se ha desarrollado una solución integral para segmentación a gran escala basada en el componente de Auto3DSeg de MONAI. Esta solución analiza la información global del *dataset*, genera las carpetas donde entrenará todos los algoritmos que le digamos contra ese *dataset* y finalmente ensamblará la mejor solución.

Se ha probado la solución con un *Autorunner* que ejecuta todos los pasos del proceso, pero también se explica componente a componente de la solución integral que es lo que hace y como puede llamar al API.

Consideramos que hemos cumplido perfectamente con el objetivo marcado puesto que hemos conseguido ejecutar y probar una solución muy novedosa que permite optimizar los tiempos de ejecución de todo el ciclo de vida de entrenamientos y obtener un resultado de una red entrenada más eficiente.

4.2.9. Objetivo 9 - Integrar y probar herramienta de ayuda al diagnóstico clínico. Casos de uso

Para cumplir con este objetivo hemos probado con dos soluciones distintas, por un lado intentar tener la herramienta cliente local en nuestro equipo, cosa que hemos podido realizar y tener el servidor en Google Colab, lo cual también hemos conseguido realizar, pero no ha sido posible llamar desde nuestro cliente al servidor de Google Colab por restricciones que impone Google.

Así que hemos probado y optado por otra solución de un despliegue completo en la nube de AWS ante la imposibilidad de tener nosotros un equipo con suficiente GPU para probar el *server*.

En la memoria se documenta todo en la sección 3.6 y con la solución de la nube hemos conseguido probar con éxito numerosos casos de uso y hemos podido visualizar con 3DSlicer el resultado de las segmentaciones automáticas como por ejemplo la segmentación para el pulmón y las vías respiratorias usando el *dataset* de MSD y un modelo UNet, la segmentación de la colum-

na vertebral y de las vértebras, la segmentación del cuerpo completo, la segmentación de 24 órganos y la segmentación del cerebro completo. Todas ellas son espectaculares aunque obviamente algunas son más específicas que otras.

Se ha comprobado la integración entre el cliente y el servidor de segmentación permitiendo incluso un *active learning*.

Como inconveniente es el coste de tener el equipo corriendo en la nube de AWS.

Se podrían probar aun más casos de uso como los distintos paquetes de MONAI MOdel Zoo que son compatibles y que reflejamos en la memoria pero que no hemos podido probar todos por minimizar el coste que nos suponía la utilización de la máquina en la nube de AWS.

Consideramos que hemos cumplido extraordinariamente con el objetivo propuesto puesto que hemos podido salvar las limitaciones de recursos hardware que teníamos y hemos podido ir encontrando soluciones a nuestros problemas para poder probar la integración de herramientas de ayuda al diagnóstico clínico con los modelos de segmentación.

4.3. Trabajo futuro

El campo de la segmentación de imágenes médicas orientadas al desarrollo de herramientas de ayuda al diagnóstico clínico es muy amplio y las posibilidades de mejora se pueden considerar en muchos de los ámbitos.

Con este trabajo se han puesto unas bases sólidas con las que seguir avanzando, se han estudiado y probado la mayoría de los elementos de base para la segmentación de datasets médicos y herramientas de diagnóstico clínico. En este apartado se pretende dar algunas ideas de posibles trabajos futuros en los que se puede investigar aunque como hemos comentado anteriormente las posibilidades son casi ilimitadas.

4.3.1. Uso de otras redes y arquitecturas de segmentación

Hemos visto las principales arquitecturas y modelos de segmentación de imágenes médicas, pero continuamente se publican artículos que mejoran el SOTA (*State-Of-The-Art*) de los anteriores o salen nuevos modelos generales de redes neuronales profundas que pueden ser aplicados al ámbito de la imagen médica.

Un tipo que se está utilizando últimamente en otros ámbitos como la generación de imágenes a través de *prompts* son las redes de difusión. Ejemplos de estos tipos de redes generalistas son StableDiffusion, DaLLE o MidJour-

ney.

Aplicado al ámbito de imágenes médicas, se publicó recientemente un artículo llamado MedSegDiff [65] que aplica estas técnicas de difusión estable y difusión latente con unos resultados bastante prometedores por lo que seguir profundizando en estas técnicas para la segmentación sea más generalizable y efectiva puede resultar de bastante interés.

4.3.2. Uso de otras herramientas de ayuda al diagnóstico clínico

En el presente trabajo hemos visto la herramienta 3d Slicer y su integración con MONAI Label.

Referente a herramientas *open source* para análisis médico nos encontramos con otras herramientas como OHIF (<https://ohif.org/>), QuPath (<https://qupath.github.io/>), CVAT y otras que puede resultar muy interesante su estudio y la integración con MONAI Label y con los algoritmos de segmentación

4.3.3. Aplicación a VR/AR y gemelos digitales

Generar gemelos digitales o aplicar tecnología de realidad virtual y realidad aumentada a los resultados de la segmentación de imágenes médicas nos puede hacer que podamos tener por ejemplo el cerebro de un paciente concreto que tiene algún tipo de cáncer antes de una intervención quirúrgica con lo que el equipo médico puede por un lado planificar con antelación como va a efectuar la intervención y puede también entrenarse previamente con ese cerebro específico para esa intervención específica para hacerla con mayores garantías de éxito para el paciente con la ayuda de la Realidad Virtual.

La Figura 4.1 muestra un gemelo digital del cerebro y posibles aplicaciones de VR/AR para entrenamiento de una intervención quirúrgica.

También sería interesante investigar en el campo de intervenciones quirúrgicas realizadas por un robot con una precisión absoluta como resultado de tener una segmentación totalmente precisa de los órganos.



Figura 4.1: Gemelo Digital y Aplicación de VR/AR

Trabajar en una plataforma para la investigación en tiempo real de IA que involucra la multitud de interfaces, señales, análisis, simulaciones y procedimientos asociados con intervenciones guiadas por imágenes y aplicaciones en el punto de atención.

Integración con sistemas de navegación robótica y quirúrgica como ROS, DVRK, OpenIGTLink y SlicerIGT, para abordar los desafíos asociados con la fusión de IA con rastreadores, sistemas de adquisición de imágenes, robots daVinci, dispositivos AR / VR y una multitud de otros sistemas médicos.

Las tareas incluyen el reconocimiento de eventos quirúrgicos, la reconstrucción en tiempo real en 3D a partir de vídeo endoscópico, el control de cámara colaborativo y la evaluación de habilidades. Una consideración importante es la gestión de recursos GPU que admita múltiples motores de inferencia que funcionen en paralelo, así como métodos para transmitir señales directamente a la memoria GPU.

4.3.4. Despliegues en entornos de producción reales

Definir cómo cerrar la brecha existente entre investigación y desarrollo y los entornos de producción clínicos llevando los modelos de IA a las aplicaciones médicas y los flujos de trabajo clínicos con el objetivo final de ayudar a mejorar la atención al paciente es una labor futura que merece la pena tener presente.

El enfoque puede incluir la definición de la arquitectura funcional de alto nivel abierta y la determinación de qué componentes y APIs estándar se requieren. Así en cómo debería ser la experiencia de extremo a extremo. Algunos de estos desafíos incluyen:

- Empaquetamiento de modelos de IA entrenados en aplicaciones médicas ejecutables
- Inferir ligeramente para validación local
- Orquestación de modelos, implementación, atención y inferencia a través de múltiples nodos de cálculo
- Consumir *pipelines* de datos multimodales, comenzando con imágenes médicas, basadas en DICOM primero, y datos de pacientes provenientes de EHR a continuación.

4.3.5. Centralización de Conjunto de Datos Médicos

Los conjuntos de datos de imágenes médicas son cruciales para poder entrenar correctamente a los algoritmos basados en redes neuronales pro-

fundas. En el presente trabajo hemos identificado los más importantes, pero sería de gran valor disponer de aún más y tenerlos todos centralizados en una base de datos central donde estén los datos públicos e incluso datos propios.

Aunque esto trae importantes retos a tener en cuenta como la privacidad de los pacientes y los recursos que se necesitan para anonimizar la información, además de que los datos se consideran como el nuevo petróleo y lo que aporta realmente valor a las compañías por lo que es complicado que quieran hacer público nuevos conjuntos de datos. Tenemos que tener en cuenta también que las imágenes médicas son pesadas y ocupan bastante espacio por lo que hay que dimensionar adecuadamente una infraestructura que soporte tal cantidad de datos.

4.3.6. Aprendizaje Federado

El aprendizaje federado está emergiendo como un enfoque prometedor para entrenar modelos de IA sin requerir que los sitios compartan datos. Esto es particularmente importante en los casos médicos, ya que existen restricciones de privacidad y reguladoras para compartir datos.

Se están desarrollando diferentes métodos para el aprendizaje federado, y actualmente son esfuerzos aislados con sinergia limitada. Sería deseable unificar los diferentes métodos de aprendizaje federado en un marco común. Esto se logrará a través de casos de uso de aprendizaje federado, así como de la generación de prácticas y documentación óptimas para permitir que la comunidad participe de manera más amplia e implemente enfoques comunes e interoperables.

4.3.7. Patología Digital

La patología digital como modalidad de imagen es un campo relativamente nuevo en comparación con la imagen radiología. La patología digital se centra en las imágenes de lámina completa (WSI), que son escaneos digitales de láminas de tejido a resolución microscópica, a menudo de 250 nm por píxel. Las WSIs son diferentes de las imágenes médicas estándar debido a su tamaño, una imagen patológica típica puede tener un tamaño de 150K x 100K píxeles.

Recientemente, se han producido avances significativos en la aplicación del análisis de imágenes y el aprendizaje automático a las imágenes de patología, también conocido como patología computacional. A pesar de esos avances, no existe una canalización estándar para el preprocesamiento, análisis y visualización de las imágenes de patología. Esta falta de normalización lleva a una barrera elevada de entrada y falta de reproducibilidad de los

métodos existentes.

4.3.8. Participación en *Challenges*

Existen diferentes plataformas que ofrecen retos de análisis de imágenes médicas donde se puede participar y además de aprender muchísimo se pueden ganar premios y sobre todo poner en práctica todos los conocimientos adquiridos.

Una de las web que tienen más retos de análisis de imágenes médicas es Grand Challenge <https://grand-challenge.org/challenges/> donde podemos encontrar retos como:

- Breast Cancer Immunohistochemical Image Generation Challenge
- Cell Segmentation in Multi-modality High-Resolution Microscopy Images
- Prostate Cancer Detection in MRI (PI-CAI)
- Endometrial Carcinoma Detection in Pipelle biopsies
- SNEMI3D: 3D Segmentation of neurites in EM images
- Ischemic Stroke Lesion Segmentation Challenge
- Etc. Etc.

También hay otras plataformas de retos de propósito más general como Kaggle (<https://www.kaggle.com/>) que suele tener también competiciones como *UW-Madison GI Tract Image Segmentation* o *Image Classification of Stroke Blood Clot Origin*.

Participar en una de estas competiciones/retos y generar a partir de ahí nuevos conocimientos específicos de una materia en concreto aplicando todo lo aprendido con este trabajo puede ser un trabajo futuro muy enriquecedor.

Bibliografía

- [1] Z. Ma, J. M. R. Tavares, and R. N. Jorge, “A review on the current segmentation algorithms for medical images,” in *Proceedings of the 1st international conference on imaging theory and applications (IMA-GAPP)*, 2009.
- [2] A. Ferreira, F. Gentil, and J. M. R. Tavares, “Segmentation algorithms for ear image data towards biomechanical studies,” *Computer methods in biomechanics and biomedical engineering*, vol. 17, no. 8, pp. 888–904, 2014.
- [3] Z. Ma, J. M. R. Tavares, R. N. Jorge, and T. Mascarenhas, “A review of algorithms for medical image segmentation and their applications to the female pelvic cavity,” *Computer Methods in Biomechanics and Biomedical Engineering*, vol. 13, no. 2, pp. 235–246, 2010.
- [4] T. Zhou, S. Ruan, and S. Canu, “A review: Deep learning for medical image segmentation using multi-modality fusion,” *Array*, vol. 3, p. 100004, 2019.
- [5] X. Liu, L. Song, S. Liu, and Y. Zhang, “A review of deep-learning-based medical image segmentation methods,” *Sustainability*, vol. 13, no. 3, p. 1224, 2021.
- [6] T. M. Buzug, “Computed tomography,” in *Springer handbook of medical technology*, pp. 311–342, Springer, 2011.
- [7] G. Bhatnagar, Q. J. Wu, and Z. Liu, “A new contrast based multimodal medical image fusion framework,” *Neurocomputing*, vol. 157, pp. 143–152, 2015.
- [8] T. A. Holly, B. G. Abbott, M. Al-Mallah, D. A. Calnon, M. C. Cohen, F. P. DiFilippo, E. P. Ficaro, M. R. Freeman, R. C. Hendel, D. Jain, *et al.*, “Single photon-emission computed tomography,” 2010.
- [9] W. Jagust, R. Thisted, M. Devous, R. Van Heertum, H. Mayberg, K. Jobst, A. Smith, and N. Borys, “SPECT perfusion imaging in the

- diagnosis of alzheimer's disease: a clinical-pathologic study," *Neurology*, vol. 56, no. 7, pp. 950–956, 2001.
- [10] D. W. Townsend, J. P. Carney, J. T. Yap, and N. C. Hall, "Pet/ct today and tomorrow," *Journal of Nuclear Medicine*, vol. 45, no. 1 suppl, pp. 4S–14S, 2004.
 - [11] A. B. Wolbarst, *Looking within: how X-ray, CT, MRI, ultrasound, and other medical images are created, and how they help physicians save lives*. Univ of California Press, 1999.
 - [12] D. M. Yousem, M. P. M. Som, M. D. B. Hackney, F. Schwaibold, and D. R. A. Hendrix, "Mr imaging versus ct'," *Radiology*, vol. 182, no. 753J, p. 59, 1992.
 - [13] J. Rowlands, "The physics of computed radiography," *Physics in medicine & biology*, vol. 47, no. 23, p. R123, 2002.
 - [14] H. J. Koo, S. Lim, J. Choe, S.-H. Choi, H. Sung, K.-H. Do, *et al.*, "Radiographic and ct features of viral pneumonia," *Radiographics*, vol. 38, no. 3, pp. 719–739, 2018.
 - [15] J. M. Provenzale and B. Sarikaya, "Comparison of test performance characteristics of mri, mr angiography, and ct angiography in the diagnosis of carotid and vertebral artery dissection: a review of the medical literature," *American Journal of Roentgenology*, vol. 193, no. 4, pp. 1167–1174, 2009.
 - [16] P. C. Gøtzsche and K. J. Jørgensen, "Screening for breast cancer with mammography," *Cochrane database of systematic reviews*, no. 6, 2013.
 - [17] J. M. Schmitt, "Optical coherence tomography (oct): a review," *IEEE Journal of selected topics in quantum electronics*, vol. 5, no. 4, pp. 1205–1215, 1999.
 - [18] Z. Guo, X. Li, H. Huang, N. Guo, and Q. Li, "Deep learning-based image segmentation on multimodal medical imaging," *IEEE Transactions on Radiation and Plasma Medical Sciences*, vol. 3, no. 2, pp. 162–169, 2019.
 - [19] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
 - [20] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.
 - [21] A. S. Lundervold and A. Lundervold, "An overview of deep learning in medical imaging focusing on mri," *Zeitschrift für Medizinische Physik*, vol. 29, no. 2, pp. 102–127, 2019.

