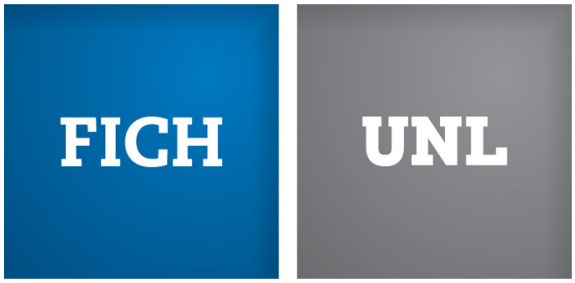

UNIVERSIDAD NACIONAL DEL LITORAL
Facultad de Ingeniería y Ciencias Hídricas

PROYECTO FINAL DE CARRERA

Informe Final

Diseño y desarrollo de una herramienta para la segmentación automática en imágenes TAC de pacientes con traumatismo de cráneo mediante redes neuronales convolucionales

Alumno: Victor Franco Matzkin
Director: Dr. Enzo Ferrante

Santa Fe, Marzo de 2019

sinc(*i*) Research Institute for Signals, Systems and Computational Intelligence (sinc.unl.edu.ar)
F. Matzkin & E. Ferrante; "Diseño y desarrollo de una herramienta para la segmentación automática en imágenes TAC de pacientes con traumatismo de cráneo mediante redes neuronales convolucionales. (Undergraduate project) "
Facultad de Ingeniería y Ciencias Hídricas - Universidad Nacional del Litoral, 2019.

Resumen

Este proyecto propone el desarrollo de una herramienta abierta y robusta para la segmentación de la cavidad craneal en imágenes de tomografía axial computada (TAC) de pacientes con traumatismo de cráneo. En particular, se trabajará con pacientes que han sido sometidos a cirugías de craniectomía descompresiva, debido a las dificultades en la segmentación de estas imágenes con las técnicas clásicas del procesamiento digital de imágenes.

Para el desarrollo de la herramienta se adoptó un enfoque basado en aprendizaje automático mediante redes neuronales convolucionales profundas. Dichas redes fueron entrenadas utilizando una serie de imágenes previamente etiquetadas. Durante el desarrollo de este proyecto, se implementaron distintas arquitecturas de redes convolucionales, y se realizó un estudio comparativo de diversas funciones de pérdida para su entrenamiento. Finalmente, se desarrolló una herramienta de uso simple que permite segmentar imágenes cerebrales de tomografía computarizada, por medio de los modelos de redes neuronales propuestos.

Palabras clave: segmentación cerebral, TAC, traumatismo de cráneo, redes neuronales convolucionales, restricciones anatómicas.

sinc(*i*) Research Institute for Signals, Systems and Computational Intelligence (sinc.unl.edu.ar)
F. Matzkin & E. Ferrante; "Diseño y desarrollo de una herramienta para la segmentación automática en imágenes TAC de pacientes con traumatismo de cráneo mediante redes neuronales convolucionales. (Undergraduate project) "
Facultad de Ingeniería y Ciencias Hídricas - Universidad Nacional del Litoral, 2019.

Agradecimientos

Agradezco en primer lugar a mi director Enzo Ferrante, quien desde un primer momento me motivó y orientó para realizar este proyecto, así como también me abrió las puertas del **s**ı**nc**(i) para desarrollarlo, y se preocupó por darme una mano siempre que fuera necesario.

A los profesores y la comunidad de la facultad, en especial a aquellos que motivaron mi interés hacia estas temáticas, y a la cátedra de Proyecto Final de Carrera, la cual estuvo a disposición cuando lo necesité.

A la universidad pública, que durante todo este tiempo supo ser mi casa, me permitió viajar y me llenó de satisfacciones en lo profesional y en lo personal.

A mis amigos y todas aquellas maravillosas personas que aportaron un granito de arena para que yo esté hoy acá. Me llenaron de fuerzas cuando veía lejano este momento, y confiaron en mí más que yo mismo.

Finalmente, gracias a mi familia, por apoyarme y acompañarme incondicionalmente en esta y todas las etapas de mi vida.

sinc(*i*) Research Institute for Signals, Systems and Computational Intelligence (sinc.unl.edu.ar)
F. Matzkin & E. Ferrante; "Diseño y desarrollo de una herramienta para la segmentación automática en imágenes TAC de pacientes con traumatismo de cráneo mediante redes neuronales convolucionales. (Undergraduate project) "
Facultad de Ingeniería y Ciencias Hídricas - Universidad Nacional del Litoral, 2019.

Índice general

sinc(*i*) Research Institute for Signals, Systems and Computational Intelligence (sinc.unl.edu.ar)
F. Matzkin & E. Ferrante; "Diseño y desarrollo de una herramienta para la segmentación automática en imágenes TAC de pacientes con traumatismo de cráneo mediante redes neuronales convolucionales. (Undergraduate project) " Facultad de Ingeniería y Ciencias Hídricas - Universidad Nacional del Litoral, 2019.

Índice de tablas

Índice de figuras

sinc(*i*) Research Institute for Signals, Systems and Computational Intelligence (sinc.unl.edu.ar)
F. Matzkin & E. Ferrante; "Diseño y desarrollo de una herramienta para la segmentación automática en imágenes TAC de pacientes con traumatismo de cráneo mediante redes neuronales convolucionales. (Undergraduate project) " Facultad de Ingeniería y Ciencias Hídricas - Universidad Nacional del Litoral, 2019.

Capítulo 1

Introducción

1.1. Justificación

En la actualidad, las tecnologías de adquisición de imágenes médicas constituyen un elemento fundamental para el diagnóstico y tratamiento de enfermedades en el ámbito clínico, ya que permiten, de una manera rápida y no invasiva, confirmar el diagnóstico de una determinada patología o definir el curso de un tratamiento específico.

Entre las distintas modalidades se encuentran las imágenes de rayos X, ecografía, tomografía axial computarizada (TAC) y resonancia magnética (IRM), entre otras. Para construir la imagen, las mismas hacen uso de diversas tecnologías como la radiación ionizante, ultrasonido o campos magnéticos. Dependiendo del equipo de captura y la necesidad clínica, las imágenes generadas pueden ser bidimensionales (como es el caso de una radiografía de fémur), o tridimensionales (como una tomografía de cabeza).

Este trabajo se centrará en el procesamiento de imágenes tridimensionales de TAC de cabeza, con foco en el caso de pacientes con traumatismo de cráneo. Dicho traumatismo se define como una lesión física del cerebro que puede ocasionar una amplia variedad de problemas médicos, psicológicos y de comportamiento. Éste puede ocurrir cuando se le aplica una fuerza al cerebro, ya sea por un impacto directo en la cabeza, o por la rápida aceleración y desaceleración, como es el caso de un latigazo cervical (ASNR, 2013).

A nivel mundial, anualmente 50 millones de personas sufren de traumatismo de cráneo (Maas y Menon, 2017), cifra que se estima que seguirá aumentando (Menon y Zahed, 2009). Además, en los países menos desarrollados existe un mayor riesgo de muerte luego de sufrir un traumatismo craneal grave, en relación a los países desarrollados (De Silva *et al.*, 2009). Por esta razón, el desarrollo de herramientas computacionales de apoyo al diagnóstico y tratamiento de dicha patología, en especial en la etapa temprana de la lesión (momento crítico en el cual, tomando las medidas correspondientes, pueden prevenirse daños futuros irreversibles) resulta fundamental.

Una de las principales complicaciones asociadas al traumatismo de cráneo en la etapa aguda del trauma, es el aumento de la presión intracraneal. Para aliviar esta presión y evitar daños irreparables en el cerebro, una de las técnicas más utilizadas es la llamada craniectomia descompresiva. Esta cirugía consiste en extraer una porción del cráneo con el objetivo de efectuar una evacuación del hematoma y cauterización de los vasos que están sangrando (Galgano *et al.*, 2017).

Posterior a esta intervención, con el objetivo de detectar potenciales complicaciones como hemorragias, isquemia cerebral, o la formación de edemas, se obtienen tomografías computarizadas de forma rutinaria (Freyschlag *et al.*, 2018). En la Figura 1.1 se muestran ejemplos de imágenes antes y después de la craniectomía.

La TAC es utilizada principalmente en la fase aguda del trauma, durante las primeras 24 horas posteriores al incidente (Lee y Newberg, 2005), con el objetivo de determinar el estado del paciente, evaluar si existe riesgo de vida y proceder a realizar una cirugía de emergencia en ese caso. Las imágenes de TAC son capturadas rápidamente, permiten detectar sangrado intra e intercerebral y los equipos de captura se encuentran disponibles en gran parte del sistema de salud público argentino. Sin embargo, las lesiones microscópicas (que suelen acarrear problemas a largo plazo en pacientes con traumatismo de cráneo) no pueden ser detectadas con los sistemas de imágenes TAC actuales. Por esta razón, luego del periodo agudo del traumatismo, se suele realizar un imagen de resonancia magnética (IRM) cerebral, para detectar microhemorragias, pequeñas contusiones o cicatrices que no son visibles en las imágenes TAC.

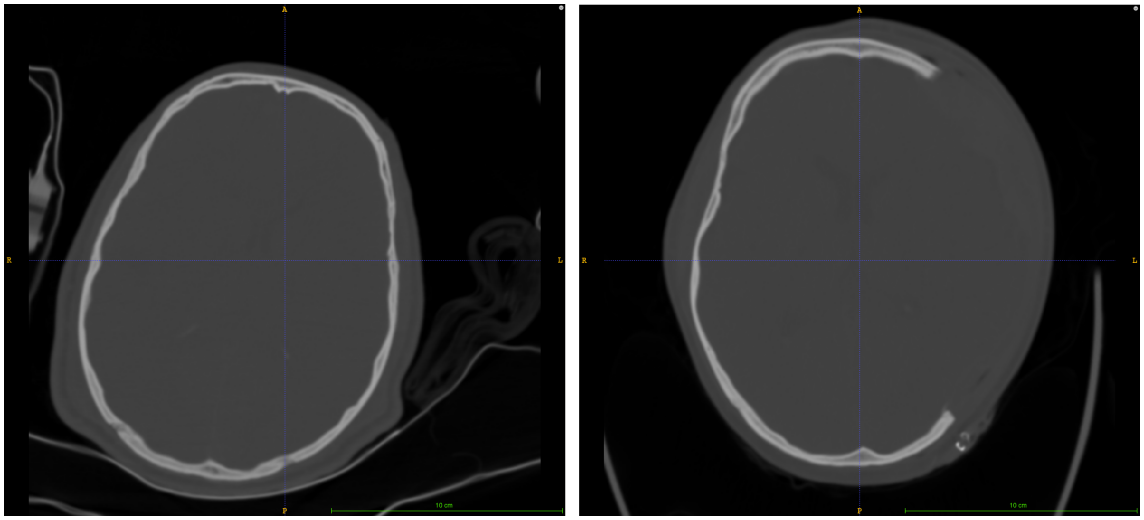

Figura 1.1: Corte axial de imágenes TAC de un paciente antes y después de haber sido sometido a una craniectomía decompresiva. En las imágenes, el cráneo se ve en color blanco. Nótese que en la imagen de la derecha, una porción del mismo ha sido extraída.