- [22] J. Long, E. Shelhamer, and T. Darrell, “Fully convolutional networks for semantic segmentation,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3431–3440, 2015.
- [23] O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” in *International Conference on Medical image computing and computer-assisted intervention*, pp. 234–241, Springer, 2015.
- [24] “Website: Neural network types.” <https://mriquestions.com/neural-network-types.html>.
- [25] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets,” *Advances in neural information processing systems*, vol. 27, 2014.
- [26] M. Tan and Q. Le, “Efficientnet: Rethinking model scaling for convolutional neural networks,” in *International conference on machine learning*, pp. 6105–6114, PMLR, 2019.
- [27] A. Myronenko, “3d mri brain tumor segmentation using autoencoder regularization,” in *International MICCAI Brainlesion Workshop*, pp. 311–320, Springer, 2018.
- [28] K. He, X. Zhang, S. Ren, and J. Sun, “Identity mappings in deep residual networks,” in *European conference on computer vision*, pp. 630–645, Springer, 2016.
- [29] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.
- [30] J. Chen, Y. Lu, Q. Yu, X. Luo, E. Adeli, Y. Wang, L. Lu, A. L. Yuille, and Y. Zhou, “Transunet: Transformers make strong encoders for medical image segmentation,” *arXiv preprint arXiv:2102.04306*, 2021.
- [31] Y. He, D. Yang, H. Roth, C. Zhao, and D. Xu, “Dints: Differentiable neural network topology search for 3d medical image segmentation,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5841–5850, 2021.
- [32] A. Hatamizadeh, Y. Tang, V. Nath, D. Yang, A. Myronenko, B. Landman, H. R. Roth, and D. Xu, “Unetr: Transformers for 3d medical image segmentation,” in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 574–584, 2022.

- [33] A. Hatamizadeh, V. Nath, Y. Tang, D. Yang, H. R. Roth, and D. Xu, “Swin unetr: Swin transformers for semantic segmentation of brain tumors in mri images,” in *Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries: 7th International Workshop, BrainLes 2021, Held in Conjunction with MICCAI 2021, Virtual Event, September 27, 2021, Revised Selected Papers, Part I*, pp. 272–284, Springer, 2022.
- [34] M. Antonelli, A. Reinke, S. Bakas, K. Farahani, A. Kopp-Schneider, B. A. Landman, G. Litjens, B. Menze, O. Ronneberger, R. M. Summers, *et al.*, “The medical segmentation decathlon,” *Nature communications*, vol. 13, no. 1, p. 4128, 2022.
- [35] A. L. Simpson, M. Antonelli, S. Bakas, M. Bilello, K. Farahani, B. Van Ginneken, A. Kopp-Schneider, B. A. Landman, G. Litjens, B. Menze, *et al.*, “A large annotated medical image dataset for the development and evaluation of segmentation algorithms,” *arXiv preprint arXiv:1902.09063*, 2019.
- [36] B. H. Menze, A. Jakab, S. Bauer, J. Kalpathy-Cramer, K. Farahani, J. Kirby, Y. Burren, N. Porz, J. Slotboom, R. Wiest, *et al.*, “The multimodal brain tumor image segmentation benchmark (brats),” *IEEE transactions on medical imaging*, vol. 34, no. 10, pp. 1993–2024, 2014.
- [37] U. Baid, S. Ghodasara, S. Mohan, M. Bilello, E. Calabrese, E. Colak, K. Farahani, J. Kalpathy-Cramer, F. C. Kitamura, S. Pati, *et al.*, “The rsna-asnr-miccai brats 2021 benchmark on brain tumor segmentation and radiogenomic classification,” *arXiv preprint arXiv:2107.02314*, 2021.
- [38] B. Landman, Z. Xu, J. Igelsias, M. Styner, T. Langerak, and A. Klein, “Miccai multi-atlas labeling beyond the cranial vault—workshop and challenge,” in *Proc. MICCAI Multi-Atlas Labeling Beyond Cranial Vault—Workshop Challenge*, vol. 5, p. 12, 2015.
- [39] Y. Tang, R. Gao, H. H. Lee, S. Han, Y. Chen, D. Gao, V. Nath, C. Bermudez, M. R. Savona, R. G. Abramson, *et al.*, “High-resolution 3d abdominal segmentation with random patch network fusion,” *Medical image analysis*, vol. 69, p. 101894, 2021.
- [40] M. Heath, K. Bowyer, D. Kopans, P. Kegelmeyer, R. Moore, K. Chang, and S. Munishkumaran, “Current status of the digital database for screening mammography,” in *Digital mammography*, pp. 457–460, Springer, 1998.
- [41] M. R. Hernandez Petzsche, E. de la Rosa, U. Hanning, R. Wiest, W. Valenzuela, M. Reyes, M. Meyer, S.-L. Liew, F. Kofler, I. Ezhov, *et al.*,

- “Isles 2022: A multi-center magnetic resonance imaging stroke lesion segmentation dataset,” *Scientific data*, vol. 9, no. 1, p. 762, 2022.
- [42] “Website: Drive - grand challenge.” <https://drive.grand-challenge.org/>.
- [43] B. Van Ginneken, M. B. Stegmann, and M. Loog, “Segmentation of anatomical structures in chest radiographs using supervised methods: a comparative study on a public database,” *Medical image analysis*, vol. 10, no. 1, pp. 19–40, 2006.
- [44] P. Bilic, P. F. Christ, E. Vorontsov, G. Chlebus, H. Chen, Q. Dou, C.-W. Fu, X. Han, P.-A. Heng, J. Hesser, *et al.*, “The liver tumor segmentation benchmark (lits),” *arXiv preprint arXiv:1901.04056*, 2019.
- [45] S. G. Armato III, G. McLennan, L. Bidaut, M. F. McNitt-Gray, C. R. Meyer, A. P. Reeves, B. Zhao, D. R. Aberle, C. I. Henschke, E. A. Hoffman, *et al.*, “The lung image database consortium (lidc) and image database resource initiative (idri): a completed reference database of lung nodules on ct scans,” *Medical physics*, vol. 38, no. 2, pp. 915–931, 2011.
- [46] D. S. Marcus, A. F. Fotenos, J. G. Csernansky, J. C. Morris, and R. L. Buckner, “Open access series of imaging studies: longitudinal mri data in nondemented and demented older adults,” *Journal of cognitive neuroscience*, vol. 22, no. 12, pp. 2677–2684, 2010.
- [47] C. G. Fonseca, M. Backhaus, D. A. Bluemke, R. D. Britten, J. D. Chung, B. R. Cowan, I. D. Dinov, J. P. Finn, P. J. Hunter, A. H. Kadish, *et al.*, “The cardiac atlas project—an imaging database for computational modeling and statistical atlases of the heart,” *Bioinformatics*, vol. 27, no. 16, pp. 2288–2295, 2011.
- [48] Q. Ha, B. Liu, and F. Liu, “Identifying melanoma images using efficientnet ensemble: Winning solution to the siim-isic melanoma classification challenge,” *arXiv preprint arXiv:2010.05351*, 2020.
- [49] M. Sonka, J. M. Fitzpatrick, *et al.*, “Handbook of medical imaging. volume 2, medical image processing and analysis,” in *Handbook of medical imaging. Volume 2, Medical image processing and analysis*, SPIE, 2000.
- [50] K. de Raad, K. van Garderen, M. Smits, S. van der Voort, F. Incekara, E. Oei, J. Hirvasniemi, S. Klein, and M. Starmans, “The effect of preprocessing on convolutional neural networks for medical image segmentation,” in *2021 IEEE 18th International Symposium on Biomedical Imaging (ISBI)*, pp. 655–658, IEEE, 2021.

- [51] L. Perez and J. Wang, “The effectiveness of data augmentation in image classification using deep learning,” *arXiv preprint arXiv:1712.04621*, 2017.
- [52] A. A. Taha, A. Hanbury, and O. A. J. del Toro, “A formal method for selecting evaluation metrics for image segmentation,” in *2014 IEEE international conference on image processing (ICIP)*, pp. 932–936, IEEE, 2014.
- [53] R. Shi, K. N. Ngan, and S. Li, “The objective evaluation of image object segmentation quality,” in *Advanced Concepts for Intelligent Vision Systems: 15th International Conference, ACIVS 2013, Poznań, Poland, October 28-31, 2013. Proceedings 15*, pp. 470–479, Springer, 2013.
- [54] A. Fenster and B. Chiu, “Evaluation of segmentation algorithms for medical imaging,” in *2005 IEEE Engineering in Medicine and Biology 27th Annual Conference*, pp. 7186–7189, IEEE, 2006.
- [55] A. A. Taha and A. Hanbury, “Metrics for evaluating 3d medical image segmentation: analysis, selection, and tool,” *BMC medical imaging*, vol. 15, no. 1, pp. 1–28, 2015.
- [56] A. Reinke, M. D. Tizabi, C. H. Sudre, M. Eisenmann, T. Rädsch, M. Baumgartner, L. Acion, M. Antonelli, T. Arbel, S. Bakas, *et al.*, “Common limitations of image processing metrics: A picture story,” *arXiv preprint arXiv:2104.05642*, 2021.
- [57] “Website: Monai modules overview.” <https://docs.monai.io/>.
- [58] “Website: Monai modules overview.” <https://docs.monai.io/en/0.9.1/highlights.html>.
- [59] A. Diaz-Pinto, S. Alle, A. Ihsani, M. Asad, V. Nath, F. Pérez-García, P. Mehta, W. Li, H. R. Roth, T. Vercauteren, D. Xu, P. Dogra, S. Ourselin, A. Feng, and M. J. Cardoso, “MONAI Label: A framework for AI-assisted Interactive Labeling of 3D Medical Images,” *arXiv e-prints*, 2022.
- [60] “Website: Monai model zoo.” <https://monai.io/model-zoo.html>.
- [61] F. Milletari, N. Navab, and S.-A. Ahmadi, “V-net: Fully convolutional neural networks for volumetric medical image segmentation,” in *2016 fourth international conference on 3D vision (3DV)*, pp. 565–571, Ieee, 2016.
- [62] I. Loshchilov and F. Hutter, “Decoupled weight decay regularization,” *arXiv preprint arXiv:1711.05101*, 2017.

- [63] Y. Tang, D. Yang, W. Li, H. R. Roth, B. Landman, D. Xu, V. Nath, and A. Hatamizadeh, “Self-supervised pre-training of swin transformers for 3d medical image analysis,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 20730–20740, 2022.
- [64] “Website: Monai auto3dseg.” <https://github.com/Project-MONAI/tutorials/tree/main/auto3dseg>.
- [65] J. Wu, H. Fang, Y. Zhang, Y. Yang, and Y. Xu, “Medsegdiff: Medical image segmentation with diffusion probabilistic model,” *arXiv preprint arXiv:2211.00611*, 2022.

Apéndice A

Cuadernos de Jupyter Desarrollados

A.1. Cuaderno: Entrenamiento desde cero

El presente cuaderno de jupyter realiza un entrenamiento desde cero de un modelo de segmentación de imágenes médicas.

Ha continuación se adjunta el código del cuaderno y la ejecución del mismo.

El cuaderno se encuentra disponible en el repositorio Github del autor del TFM, a través de la URL:

[https://github.com/franguerrero/TFM_SegmentacionImagenesMedicas/
blob/main/cuadernos_jupyter/TFM_EntrenamientoDesdeCero_BTCV.ipynb](https://github.com/franguerrero/TFM_SegmentacionImagenesMedicas/blob/main/cuadernos_jupyter/TFM_EntrenamientoDesdeCero_BTCV.ipynb)

▼ Entrenamiento desde cero para segmentación de Imágenes Médicas

Se utiliza el dataset de BTCV que se puede descargar de <https://www.synapse.org/#ISynapse:syn3193805/wiki/217752>.

Este dataset contiene 13 órganos abdominales incluyendo 1. Bazo 2. Riñón derecho 3. Riñón izquierdo 4. Vesícula biliar 5. Esófago 6. Hígado 7. Estómago 8. Aorta 9. VCI 10. Venas portal y esplénica 11. Páncreas 12. Glándula adrenal derecha 13. Glándula adrenal izquierda.

Modalidad: CT (Tomografía Computarizada) Size: 30 3D volumes (24 Training + 6 Testing)

Challenge: BTCV MICCAI Challenge

Este cuaderno puede ser ejecutado en Google Colab, aunque no va a funcionar en la versión gratuita de Google Colab. Se necesita al menos Google Colab PRO con el entorno de ejecución de GPU Estandar y Alta capacidad RAM. Se recomienda el entorno GPU Premium.

Si se ejecuta en local, los requisitos mínimos son GPU de 16GB y Memoria RAM de 24GB.

[Open in Colab](#)

▼ Setup del entorno

Verificamos si está conectado con la GPU y mostramos la información de la GPU

```
gpu_info = !nvidia-smi
gpu_info = '\n'.join(gpu_info)
if gpu_info.find('failed') >= 0:
    print('No conectado a GPU')
else:
    print(gpu_info)

Fri Feb 3 22:06:13 2023
+-----+
| NVIDIA-SMI 510.47.03   Driver Version: 510.47.03   CUDA Version: 11.6 |
+-----+
| GPU  Name      Persistence-M! Bus-Id     Disp.A  | Volatile Uncorr. ECC | | | |
| Fan  Temp  Perf  Pwr:Usage/Cap | Memory-Usage | GPU-Util  Compute M. |
|          |          |          |          |      MIG M. |
|-----+
| 0  NVIDIA A100-SXM... Off | 00000000:00:04.0 Off |          0 | | |
| N/A  30C   P0  52W / 400W |  0MiB / 40960MiB |   0% Default |
|          |          |          |          | Disabled |
+-----+
+-----+
| Processes:                               |
| GPU  GI CI      PID  Type  Process name        GPU Memory |
|   ID  ID          ID          Usage          |
|-----|
| No running processes found               |
+-----+


# Instalamos las librerías de MONAI en su ultima versión estable que en el momento de la realización de este cuaderno es la 1.1.0
!pip install -q "monai[nibabel, tqdm, einops]==1.1.0"
!python -c "import matplotlib" || pip install -q matplotlib
%matplotlib inline

          1.2/1.2 MB 16.0 MB/s eta 0:00:00
          41.6/41.6 KB 3.3 MB/s eta 0:00:00

# Importamos aquellas librerías que nos van a hacer falta tanto de MONAI como otras como de numpy o matplotlib
# Finalizamos viendo las versiones instaladas de cada una de ellas

import os
import shutil
import tempfile

import matplotlib.pyplot as plt
import numpy as np
from tqdm import tqdm

from monai.losses import DiceCELoss
from monai.inferers import sliding_window_inference
from monai.transforms import (
    AsDiscrete,
    EnsureChannelFirstd,
    Compose,
    CropForegroundd,
    LoadImaged,
    Orientationd,
```

```

        RandFlipd,
        RandCropByPosNegLabeld,
        RandShiftIntensityd,
        ScaleIntensityRanged,
        Spacingd,
        RandRotate90d,
    )

from monai.config import print_config
from monai.metrics import DiceMetric
from monai.networks.nets import UNETR

from monai.data import (
    DataLoader,
    CacheDataset,
    load_decathlon_datalist,
    decollate_batch,
)

import torch
print_config()

MONAI version: 1.1.0
Numpy version: 1.21.6
Pytorch version: 1.13.1+cu116
MONAI flags: HAS_EXT = False, USE_COMPILED = False, USE_META_DICT = False
MONAI rev id: a2ec3752f54bfc3b40e7952234f0eb5452ed63e3
MONAI __file__: /usr/local/lib/python3.8/dist-packages/monai/__init__.py

Optional dependencies:
Pytorch Ignite version: NOT INSTALLED or UNKNOWN VERSION.
Nibabel version: 3.0.2
scikit-image version: 0.18.3
Pillow version: 7.1.2
Tensorboard version: 2.9.1
gdown version: 4.4.0
TorchVision version: 0.14.1+cu116
tqdm version: 4.64.1
lmdb version: 0.99
psutil version: 5.4.8
pandas version: 1.3.5
einops version: 0.6.0
transformers version: NOT INSTALLED or UNKNOWN VERSION.
mlflow version: NOT INSTALLED or UNKNOWN VERSION.
pynrrd version: NOT INSTALLED or UNKNOWN VERSION.

For details about installing the optional dependencies, please visit:
https://docs.monai.io/en/latest/installation.html#installing-the-recommended-dependencies

```

▼ Setup directorios de trabajo

Establecemos la conexión con la cuenta de Google Drive y le especificamos la carpeta donde vamos a almacenar los resultados del modelo que obtengamos

```

from google.colab import drive
drive.mount('/content/gdrive')
Mounted at /content/gdrive

# Ruta de Google Drive donde vamos a almacenar los resultados
os.environ['MONAI_DATA_DIRECTORY'] = '/content/gdrive/MyDrive/TFM_SegmentacionImagenesMedicas/Resultados/'

directory = os.environ.get("MONAI_DATA_DIRECTORY")
root_dir = tempfile.mkdtemp() if directory is None else directory
print(root_dir)
/content/gdrive/MyDrive/TFM_SegmentacionImagenesMedicas/Resultados/

```

▼ Setup de las transformaciones para entrenamiento (training) y validacion

```

train_transforms = Compose(
[
    LoadImaged(keys=["image", "label"]),
    EnsureChannelFirstd(keys=["image", "label"]),
    Orientationd(keys=["image", "label"], axcodes="RAS"),
]
)

```

```

Spacingd(
    keys=["image", "label"],
    pixdim=(1.5, 1.5, 2.0),
    mode=("bilinear", "nearest"),
),
ScaleIntensityRanged(
    keys=["image"],
    a_min=-175,
    a_max=250,
    b_min=0.0,
    b_max=1.0,
    clip=True,
),
CropForegroundd(keys=["image", "label"], source_key="image"),
RandCropByPosNegLabeld(
    keys=["image", "label"],
    label_key="label",
    spatial_size=(96, 96, 96),
    pos=1,
    neg=1,
    num_samples=4,
    image_key="image",
    image_threshold=0,
),
RandFlipd(
    keys=["image", "label"],
    spatial_axis=[0],
    prob=0.10,
),
RandFlipd(
    keys=["image", "label"],
    spatial_axis=[1],
    prob=0.10,
),
RandFlipd(
    keys=["image", "label"],
    spatial_axis=[2],
    prob=0.10,
),
RandRotate90d(
    keys=["image", "label"],
    prob=0.10,
    max_k=3,
),
RandShiftIntensityd(
    keys=["image"],
    offsets=0.10,
    prob=0.50,
),
),
),
)
),
val_transforms = Compose(
[
    LoadImaged(keys=["image", "label"]),
    EnsureChannelFirstd(keys=["image", "label"]),
    Orientationd(keys=["image", "label"], axcodes="RAS"),
    Spacingd(
        keys=["image", "label"],
        pixdim=(1.5, 1.5, 2.0),
        mode=("bilinear", "nearest"),
),
    ScaleIntensityRanged(
        keys=["image"], a_min=-175, a_max=250, b_min=0.0, b_max=1.0, clip=True
),
    CropForegroundd(keys=["image", "label"], source_key="image"),
]
)
)

```

▼ Descarga del dataset y almacenamiento en Google Drive

El dataset ha sido descargado de <https://www.synapse.org/#/Synapse:syn3193805/wiki/217752> y se ha subido a la aplicacion de google drive.

En concreto a la ruta \MyDrive\TFM_SegmentacionImagenesMedicas\Datasets\BTCV

Es importante señalar que las imágenes del dataset de BTCV van a la carpeta \textbackslash imagesTr y las etiquetas van a la carpeta \textbackslash labelsTr.

Además también se encuentra el fichero dataset_0.json que contiene la descripción de las imágenes y sus correspondientes etiquetas de ground truth que van a ser usadas para el entrenamiento y las que van a ser usadas para validación. En nuestro caso se han usado 24

imágenes (y correspondientes etiquetas) para entrenamiento y 6 para validación. Las de validación se corresponden con la numeración desde la 35 a la 40 ambas incluidas.

```
# Establecemos la ruta donde se encuentran las imágenes y etiquetas de BTCV descargadas
data_dir = "/content/gdrive/MyDrive/TFM_SegmentacionImagenesMedicas/Datasets/BTCV/"
split_JSON = "dataset_0.json"

datasets = data_dir + split_JSON
datalist = load_decathlon_datalist(datasets, True, "training")
val_files = load_decathlon_datalist(datasets, True, "validation")

# Cacheamos los datos de entrenamiento y validación
train_ds = CacheDataset(
    data=datalist,
    transform=train_transforms,
    cache_num=24,
    cache_rate=1.0,
    num_workers=8,
)
train_loader = DataLoader(
    train_ds, batch_size=1, shuffle=True, num_workers=8, pin_memory=True
)
val_ds = CacheDataset(
    data=val_files, transform=val_transforms, cache_num=6, cache_rate=1.0, num_workers=4
)
val_loader = DataLoader(
    val_ds, batch_size=1, shuffle=False, num_workers=4, pin_memory=True
)

Loading dataset: 100%|██████████| 23/23 [00:17<00:00,  1.33it/s]
Loading dataset: 100%|██████████| 5/5 [00:07<00:00,  1.51it/s]
```

▼ Visualizamos imagen y etiquetas y comprobamos la forma de los datos

```
# Definimos un mapeo de los datos de validación
slice_map = {
    "img0035.nii.gz": 170,
    "img0036.nii.gz": 230,
    "img0037.nii.gz": 204,
    "img0038.nii.gz": 204,
    "img0039.nii.gz": 204,
    "img0040.nii.gz": 180,
}

# Seleccionamos uno cualquiera, por ejemplo el que esta en el lugar 0 del map
case_num = 0
img_name = os.path.split(val_ds[case_num]["image"].meta["filename_or_obj"])[1]
img = val_ds[case_num]["image"]
label = val_ds[case_num]["label"]
img_shape = img.shape
label_shape = label.shape
print(f"forma de la imagen: {img_shape}, forma de la etiqueta: {label_shape}")
print(f"Nombre de la Imagen: {img_name}")
plt.figure("Imagen", (18, 6))
plt.subplot(1, 2, 1)
plt.title("Imagen")
plt.imshow(img[0, :, :, slice_map[img_name]].detach().cpu(), cmap="gray")
plt.subplot(1, 2, 2)
plt.title("Etiqueta Ground Truth")
plt.imshow(label[0, :, :, slice_map[img_name]].detach().cpu())
plt.show()
```

forma de la imagen: torch.Size([1, 314, 214, 234]), forma de la etiqueta: torch.Size([1, 96, 96, 96])
 Nombre de la Imagen: img0035.nii.gz



▼ Definimos el modelo, la función de pérdida y el optimizador

```
os.environ["CUDA_DEVICE_ORDER"] = "PCI_BUS_ID"
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Definimos el modelo de arquitectura de segmentación, en este caso una UNETR
model = UNETR(
    in_channels=1,
    out_channels=14,
    img_size=(96, 96, 96),
    feature_size=16,
    hidden_size=768,
    mlp_dim=3072,
    num_heads=12,
    pos_embed="perceptron",
    norm_name="instance",
    res_block=True,
    dropout_rate=0.0,
).to(device)

# Definimos la función de pérdida
loss_function = DiceCELoss(to_onehot_y=True, softmax=True)
torch.backends.cudnn.benchmark = True

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

▼ Ejecutamos un proceso de entrenamiento de red neuronal típico de PyTorch una vez que previamente hemos definido modelo, función de pérdida y optimizador

```
def validation(epoch_iterator_val):
    model.eval()
    with torch.no_grad():
        for step, batch in enumerate(epoch_iterator_val):
            val_inputs, val_labels = (batch["image"].cuda(), batch["label"].cuda())
            val_outputs = sliding_window_inference(val_inputs, (96, 96, 96), 4, model)
            val_labels_list = decollate_batch(val_labels)
            val_labels_convert = [
                post_label(val_label_tensor) for val_label_tensor in val_labels_list
            ]
            val_outputs_list = decollate_batch(val_outputs)
            val_output_convert = [
                post_pred(val_pred_tensor) for val_pred_tensor in val_outputs_list
            ]
            dice_metric(y_pred=val_output_convert, y=val_labels_convert)
            epoch_iterator_val.set_description(
                "Validar (%d / %d Steps)" % (global_step, 10.0)
            )
        mean_dice_val = dice_metric.aggregate().item()
        dice_metric.reset()
    return mean_dice_val

def train(global_step, train_loader, dice_val_best, global_step_best):
    model.train()
    epoch_loss = 0
    step = 0
    epoch_iterator = tqdm(
        train_loader, desc="Training (X / X Steps) (loss=X.X)", dynamic_ncols=True
    )
    for step, batch in enumerate(epoch_iterator):
        step += 1
        x, y = (batch["image"].cuda(), batch["label"].cuda())
        logit_map = model(x)
        loss = loss_function(logit_map, y)
        loss.backward()
        epoch_loss += loss.item()
        optimizer.step()
        optimizer.zero_grad()
    epoch_iterator.set_description(
        "Training (%d / %d Steps) (Perdida loss=%2.5f)" % (global_step, max_iterations, loss)
```

```

        )
    if (
        global_step % eval_num == 0 and global_step != 0
    ) or global_step == max_iterations:
        epoch_iterator_val = tqdm(
            val_loader, desc="Validar (X / X Steps) (dice=X.X)", dynamic_ncols=True
        )
        dice_val = validation(epoch_iterator_val)
        epoch_loss /= step
        epoch_loss_values.append(epoch_loss)
        metric_values.append(dice_val)
    # Guardamos el modelo que tiene mejor metrica DICE
    if dice_val > dice_val_best:
        dice_val_best = dice_val
        global_step_best = global_step
        torch.save(
            model.state_dict(), os.path.join(root_dir, "mejor_modelo_metrica.pth")
        )
        print(
            "Se ha salvado el modelo ! Mejor Avg. Dice: {} Actual Avg. Dice: {}".format(
                dice_val_best, dice_val
            )
        )
    else:
        print(
            "No se ha salvado el modelo ! Mejor Avg. Dice: {} Actual Avg. Dice: {}".format(
                dice_val_best, dice_val
            )
        )
    global_step += 1
return global_step, dice_val_best, global_step_best

# Numero máximo de iteraciones. Para buenos resultados se recomienda 25000
max_iterations = 2000

# Cada cuantos pasos evaluamos
eval_num = 500
post_label = AsDiscrete(to_onehot=14)
post_pred = AsDiscrete(argmax=True, to_onehot=14)

# Parametrizamos la métrica DICE a usar
dice_metric = DiceMetric(include_background=True, reduction="mean", get_not_nans=False)
global_step = 0
dice_val_best = 0.0
global_step_best = 0
epoch_loss_values = []
metric_values = []

# Mandamos el modelo a entrenar
while global_step < max_iterations:
    global_step, dice_val_best, global_step_best = train(
        global_step, train_loader, dice_val_best, global_step_best
    )

# Cargamos el mejor modelo conseguido
model.load_state_dict(torch.load(os.path.join(root_dir, "mejor_modelo_metrica.pth")))

```

```

Training (1669 / 2800 Steps) (Perdida loss=1.51841): 100% |██████████| 23/23 [00:07<00:00, 3.00it/s]
Training (1678 / 2800 Steps) (Perdida loss=1.72019): 100% |██████████| 23/23 [00:07<00:00, 3.12it/s]
Training (1701 / 2800 Steps) (Perdida loss=1.51839): 100% |██████████| 23/23 [00:07<00:00, 3.97it/s]
Training (1724 / 2800 Steps) (Perdida loss=1.69693): 100% |██████████| 23/23 [00:07<00:00, 3.08it/s]
Training (1747 / 2800 Steps) (Perdida loss=1.44773): 100% |██████████| 23/23 [00:07<00:00, 3.11it/s]
Training (1770 / 2800 Steps) (Perdida loss=1.51677): 100% |██████████| 23/23 [00:07<00:00, 2.98it/s]
Training (1793 / 2800 Steps) (Perdida loss=1.66955): 100% |██████████| 23/23 [00:07<00:00, 3.00it/s]
Training (1816 / 2800 Steps) (Perdida loss=1.45571): 100% |██████████| 23/23 [00:07<00:00, 3.00it/s]
Training (1839 / 2800 Steps) (Perdida loss=1.58925): 100% |██████████| 23/23 [00:07<00:00, 2.92it/s]
Training (1862 / 2800 Steps) (Perdida loss=1.33727): 100% |██████████| 23/23 [00:07<00:00, 3.18it/s]
Training (1885 / 2800 Steps) (Perdida loss=1.46475): 100% |██████████| 23/23 [00:07<00:00, 2.97it/s]
Training (1908 / 2800 Steps) (Perdida loss=1.60801): 100% |██████████| 23/23 [00:07<00:00, 3.03it/s]
Training (1931 / 2800 Steps) (Perdida loss=1.42488): 100% |██████████| 23/23 [00:07<00:00, 3.03it/s]
Training (1954 / 2800 Steps) (Perdida loss=1.43964): 100% |██████████| 23/23 [00:07<00:00, 2.94it/s]
Training (1977 / 2800 Steps) (Perdida loss=1.38371): 100% |██████████| 23/23 [00:07<00:00, 2.98it/s]
Training (2000 / 2800 Steps) (Perdida loss=1.70094): 96% |██████████| 22/23 [00:07<00:00, 4.50it/s]
Validar (X / X Steps) (dice=X.X): 0% |██████████| 0/5 [00:00<, ?it/s]
Validar (1978 / 10 Steps): 0% | 0/5 [00:01<, ?it/s]
Validar (1979 / 10 Steps): 20% |████ 1/5 [00:01<00:05, 1.34it/s]
Validar (1979 / 10 Steps): 20% |████ 1/5 [00:01<00:05, 1.34it/s]
Validar (1978 / 10 Steps): 40% |████████ 2/5 [00:01<00:02, 1.20it/s]
Validar (1978 / 10 Steps): 40% |████████ 2/5 [00:01<00:02, 1.20it/s]
Validar (1978 / 10 Steps): 60% |██████████ 3/5 [00:02<00:01, 1.07it/s]
Validar (1978 / 10 Steps): 60% |██████████ 3/5 [00:02<00:01, 1.07it/s]
Validar (1978 / 10 Steps): 80% |███████████ 4/5 [00:03<00:00, 1.24it/s]
Validar (1978 / 10 Steps): 80% |███████████ 4/5 [00:03<00:00, 1.24it/s]
Validar (1978 / 10 Steps): 100% |███████████ 5/5 [00:04<00:00, 1.17it/s]
Validar (1978 / 10 Steps): 100% |███████████ 5/5 [00:04<00:00, 1.17it/s]

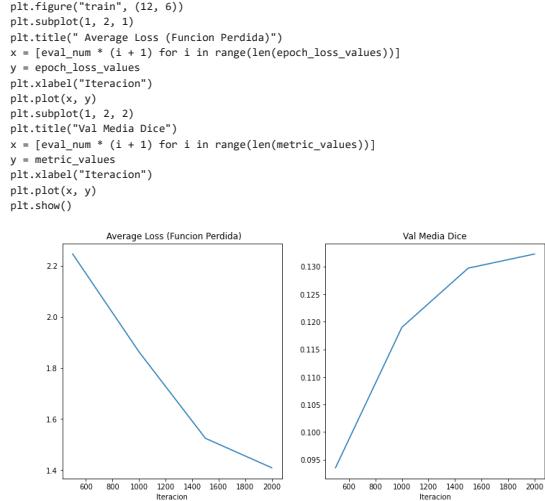
Training (2000 / 2800 Steps) (Perdida loss=1.70094): 100% |██████████| 23/23 [00:17<00:00, 1.77it/s] ha salvado el modelo ! Model saved.