Pese a presentar un diagnóstico más detallado, las IRM suelen ser más costosas, lentas y de menor disponibilidad de equipamiento en el sistema de salud público que la tecnología TAC. Por tal motivo, durante los últimos años se han presentado diversos estudios tendientes a explotar al máximo las imágenes TAC de pacientes con traumatismo de cráneo, intentando predecir el desarrollo futuro de la patología en base a indicadores cuantitativos extraídos de las TAC adquiridas en la fase aguda del trauma (Yuh *et al.*, 2012). Para llevar a cabo la cuantificación de dichas lesiones, un paso fundamental consiste en segmentar la región de la imagen que corresponde al cerebro, lo cual se ilustra en la figura 1.2. Esta tarea, conocida como segmentación de la cavidad craneal (o *skull stripping* en inglés) es fundamental para eliminar zonas de la imagen como el cráneo, ojos y demás partes que no resultan de interés en el análisis de las lesiones. Existen numerosos trabajos y herramientas que abordan esta problemática en IRM (Kalavathi y Prasath, 2016).

Sin embargo, y pese a que han sido publicados algunos trabajos científicos en los que se estudia este problema en imágenes TAC (Patel *et al.*, 2017), no existe actualmente ninguna herramienta abierta (en el sentido de que su código fuente se encuentre disponible) que permita realizar segmentación de la cavidad craneal en imágenes TAC, y mucho menos que sea capaz de lidiar con casos que han sido sometidos a una craniectomía descompresiva.

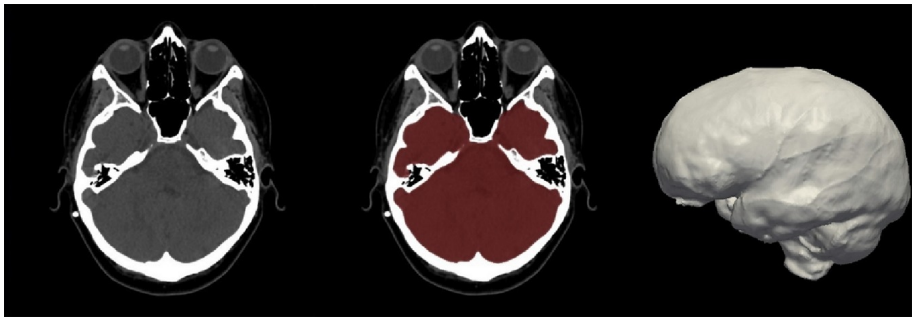

Figura 1.2: TAC de cráneo. En rojo se puede ver la región del cerebro que se busca segmentar (Patel *et al.*, 2017).

Este proyecto propone en consecuencia el desarrollo e implementación de una herramienta abierta para la segmentación de la cavidad craneal en imágenes TAC de pacientes con traumatismo de cráneo. En particular, el interés radica en pacientes que han sido sometidos a cirugías de craniectomía descompresiva, dado que este tipo de imágenes dificulta el funcionamiento de los algoritmos existentes (Patel *et al.*, 2017). Este problema ha sido abordado por otros autores mediante diferentes técnicas, tales como registración de imágenes (Patel *et al.*, 2017) y algoritmos de segmentación basados en modelos deformables (Muschelli *et al.*, 2015). En este trabajo se adoptará un enfoque basado en aprendizaje automático mediante redes neuronales convolucionales profundas (LeCun *et al.*, 2015), a partir de una serie de imágenes[1] de las cuales ya se conoce su segmentación. Las redes neuronales convolucionales profundas han demostrado ser sumamente precisas en diversas tareas de segmentación de imágenes médicas (Litjens *et al.*, 2017). Sin embargo, aún no han sido utilizadas en el contexto propuesto. Durante el desarrollo de este trabajo, se estudiarán diferentes modelos existentes, y se evaluará la incorporación de restricciones anatómicas (Oktay *et al.*, 2017) para mejorar la robustez de la herramienta desarrollada.

Como se mencionó anteriormente, el traumatismo de cráneo afecta a un número elevado de personas y acarrea un riesgo en la vida de quien lo padece, por lo que es necesario desarrollar técnicas que permitan detectar en una fase temprana cualquier tipo de lesión que se pueda manifestar luego de ocurrido el accidente y tomar medidas en el caso que sea necesario.

[1]El conjunto de imágenes con su respectiva segmentación cerebral es provisto por investigadores del Departamento de Anestesia de la Universidad de Cambridge (Reino Unido) en el marco de una colaboración existente entre el director del proyecto final, Dr. Enzo Ferrante (Sinc) y los Dres. Virginia Newcombe y David Menon (Universidad de Cambridge).

En particular, se espera que esta herramienta funcione como etapa de pre-procesamiento dentro de un *framework* de procesamiento de imágenes médicas más amplio, donde será utilizada para aislar la masa cerebral y proceder, posteriormente, a efectuar un proceso de detección de lesiones, útil tanto en la etapa de diagnóstico como en la de seguimiento de la evolución de la patología.

1.2. Objetivos

Objetivo general

Diseñar y desarrollar una herramienta robusta y abierta para la segmentación de la cavidad craneal en imágenes de tomografía axial computarizada de cabeza, utilizando redes neuronales convolucionales con restricciones anatómicas.

Objetivos específicos

1. Pre-procesar (registrar y normalizar) el conjunto de imágenes TAC de pacientes con traumatismo de cráneo provisto por la Universidad de Cambridge.

2. Diseñar e implementar una arquitectura básica de redes neuronales convolucionales profundas para la segmentación de cavidad craneal.

3. Diseñar e implementar una arquitectura de redes neuronales convolucionales profundas con restricciones anatómicas para la segmentación de cavidad craneal.

4. Comparar las implementaciones propuestas con algoritmos del estado del arte.

5. Documentar la herramienta desarrollada.

1.3. Alcance

Exclusiones

En este trabajo se pretende implementar un prototipo de herramienta que pueda ser utilizado en pipelines de procesamiento de imágenes TAC de pacientes con traumatismo de cráneo. Queda fuera del alcance de este proyecto la implementación de etapas posteriores tales como cuantificación de lesiones o predicción de la evolución del paciente, así como también el despliegue de la herramienta en el ámbito clínico real.

Alcances Funcionales

- Realizar el pre-procesamiento de los datos de entrada
- Segmentar la región de la imagen correspondiente al cerebro en imágenes TAC de cabeza.
- Permitir la segmentación de imágenes TAC de cabeza en pacientes que han sido sometidos a craniectomía.

Alcances No funcionales

- El lenguaje utilizado será Python, y se utilizará la biblioteca de aprendizaje profundo PyTorch para la implementación de las redes neuronales convoluciones.
- El sistema deberá ser *robusto* con respecto a imágenes provenientes de diferentes tomógrafos.
- El sistema deberá ser *modular* y *extensible*, en el sentido de que permita incorporar fácilmente nuevos algoritmos de segmentación y arquitecturas de redes neuronales.
- El sistema deberá proveer una interfaz por línea de comandos para facilitar su uso en Scripts.
- El diseño será orientado a objetos para facilitar su uso como parte de otros frameworks.

- La herramienta será desarrollada para el sistema operativo Linux.

Supuestos

- Se contará con placas de procesamiento gráfico suficientemente poderosas como para procesar las imágenes TAC volumétricas.
- Se dispondrá del conjunto de imágenes con sus respectivas segmentaciones al momento de comenzar el desarrollo del proyecto.
- Se contará con la colaboración permanente del Director del Proyecto, quien posee experiencia en el área de análisis de imágenes biomédicas.

Capítulo 2

Fundamentos teóricos

Este capítulo incluye una introducción a las tecnologías de adquisición de imágenes médicas más utilizadas, y presenta conceptos básicos de visión artificial, procesamiento de imágenes y aprendizaje profundo con el objetivo de poner en contexto al lector.

2.1. Imágenes médicas

Las imágenes médicas permiten visualizar el interior del cuerpo humano sin la necesidad de realizar una intervención quirúrgica. Éstas pueden ser adquiridas con diferentes propósitos tales como el diagnóstico, navegación intraoperatoria y monitoreo posoperatiorio (Preim y Bartz, 2007).

Dependiendo de la parte del cuerpo a estudiar y los medios de adquisición con los que se cuente, se podrán tener distintos tipos de imágenes con diferentes niveles de precisión, costos, y que reflejen o permitan contrastar de una manera complementaria las imágenes producidas por otros métodos, y así modificar un diagnóstico determinado de ser necesario.

A continuación se detallarán algunas de las tecnologías de uso más frecuente en la actualidad.

Rayos X Esta técnica sentó las bases del diagnóstico por imágenes. Descubierta en 1895, utiliza un tipo de radiación electromagnética que tiene capacidad de interacción con la materia. Su principio de funcionamiento se basa en disparar los rayos X hacia una placa, los cuales se atenúan a medida que pasan a través del cuerpo de la persona, produciendo de esta forma la imagen (un ejemplo se muestra en la Figura 2.1).

Entre las principales aplicaciones se encuentran el estudio de los sistemas sistemas esquelético, respiratorio, cardiovascular, urinario y gastrointestinal (Díaz, 2014).

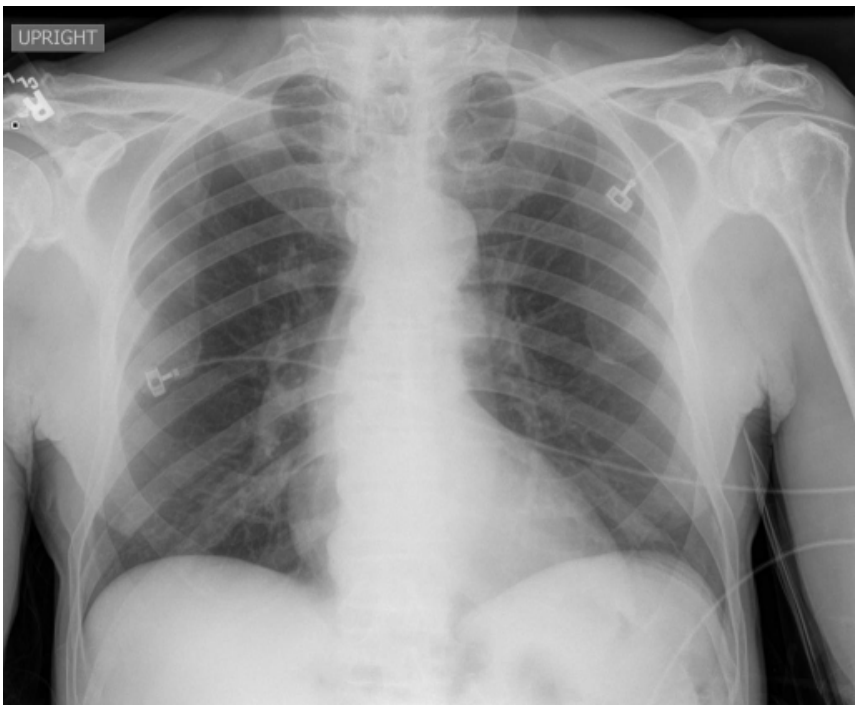

Figura 2.1: Radiografía de Tórax (Kohli y Rosenman, 2013).