# Informamos por pantalla que ha concluido el entrenamiento y la mejor métrica
print(
    f"entrenamiento completado, mejor metrica conseguida: {dice_val_best:.4f} "
    f"en la iteracion: {global_step_best}"
)
)

entrenamiento completado, mejor metrica conseguida: 0.1323 en la iteracion: 2000

```

▼ Dibujamos las gráficas de la función de pérdidas y de la métrica DICE que vamos obteniendo en las iteraciones



▼ Comprobación de la salida del mejor modelo con inferencia de una imagen y la etiqueta de entrada de una imagen de validación.

```
# Seleccionamos una imagen de validacion cualquiera, por ejemplo la que esta en cuarto lugar
```

```

case_num = 4
# Cargamos el modelo que hemos entrenado y que se ha guardado al ser el de mejor métrica DICE
model.load_state_dict(torch.load(os.path.join(root_dir, "mejor_modelo_métrica.pth")))
# Lanzamos la evaluacion del modelo en PyTorch
model.eval()
with torch.no_grad():
    img_name = os.path.split(val_ds[case_num]["image"].meta["filename_or_obj"])[1]
    img = val_ds[case_num]["image"]
    label = val_ds[case_num]["label"]
    val_inputs = torch.unsqueeze(img, 1).cuda()
    val_labels = torch.unsqueeze(label, 1).cuda()
    val_outputs = sliding_window_inference(
        val_inputs, (96, 96, 96), 4, model, overlap=0.8
    )
    plt.figure("check", (18, 6))
    plt.subplot(1, 3, 1)
    plt.title("Imagen Validacion")
    plt.imshow(val_inputs.cpu().numpy()[0, 0, :, :, slice_map[img_name]], cmap="gray")
    plt.subplot(1, 3, 2)
    plt.title("Etiqueta Ground Truth")
    plt.imshow(val_labels.cpu().numpy()[0, 0, :, :, slice_map[img_name]])
    plt.subplot(1, 3, 3)
    plt.title("Inferencia Obtenida")
    plt.imshow(
        torch.argmax(val_outputs, dim=1).detach().cpu()[0, :, :, slice_map[img_name]]
    )
    plt.show()

```

✓ 11 s completado a las 23:19

● ×

A.2. Cuaderno: Inferencias con modelos entrenados

El presente cuaderno de jupyter realiza una inferencia a una imagen de entrada para obtener una segmentacion usando un modelo preentrenado.

Se puede usar tanto los modelos que hemos entrenados nosotros como modelos que puedes descargar de MONAI Model Zoo que sean compatibles al dataset o cualquier otro modelo compatible proveniente de otras fuentes.

Ha continuación se adjunta el código del cuaderno y la ejecución del mismo.

El cuaderno se encuentra disponible en el repositorio Github del autor del TFM, a través de la URL:

[https://github.com/franguerrero/TFM_SegmentacionImagenesMedicas/
blob/main/cuadernos_jupyter/TFM_InferenciaModelosEntrenados.ipynb](https://github.com/franguerrero/TFM_SegmentacionImagenesMedicas/blob/main/cuadernos_jupyter/TFM_InferenciaModelosEntrenados.ipynb)

▼ Setup del entorno

Verificamos si está conectado con la GPU y mostramos la información de la GPU

```
gpu_info = !nvidia-smi
gpu_info = '\n'.join(gpu_info)
if gpu_info.find('failed') >= 0:
    print('Not connected to a GPU')
else:
    print(gpu_info)

Sat Feb 4 10:33:36 2023
+-----+
| NVIDIA-SMI 510.47.03 Driver Version: 510.47.03 CUDA Version: 11.6 |
+-----+
| GPU Name Persistence-M| Bus-Id Disp.A | Volatile Uncorr. ECC | | |
| Fan Temp Perf Pwr:Usage/Cap| Memory-Usage | GPU-Util Compute M. |
| | | | | MIG M. |
+-----+
| 0 Tesla T4 Off 00000000:00:04.0 Off | 0 |
| N/A 47C P0 23W / 70W | 0MiB / 15360MiB | 0% Default | N/A |
+-----+

+-----+
| Processes:
| GPU GI CI PID Type Process name GPU Memory |
| ID ID | Usage |
+-----+
| No running processes found
+-----+



# Instalamos las librerías de MONAI en su ultima versión estable que en el momento de la realización de este cuaderno es la 1.1.0
!pip install -q "monai[nibabel, tqdm, einops]"==1.1.0
%matplotlib inline

          1.2/1.2 MB 20.0 MB/s eta 0:00:00
          41.6/41.6 KB 2.3 MB/s eta 0:00:00


# Importamos aquellas librerías que nos van a hacer falta tanto de MONAI como otras como de numpy o matplotlib
# Finalizamos viendo las versiones instaladas de cada una de ellas

import os
import shutil
import tempfile

import matplotlib.pyplot as plt
import numpy as np
from tqdm import tqdm

from monai.losses import DiceCELoss
from monai.inferers import sliding_window_inference
from monai.transforms import (
    EnsureChannelFirstd,
    Compose,
    CropForegroundd,
    LoadImaged,
    Orientationd,
    ScaleIntensityRanged,
    Spacingd,
)

from monai.config import print_config
from monai.metrics import DiceMetric
from monai.networks.nets import UNETR
from monai.networks.nets import SwinUNETR

from monai.data import (
    DataLoader,
    CachedDataset,
    load_decathlon_datalist,
    decollate_batch,
)

import torch

print_config()

MONAI version: 1.1.0
Numpy version: 1.21.6
Pytorch version: 1.13.1+cu116
MONAI flags: HAS_EXT = False, USE_COMPILED = False, USE_META_DICT = False
MONAI rev id: a2ec3752f54bfc3b40e7952234fbebe5452ed63e3
```

```

MONAI __file__: /usr/local/lib/python3.8/dist-packages/monai/__init__.py

Optional dependencies:
Pytorch Ignite version: NOT INSTALLED or UNKNOWN VERSION.
Nibabel version: 3.0.2
scikit-image version: 0.18.3
Pillow version: 7.1.2
Tensorboard version: 2.9.1
gdown version: 4.4.0
TorchVision version: 0.14.1+cu116
tqdm version: 4.64.1
lmdb version: 0.99
psutil version: 5.4.8
pandas version: 1.3.5
einops version: 0.6.0
transformers version: NOT INSTALLED or UNKNOWN VERSION.
miflow version: NOT INSTALLED or UNKNOWN VERSION.
pynrrd version: NOT INSTALLED or UNKNOWN VERSION.

For details about installing the optional dependencies, please visit:
https://docs.monai.io/en/latest/installation.html#installing-the-recommended-dependencies

```

▼ Setup directorios de trabajo

Establecemos la conexión con la cuenta de Google Drive y le especificamos la carpeta donde vamos a almacenar los resultados del modelo que obtengamos

```

from google.colab import drive
drive.mount('/content/gdrive')

Mounted at /content/gdrive

os.environ['MONAI_DATA_DIRECTORY'] = '/content/gdrive/MyDrive/TFM_SegmentacionImagenesMedicas/Resultados/'

directory = os.environ.get("MONAI_DATA_DIRECTORY")
root_dir = tempfile.mkdtemp() if directory is None else directory
print(root_dir)

/content/gdrive/MyDrive/TFM_SegmentacionImagenesMedicas/Resultados/

```

▼ Definimos el modelo porque luego es necesario para la imagen de validacion

```

os.environ["CUDA_DEVICE_ORDER"] = "PCI_BUS_ID"
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Modelo de UNETR para usar con las redes entrenadas con el Cuaderno Jupyter desde cero
model = UNETR(
    in_channels=1,
    out_channels=14,
    img_size=(96, 96, 96),
    feature_size=16,
    hidden_size=768,
    mlp_dim=3072,
    num_heads=12,
    pos_embed="perceptron",
    norm_name="Instance",
    res_block=True,
    dropout_rate=0.0,
).to(device)

"""
# Modelo de SwinUNETR para el modelo preentrenado de model.pt
"""

model = SwinUNETR(
    img_size=(96, 96, 96),
    in_channels=1,
    out_channels=14,
    feature_size=48,
    use_checkpoint=True,
).to(device)

"""
' \n# Modelo de SwinUNETR para el modelo preentrenado de model.pt\n\nmodel = Sw
inINFTR(\n    img_size=(96, 96, 96),\n    in_channels=1,\n    out_channels=14,\n
val_transforms = Compose(
    [
        LoadImage(keys=["image", "label"]),
        EnsureChannelFirst(keys=["image", "label"]),
    ]
)

```

```

        Orientationd(keys=["image", "label"], axcodes="RAS"),
        Spacingd(
            keys=["image", "label"],
            pixdim=(1.5, 1.5, 2.0),
            mode="bilinear", "nearest"),
        ),
        ScaleIntensityRanged(
            keys=["image"], a_min=-175, a_max=250, b_min=0.0, b_max=1.0, clip=True
        ),
        CropForegroundd(keys=["image", "label"], source_key="image"),
    ]
)
)

```

▼ Cargamos y cacheamos las imágenes que vamos a inferir luego para segmentar usando el modelo

```

# Establecemos la ruta donde se encuentran las imágenes y etiquetas de BTCV descargadas
data_dir = "/content/gdrive/MyDrive/TFM_SegmentacionImagenesMedicas/Datasets/BTCV/"
split_JSON = "dataset_0.json"

datasets = data_dir + split_JSON

val_files = load_decathlon_datalist(datasets, True, "validation")

val_ds = CacheDataset(
    data=val_files, transform=val_transforms, cache_num=6, cache_rate=1.0, num_workers=4
)
val_loader = DataLoader(
    val_ds, batch_size=1, shuffle=False, num_workers=4, pin_memory=True
)

Loading dataset: 100%[██████████] 5/5 [00:19<00:00, 3.89s/it]
/usr/local/lib/python3.8/dist-packages/torch/utils/data/dataloader.py:554: UserWarning: This DataLoader will create 4 worker processes in total. 0
warnings.warn(_create_warning_msg
)
slice_map = {
    "img0035.nii.gz": 170,
    "img0036.nii.gz": 230,
    "img0037.nii.gz": 204,
    "img0038.nii.gz": 204,
    "img0039.nii.gz": 204,
    "img0040.nii.gz": 180,
}

```

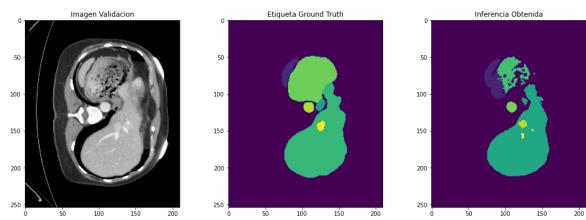
▼ Comprobación de inferencia de una imagen al aplicarle el modelo el resultado que se obtiene y su comparación con la etiqueta de ground truth del especialista

```

case_num = 4
# Modelo entranado desde cero por nosotros con 10.000 iteraciones
model.load_state_dict(torch.load(os.path.join(root_dir, "mejor_modelo_metrica_IT10000.pth")))

# Modelo preentrenado con Swin-UNETR descargado de MONAI Model Zoo
#model.load_state_dict(torch.load(os.path.join(root_dir, "model.pt")))
model.eval()
with torch.no_grad():
    img_name = os.path.split(val_ds[case_num]["image"].meta["filename_or_obj"])[1]
    img = val_ds[case_num]["image"]
    label = val_ds[case_num]["label"]
    val_inputs = torch.unsqueeze(img, 1).cuda()
    val_labels = torch.unsqueeze(label, 1).cuda()
    val_outputs = sliding_window_inference(
        val_inputs, (96, 96, 96), 4, model, overlap=0.8
    )
    plt.figure("check", (18, 6))
    plt.subplot(1, 3, 1)
    plt.title("Imagen Validacion")
    plt.imshow(val_inputs.cpu().numpy()[0, 0, :, :, slice_map[img_name]], cmap="gray")
    plt.subplot(1, 3, 2)
    plt.title("Etiqueta Ground Truth")
    plt.imshow(val_labels.cpu().numpy()[0, 0, :, :, slice_map[img_name]])
    plt.subplot(1, 3, 3)
    plt.title("Inferencia Obtenida")
    plt.imshow(
        torch.argmax(val_outputs, dim=1).detach().cpu()[0, :, :, slice_map[img_name]]
    )
    plt.show()

```



Productos de pago de Colab - Cancelar contratos
✓ 1 min 24 s completado a las 11:36

● ✕

A.3. Cuaderno: Entrenamiento con Transfer Learning

El presente cuaderno de jupyter realiza un entrenamiento de una red de segmentación de imágenes médicas basado en el modelo de Swin-UNETR utilizando una red previa preentrenada. A esa red con pesos pre-existentes, se le vuelve a entrenar con las imágenes de referencia del dataset de BTCV.

Se comprueba como para el mismo número de iteraciones respecto a entrenamiento desde cero, se obtiene una métrica de evaluación superior. Obteniendo por tanto mayor precisión en la segmentación cuando estamos considerando un alto número de iteraciones. Se obtiene en total mejor DICE si se usa redes preentrenadas que con un entrenamiento desde cero.

Ha continuación se adjunta el código del cuaderno y la ejecución del mismo.

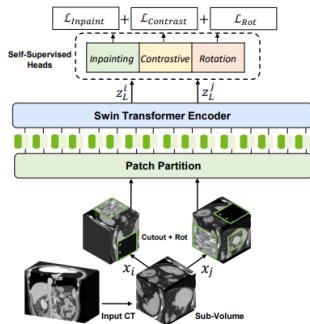
El cuaderno se encuentra disponible en el repositorio Github del autor del TFM, a través de la URL:

[https://github.com/franguerrero/TFM_SegmentacionImagenesMedicas/
blob/main/cuadernos_jupyter/TFM_EntrenamientoFIIno_DesdeModeloPreentrenado.
ipynb](https://github.com/franguerrero/TFM_SegmentacionImagenesMedicas/blob/main/cuadernos_jupyter/TFM_EntrenamientoFIIno_DesdeModeloPreentrenado.ipynb)

[Open in Colab](#)

▼ Modelo Pre-Entrenado de Swin UNETR

Utilizamos pesos de preentrenamiento auto-supervisado del codificador Swin UNETR (Transformador Swin 3D) en una cohorte de 5050 escaneos de TC de conjuntos de datos disponibles públicamente. El codificador se preentrena usando tareas previas de reconstrucción, predicción de rotación y aprendizaje contrastivo como se muestra a continuación.



▼ Setup del entorno

Verificamos si está conectado con la GPU y mostramos la información de la GPU

```
gpu_info = !nvidia-smi
gpu_info = '\n'.join(gpu_info)
if gpu_info.find('failed') >= 0:
    print('Not connected to a GPU')
else:
    print(gpu_info)

Sat Feb 4 15:46:44 2023
+-----+
| NVIDIA-SMI 510.47.03   Driver Version: 510.47.03   CUDA Version: 11.6      |
+-----+
| GPU Name Persistence-M| Bus-Id Disp.A | Volatile Uncorr. ECC |
| Fan Temp Perf Pwr:Usage/Cap| Memory-Usage | GPU-Util Compute M. |
|                               |             | MIG M. |
+-----+
| 0  NVIDIA A100-SXM... Off | 00000000:00:04.0 Off |          0 |
| N/A 33C P0 55W / 400W | 0MiB / 40960MiB | 0% Default |
+-----+
+-----+
| Processes:                   GPU Memory |
| GPU GI CI PID Type Process name     Usage |
| ID ID          |
+-----+
| No running processes found |
+-----+


# Instalamos las librerías de MONAI en su ultima versión estable que en el momento de la realización de este cuaderno es la 1.1.0
!pip install -q "monai[nibabel, tqdm, einops]"==1.1.0
!python -c "import matplotlib" || pip install -q matplotlib
%matplotlib inline
===== 1.2/1.2 MB 15.1 MB/s eta 0:00:00
41.6/41.6 KB 3.5 MB/s eta 0:00:00

# Importamos aquellas librerías que nos van a hacer falta tanto de MONAI como otras como de numpy o matplotlib
# Finalizamos viendo las versiones instaladas de cada una de ellas

import os
import shutil
import tempfile

import matplotlib.pyplot as plt
from tqdm import tqdm
```

```

from monai.losses import DiceCELoss
from monai.inferers import sliding_window_inference
from monai.transforms import (
    AsDiscrete,
    Compose,
    CropForegroundd,
    LoadImaged,
    Orientationd,
    RandFlipd,
    RandCropByPosNegLabeld,
    RandShiftIntensityd,
    ScaleIntensityRanged,
    Spacingd,
    RandRotate90d,
    EnsureTyped,
)

from monai.config import print_config
from monai.metrics import DiceMetric
from monai.networks.nets import SwinUNETR

from monai.data import (
    ThreadDataLoader,
    CachedDataset,
    load_decathlon_datalist,
    decollate_batch,
    set_track_meta,
)

import torch

print_config()
    MONAI version: 1.1.0
    Numpy version: 1.21.6
    Pytorch version: 1.13.1+cu116
    MONAI flags: HAS_EXT = False, USE_COMPILED = False, USE_META_DICT = False
    MONAI rev id: a2ec3752f54bfc5b40e7952234fbef5452ed63e3
    MONAI __file__: /usr/local/lib/python3.8/dist-packages/monai/__init__.py

Optional dependencies:
Pytorch Ignite version: NOT INSTALLED or UNKNOWN VERSION.
Nibabel version: 3.0.2
scikit-image version: 0.18.3
Pillow version: 7.1.2
Tensorboard version: 2.9.1
gdown version: 4.4.0
TorchVision version: 0.14.1+cu116
tqdm version: 4.64.1
imdb version: 0.99
psutil version: 5.4.8
pandas version: 1.3.5
einops version: 0.6.0
transformers version: NOT INSTALLED or UNKNOWN VERSION.
miflow version: NOT INSTALLED or UNKNOWN VERSION.
pynrrd version: NOT INSTALLED or UNKNOWN VERSION.

For details about installing the optional dependencies, please visit:
https://docs.monai.io/en/latest/installation.html#installing-the-recommended-dependencies

```

▼ Setup directorios de trabajo

Establecemos la conexión con la cuenta de Google Drive y le especificamos la carpeta donde vamos a almacenar los resultados del modelo que obtengamos

```

from google.colab import drive
drive.mount('/content/gdrive')

Mounted at /content/gdrive

os.environ['MONAI_DATA_DIRECTORY'] = '/content/gdrive/MyDrive/TFM_SegmentacionImagenesMedicas/Resultados'

directory = os.environ.get("MONAI_DATA_DIRECTORY")
root_dir = tempfile.mkdtemp() if directory is None else directory
print(root_dir)

/content/gdrive/MyDrive/TFM_SegmentacionImagenesMedicas/Resultados/

```

▼ Setup de las transformaciones para entrenamiento (training) y validacion

```

num_samples = 4

os.environ["CUDA_DEVICE_ORDER"] = "PCI_BUS_ID"

```

```

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

train_transforms = Compose(
    [
        LoadImaged(keys=["image", "label"], ensure_channel_first=True),
        ScaleIntensityRanged(
            keys=["image"],
            a_min=-175,
            a_max=250,
            b_min=0.0,
            b_max=1.0,
            clip=True,
        ),
        CropForegroundd(keys=["image", "label"], source_key="image"),
        Orientationd(keys=["image", "label"], axcodes="RAS"),
        Spacingd(
            keys=["image", "label"],
            pixdim=(1.5, 1.5, 2.0),
            mode=("bilinear", "nearest"),
        ),
        EnsureTyped(keys=["image", "label"], device=device, track_meta=False),
        RandCropByPosNegLabeld(
            keys=["image", "label"],
            label_key="label",
            spatial_size=(96, 96, 96),
            pos=1,
            neg=1,
            num_samples=num_samples,
            image_key="image",
            image_threshold=0,
        ),
        RandFlipd(
            keys=["image", "label"],
            spatial_axis=[0],
            prob=0.10,
        ),
        RandFlipd(
            keys=["image", "label"],
            spatial_axis=[1],
            prob=0.10,
        ),
        RandFlipd(
            keys=["image", "label"],
            spatial_axis=[2],
            prob=0.10,
        ),
        RandRotate90d(
            keys=["image", "label"],
            prob=0.10,
            max_k=3,
        ),
        RandShiftIntensityd(
            keys=["image"],
            offsets=0.10,
            prob=0.50,
        ),
    ],
)
val_transforms = Compose(
    [
        LoadImaged(keys=["image", "label"], ensure_channel_first=True),
        ScaleIntensityRanged(keys=["image"], a_min=-175, a_max=250, b_min=0.0, b_max=1.0, clip=True),
        CropForegroundd(keys=["image", "label"], source_key="image"),
        Orientationd(keys=["image", "label"], axcodes="RAS"),
        Spacingd(
            keys=["image", "label"],
            pixdim=(1.5, 1.5, 2.0),
            mode="bilinear", "nearest"),
        ),
        EnsureTyped(keys=["image", "label"], device=device, track_meta=True),
    ],
)

```

▼ Descarga del dataset y almacenamiento en Google Drive

El dataset ha sido descargado de <https://www.synapse.org/#/Synapse.syn3193805/wiki/217752> y se ha subido a la aplicacion de google drive.

En concreto a la ruta \MyDrive\TFM_Segmentacion\ImagenesMedicas\Datasets\BTCV

Es importante señalar que las imágenes del dataset de BTCV van a la carpeta \textbackslash imagesTr y las etiquetas van a la carpeta \textbackslash labelsTr.

Además también se encuentra el fichero dataset_0.json que contiene la descripción de las imágenes y sus correspondientes etiquetas de ground truth que van a ser usadas para el entrenamiento y las que van a ser usadas para validación. En nuestro caso se han usado 24 imágenes (y correspondientes etiquetas) para entrenamiento y 6 para validación. Las de validación se corresponden con la numeración desde la 35 a la 40 ambas incluidas.

```

#data_dir = "data/"
#split_json = "dataset_0.json"

# Establecemos la ruta donde se encuentran las imágenes y etiquetas de BTCV descargadas
data_dir = "/content/gdrive/MyDrive/TFM_SegmentacionImagenesMedicas/Datasets/BTCV/"
split_json = "dataset_0.json"

datasets = data_dir + split_json
datalist = load_decaction_datalist(datasets, True, "training")
val_files = load_decathon_datalist(datasets, True, "validation")

# Cacheamos los datos de entrenamiento y validación
train_ds = CachedDataset(
    data=datalist,
    transform=train_transforms,
    cache_num=24,
    cache_rate=1.0,
    num_workers=8,
)
train_loader = ThreadDataLoader(train_ds, num_workers=0, batch_size=1, shuffle=True)
val_ds = CachedDataset(data=val_files, transform=val_transforms, cache_num=6, cache_rate=1.0, num_workers=4)
val_loader = ThreadDataLoader(val_ds, num_workers=0, batch_size=1)

# we want cached training images to not have metadata, and validations to have metadata
# the EnsureTyped transforms allow us to make this distinction
# on the other hand, set_track_meta is a global API; doing so here makes sure subsequent transforms (i.e., random transforms for training)
# will be carried out as Tensors, not MetaTensors
set_track_meta(False)

Loading dataset: 100%|██████████| 23/23 [00:20<00:00,  1.12it/s]
Loading dataset: 100%|██████████| 5/5 [00:06<00:00,  1.35s/it]

```

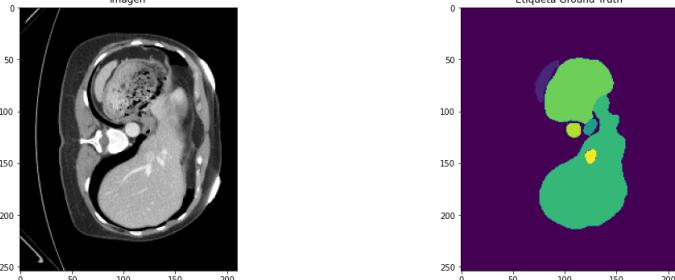
▼ Visualizamos imagen y etiquetas y comprobamos la forma de los datos

```

slice_map = {
    "img0035.nii.gz": 170,
    "img0036.nii.gz": 230,
    "img0037.nii.gz": 204,
    "img0038.nii.gz": 204,
    "img0039.nii.gz": 204,
    "img0040.nii.gz": 180,
}
# Seleccionamos uno cualquiera, por ejemplo el que esta en el lugar 0 del map
case_num = 4
img_name = os.path.split(val_ds[case_num]["image"])[1].meta["filename_or_obj"][1]
img = val_ds[case_num]["image"]
label = val_ds[case_num]["label"]
img_shape = img.shape
label_shape = label.shape
print(f"Forma de la imagen: {img_shape}, forma de la etiqueta: {label_shape}")
plt.figure("Imagen", (18, 6))
plt.subplot(1, 2, 1)
plt.title("Imagen")
plt.imshow(img[0, :, :, slice_map[img_name]].detach().cpu(), cmap="gray")
plt.subplot(1, 2, 2)
plt.title("Etiqueta Ground Truth")
plt.imshow(label[0, :, :, slice_map[img_name]].detach().cpu())
plt.show()

forma de la imagen: torch.Size([1, 254, 210, 292]), forma de la etiqueta: torch.Size([1, 254, 210, 292])

```



▼ Creación del modelo Swin UNETR

En esta sección, creamos un modelo Swin UNETR para la segmentación multi-órgano de 14 clases. Usamos un tamaño de característica de 48, que es compatible con los pesos preentrenados auto-supervisados. También usamos el puntos de control de gradiente (use_checkpoint) para un entrenamiento más eficiente en memoria.

```
os.environ["CUDA_DEVICE_ORDER"] = "PCI_BUS_ID"
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

model = SwinUNETR(
    img_size=(96, 96, 96),
    in_channels=1,
    out_channels=14,
    feature_size=48,
    use_checkpoint=True,
).to(device)
```

▼ Inicializar el codificador Swin UNETR desde pesos preentrenados auto-supervisados

En esta sección, inicializamos el codificador Swin UNETR desde los pesos preentrenados. Los pesos se pueden descargar usando el comando wget a continuación..

```
# !wget https://github.com/Project-MONAI/MONAI-extra-test-data/releases/download/0.8.1/model_swinvit.pt

# Cargamos los pesos del preentrenado Swin-UNETR
weight = torch.load(os.path.join(root_dir, "model_swinvit.pt"))
model.load_from(weights=weight)
print("Usando pesos del backbone Swin UNETR preentrenados auto-supervisados")

Usando pesos del backbone Swin UNETR preentrenados auto-supervisados
```

▼ Función de pérdida y el optimizador

```
torch.backends.cudnn.benchmark = True
loss_function = DiceCELoss(to_onehot_y=True, softmax=True)
optimizer = torch.optim.AdamW(model.parameters(), lr=1e-4, weight_decay=1e-5)
scaler = torch.cuda.amp.GradScaler()
```

▼ Ejecutamos un proceso de entrenamiento de red neuronal típico de PyTorch

```
def validation(epoch_iterator_val):
    model.eval()
    with torch.no_grad():
        for batch in epoch_iterator_val:
            val_inputs, val_labels = (batch["image"].cuda(), batch["label"].cuda())
            with torch.cuda.amp.autocast():
                val_outputs = sliding_window_inference(val_inputs, (96, 96, 96), 4, model)
            val_labels_list = decollate_batch(val_labels)
            val_labels_convert = [post_label(val_label_tensor) for val_label_tensor in val_labels_list]
            val_outputs_list = decollate_batch(val_outputs)
            val_outputs_convert = [post_pred(val_pred_tensor) for val_pred_tensor in val_outputs_list]
            dice_metric(y_pred=val_outputs_convert, y=val_labels_convert)
            epoch_iterator_val.set_description("Validar (%d / %d Steps)" % (global_step, 10.0))
        mean_dice_val = dice_metric.aggregate().item()
        dice_metric.reset()
    return mean_dice_val

def train(global_step, train_loader, dice_val_best, global_step_best):
    model.train()
    epoch_loss = 0
    step = 0
    epoch_iterator = tqdm(train_loader, desc="Training (X / X Steps) (Perdida loss=X.X)", dynamic_ncols=True)
    for step, batch in enumerate(epoch_iterator):
        step += 1
        x, y = (batch["image"].cuda(), batch["label"].cuda())
        with torch.cuda.amp.autocast():
            logit_map = model(x)
            loss = loss_function(logit_map, y)
            scaler.scale(loss).backward()
        epoch_loss += loss.item()
        scaler.unscale_(optimizer)
        scaler.step(optimizer)
        scaler.update()
        optimizer.zero_grad()
    epoch_iterator.set_description(f"Training ({global_step} / {max_iterations} Steps) (Perdida loss={loss:2.5f})")
    if (global_step % eval_num == 0 and global_step != 0) or global_step == max_iterations:
        epoch_iterator_val = tqdm(val_loader, desc="Validar (X / X Steps) (dice=X.X)", dynamic_ncols=True)
        dice_val = validation(epoch_iterator_val)
        epoch_loss /= step
        epoch_loss_values.append(epoch_loss)
        if dice_val > dice_val_best:
            dice_val_best = dice_val
            global_step_best = global_step
```