Ultrasonido En esta técnica se aprovechan las características físicas del sonido para producir las imágenes. En base al concepto de sonido[1], el Ultrasonido puede definirse como una serie de ondas mecánicas organizadas por la vibración de un cuerpo elástico (un cristal piezoeléctrico) y propagadas por un medio material (tejidos corporales), cuya frecuencia supera la del sonido audible por el humano.

Su principio de funcionamiento es el siguiente: un transmisor produce un pulso corto (de millonésimas de segundo de duración) de oscilaciones eléctricas de alta frecuencia (de 1 a

[1]Sonido: ondas mecánicas producidas por la vibración de un cuerpo elástico y propagado a partir de un medio material a través de compresiones y dilataciones de éste

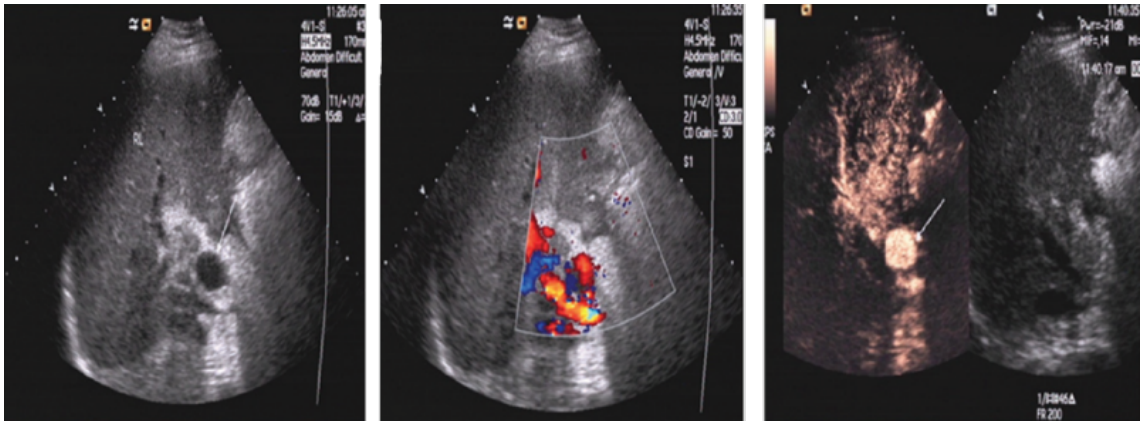

Figura 2.2: Ultrasonido utilizado para el diagnóstico de aneurismas en la arteria hepática (Ren *et al.*, 2016).

10 MHz), y un transductor, ubicado sobre el cuerpo humano, convierte esta señal eléctrica en un pulso de vibraciones mecánicas con la misma frecuencia y duración, las cuales atraviesan de diferentes maneras a los órganos del cuerpo. Los ecos de los órganos, vasos sanguíneos y demás estructuras son amplificados y procesados por un receptor y enviados a una computadora, quien lleva un registro de sus tiempos de retorno y amplitudes y con esta información construye la imagen (Wolbarst y Cook, 1999).

Esta técnica es útil para el estudio de tejidos blandos y órganos que son radiológicamente similares (esto es, aquellas imágenes en donde los coeficientes de atenuación de Rayos X sean prácticamente iguales), permitiendo monitorear el flujo sanguíneo en venas y arterias (un ejemplo se puede observar en la Figura 2.2), fines obstétricos o ginecológicos, cardiovasculares entre otros (Rawat *et al.*, 2018).

Como ventajas se pueden mencionar que provee una buena visualización de tejidos blandos, diferenciación entre sólidos y líquidos y su uso no implica ningún tipo de riesgo (en comparación con otras técnicas que utilizan radiación ionizante).

Tomografía Computarizada En las imágenes de rayos X descriptas anteriormente, la información tridimensional del cuerpo se proyecta en un plano bidimensional, por lo que pueden existir estructuras de interés que se solapen, y los tejidos de diferentes densidades se pueden confundir u ocultar debido a las variaciones en intensidad de la radiación en in-

Aire	Grasa	Agua	CSF[I]	Sangre	Músculo	Materia Blanca	Materia Gris	Hueso
-1000	-100	0	15	30-45	40	20-30	37-45	> 150

[I] CSF: Líquido cefalorraquídeo o líquido cerebroespinal.

Tabla 2.1: Escala de Hounsfield.

teracción con determinadas estructuras, por lo que la tomografía computarizada (también llamada Tomografía Axial Computarizada o TAC) viene a proporcionar una estrategia diferente para extraer información tridimensional del cuerpo superando estas limitaciones.

Mediante esta técnica se obtienen cortes o secciones de un objeto (como el que se puede observar en la Figura 2.3) a través de una exploración de rayos X. El funcionamiento básico consiste en utilizar una fuente de rayos X, la cual rota y se desplaza por una estructura circular llamada *gantry*. Durante el examen, el paciente permanece acostado en una camilla que se desplaza lentamente por el *gantry*, y a la vez el tubo de rayos X rota alrededor del paciente. Los rayos impactan sobre un detector de rayos X luego de atravesar el cuerpo, y son transmitidos a una computadora, la cual construye la imagen del corte axial correspondiente. Si bien la exposición prolongada a rayos X es contraproducente (Herzog y Rieger, 2004), el uso de TAC previene la necesidad de realizarse cirugías exploratorias, por lo que se utiliza ampliamente en el sistema de salud (salvo en determinados pacientes).

Durante la evolución de esta técnica en los últimos 40 años, tanto la calidad de la imagen producida como la velocidad en el proceso de adquisición, la reconstrucción de la misma y la eficiencia en el aprovechamiento de los rayos X ha sido mejorada (Goldman, 2007).

Los valores de intensidad en las imágenes generadas por la computadora se corresponden con los coeficientes de atenuación de los rayos X percibidos, en una escala normalizada llamada *Escala de Hounsfield* en la cual se puede reconocer cada tipo de tejido en un rango de valores determinado. En la Tabla 2.1 se muestran las unidades para diferentes tejidos.

Los principales estudios que se realizan con esta técnica son de cabeza, área abdominal, sistema genitourinario, miembros superiores e inferiores y sistema musculoesquelético.

Frente a las imágenes de resonancia magnética, las imágenes de TAC proporcionan un

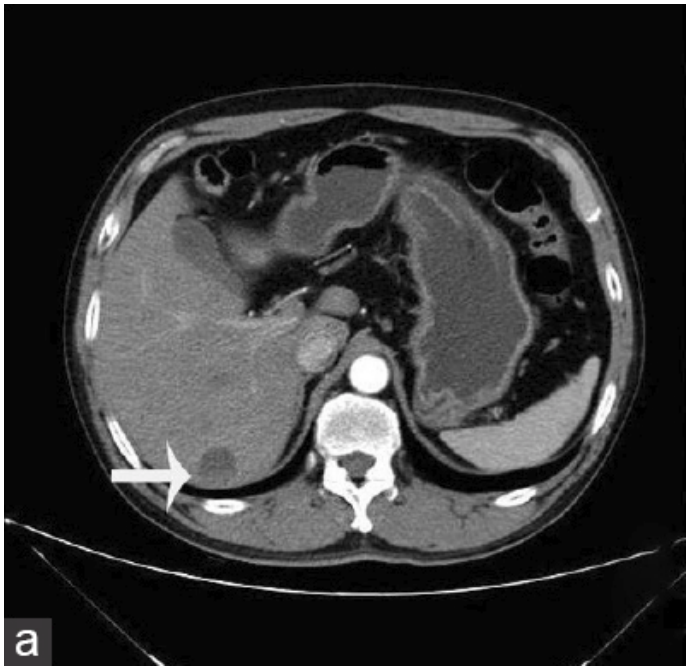

Figura 2.3: Corte de tomografía computarizada del hígado (Sun *et al.*, 2018).

menor nivel de detalle, pero por otro lado la obtención resulta más rápida y económica.

Resonancia magnética Frente a las limitaciones de la tomografía computarizada como la incapacidad para observar con detalle los tejidos internos y la emisión de radiación ionizante, una alternativa es la resonancia magnética, que mediante la utilización de un campo magnético y ondas de radio permite producir imágenes de órganos, tejidos blandos, huesos y otras estructuras del cuerpo de una manera más clara y precisa, como se muestra en la Figura 2.4.

El principio de funcionamiento de esta técnica se basa en el uso de campos magnéticos. Las moléculas de agua del cuerpo humano se alinean en un campo magnético al interior del resonador. Los protones absorben la energía del campo magnético y cambian su espín, de manera que al apagar el campo magnético, los protones vuelvan a su espín normal produciendo una señal de radio medible por los receptores en el escáner. Debido a que los protones de los distintos órganos vuelven a su estado normal en tiempos distintos, el escáner puede distinguir entre los diferentes tipos de tejidos, construyendo de esta forma la imagen (Caverly, 2015).

La calidad de las imágenes obtenidas se puede mejorar utilizando materiales de contraste,

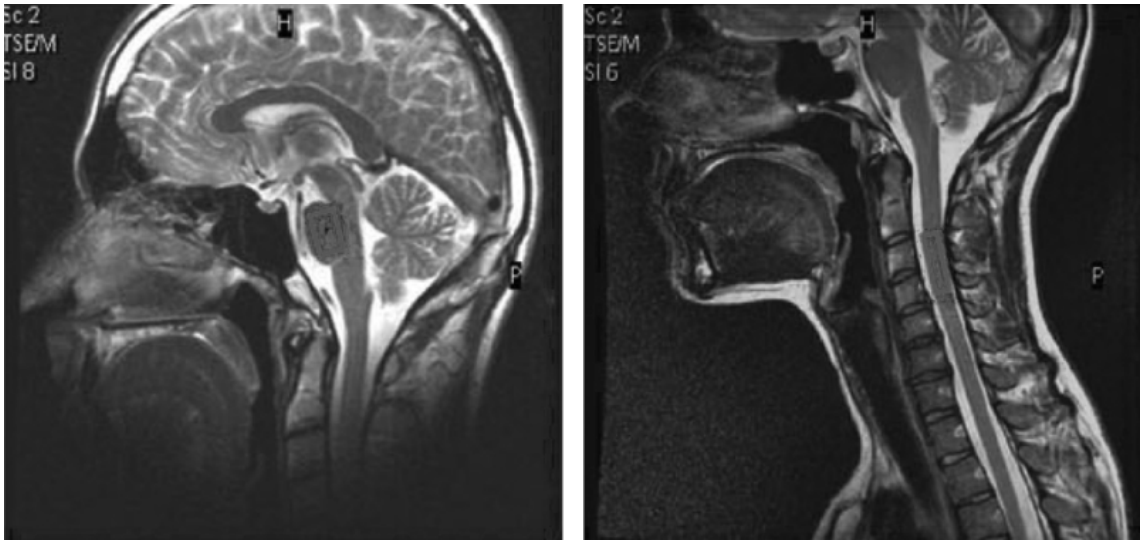

Figura 2.4: Resonancia Magnética de diferentes partes de la médula espinal (Wyss *et al.*, 2017).

los cuales son inyectados al paciente previo al estudio. Esto permite distinguir anormalidades, haciendo que se resalten más fácilmente.