```

metric_values.append(dice_val)
if dice_val > dice_val_best:
    dice_val_best = dice_val
    global_step_best = global_step
    torch.save(model.state_dict(), os.path.join(root_dir, "best_metric_model_usandopreentrenado.pth"))
    print(
        "Se ha salvado el modelo ! Mejor Avg. Dice: {} Actual Avg. Dice: {}".format(dice_val_best, dice_val)
    )
else:
    print(
        "No se ha salvado el modelo ! Mejor Avg. Dice: {} Actual Avg. Dice: {}".format(
            dice_val_best, dice_val
        )
    )
global_step += 1
return global_step, dice_val_best, global_step_best

max_iterations = 2000
eval_num = 500
post_label = AsDiscrete(to_onehot=14)
post_pred = AsDiscrete(argmax=True, to_onehot=14)
dice_metric = DiceMetric(include_background=True, reduction="mean", get_not_nans=False)
global_step = 0
dice_val_best = 0.0
global_step_best = 0
epoch_loss_values = []
metric_values = []
while global_step < max_iterations:
    global_step, dice_val_best, global_step_best = train(global_step, train_loader, dice_val_best, global_step_best)
    model.load_state_dict(torch.load(os.path.join(root_dir, "best_metric_model_usandopreentrenado.pth")))
    Training (1455 / 2000 Steps) (Menor loss=1.54481): 100% | 44/45 [00:19<00:00, 2.44it/s]
    Training (1356 / 2000 Steps) (Perdida loss=1.10094): 100% | 23/23 [00:10<00:00, 2.24it/s]
    Training (1379 / 2000 Steps) (Perdida loss=1.96851): 100% | 23/23 [00:10<00:00, 2.25it/s]
    Training (1402 / 2000 Steps) (Perdida loss=1.02706): 100% | 23/23 [00:10<00:00, 2.24it/s]
    Training (1425 / 2000 Steps) (Perdida loss=1.07510): 100% | 23/23 [00:10<00:00, 2.24it/s]
    Training (1448 / 2000 Steps) (Perdida loss=1.02120): 100% | 23/23 [00:10<00:00, 2.24it/s]
    Training (1471 / 2000 Steps) (Perdida loss=1.07682): 100% | 23/23 [00:10<00:00, 2.24it/s]
    Training (1494 / 2000 Steps) (Perdida loss=1.09756): 100% | 23/23 [00:10<00:00, 2.24it/s]
    Training (1500 / 2000 Steps) (Perdida loss=1.10699): 22% | 5/23 [00:02<00:00, 2.24it/s]
    Validar (X / X Steps) (dice=X.X): 0% | 0/5 [00:00<00:, ?it/s]
    Validar (X / X Steps): 0% | 0/5 [00:01<00:, ?it/s]
    Validar (1495 / 10 Steps): 20% | 1/5 [00:01<00:05, 1.35s/it]
    Validar (1495 / 10 Steps): 20% | 1/5 [00:02<00:05, 1.35s/it]
    Validar (1495 / 10 Steps): 40% | 2/5 [00:02<00:03, 1.18s/it]
    Validar (1495 / 10 Steps): 40% | 2/5 [00:04<00:03, 1.18s/it]
    Validar (1495 / 10 Steps): 60% | 3/5 [00:04<00:03, 1.72s/it]
    Validar (1495 / 10 Steps): 60% | 3/5 [00:06<00:03, 1.72s/it]
    Validar (1495 / 10 Steps): 80% | 4/5 [00:06<00:01, 1.57s/it]
    Validar (1495 / 10 Steps): 80% | 4/5 [00:07<00:01, 1.57s/it]
    Validar (1495 / 10 Steps): 100% | 5/5 [00:07<00:00, 1.51s/it]
    Training (1500 / 2000 Steps) (Perdida loss=1.10699): 26% | 6/23 [00:10<00:54, 3.22s/it]Se ha salvado el modelo ! Mejor Avg. Dice: 6
    Training (1517 / 2000 Steps) (Perdida loss=1.01888): 100% | 23/23 [00:10<00:00, 1.25it/s]
    Training (1540 / 2000 Steps) (Perdida loss=0.96344): 100% | 23/23 [00:10<00:00, 2.24it/s]
    Training (1563 / 2000 Steps) (Perdida loss=0.95963): 100% | 23/23 [00:10<00:00, 2.24it/s]
    Training (1586 / 2000 Steps) (Perdida loss=0.98843): 100% | 23/23 [00:10<00:00, 2.24it/s]
    Training (1609 / 2000 Steps) (Perdida loss=0.98661): 100% | 23/23 [00:10<00:00, 2.24it/s]
    Training (1632 / 2000 Steps) (Perdida loss=1.11914): 100% | 23/23 [00:10<00:00, 2.24it/s]
    Training (1655 / 2000 Steps) (Perdida loss=1.03947): 100% | 23/23 [00:10<00:00, 2.24it/s]
    Training (1678 / 2000 Steps) (Perdida loss=1.02445): 100% | 23/23 [00:10<00:00, 2.24it/s]
    Training (1701 / 2000 Steps) (Perdida loss=1.18941): 100% | 23/23 [00:10<00:00, 2.24it/s]
    Training (1724 / 2000 Steps) (Perdida loss=0.94423): 100% | 23/23 [00:10<00:00, 2.24it/s]
    Training (1747 / 2000 Steps) (Perdida loss=0.96737): 100% | 23/23 [00:10<00:00, 2.24it/s]
    Training (1770 / 2000 Steps) (Perdida loss=0.91517): 100% | 23/23 [00:10<00:00, 2.24it/s]
    Training (1793 / 2000 Steps) (Perdida loss=0.95751): 100% | 23/23 [00:10<00:00, 2.24it/s]
    Training (1816 / 2000 Steps) (Perdida loss=1.01148): 100% | 23/23 [00:10<00:00, 2.25it/s]
    Training (1839 / 2000 Steps) (Perdida loss=0.98874): 100% | 23/23 [00:10<00:00, 2.24it/s]
    Training (1862 / 2000 Steps) (Perdida loss=0.92786): 100% | 23/23 [00:10<00:00, 2.24it/s]
    Training (1885 / 2000 Steps) (Perdida loss=0.99397): 100% | 23/23 [00:10<00:00, 2.24it/s]
    Training (1908 / 2000 Steps) (Perdida loss=0.94047): 100% | 23/23 [00:10<00:00, 2.24it/s]
    Training (1931 / 2000 Steps) (Perdida loss=1.01874): 100% | 23/23 [00:10<00:00, 2.24it/s]
    Training (1954 / 2000 Steps) (Perdida loss=1.08393): 100% | 23/23 [00:10<00:00, 2.24it/s]
    Training (1977 / 2000 Steps) (Perdida loss=1.04224): 100% | 23/23 [00:10<00:00, 2.24it/s]
    Training (2000 / 2000 Steps) (Perdida loss=0.86051): 96% | 22/23 [00:10<00:00, 2.27it/s]
    Validar (X / X Steps) (dice=X.X): 0% | 0/5 [00:00<00:, ?it/s]
    Validar (1978 / 10 Steps): 20% | 1/5 [00:01<00:, ?it/s]
    Validar (1978 / 10 Steps): 20% | 1/5 [00:01<00:05, 1.35s/it]
    Validar (1978 / 10 Steps): 40% | 2/5 [00:02<00:05, 1.35s/it]
    Validar (1978 / 10 Steps): 40% | 2/5 [00:04<00:03, 1.18s/it]
    Validar (1978 / 10 Steps): 60% | 3/5 [00:04<00:03, 1.72s/it]
    Validar (1978 / 10 Steps): 60% | 3/5 [00:06<00:03, 1.72s/it]
    Validar (1978 / 10 Steps): 80% | 4/5 [00:06<00:01, 1.57s/it]
    Validar (1978 / 10 Steps): 80% | 4/5 [00:07<00:00, 1.57s/it]
    Validar (1978 / 10 Steps): 100% | 5/5 [00:07<00:00, 1.51s/it]
    Training (2000 / 2000 Steps) (Perdida loss=0.86051): 100% | 23/23 [00:18<00:00, 1.25it/s]Se ha salvado el modelo ! Mejor Avg. Dice: 0.5575
    <All keys matched successfully>

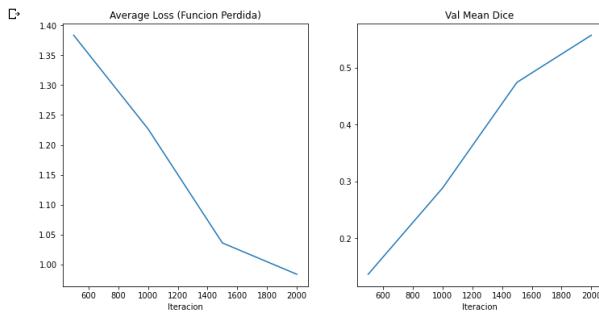
```

- ▼ Dibujamos las gráficas de la función de pérdidas y de la métrica DICE que vamos obteniendo en las iteraciones

```

plt.figure("train", (12, 6))
plt.subplot(1, 2, 1)
plt.title("Average Loss (Funcion Perdida)")
x = [eval_num * (i + 1) for i in range(len(epoch_loss_values))]
y = epoch_loss_values
plt.xlabel("Iteracion")
plt.plot(x, y)
plt.subplot(1, 2, 2)
plt.title("Val Mean Dice")
x = [eval_num * (i + 1) for i in range(len(metric_values))]
y = metric_values
plt.xlabel("Iteracion")
plt.plot(x, y)
plt.show()

```

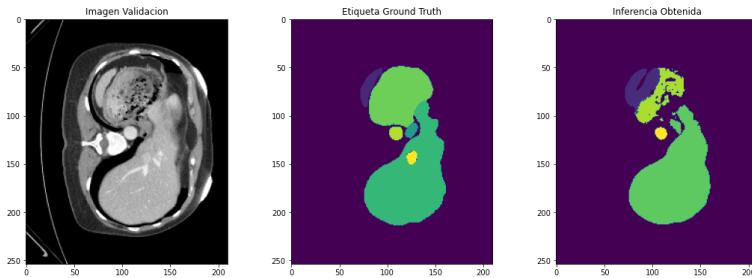


- ▼ Check best model output with the input image and label

```

case_num = 4
model.load_state_dict(torch.load(os.path.join(root_dir, "best_metric_model_usandopreentrenado.pth")))
model.eval()
with torch.no_grad():
    img_name = os.path.split(val_ds[case_num]["image"].meta["filename_or_obj"])[1]
    img = val_ds[case_num]["image"]
    label = val_ds[case_num]["label"]
    val_inputs = torch.unsqueeze(img, 1).cuda()
    val_labels = torch.unsqueeze(label, 1).cuda()
    val_outputs = sliding_window_inference(val_inputs, (96, 96, 96), 4, model, overlap=0.8)
    plt.figure("check", (18, 6))
    plt.subplot(1, 3, 1)
    plt.title("Imagen Validacion")
    plt.imshow(val_inputs.cpu().numpy()[0, 0, :, :, slice_map[img_name]], cmap="gray")
    plt.subplot(1, 3, 2)
    plt.title("Etiqueta Ground Truth")
    plt.imshow(val_labels.cpu().numpy()[0, 0, :, :, slice_map[img_name]])
    plt.subplot(1, 3, 3)
    plt.title("Inferencia Obtenida")
    plt.imshow(torch.argmax(val_outputs, dim=1).detach().cpu()[0, :, :, slice_map[img_name]])
    plt.show()

```



✓ 37 s completado a las 17:04

● ×

A.4. Cuaderno: Solución integral Segmentación Autorunner

El presente cuaderno de jupyter proporciona una demostración simple de cómo utilizar Auto3DSeg AutoRunner para procesar un conjunto de datos simulados y generar resultados en muy pocos minutos.

Para disminuir los tiempos de entrenamiento, en lugar de usar imágenes reales de CT que llevarían mucho tiempo de entrenar, se van a simular un conjunto de datos de entrenamiento con una geometría mucho más simple de la que sería un dato real y se va a entrenar al modelo usando muy pocas iteraciones (épocas) con el objetivo de demostrar todos el proceso. Los resultados que se obtendrán no serán nada precisos, pero el objetivo ahora sería ver todo el proceso en un tiempo razonable independientemente de la calidad de precisión del modelo resultante.

Nos creamos también la variable sim_datalist que proporciona la información de los conjuntos de datos simulados. En él se listan 12 imágenes de entrenamiento y 2 de pruebas.

AutoRunner automáticamente configurará cuatro redes neuronales diferentes y realizará un entrenamiento multigrupo para lograr el mejor rendimiento. El módulo es altamente configurable y solo requiere una entrada mínima.

Podemos ver que los algoritmos comienzan a aprender los conjuntos de datos y la predicción de vértices de fondo y primer plano. Auto3DSeg y AutoRunner son altamente configurables. Para obtener mejores resultados, se puede aumentar el tiempo de entrenamiento, aplicar un método de ensamblado diferente o utilizar la optimización de hiperparámetros a través de las APIs de módulo AutoRunner o Auto3DSeg.

Ha continuación se adjunta el código del cuaderno y la ejecución del mismo.

El cuaderno se encuentra disponible en el repositorio Github del autor del TFM, a través de la URL:

[https://github.com/franguerrero/TFM_SegmentacionImagenesMedicas/
blob/main/cuadernos_jupyter/TFM_SolucionIntegralSegmentacion_Auto3DSeg-
Autorunner.ipynb](https://github.com/franguerrero/TFM_SegmentacionImagenesMedicas/blob/main/cuadernos_jupyter/TFM_SolucionIntegralSegmentacion_Auto3DSeg-Autorunner.ipynb)

▼ Solución Integral Segmentación. Auto3DSeg. Ejemplo simple con Autorunner

En este cuaderno, proporcionaremos una demostración simple de cómo utilizar Auto3DSeg AutoRunner para procesar un conjunto de datos simulados y generar resultados en muy pocos minutos

[Open in Colab](#)

▼ Setup del entorno

Verificamos si está conectado con la GPU y mostramos la información de la GPU

```
gpu_info = !nvidia-smi
gpu_info = '\n'.join(gpu_info)
if gpu_info.find('falló') >= 0:
    print('No conectado a GPU')
else:
    print(gpu_info)

Sun Feb  5 08:57:19 2023
+-----+
| NVIDIA-SMI 510.47.03   Driver Version: 510.47.03   CUDA Version: 11.6 |
+-----+
| GPU Name Persistence-M| Bus-Id Disp.A | Volatile Uncorr. ECC | | | | |
| Fan Temp Perf Pwr:Usage/Cap| Memory-Usage | GPU-Util Compute M. |
| | | | | | | MIG M. |
+-----+
| 0 Tesla T4          Off | 00000000:00:04.0 Off |          0 | | | | |
| N/A 47C P0 26W / 70W |     0MiB / 15360MiB |      0% Default |
| | | | | | | N/A |
+-----+
+-----+
| Processes:           |
| GPU GI CI PID Type Process name          GPU Memory |
| ID ID ID   | Usage |
| =========== |
| No running processes found |
+-----+  
  
import os
Se ha guardado correctamente
# Necesitamos estas versiones para que funcione correctamente
!pip install nibabel==4.0.2 fire==0.4.0 scikit-image==0.19.3

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting nibabel==4.0.2
  Downloading nibabel-4.0.2-py3-none-any.whl (3.3 MB)
    3.3/3.3 MB 26.7 MB/s eta 0:00:00
Collecting fire==0.4.0
  Downloading fire-0.4.0.tar.gz (87 kB)
    87.7/87.7 KB 6.0 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Collecting scikit-image==0.19.3
  Downloading scikit_image-0.19.3-cp38-cp38-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (14.0 MB)
    14.0/14.0 MB 17.0 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.8/dist-packages (from nibabel==4.0.2) (1.21.6)
Requirement already satisfied: packaging>=17.0 in /usr/local/lib/python3.8/dist-packages (from nibabel==4.0.2) (23.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.8/dist-packages (from nibabel==4.0.2) (57.4.0)
Requirement already satisfied: six in /usr/local/lib/python3.8/dist-packages (from fire==0.4.0) (1.15.0)
Requirement already satisfied: termcolor in /usr/local/lib/python3.8/dist-packages (from fire==0.4.0) (2.2.0)
Requirement already satisfied: scipy>=1.4.1 in /usr/local/lib/python3.8/dist-packages (from scikit-image==0.19.3) (1.7.3)
Requirement already satisfied: pillow!=7.1.0,!7.1.1,!8.3.0,>=6.1.0 in /usr/local/lib/python3.8/dist-packages (from scikit-image==0.19.3) (7.1.2)
Requirement already satisfied: PyWavelets>=1.1.1 in /usr/local/lib/python3.8/dist-packages (from scikit-image==0.19.3) (1.4.1)
Requirement already satisfied: imageio>=2.4.1 in /usr/local/lib/python3.8/dist-packages (from scikit-image==0.19.3) (2.9.0)
Requirement already satisfied: networkxx>=2.2 in /usr/local/lib/python3.8/dist-packages (from scikit-image==0.19.3) (3.0)
Requirement already satisfied: tifffile>=2019.7.26 in /usr/local/lib/python3.8/dist-packages (from scikit-image==0.19.3) (2023.1.23.1)
Building wheels for collected packages: fire
  Building wheel for fire (setup.py) ... done
    Created wheel for fire: filename=fire-0.4.0-py2.py3-none-any.whl size=115943 sha256=33213b87b1a2b9bc039e6adea043b3fb2c99c0088aec9aa911dedf5284db
    Stored in directory: /root/.cache/pip/wheels/1f/10/06/2a990e4d73a8479fe2922445e8a876d3cfbfbed052284c6a1
Successfully built fire
Installing collected packages: nibabel, fire, scikit-image
  Attempting uninstall: nibabel
    Found existing installation: nibabel 3.0.2
    Uninstalling nibabel-3.0.2:
      Successfully uninstalled nibabel-3.0.2
  Attempting uninstall: scikit-image
    Found existing installation: scikit-image 0.18.3
    Uninstalling scikit-image-0.18.3:
      Successfully uninstalled scikit-image-0.18.3
Successfully installed fire-0.4.0 nibabel-4.0.2 scikit-image-0.19.3
```

Instalamos las librerías de MONAI en su última versión estable que en el momento de la realización de este cuaderno es la 1.1.0

```

!pip install -q "monai[nibabel, tqdm, einops, yaml]==1.1.0"
#!python -c "import monai" || pip install -q "monai-weekly[nibabel, yaml, tqdm]"
#!python -c "import monai" || pip install -q "monai[all]"

```

1.2/1.2 MB 19.5 MB/s eta 0:00:00
WARNING: monai 1.1.0 does not provide the extra 'yaml'
41.6/41.6 KB 3.1 MB/s eta 0:00:00

▼ Setup de imports

```

# Importamos aquellas librerías que nos van a hacer falta tanto de MONAI como otras como de numpy o matplotlib
# Finalizamos viendo las versiones instaladas de cada una de ellas

import os
import json
import nibabel as nib
import numpy as np
import matplotlib.pyplot as plt
import torch

from monai.apps.auto3dseg import AutoRunner
from monai.config import print_config
from monai.data import create_test_image_3d

print_config()

MONAI version: 1.1.0
Numpy version: 1.21.6
Pytorch version: 1.13.1+cu116
MONAI flags: HAS_EXT = False, USE_COMPILED = False, USE_META_DICT = False
MONAI rev id: a2ec3752f54bfc3b40e795223afbeb5452ed63e3
MONAI __file__: /usr/local/lib/python3.8/dist-packages/monai/__init__.py

Optional dependencies:
Pytorch Ignite version: NOT INSTALLED or UNKNOWN VERSION.
Nibabel version: 4.0.2
scikit-image version: 0.19.3
Pillow version: 7.1.2
Tensorboard version: 2.9.1
gdown version: 4.4.0
TorchVision version: 0.14.1+cu116
tqdm version: 4.64.1
lmdb version: 0.99

ha guardado correctamente ✘
transformers version: NOT INSTALLED or UNKNOWN VERSION.
miflow version: NOT INSTALLED or UNKNOWN VERSION.
pynrrd version: NOT INSTALLED or UNKNOWN VERSION.