Debido a su funcionamiento, un estudio de resonancia magnética puede durar entre 30 minutos y una hora, siendo además una técnica más costosa que las TAC.

En cuanto a las restricciones, esta técnica está contraindicada en pacientes con marcapasos, personas con implantes de metal, claustrofobia, implantes cocleares, entre otros (Narvaez *et al.*, 2009).

2.2. Sistemas de visión artificial

Las imágenes médicas descriptas en la sección anterior, al igual que las imágenes naturales capturadas con una cámara fotográfica, son almacenadas de forma digital en matrices de píxeles que pueden variar dependiendo el tipo de imagen. En el caso de las imágenes bidimensionales estas matrices poseen un tamaño de $h \times w \times c$, siendo h la altura, w el ancho y c la cantidad de canales de color de la misma. Para las imágenes tridimensionales se incluye la profundidad d, resultando de tamaño $h \times w \times d \times c$. La unidad de representación mínima de información en imágenes tridimensionales es el vóxel, el cual representa el

volumen mínimo capaz de ser representado en la imagen. Para un canal determinado y una posición de ese espacio tridimensional, un vóxel tomará un determinado nivel de intensidad discreto.

La visión artificial es una disciplina que se ocupa de imitar el comportamiento del sistema visual humano a través de una serie de procesos que implica la adquisición, el procesamiento y el análisis de una imagen digital o un conjunto de ellas.

Imitar determinadas características de la visión humana puede resultar una tarea compleja para una computadora. Szeliski (2010) atribuye esta dificultad a que la visión es un problema inverso, esto es, a partir de información insuficiente se busca recuperar algunas incógnitas para determinar una solución. Por esta razón, la investigación en visión por computadora es uno de los sub-campos de las ciencias de la computación más activos en la actualidad.

2.2.1. Procesamiento de imágenes

Dentro de un sistema de visión artificial existe en general una capa de procesamiento de imágenes, la cual incluye métodos que realizan operaciones sobre la representación digital de la imagen, con el objetivo de mejorarla (en base a la subjetividad del usuario), o permitir que se pueda extraer información de ella más fácilmente. Gonzalez y Woods (2006) delimitan la separación entre el procesamiento de imágenes y otras disciplinas en que en ésta tanto la entrada como la salida de cualquier proceso son imágenes, es decir, ocurre un proceso que se encarga de efectuar una transformación en los píxeles de la misma.

Al ser un concepto tan general, las aplicaciones son muchas, desde el procesamiento por píxel, pasando por transformaciones de intensidad, filtrado en los dominios espacial y frecuencial, restauración y reconstrucción de la imagen, compresión, operaciones morfológicas, entre otros.

Estas operaciones sobre las imágenes pueden estar contempladas en una etapa de *preprocesamiento*, mediante la cual las imágenes son llevadas a una estructura determinada

en donde se comparte, por ejemplo, un rango de tamaños con los que se puede trabajar, valores de intensidad permitidos, o se descarta información que no es de utilidad para el problema, siempre y cuando sea posible.

2.2.2. Segmentación de imágenes

La segmentación de imágenes es uno de los problemas más abordados en el área de visión artificial, el cual se ocupa de subdividir una imagen de entrada en objetos o regiones que la componen en base a determinadas características de la misma. Éste es un paso útil en muchas aplicaciones en las que se quiere trabajar sólo con determinadas regiones de una imagen o mostrar que algunas partes de la imagen comparten un mismo conjunto de características. Por ejemplo, en este proyecto final de carrera nos centraremos en la segmentación del cerebro en imágenes TAC de pacientes con craniectomía.

Como salida de este proceso, se puede modificar la imagen original para delimitar las regiones, o bien se puede generar una nueva imagen de iguales dimensiones que entrada, pero con una cantidad de colores equivalente a la cantidad de clases o etiquetas posibles. En la Figura 2.5 se muestra un ejemplo de segmentación y la superposición entre las imágenes. Existe una gran variedad de métodos para realizar esta tarea. A continuación se mencionarán algunos de ellos.

Umbralizado

Es el método más simple, ilustrado en la Figura 2.6, consiste en convertir una imagen en escala de grises en una imagen binaria (o de dos clases) al establecer un valor de umbral, por lo que las intensidades menores a éste se correspondan con el valor mínimo y las intensidades mayores o iguales al umbral tengan el valor máximo. Para una imagen bidimensional f, al fijar el umbral en t, el valor de intensidad g en el punto (x, y) resulta:

$$g(x,y) = \begin{cases} 0 & \text{si } f < t \\ 1 & \text{si } f \geq t \end{cases} \tag{2.1}$$

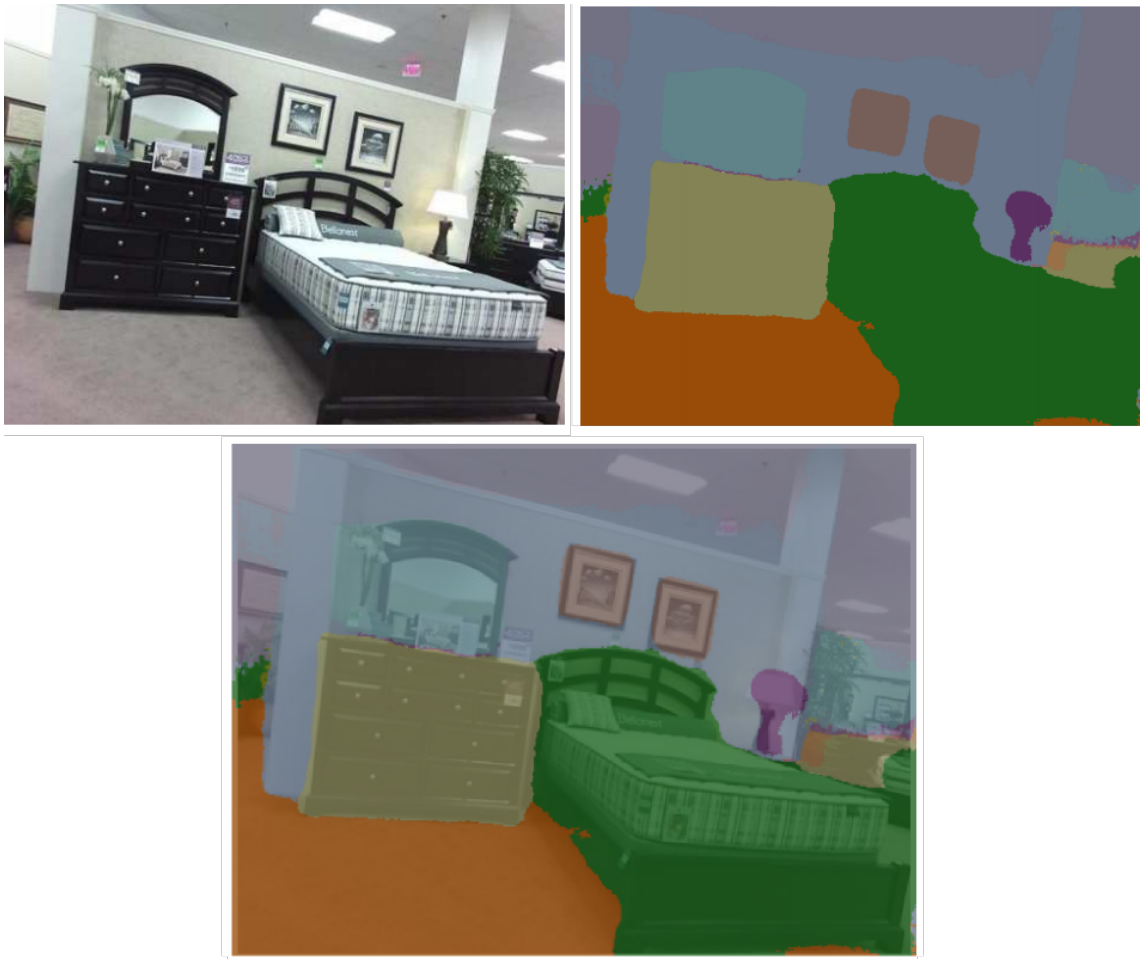

Figura 2.5: Ejemplo de segmentación de imágenes. Izquierda: imagen original, Derecha: segmentación de los objetos, Abajo: superposición. (Badrinarayanan *et al.*, 2017)

Figura 2.6: Umbralizado en una imagen de una moneda en el valor de intensidad 128.

Este valor de umbral puede ser arbitrario o estar relacionado con alguna característica de la imagen, un ejemplo de esto es el método de Otsu (1979), el cual se utiliza en imágenes cuyas intensidades siguen una distribución bimodal y aplica como umbral la intensidad que minimiza la varianza entre modas.

Crecimiento de regiones

En este método, los píxeles se agrupan en regiones en base a propiedades que estos tengan en común, como por ejemplo, un criterio de conectividad o adyacencia. Para comenzar se comienza con una inicialización de la segmentación llamada semilla, y a continuación esta inicialización se expande, agregando los píxeles vecinos que cumplan dicho criterio. En el momento en que no existen más píxeles vecinos que cumplan la condición, el proceso termina. Una imagen segmentada mediante este método se puede ver en la Figura 2.7. Notar que cada semilla puede pertenecer a una clase diferente, superando así el límite a dos clases que ofrece el umbralizado.

Al utilizar la vecindad, este método resulta de interés para separar objetos que comparten determinadas características pero son similares en valores de intensidad (Gonzalez y Woods, 2006).

Figura 2.7: Ejemplo se segmentación basada en crecimiento de regiones, insertando una semilla en cada moneda.

Transformación divisoria (watershed)

En esta técnica (ilustrada en la Figura 2.8) la imagen es vista como un relieve topográfico en el cual las intensidades son interpretadas como las alturas del mismo. Además, se considera que si en los valles se arroja agua, ésta caerá hacia las zonas de baja intensidad, inundando las distintas cuencas que puedan formarse. Este proceso continúa hasta que dos cuencas distintas se unen, formando en esa unión la frontera de la segmentación, que podría separar a dos o más clases distintas (Beucher, 1994).

Esta técnica se suele combinar con la aplicación de operaciones morfológicas con el objetivo de superar las limitaciones que se presentan en imágenes con ruido, en donde aparecen demasiados mínimos locales.

Conjuntos de nivel (level sets)

El método de conjuntos de nivel utiliza el concepto de curvas de nivel, aplicado a la región que se desea segmentar: si dicha región es una curva cerrada en un espacio bidimensional, esta puede representarse como una curva implícita, en donde los puntos interiores de la misma tengan valores positivos, los puntos exteriores tengan valores negativos y el contorno, o *curva de nivel cero* valga cero.