For details about installing the optional dependencies, please visit:
https://docs.monai.io/en/latest/installation.html#installing-the-recommended-dependencies

```

▼ Conjunto de datos simulados

Para disminuir los tiempos de entrenamiento, en lugar de usar imágenes reales de CT que llevarían mucho tiempo de entrenar, se van a simular un conjunto de datos de entrenamiento con una geometría mucho más simple de la que sería un dato real y se va a entrenar al modelo usando muy pocas iteraciones (épocas) con el objetivo de demostrar todos el proceso. Los resultados que se obtendrán no serán nada precisos, pero el objetivo ahora sería ver todo el proceso en un tiempo razonable independientemente de la calidad de precisión del modelo resultante.

Nos creamos también la variable `sim_datalist` que proporciona la información de los conjuntos de datos simulados. En él se listan 12 imágenes de entrenamiento y 2 de pruebas.

Los datos de entrenamiento se dividen en 3 grupos. Cada grupo utilizará 8 imágenes para entrenar y 4 para validar. El tamaño de la dimensión se define por `sim_dim`. Cada conjunto de validación solo tiene 4 imágenes en un grupo de entrenamiento.

`sim_datalist` es el equivalente a `dataset_0.json` que hemos usado en los otros cuadernos

```

sim_datalist = {
    "testing": [
        {"image": "test_image_001.nii.gz", "label": "test_label_001.nii.gz"},
        {"image": "test_image_002.nii.gz", "label": "test_label_002.nii.gz"},
    ],
    "training": [
        {"fold": 0, "image": "tr_image_001.nii.gz", "label": "tr_label_001.nii.gz"},
        {"fold": 0, "image": "tr_image_002.nii.gz", "label": "tr_label_002.nii.gz"},
        {"fold": 0, "image": "tr_image_003.nii.gz", "label": "tr_label_003.nii.gz"},
        {"fold": 0, "image": "tr_image_004.nii.gz", "label": "tr_label_004.nii.gz"},
        {"fold": 0, "image": "tr_image_005.nii.gz", "label": "tr_label_005.nii.gz"},
        {"fold": 1, "image": "tr_image_006.nii.gz", "label": "tr_label_006.nii.gz"},
        {"fold": 1, "image": "tr_image_007.nii.gz", "label": "tr_label_007.nii.gz"},
        {"fold": 1, "image": "tr_image_008.nii.gz", "label": "tr_label_008.nii.gz"},
        {"fold": 2, "image": "tr_image_009.nii.gz", "label": "tr_label_009.nii.gz"},
    ]
}

```

```

        {"fold": 2, "image": "tr_image_010.nii.gz", "label": "tr_label_010.nii.gz"},  

        {"fold": 2, "image": "tr_image_011.nii.gz", "label": "tr_label_011.nii.gz"},  

        {"fold": 2, "image": "tr_image_012.nii.gz", "label": "tr_label_012.nii.gz"},  

    ],  

}  
  

sim_dim = (64, 64, 64)  
  

# debemos establecer un límite en el número total de GPUs que estamos utilizando. Al ser con Colab solo tendremos 1 GPU  

if torch.cuda.device_count() > 4:  

    os.environ["CUDA_DEVICE_ORDER"] = "PCI_BUS_ID"  

    os.environ["CUDA_VISIBLE_DEVICES"] = "0,1,2,3"

```

▼ Generar imágenes y etiquetas simuladas

Utilizamos las funciones 'create_test_image_3d' de MONAI y 'nib.Nifti1Image' para generar las imágenes 3D simuladas en el directorio de trabajo './solucionintegralsegmentacion'.

```

work_dir = "./solucionintegralsegmentacion"  

if not os.path.isdir(work_dir):  

    os.makedirs(work_dir)  
  

dataroot_dir = os.path.join(work_dir, "sim_dataroot")  

if not os.path.isdir(dataroot_dir):  

    os.makedirs(dataroot_dir)  
  

datalist_file = os.path.join(work_dir, "sim_datalist.json")  

with open(datalist_file, "w") as f:  

    json.dump(sim_datalist, f)  
  

for d in sim_datalist["testing"] + sim_datalist["training"]:  

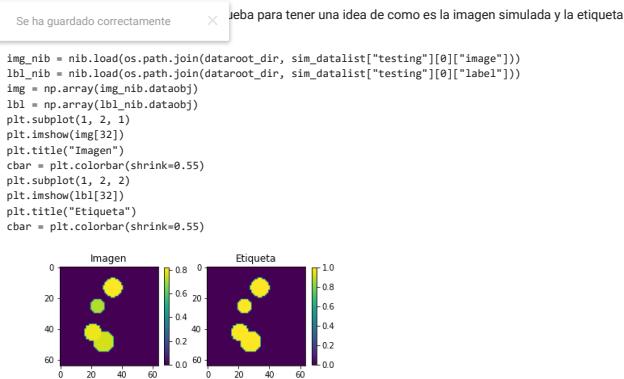
    im, seg = create_test_image_3d(  

        sim_dim[0], sim_dim[1], sim_dim[2], rad_max=10, num_seg_classes=1, random_state=np.random.RandomState(42)
    )  

    image_fpath = os.path.join(dataroot_dir, d["image"])
    label_fpath = os.path.join(dataroot_dir, d["label"])
    nib.save(nib.Nifti1Image(im, affine=np.eye(4)), image_fpath)
    nib.save(nib.Nifti1Image(seg, affine=np.eye(4)), label_fpath)

```

▼ Inspeccionar los datos simulados



▼ Ejecutar el AutoRunner de la Solución Integral de Auto3DSeg

AutoRunner automáticamente configurará cuatro redes neuronales diferentes y realizará un entrenamiento multigrupo para lograr el mejor rendimiento. El módulo es altamente configurable y solo requiere una entrada mínima.

Configuramos el Autorunner:

```

runner = AutoRunner(  

    work_dir=work_dir,  

    input={  

        "modality": "MRI",  

        "datalist": datalist_file,  

        "dataroot": dataroot_dir,  

    },  

)

```

```
2023-02-05 08:58:02,648 - INFO - AutoRunner using work directory ./solucionintegralsegmentacion
2023-02-05 08:58:02,653 - INFO - The output_dir is not specified. ./content/solucionintegralsegmentacion/ensemble_output will be used to save ensemble
2023-02-05 08:58:02,654 - INFO - Directory /content/solucionintegralsegmentacion/ensemble_output is created to save ensemble predictions
```

▼ Configuracion para ejecución rápida

Sobrescribimos los parámetros de entrenamiento para que podamos completar el pipeline en pocos minutos.

```
max_epochs = 2

# safeguard to ensure max_epochs is greater or equal to 2
# aseguramos que el numero de epochas sea al menos de 2
max_epochs = max(max_epochs, 2)

train_param = {
    "CUDA_VISIBLE_DEVICES": [0], # usamos solo una GPU puesto que es lo que tiene colab
    "num_iterations": 4 * max_epochs,
    "num_iterations_per_validation": 2 * max_epochs,
    "num_images_per_batch": 2,
    "num_epochs": max_epochs,
    "num_warmup_iterations": 2 * max_epochs,
}
runner.set_training_params(train_param)
runner.set_num_fold(num_fold=1)
```

▼ Ejecutamos el Autorunner

Autorunner realiza el análisis de datos, generación de algoritmos, entrenamiento y ensamblado de modelos.

```
runner.run()

2023-02-05 08:58:02,689 - INFO - Running data analysis...
100%[██████████] 12/12 [00:06<00:00, 1.76it/s]
2023-02-05 08:58:10,213 - INFO - Expected md5 is None, skip md5 check for file /tmp/tmp8qt_n4eb/algo_temp/templates.tar.gz.
2023-02-05 08:58:10,214 - INFO - Downloaded: /tmp/tmp8qt_n4eb/algo_temp/templates.tar.gz.

2023-02-05 08:58:11,075 - INFO - /content/solucionintegralsegmentacion/segresnet2d_0
2023-02-05 08:58:11,572 - INFO - /content/solucionintegralsegmentacion/dints_0
2023-02-05 08:58:11,872 - INFO - /content/solucionintegralsegmentacion/swinunetr_0
2023-02-05 08:58:11,875 - INFO - /content/solucionintegralsegmentacion/segresnet_0
2023-02-05 08:58:12,616 - INFO - /content/solucionintegralsegmentacion/dints_0/scripts/search.py run --config_file='/content/solucionintegralsegmentacion/dints_0/config.yaml'
2023-02-05 08:58:17,747 - INFO - Launching: python /content/solucionintegralsegmentacion/dints_0/scripts/train.py run --config_file='/content/solucionintegralsegmentacion/dints_0/config.yaml'
2023-02-05 08:59:21,002 - INFO - Launching: python /content/solucionintegralsegmentacion/segresnet2d_0/scripts/train.py run --config_file='/content/solucionintegralsegmentacion/segresnet2d_0/config.yaml'
2023-02-05 08:59:37,933 - INFO - Launching: python /content/solucionintegralsegmentacion/segresnet_0/scripts/train.py run --config_file='/content/solucionintegralsegmentacion/segresnet_0/config.yaml'
2023-02-05 09:00:00,889 - INFO - Launching: python /content/solucionintegralsegmentacion/swinunetr_0/scripts/train.py run --config_file='/content/solucionintegralsegmentacion/swinunetr_0/config.yaml'
[info] ha guardado correctamente
[info] checkpoint /content/solucionintegralsegmentacion/swinunetr_0/model_fold0/best_metric_model.pt loaded
2023-02-05 09:00:36,538 INFO image_writer.py:194 - writing: /content/solucionintegralsegmentacion/swinunetr_0/prediction_testing/test_image_001.tif
[info] checkpoint /content/solucionintegralsegmentacion/swinunetr_0/model_fold0/best_metric_model.pt loaded
2023-02-05 09:00:37,544 INFO image_writer.py:194 - writing: /content/solucionintegralsegmentacion/swinunetr_0/prediction_testing/test_image_002.tif
Auto3Dseg picked the following networks to ensemble:
swinunetr_0
2023-02-05 09:00:37,574 INFO image_writer.py:194 - writing: /content/solucionintegralsegmentacion/ensemble_output/test_image_001/test_image_001_en
2023-02-05 09:00:37,588 INFO image_writer.py:194 - writing: /content/solucionintegralsegmentacion/ensemble_output/test_image_002/test_image_002_en
2023-02-05 09:00:37,599 - INFO - Auto3Dseg ensemble prediction outputs are saved in /content/solucionintegralsegmentacion/ensemble_output.
2023-02-05 09:00:37,603 - INFO - Auto3Dseg pipeline is complete successfully.
```

▼ Inspeccionar las predicciones ensambladas del algoritmo y compararlas con el Ground Truth (GT)

```
image_name = sim_datalist["testing"][[0]]["image"].split(".")[0]
prediction_nib = nib.load(os.path.join(work_dir, "ensemble_output", image_name, image_name + "_ensemble" + ".nii.gz"))
pred = np.array(prediction_nib.dataobj)

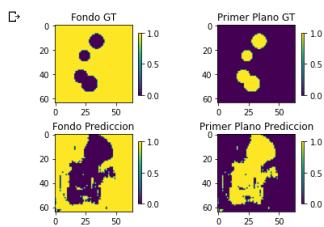
img_slice32 = lbl[32] == 0
label_slice32 = lbl[32] == 1
background_slice32 = pred[32, :, :, 0] if pred.ndim == 4 else pred[32] == 0
foreground_slice32 = pred[32, :, :, 1] if pred.ndim == 4 else pred[32] == 1

plt.subplot(2, 2, 1)
plt.imshow(img_slice32)
plt.title("Fondo GT")
cbar = plt.colorbar(shrink=0.8)
plt.subplot(2, 2, 2)
plt.imshow(label_slice32)
plt.title("Primer Plano GT")
cbar = plt.colorbar(shrink=0.8)
plt.subplot(2, 2, 3)
plt.imshow(background_slice32)
plt.title("Fondo Prediccion")
plt.subplot(2, 2, 4)
plt.title("Primer Plano Prediccion")
```

```

cbar = plt.colorbar(shrink=0.8)
plt.subplot(2, 2, 4)
plt.imshow(foreground_slice32)
plt.title("Primer Plano Prediccion")
cbar = plt.colorbar(shrink=0.8)
# set the spacing between subplots
plt.subplots_adjust(left=0.1, bottom=0.1, right=0.9, top=0.9, wspace=0.4, hspace=0.4)
plt.show()

```



Conclusion

Podemos ver que los algoritmos comienzan a aprender los conjuntos de datos y la predicción de véxeles de fondo y primer plano. Auto3DSeg y AutoRunner son altamente configurables. Para obtener mejores resultados, se puede aumentar el tiempo de entrenamiento, aplicar un método de ensamblado diferente o utilizar la optimización de hiperparámetros a través de las APIs de módulo AutoRunner o Auto3DSeg.

Se ha guardado correctamente

Productos de pago de Colab - Cancelar contratos

✓ 0 s completado a las 10:00

● ✕

A.5. Cuaderno: Servidor de MONAI Label

El presente cuaderno de jupyter

Durante la ejecución del cuaderno, se pide que se reinicie el entorno. Además es importante comentar que se genera un proxy para poder conectarse al servidor, aunque este proxy solo es accesible desde un navegador de Chrome que tenga iniciada la sesión con el mismo usuario que el cuaderno de Google Colab.

Ha continuación se adjunta el código del cuaderno y la ejecución del mismo.

El cuaderno se encuentra disponible en el repositorio Github del autor del TFM, a través de la URL:

[https://github.com/franguerrero/TFM_SegmentacionImagenesMedicas/
blob/main/cuadernos_jupyter/TFM_ServidorMonaiLabel.ipynb](https://github.com/franguerrero/TFM_SegmentacionImagenesMedicas/blob/main/cuadernos_jupyter/TFM_ServidorMonaiLabel.ipynb)

[Open in Colab](#)