Por lo tanto, en este método se busca representar el contorno de un objeto mediante una curva de nivel cero inicial, la cual se ajustará a dicho contorno minimizando algún criterio de energía (Osher y Paragios, 2003). La idea del algoritmo es deformar la curva utilizando

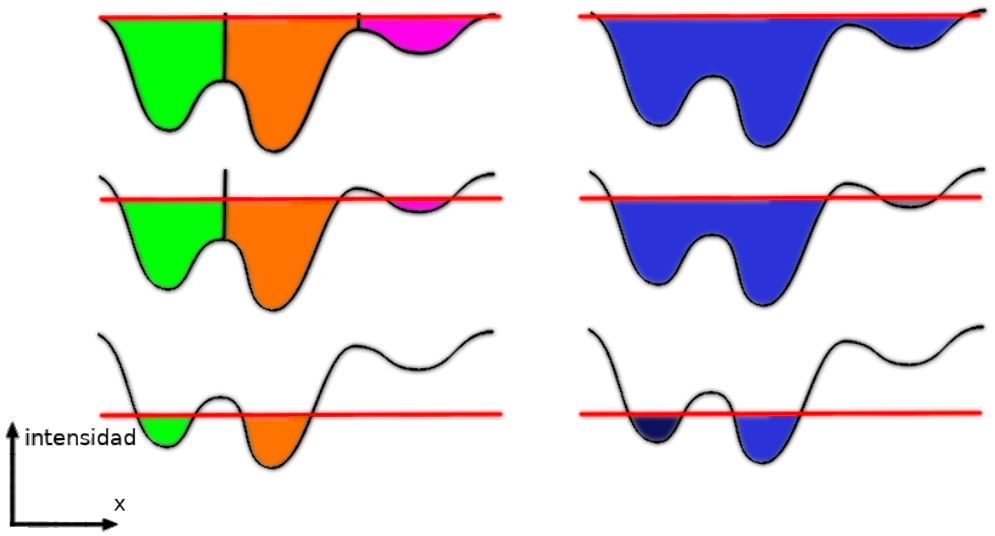

Figura 2.8: Transformación divisoria en una dimensión. A la derecha se ve como el agua inunda las cuencas. A la izquierda, las cuencas se pintan de colores distintos y se traza la frontera al unirse las aguas.

las propiedades de la imagen a segmentar, de modo tal que la curva de nivel cero se ajuste a los bordes del objeto de interés.

Una de las principales ventajas de este método es que al trabajar con curvas implícitas no existe la necesidad de parametrizar los objetos a segmentar. Además, el método incorpora cambios en la topología de los objetos de forma natural.

Un ejemplo de la aplicación de esta técnica para segmentar una imagen 2D puede observarse en la Figura 2.9.

Otros métodos

Además de los mencionados, existen otros métodos dentro de los campos de visión artificial y procesamiento digital clásicos, tales como los basados en movimiento, en el dominio frecuencial, de detección de bordes, basados en grafos, basados en histograma, entre otros, cada uno con sus ventajas y desventajas dependiendo el problema al que se los quiera aplicar.

Por otro lado, durante las últimas décadas, los avances en inteligencia artificial y aprendizaje automático han dado lugar al desarrollo de nuevas técnicas de segmentación au-

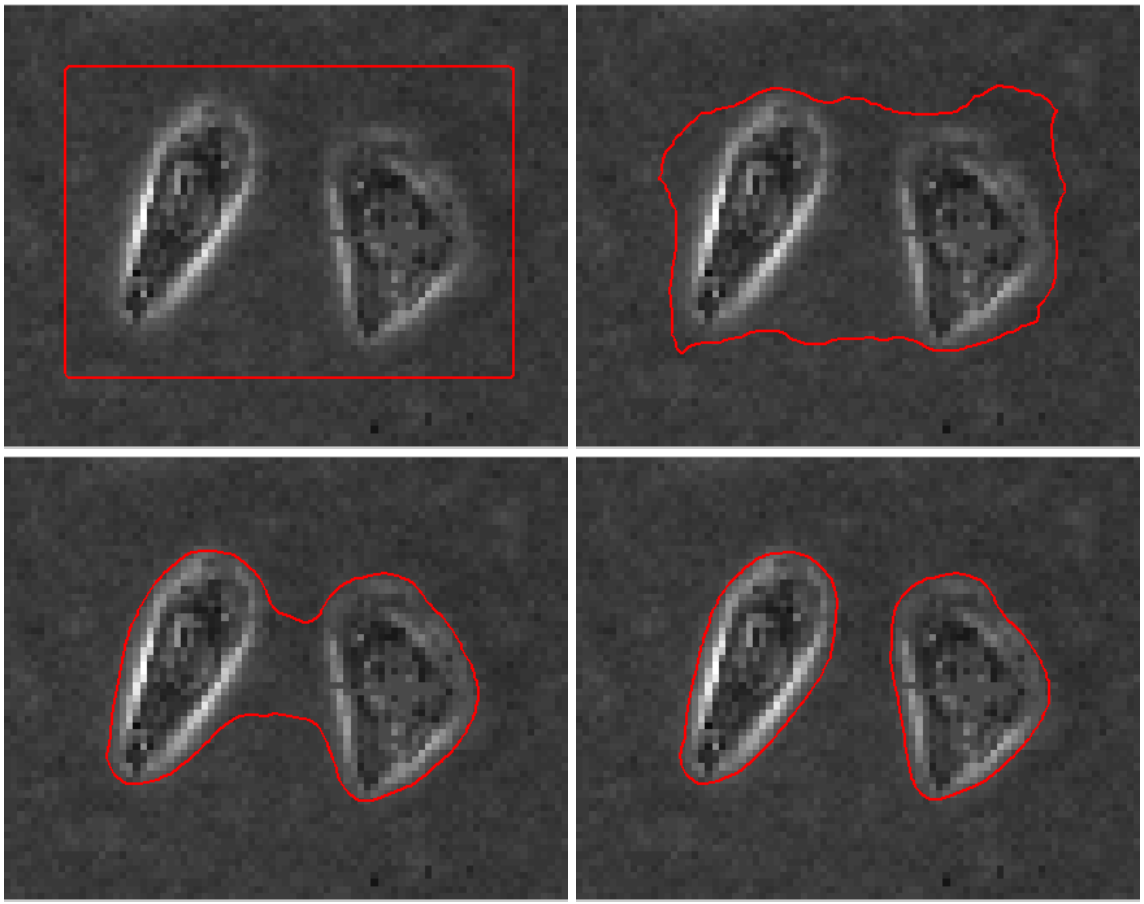

Figura 2.9: Ejemplo de diferentes instantes de la aplicación del método de Conjuntos de Nivel. En rojo, la curva inicial, avanza hasta llegar a los bordes del objeto (tomadas de `http://www.imagecomputing.org/~cmli/DRLSE/`).

tomática capaces de aprender a realizar dicha tarea a partir de un conjunto de imágenes con anotaciones, siguiendo lo que se conoce como paradigma supervisado de aprendizaje. La herramienta desarrollada en este proyecto final de carrera se enmarca en este contexto, por lo que a continuación se introducirán los conceptos básicos sobre inteligencia artificial, aprendizaje automático y aprendizaje profundo que servirán de base para avanzar luego en mayor detalle sobre el método propuesto.

2.3. Generalidades de la Inteligencia Artificial

Inteligencia artificial, aprendizaje automático y aprendizaje profundo son términos bien definidos, pero que suelen ser utilizados indistintamente en el lenguaje corriente. A continuación, se brinda una descripción de dichos términos ya que serán necesarios para introducir diversos conceptos necesarios en este proyecto.

2.3.1. Inteligencia Artificial

El término quizás más conocido y, más antiguo en este caso, es el de *Inteligencia Artificial* (IA), el cual nació en 1956 en la Conferencia de Dartmouth (McCorduck, 2004) como un término que pueda agrupar a los diversos nombres de las "máquinas pensantes" (*thinking machines* en inglés) tales como la cibernética, teoría de autómatas y procesamiento de información compleja, y otros que estén contenidos en su significado.

El objetivo de la IA es construir máquinas complejas que tengan características de la inteligencia humana (Russell y Norvig, 2002), esto es, máquinas que puedan razonar, sentir y pensar tal como lo hacen los humanos.

Dentro de este concepto, vale realizar otra distinción: por un lado existe la IA *General*, que tiene como objetivo igualar o superar la inteligencia humana (Monostori, 2003), pudiendo aplicar inteligencia a cualquier problema (concepto que por el momento sólo puede ser visto en la ficción), mientras que la IA *débil* (o narrow en inglés) busca lograr resolver un problema específico o tareas cognitivas las cuales no abarcan, o en muchos casos están

completamente por fuera de las aptitudes cognitivas humanas posibles (Sarangi y Sharma, 2018).

2.3.2. Aprendizaje Automático

El aprendizaje automático es un conjunto de métodos comprendidos dentro de la Inteligencia Artificial, cuyo objetivo radica en aprender un modelo en base a patrones existentes en los datos que le son suministrados mediante técnicas estadísticas.

Tipos de aprendizaje

En base a cómo experimenta el algoritmo con los datos que le son proporcionados, el aprendizaje que éste adquiere se puede clasificar de dos maneras: supervisado y no supervisado.

En el aprendizaje *no supervisado* se alimenta un modelo con un conjunto de entradas las cuales son organizadas en base a características comunes. Estas entradas son *no anotadas*, es decir, no se tiene información acerca de a qué clases pueden pertenecer los datos. Los principales usos de este tipo de aprendizaje son el clustering, y la reducción de dimensionalidad.

Por otro lado, en el aprendizaje *supervisado* el modelo observa una serie de ejemplos de entradas y salidas posibles y aprende una función que mapea entradas a salidas, a través de un modelo predictivo (Russell y Norvig, 2002). Este es el tipo de aprendizaje utilizado para problemas de clasificación y regresión, y fue el enfoque utilizado en este proyecto final de carrera para abordar el problema de segmentación de imágenes cerebrales.

Otros tipos de aprendizaje que se encuentran entre estas dos categorías son el aprendizaje por *refuerzo* (en donde el modelo aprende a partir de recompensas o castigos) y el aprendizaje *semi supervisado* (en donde sólo algunas entradas están etiquetadas).

2.3.3. Aprendizaje Profundo

En la sección anterior se mencionó que los métodos de aprendizaje automático buscan patrones en los datos suministrados. En el caso de las imágenes, estos patrones suelen estar asociados a características tales como combinaciones específicas de bordes, colores o texturas. Los métodos clásicos de visión artificial basados en aprendizaje automático supervisado suelen realizar la extracción de dichas características utilizando operadores específicos (operador de Canny, transformada de Hough, histograma de gradientes orientados, Bancos de Gabor, entre otros), y posteriormente emplear algoritmos de clasificación para determinar la pertenencia de una imagen a una u otra clase. Sin embargo, la selección y el diseño de estos operadores suele ser una tarea compleja, y usualmente aquellos que resultan más útiles para resolver un problema no generalizan a contextos diferentes. La idea detrás del aprendizaje profundo en el contexto de la visión artificial es permitirle al algoritmo descubrir por su cuenta qué patrones o estructuras de alto nivel resultan más discriminativos para cada problema. Por ejemplo, para un problema de reconocimiento de vehículos en imágenes los patrones más distintivos pueden ser ciertas combinaciones de líneas, colores y texturas que se asocian unívocamente a vehículos.

Los métodos de aprendizaje profundo se basan en aprender representaciones de los datos en distintos niveles de abstracción, y en general condensan los procesos de extracción de características y clasificación en un único modelo (ver Figura 2.10). Las redes neuronales artificiales (RNA) que se presentarán en la siguiente sección permiten construir dichos modelos de forma simple y modular, y serán empleados en este trabajo.

2.4. Redes Neuronales Artificiales

Una Red Neuronal Artificial (RNA) es un modelo computacional inspirado en el funcionamiento del cerebro humano, compuesto por una función ϕ, la cual efectúa un mapeo (en general no lineal) de un espacio de entradas $\mathbb{R}^N$ a otro de salidas $\mathbb{R}^K$, en donde N y K

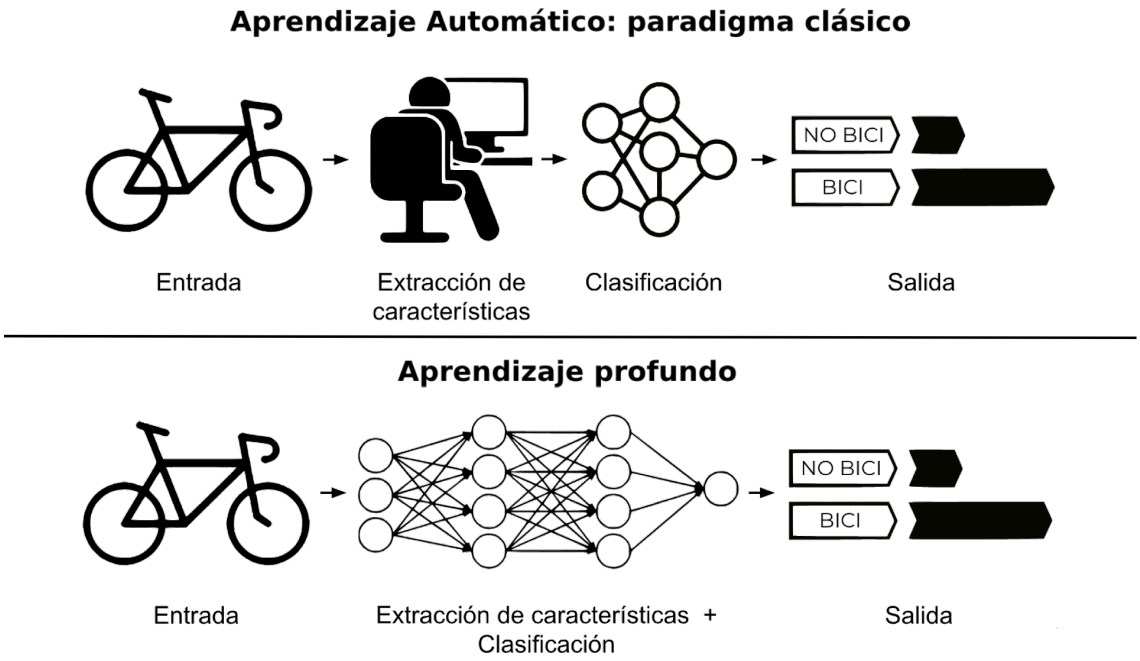

Figura 2.10: Comparación entre el paradigma clásico del aprendizaje automático y el aprendizaje profundo

son las dimensiones de los espacios en la entrada y la salida, respectivamente:

$$\phi : \mathbb{R}^I \rightarrow \mathbb{R}^K. \tag{2.2}$$

Esta red está formada por un conjunto de neuronas (formuladas como un modelo simplificado de la neurona biológica) representadas por funciones matemáticas simples (Engelbrecht, 2007). A través de la combinación de una cantidad finita de neuronas, el *Teorema de Aproximación Universal* (Cybenko, 1989) establece que es posible lograr la aproximación de funciones continuas en espacios compactos de $\mathbb{R}^N$ con una precisión ϵ determinada.

2.4.1. Modelos de perceptrón

Perceptrón Simple

En la corteza cerebral humana, cada neurona puede activarse dependiendo de los diferentes estímulos que recibe el cuerpo humano, y la activación conjunta de diferentes neuronas

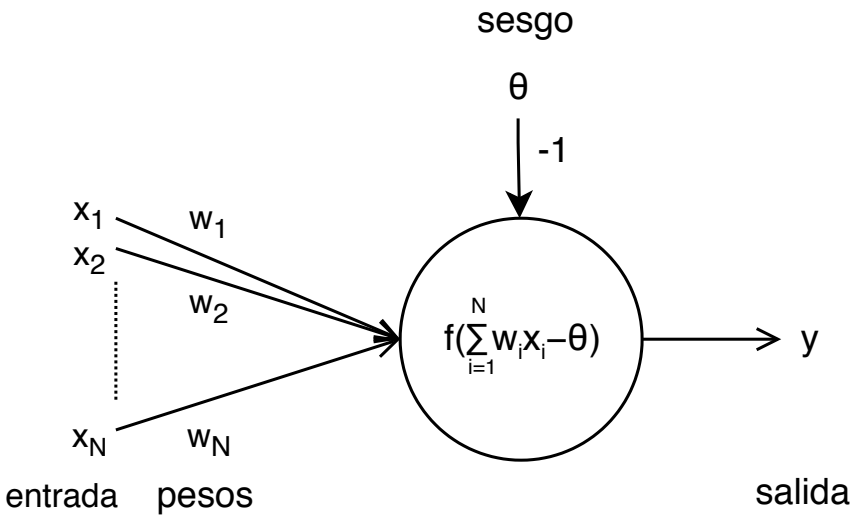

Figura 2.11: Modelo de neurona artificial

puede ser interpretado de manera diferenciada por el cerebro, otorgándole un significado distinto a cada activación conjunta. En base a este concepto básico puede plantearse un modelo simple de neurona artificial (ilustrado en la Figura 2.11), en el cual las señales de entrada representen los estímulos del exterior, los pesos den una medida de la "importancia" que se le da a cada uno de estos estímulos, y en base a una operación entre estas entradas y estímulos, se obtenga un valor el cual puede activar o no la neurona, en base a una función de activación f o un umbral.

Este modelo simple, también denominado *perceptrón simple*, puede escribirse matemáticamente como:

$$y = f(\sum_{i=1}^{N} w_i x_i - \theta), \tag{2.3}$$

donde y es la salida, w_i los pesos del perceptrón, x_i la entrada, n la cantidad de neuronas y θ el sesgo.

Si bien este modelo permite implementar una función que mapea una entrada real multidimensional a una salida binaria, las funciones que pueden ser aprendidas sólo resultan adecuadas para resolver problemas cuyos patrones sean linealmente separables (Raudys, 1998). Un ejemplo de esto puede verse en la Figura 2.12. Allí, puede observarse el caso

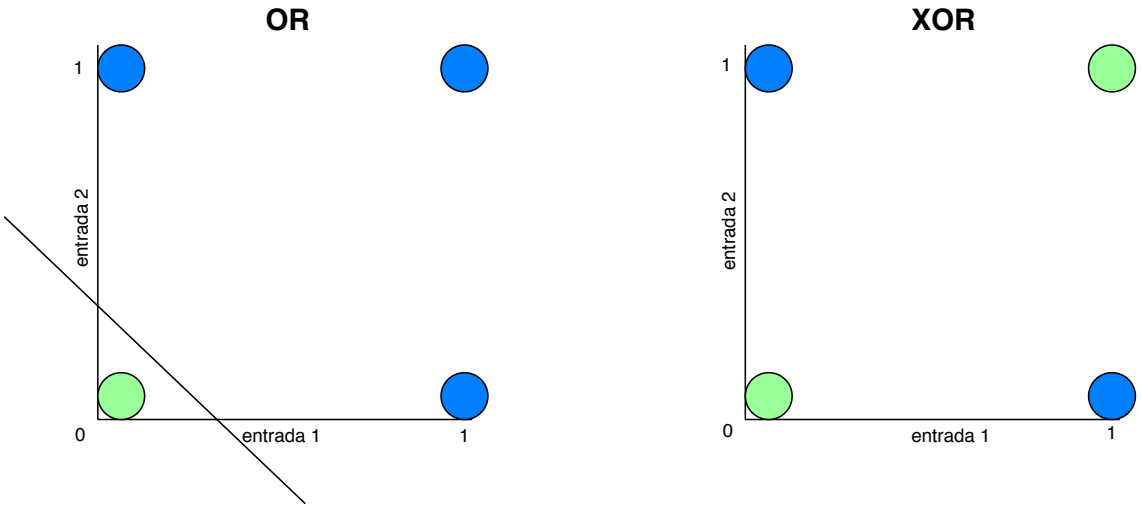

Figura 2.12: Separabilidad lineal de las operaciones lógicas OR y XOR (azul indica verdadero y verde indica falso). Mientras que la operación OR es linealmente separable con una única recta, la operación XOR no puede ser representada con un modelo tan simple.

de las operaciones lógicas OR y XOR. Para el primer caso, un perceptrón simple con dos entradas (una para cada variable que participa en la operación) puede hallar más de una recta que separe entre las dos clases posibles con un error nulo, mientras que para el problema del XOR no. Esto motiva el desarrollo de arquitecturas de redes neuronales más complejas, como el caso del perceptrón multicapa que se presenta continuación.

Perceptrón multicapa

Para aumentar el poder de expresión del perceptrón simple, se pueden combinar varias de estas estructuras en una serie de al menos tres capas: una de entrada, una capa oculta, y otra de salida, formando así un *perceptrón multicapa* pudiendo aproximar cualquier tipo de función.

Cada salida del perceptrón (ya sea de una capa oculta como de la capa de salida) se corresponde con una ecuación, que relaciona las entradas (que pueden ser la entrada de la red o la salida de una capa oculta), los pesos y el sesgo (correspondientes a una capa

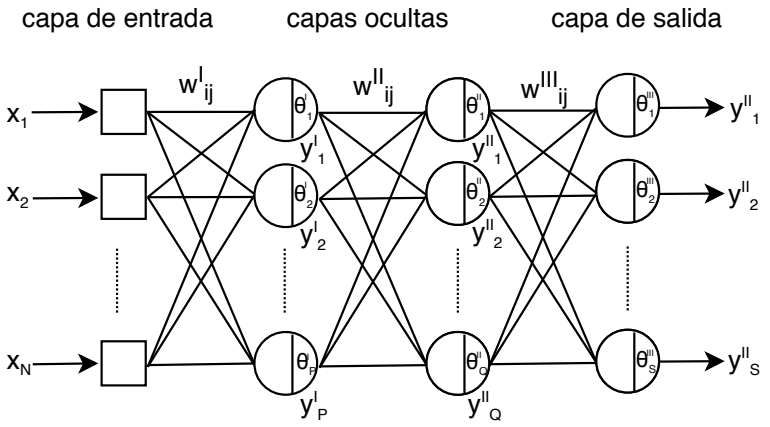

Figura 2.13: Perceptrón multicapa

determinada). Por ejemplo, para la primer capa oculta (I):

$$y_j^I = f(\sum_{i=1}^{N} w_{ij}^I x_i - \theta_j^I) \tag{2.4}$$

siendo y_j la salida de la neurona j en la capa I, w_{ij}^I los pesos del perceptrón, x_i la entrada, N la cantidad de neuronas en la capa I y θ^I el sesgo en la capa I. Para el resto de las capas la ecuación es similar, salvo que la entrada en una capa sucesiva será ahora la salida de la capa anterior a ésta.