```
#Instalo Monai Label
# Voy a tener que reiniciar el entorno, lo pide al instalar
!pip install monailabel

#Requirement already satisfied: attrs>=1.4.0 in /usr/local/lib/python3.8/dist-packages (from jsonschema<4.0.0,>=3.2.0->pydicom-seg==0.4.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.8/dist-packages (from jsonschema<4.0.0,>=3.2.0->pydicom-seg==0.4.0)
Requirement already satisfied: matplotlib>=3.0.0,>>2.0.0 in /usr/local/lib/python3.8/dist-packages (from scikit-image>=0.14.2->monai[fire,gdown,i])
Requirement already satisfied: networkx>2.0 in /usr/local/lib/python3.8/dist-packages (from scikit-image>=0.14.2->monai[fire,gdown,ig])
Requirement already satisfied: lifffile>=2019.7.0 in /usr/local/lib/python3.8/dist-packages (from scikit-image>=0.14.2->monai[fire,gd])
Requirement already satisfied: PyWavelets>1.1.1 in /usr/local/lib/python3.8/dist-packages (from scikit-image>=0.14.2->monai[fire,gdw])
Requirement already satisfied: temcolor in /usr/local/lib/python3.8/dist-packages (from fire->monai[fire,gdown,ignite,itk,lmb,mflow])
Requirement already satisfied: pilLOW>2.1 in /usr/local/lib/python3.8/dist-packages (from mflow->monai[fire,gdown,ignite,itk,lmb,mflow])
Requirement already satisfied: pilLOW>1.1.1 in /usr/local/lib/python3.8/dist-packages (from mflow->monai[fire,gdown,ignite,itk,lmb,mflow])
Requirement already satisfied: Flask3 in /usr/local/lib/python3.8/dist-packages (from mflow->monai[fire,gdown,ignite,itk,lmb,mflow])
Requirement already satisfied: importlib-metadata>4.7.0,>>4.7.0 in /usr/local/lib/python3.8/dist-packages (from mflow->monai[fire,gdown,ignite,itk])
Requirement already satisfied: pyarrow<11,>>4.0.0 in /usr/local/lib/python3.8/dist-packages (from mflow->monai[fire,gdown,ignite,itk,lmb,mflow])
Requirement already satisfied: cloudpickle<2.0 in /usr/local/lib/python3.8/dist-packages (from mflow->monai[fire,gdown,ignite,itk,lmb])
Requirement already satisfied: markdown<4,>>3.3 in /usr/local/lib/python3.8/dist-packages (from mflow->monai[fire,gdown,ignite,itk,lmb])
Requirement already satisfied: databricks-clix1,>>0.8.7 in /usr/local/lib/python3.8/dist-packages (from mflow->monai[fire,gdown,ignite,itk,lmb,mflow])
Requirement already satisfied: alembic<2 in /usr/local/lib/python3.8/dist-packages (from mflow->monai[fire,gdown,ignite,itk,lmb,mfl])
Requirement already satisfied: querystring-parser<2 in /usr/local/lib/python3.8/dist-packages (from mflow->monai[fire,gdown,ignite,itk])
Requirement already satisfied: shap<1,>>0.40 in /usr/local/lib/python3.8/dist-packages (from mflow->monai[fire,gdown,ignite,itk,lmb])
Requirement already satisfied: unicorn21 in /usr/local/lib/python3.8/dist-packages (from mflow->monai[fire,gdown,ignite,itk,lmb,mfl])
Requirement already satisfied: sqlalchemy<2,>>1.4.0 in /usr/local/lib/python3.8/dist-packages (from mflow->monai[fire,gdown,ignite,itk])
Requirement already satisfied: gitpython<4,>>2.1.0 in /usr/local/lib/python3.8/dist-packages (from mflow->monai[fire,gdown,ignite,itk])
Requirement already satisfied: docker7,>>4.0.0 in /usr/local/lib/python3.8/dist-packages (from mflow->monai[fire,gdown,ignite,itk,lmb])
Requirement already satisfied: entrypoints<1 in /usr/local/lib/python3.8/dist-packages (from mflow->monai[fire,gdown,ignite,itk,lmb])
Requirement already satisfied: Jinja2<4,>>2.11 in /usr/local/lib/python3.8/dist-packages (from mflow->monai[fire,gdown,ignite,itk,lmb])
Requirement already satisfied: protobuf<5,>>3.12.0 in /usr/local/lib/python3.8/dist-packages (from mflow->monai[fire,gdown,ignite,itk])
Requirement already satisfied: pytz<2023 in /usr/local/lib/python3.8/dist-packages (from mflow->monai[fire,gdown,ignite,itk,lmb,mfl])
Requirement already satisfied: grpcio<1.24.3 in /usr/local/lib/python3.8/dist-packages (from tensorflow->monai[fire,gdown,ignite,itk])
Requirement already satisfied: google-auth-oauthlib<0.5,>>0.4.1 in /usr/local/lib/python3.8/dist-packages (from tensorflow->monai[fire,gdown,ignite,itk])
Requirement already satisfied: wheel<0.26 in /usr/local/lib/python3.8/dist-packages (from tensorflow->monai[fire,gdown,ignite,itk,lmb])
Requirement already satisfied: tensorboard>2.0 in /usr/local/lib/python3.8/dist-packages (from tensorflow->monai[fire,gdown,ignite,itk])
Requirement already satisfied: tensorboard-plugin-wit>1.6.0 in /usr/local/lib/python3.8/dist-packages (from tensorflow->monai[fire,gd])
Requirement already satisfied: werkzeug<1.0.1 in /usr/local/lib/python3.8/dist-packages (from tensorboard->monai[fire,gdown,ignite,itk])
Requirement already satisfied: absl-py<0.4 in /usr/local/lib/python3.8/dist-packages (from tensorboard->monai[fire,gdown,ignite,itk,l])
Requirement already satisfied: google-auth<3,>>1.6.3 in /usr/local/lib/python3.8/dist-packages (from tensorboard->monai[fire,gdown,ign])
Requirement already satisfied: Make in /usr/local/lib/python3.8/dist-packages (from alembic<2>-->monai[fire,gdown,ignite,itk,lmb])
Requirement already satisfied: importlib-resource in /usr/local/lib/python3.8/dist-packages (from alembic<2>-->monai[fire,gdown,ignite,itk])
Requirement already satisfied: pyjwt<1.7.0 in /usr/local/lib/python3.8/dist-packages (from databricks-clix1,>>0.8.7->mflow->monai[fi])
Requirement already satisfied: tabulate>0.7.7 in /usr/local/lib/python3.8/dist-packages (from databricks-clix1,>>0.8.7->mflow->monai)
Requirement already satisfied: authlib<3.1.0 in /usr/local/lib/python3.8/dist-packages (from tensorflow->monai[fire,gd])
Requirement already satisfied: websocket-client<0.32.0 in /usr/local/lib/python3.8/dist-packages (from docker7,>>4.0.0->mflow->monai)
Requirement already satisfied: itsdangerous<2.0,>>0.2.0 in /usr/local/lib/python3.8/dist-packages (from Flask<3>->mflow->monai[fire,gd])
Requirement already satisfied: gitdb5,>>4.0.1 in /usr/local/lib/python3.8/dist-packages (from gitpython<4,>>2.1.0->mflow->monai[fire])
Requirement already satisfied: pyasn1-modules<0.2.1 in /usr/local/lib/python3.8/dist-packages (from google-auth<3,>>1.6.3->tensorboar)
Requirement already satisfied: requests-oauthlib<0.7.0 in /usr/local/lib/python3.8/dist-packages (from google-auth-oauthlib<0.5,>>0.4)
Requirement already satisfied: zipp>0.5 in /usr/local/lib/python3.8/dist-packages (from importlib-metadata<4.7.0,>>4.7.0->3.7.0->mflow)
Requirement already satisfied: MarkupSafe<0.23 in /usr/local/lib/python3.8/dist-packages (from Jinja2<4,>>2.11->mflow->monai[fire,gd])
Requirement already satisfied: cylcer>=10 in /usr/local/lib/python3.8/dist-packages (from matplotlib<3.0.0,>>2.0.0->scikit-image>0)
Requirement already satisfied: kiwisolver>1.0.1 in /usr/local/lib/python3.8/dist-packages (from matplotlib<3.0.0,>>2.0.0->scikit-im)
Requirement already satisfied: python-dateutil>2.1 in /usr/local/lib/python3.8/dist-packages (from matplotlib<3.0.0,>>2.0.0->scikit-im)
Requirement already satisfied: pyParsing!>2.0.4,>=2.1.2,>=2.6,>=2.0.1 in /usr/local/lib/python3.8/dist-packages (from matplotlib<3.0.0,>>2.0.0->scikit-im)
Requirement already satisfied: silencer<0.3.0 in /usr/local/lib/python3.8/dist-packages (from matplotlib<3.0.0,>>2.0.0->scikit-im)
Requirement already satisfied: greenlet<0.4.17 in /usr/local/lib/python3.8/dist-packages (from shapeless,>>0.40->mflow->monai[fire,gdown,ignite,it])
Requirement already satisfied: greenlet<0.4.17 in /usr/local/lib/python3.8/dist-packages (from sqlalchemy<2,>>1.4.0->mflow->monai[fi])
Requirement already satisfied: PySocks<1.5.7,>>1.5.6 in /usr/local/lib/python3.8/dist-packages (from requests<=2.28.1->monailabel<1)
Requirement already satisfied: smmap6,>>3.0.1 in /usr/local/lib/python3.8/dist-packages (from gitdb5,>>4.0.1->gitpython<4,>>2.1.0->m)
Requirement already satisfied: llvmlite<0.40,>>0.39.0dev0 in /usr/local/lib/python3.8/dist-packages (from numba->shap1,>>0.40->mflow
```

Clono el repo de github de monai label ya que necesito tener las sample-apps porque no puedo usar directamente el comando de monailabel apps
!git clone https://github.com/Project-MONAI/MONAILabel.git

fatal: destination path 'MONAILabel' already exists and is not an empty directory.

Me creo una carpeta de test y copio ahí las sample-apps para no tener problemas

```
!mkdir /content/test
!mkdir /content/test/monailabel
!cp -r /content/MONAILabel/sample-apps/ /content/test/monailabel/
mkdir: cannot create directory '/content/test': File exists
mkdir: cannot create directory '/content/test/monailabel': File exists
```

Descargo la APP haciendo referencia en el prefix a la carpeta que hemos bajado antes
!monailabel apps --download --name radiology --output /content/sample-apps/ --prefix /content/test

```
Using PYTHONPATH=/usr:/env/python
radiology is copied at: /content/sample-apps/radiology

# Descargo los datos de prueba de MSD de Spleen
!monailabel datasets --download --name Task09_Spleen --output .

Using PYTHONPATH=/usr:/env/python
Task09_Spleen.tar: 1.50GB [02:15, 11.9MB/s]
2023-02-05 12:38:55,382 - INFO - Downloaded: Task09_Spleen.tar
2023-02-05 12:38:58,281 - INFO - Verified 'Task09_Spleen.tar', md5: 410d4a301da4e5b2ff86ec3ddba524e.
2023-02-05 12:38:58,281 - INFO - Non-empty folder exists in /content/Task09_Spleen, skipped extracting.
Task09_Spleen is downloaded at: ./Task09_Spleen

from google.colab.output import eval_js
print(eval_js("google.colab.kernel.proxyPort(8000)"))

https://mjoinh0j2jk-496ff2e9cd22116-8000-colab.googleusercontent.com/

#Inicializo el servidor
!monailabel start_server --app /content/sample-apps/radiology -studies Task09_Spleen/imagesTr --conf models deepedit
2023-02-05 12:39:07,802 - ENV SETTINGS:: MONAI_LABEL_TASKS_MAIN = True
2023-02-05 12:39:07,803 - ENV SETTINGS:: MONAI_LABEL_TASKS_STRATEGY = True
2023-02-05 12:39:07,803 - ENV SETTINGS:: MONAI_LABEL_TASKS_SCORING = True
2023-02-05 12:39:07,803 - ENV SETTINGS:: MONAI_LABEL_TASKS_BATCH_INFERENCE = True
2023-02-05 12:39:07,803 - ENV SETTINGS:: MONAI_LABEL_DATASTORE =
2023-02-05 12:39:07,803 - ENV SETTINGS:: MONAI_LABEL_DATASTORE_URL =
2023-02-05 12:39:07,804 - ENV SETTINGS:: MONAI_LABEL_DATASTORE_USERNAME =
2023-02-05 12:39:07,804 - ENV SETTINGS:: MONAI_LABEL_DATASTORE_PASSWORD =
2023-02-05 12:39:07,804 - ENV SETTINGS:: MONAI_LABEL_DATASTORE_API_KEY =
```

```
2023-02-05 12:39:07,805 - ENV SETTINGS:: MONAI_ZOO_SOURCE = "github"
2023-02-05 12:39:07,805 - ENV SETTINGS:: MONAI_ZOO_REPO = Project-MONAI/model-zoo/hosting_storage_v1
2023-02-05 12:39:07,805 - ENV SETTINGS:: MONAI_ZOO_AUTH_TOKEN =
2023-02-05 12:39:07,805 - ENV SETTINGS:: MONAI_LABEL_AUTO_UPDATE_SCORING = True
2023-02-05 12:39:07,805 -
Allow Origins: ['*']
[2023-02-05 12:39:08,960] [2096] [MainThread] [INFO] (uvicorn.error:75) - Started server process [2096]
[2023-02-05 12:39:08,960] [2096] [MainThread] [INFO] (uvicorn.error:45) - Waiting for application startup.
[2023-02-05 12:39:08,961] [2096] [MainThread] [INFO] (monailabel.interfaces.utils.app:38) - Initializing App from: /content/sample-app
[2023-02-05 12:39:09,851] [2096] [MainThread] [INFO] (monailabel.utils.others.class_utils:37) - Subclass for MONAILabelApp Found: <class
[2023-02-05 12:39:09,865] [2096] [MainThread] [INFO] (monailabel.utils.others.class_utils:37) - Subclass for TaskConfig Found: <class
[2023-02-05 12:39:09,866] [2096] [MainThread] [INFO] (monailabel.utils.others.class_utils:37) - Subclass for TaskConfig Found: <class
[2023-02-05 12:39:09,868] [2096] [MainThread] [INFO] (monailabel.utils.others.class_utils:37) - Subclass for TaskConfig Found: <class
[2023-02-05 12:39:09,869] [2096] [MainThread] [INFO] (monailabel.utils.others.class_utils:37) - Subclass for TaskConfig Found: <class
[2023-02-05 12:39:09,870] [2096] [MainThread] [INFO] (monailabel.utils.others.class_utils:37) - Subclass for TaskConfig Found: <class
[2023-02-05 12:39:09,872] [2096] [MainThread] [INFO] (monailabel.utils.others.class_utils:37) - Subclass for TaskConfig Found: <class
[2023-02-05 12:39:09,873] [2096] [MainThread] [INFO] (monailabel.utils.others.class_utils:37) - Subclass for TaskConfig Found: <class
[2023-02-05 12:39:09,874] [2096] [MainThread] [INFO] (monailabel.utils.others.class_utils:37) - Subclass for TaskConfig Found: <class
[2023-02-05 12:39:09,874] [2096] [MainThread] [INFO] (main:9) - *** Admin Model. AdminEdit --> 1th runfile AdminEdit RunEdit
```

Colab paid products - Cancel contracts here
Executing (2s) Cell > system() > _system_compat() > _run_command() > _monitor_process() > _poll_process()