Un detalle que vale la pena destacar, es que estos modelos también son denominados *totalmente conectados* (fully connected en inglés), lo cual implica que cada neurona posee una conexión (a través de un peso) con todos los elementos de la capa anterior. Este es un detalle a considerar cuando se quiere trabajar con datos con grandes dimensiones, ya que influye en la cantidad de parámetros final de la red.

2.4.2. Conceptos asociados al entrenamiento del modelo

Al entrenar una red neuronal se busca que la misma posea buena *capacidad de generalización*, esto es, que al suministrarle nuevos datos, la misma tenga un rendimiento lo más similar posible al que tiene con los datos con que fue entrenada. En esta sección se mencionarán algunos conceptos básicos asociados a dicha capacidad y al entrenamiento en general.

Entrenamiento y prueba de un modelo

El proceso mediante el cual los pesos w_{ij} son aprendidos a partir de un conjunto de datos se denomina entrenamiento. Para entrenar un modelo de RNA como el perceptrón, se le suministra a la capa de entrada uno o varios[2] datos de entrada x, los cuales son multiplicados por los pesos w que los conectan hacia las neuronas de la primera capa. Estas neuronas transforman la entrada en base a los pesos, produciendo como salida la entrada en la capa siguiente (en el caso de tener más capas) o la salida de la red (en el caso de ser la última capa). Este proceso es conocido como pasada *hacia adelante* de los datos (forward pass), dado que los mismos fluyen por la red en una única dirección. Por esta razón, los perceptrones simple y multicapa son conocidos como redes neuronales *pre-alimentadas* (feed forward).

En un escenario de aprendizaje supervisado, a partir de la salida del modelo, se puede calcular la tasa de acierto comparando la salida de la red con el valor esperado (o *ground-truth*) por medio de una función de pérdida. Dicha función devolverá un valor pequeño cuando la salida de la red sea la deseada, y un valor más grande en caso contrario. Así, el entrenamiento de una red neuronal artificial se reduce a un problema de optimización, donde los parámetros a optimizar son los pesos w_{ij} de la red, y la función objetivo está dada por la función de pérdida. Dicho problema de optimización es resuelto utilizando el método del gradiente descendiente (ver Sección 2.4.5), y dicho gradiente es calculado por medio del algoritmo de *retropropagación*. Los pesos son actualizados iterativamente, utilizando todos los datos de entrenamiento (lo cual se denomina una *época*) tantas veces como sea necesario hasta cumplir algún criterio de convergencia pre-establecido. Una vez finalizado el proceso de entrenamiento, se utilizan los pesos finales obtenidos en la última actualización para realizar predicciones en un conjunto de datos distinto al de entrenamiento, llamado conjunto de prueba.

[2]Los datos pueden ser procesados de a uno, o en lotes (o *batches*), con el objetivo de aprovechar los recursos disponibles y aumentar el rendimiento del algoritmo.

Hiperparámetros

Los hiperparámetros son aquellos parámetros utilizados para controlar el comportamiento del algoritmo de aprendizaje y asegurar la capacidad de generalización del modelo. Estos deben fijarse antes de comenzar el entrenamiento, ya que no son aprendidos mediante dicho proceso (Alpaydin, 2014).

Algunos ejemplos de hiperarámetros pueden ser la velocidad de aprendizaje, la cantidad de neuronas y capas, la inicialización de los pesos, el numero de épocas, el tamaño del batch, entre otros.

Particiones de los datos

En general, al utilizar un algoritmo de aprendizaje supervisado, los datos (anotados) son divididos en tres particiones disjuntas para realizar el entrenamiento del modelo y una vez entrenado probar su precisión (Engelbrecht, 2007):

- **Entrenamiento:** Esta partición está compuesta por aquellas entradas que son utilizadas para actualizar los pesos del modelo.

- **Validación:** Debido a que el aprendizaje no depende únicamente de los pesos de la red, se utiliza esta partición para observar la capacidad de generalización del modelo, y si es necesario, modificar los hiperparámetros para mejorar la misma.

- **Prueba:** Ya que los pesos de la red dependen de la partición *de entrenamiento* y los hiperparámetros son ajustados utilizando la partición *de validación*, es necesario establecer una partición *de prueba*, la cual no depende de ninguna variable utilizada antes o durante el entrenamiento. Así, es posible confirmar la capacidad de generalización obtenida, o calcular métricas que sirvan para evaluar la calidad de las predicciones generadas por el modelo.

Sobreajuste y detención temprana

Como se mencionó anteriormente, al entrenar una red neuronal se busca que la misma posea buena capacidad de *generalización*. Esta capacidad puede perderse si el modelo se sobreajusta a los datos de entrenamiento (*overfitting* en inglés), lo cual implica que a medida que se avance en la cantidad de épocas, este comience a resignar su capacidad de generalización y se enfoque en aprender lo mejor posible la partición de entrenamiento. Este comportamiento se puede reconocer observando cómo se comporta el error con las dos particiones (entrenamiento y validación) en simultáneo. En el punto en el cual el error para la partición de validación deja de decrecer a una tasa similar a la que decrece el error con los datos de entrenamiento, o cuando el error de validación empieza a crecer en un punto mientras el error de entrenamiento decrece (ver Figura 2.14), es recomendable cortar el proceso de entrenamiento, ya que luego de ese momento comienza a perderse la capacidad de generalización. Este proceso de analizar las curvas de error y detenerse en este punto es conocido como detención temprana (o *early stopping* en inglés).

2.4.3. Funciones de activación

En las definiciones del perceptrón dadas en las ecuaciones 2.3 y 2.4, la función f se denomina *función de activación*, y determina la salida de la neurona, esto es, si se enciende o no y con qué valor. Además, dichas funciones permiten introducir no-linealidad en los modelos, aumentando su poder expresivo. La Tabla 2.2 incluye algunos ejemplos de funciones de activación clásicas utilizadas en el modelo de redes neuronales.

Otras funciones importantes

Función exponencial normalizada Esta función (llamada softmax en inglés), es una generalización de una función logística que comprime un vector k-dimensional $\mathbf{z}$ de valores reales en un vector k-dimensional $\sigma(\mathbf{z})$ con valores en el intervalo $(0, 1)$ y cuya suma es 1,

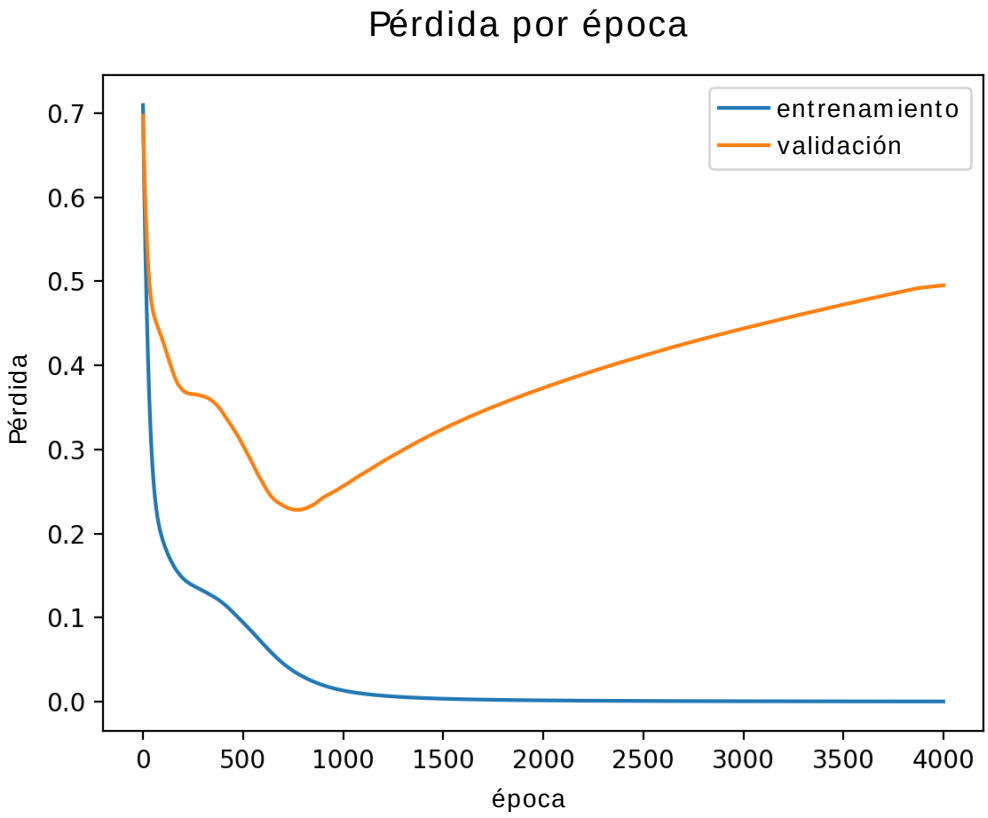

Figura 2.14: Detención temprana: en este caso, sería recomendable terminar con el entrenamiento cerca de la época 750, en donde el error de validación llega a un mínimo y luego comienza a crecer.

Nombre	Función
Sigmoidea	$f(x) = \frac{1}{e + \exp(-x)}$
ReLU	$f(x) = max(0, x)$
Tangente hiperbólica	$f(x) = \frac{2}{1 + \exp(-2x)} - 1$

Tabla 2.2: Ejemplo de funciones de activación.

es decir $\sigma : \mathbb{R}^K \to \left\{ z \in \mathbb{R}^K | z_i > 0, \sum_{i=1}^{K} z_i = 1 \right\}$. La función σ se define entonces como:

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}, \qquad j = 1, \ldots, K \tag{2.5}$$

Esta función es de utilidad en problemas de clasificación, ya que si es utilizada como función de activación en la última capa de un perceptrón multicapa, permite interpretar la salida de dicho modelo como una distribución de probabilidades (por ejemplo, indicando las probabilidades asociadas a una u otra clase en un problema de clasificación).

2.4.4. Funciones de pérdida

En base al tipo de problema que se busca resolver (por ejemplo, de clasificación binaria/multi clase o regresión) existen diferentes funciones de pérdida (también denominadas funciones objetivo, de costo, o de error) de alguna forma estiman la calidad de las predicciones y se desean minimizar o maximizar. A continuación se incluye la definición de algunas de las funciones de pérdida que se serán utilizadas en este proyecto:

- Entropía cruzada: Al utilizar esta función de pérdida es posible interpretar la salida de la red neuronal como una distribución de probabilidad $q(c)$, definida sobre un conjunto de clases $c \in \mathcal{C}$, por lo que, a partir de dos distribuciones de probabilidad p y q, de las cuales p representa el *Ground Truth*[3], q la predicción realizada y c una clase determinada, la entropía cruzada se define como:

$$CE(p, q) = -\sum_{c \in \mathcal{C}} p(c) \log q(c) \tag{2.6}$$

  Esta función suele ser utilizada en problemas de clasificación donde se espera que la salida del modelo sea categórica.

- Error cuadrático medio (Mean squared error o MSE): Cuando la salida de la red neuronal corresponde a un valor continuo (generalmente cuando se está utilizando la red para hacer regresión), una medida comúnmente utilizada es el error cuadrático medio (*error* ℓ^2), se calcula a través de la comparación elemento a elemento de la

[3]El *Ground Truth* está compuesto por aquellas etiquetas, imágenes o patrones que son obtenidos por observación directa (empíricamente), y pueden ser usados para validar las salidas de un modelo, quien infiere dicho *Ground Truth* (Kirk, 2017).

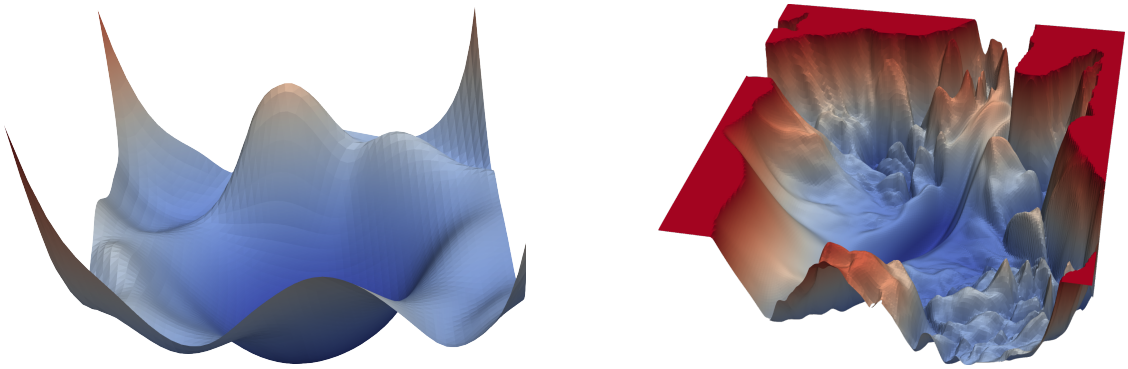

Figura 2.15: Superficies de error de ejemplo (tomadas de `https://www.cs.umd.edu/~tomg/projects/landscapes/`)

predicción $\hat{Y}$ y el ground truth Y.

$$\mathrm{MSE}(\mathrm{Y}, \hat{\mathrm{Y}}) = \frac{1}{M} \sum_{i=1}^{M} (\hat{Y}_i - Y_i)^2, \tag{2.7}$$

donde M corresponde a la cantidad de elementos en Y.

- Error ℓ^1: También utilizada en ocasiones para problemas de regresión, sea la predicción $\hat{Y}$ y el ground truth Y, se define como:

$$\mathrm{L}(\mathrm{Y}, \hat{\mathrm{Y}}) = \frac{1}{M} \sum_{i=1}^{M} |\hat{Y}_i - Y_i|, \tag{2.8}$$

siendo M la cantidad de elementos en Y.

En base a los P parámetros que conforman el modelo, la función de pérdida puede verse como una hiper-superficie en un espacio P-dimensional. En una determinada etapa del entrenamiento, el modelo estará en un punto de esa superficie, la cual puede poseer muchos valores extremos. En la Figura 2.15 se muestran dos hipotéticas superficies de error correspondientes a un modelo con dos parámetros. Como puede observarse, encontrar un valor mínimo en una superficie de este tipo no resulta trivial, más aún cuando la cantidad de parámetros es mayor a 2, alcanzando en ocasiones los miles o millones.

2.4.5. Optimización

Una vez definida la parametrización del modelo y establecida una función de pérdida para el entrenamiento, se debe encontrar una combinación de pesos que se corresponda con un mínimo en esta superficie de error P-dimensional. Una estrategia puede ser evaluar la función en puntos aleatorios, lo cual resulta ineficiente, ya que estos espacios P-dimensionales son demasiado extensos, y no es esperable que en un conjunto determinado de intentos se encuentre un mínimo aleatoriamente.

Una mejor alternativa es la que resulta al explorar la superficie de error moviéndose (de a pasos pequeños y a partir de una inicialización aleatoria) en dirección opuesta al gradiente de la función de pérdida. Así, los pesos del modelo son actualizados iterativamente utilizando una fracción del gradiente como valor de actualización. Este método se conoce como gradiente descendiente (o Gradient Descent en inglés) y posee diversas variantes:

- Gradiente descendiente clásico: Para hacer una actualización en los pesos θ de la red, calcula el gradiente $\nabla_\theta J(\theta)$ de la función de costo J respecto a dichos pesos θ, utilizando todos los datos de entrenamiento. Dicho gradiente es luego multiplicado por la tasa de aprendizaje η y utilizado para actualizar los pesos tal como indica la siguiente ecuación:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta) \tag{2.9}$$

  Debido a que para realizar una actualización en los pesos hay que calcular el gradiente de la función de pérdida evaluándola considerando la salida del modelo para todo los datos de entrenamiento $\{x^{(i)}, y^{(i)}\}$, este método puede resultar muy lento.

- Gradiente descendiente estocástico: A diferencia del anterior, utiliza sólo un dato de entrenamiento $x^{(i)}$ con su respectiva etiqueta $y^{(i)}$ a la vez, resultando en una actualización más veloz de los pesos, aunque más inestable.

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i)}; y^{(i)}) \tag{2.10}$$

- Gradiente descendiente por mini-batches: Es una combinación de los dos anteriores, en la cual se utiliza un subconjunto pequeño n de los datos de entrenamiento (desde i hasta $i+n$) para realizar la actualización de los pesos:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i,i+n)}; y^{(i,i+n)}) \tag{2.11}$$

2.5. Redes Neuronales Convolucionales

2.5.1. Introducción

En las redes neuronales artificiales presentadas anteriormente, se mencionó el problema del aumento en la cantidad de parámetros a medida que crecen las dimensiones de los datos. Este es un tema no menor, ya que en visión artificial y procesamiento de imágenes es común trabajar con resoluciones de varios cientos de píxeles de longitud, y esto representa un potencial problema en la necesidad de recursos para entrenar un modelo en particular.

Las redes neuronales convolucionales (LeCun *et al.*, 1999) resuelven este problema por medio de la operación de convolución, pero conservan las características básicas de las redes presentadas anteriormente: están compuestas por neuronas que se activan mediante una función no lineal, se organizan en capas, y poseen pesos que son aprendidos a través del cálculo de un error optimizando una función de pérdida. Estas redes están inspiradas en el funcionamiento de la corteza visual primaria en mamíferos (Hubel y Wiesel, 1959), compuesta por neuronas que se organizan en forma de *capas*, donde las primeras capas están asociadas al reconocimiento de patrones básicos (como líneas o bordes) y poseen un *campo receptivo* acotado (es decir, su comportamiento es afectado sólo por una región localizada de la señal de entrada) mientras que las capas superiores se especializan en detectar patrones más complejos y su campo receptivo es más amplio.

El funcionamiento de estas redes se basa en la operación de convolución, en la cual un filtro (compuesto por pesos) se desliza sobre la imagen, multiplicándose con las intensidades de la imagen por la que éste circula y sumándolos, produciendo como resultado un *mapa de características* (o feature map en inglés) que contendrá los valores de estas sumas por

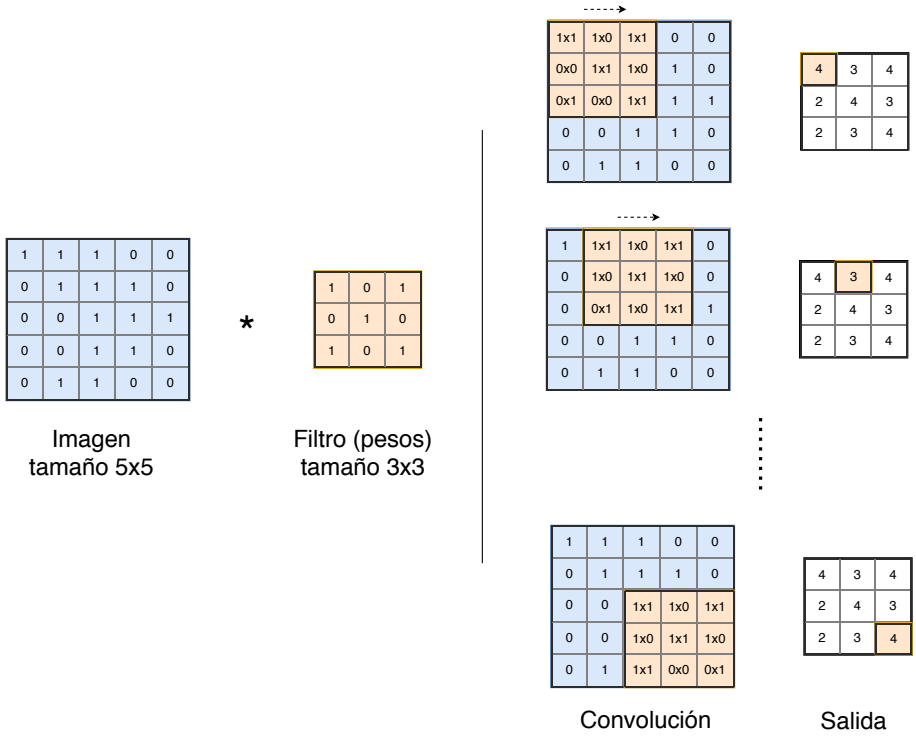

Figura 2.16: Aplicación de la operación de convolución sobre una matriz de una imagen

cada posición por la que se deslizó el filtro (ver Figura 2.16). Matemáticamente, este funcionamiento se corresponde con la operación de correlación[4]:

$$g(x,y) = \sum_{s=0}^{a}\sum_{t=0}^{b} w(x+s, y+t) f(s,t), \tag{2.12}$$

donde w es la imagen, f es un filtro de tamaño $a \times b$, (s,t) las posiciones en el filtro y (x,y) las posiciones de los puntos en la imagen de salida g.

Notar que en la primera capa de una red convolucional, la entrada estará dada por la imagen a procesar. Sin embargo, la entrada en las capas sucesivas de la red estará dada por los mapas de características generados en la capa anterior.

Los parámetros que se deben definir en una red de este tipo son:

- Cantidad y tamaño de los filtros: en cada capa convolucional se debe establecer

[4]Si bien este tipo de redes es conocida popularmente como redes neuronales convolucionales, en la práctica, la operación que implementan se corresponde con una correlación y no una convolución. En este trabajo, al igual que en gran parte de la literatura sobre aprendizaje profundo existente, utilizaremos el término "convolución" para referirnos a este tipo de operaciones.

cuántos filtros habrá y de qué tamaño.

Las dimensiones del filtro definen cuánto van a ser observadas las diferentes regiones de la imagen de entrada, determinando así el *campo receptivo* de una neurona, el cual puede definirse como la región de la imagen de entrada que afecta el valor de una determinada neurona. Notar que el campo receptivo de las neuronas ubicadas en las capas más profundas es más amplio que el de las iniciales, dada la aplicación sucesiva de la operación de convolución.

- Paso (*stride*): establece cuántos píxeles se desplaza el filtro al ser aplicado.
- Padding: Al convolucionar un filtro con una imagen, el tamaño del mismo junto con el paso definido afectarán el tamaño de la salida. Para evitar este efecto, es posible generar nuevos datos alrededor de la imagen de entrada (esto es, realizar padding sobre la entrada) con el objetivo de conservar los tamaños. Dichos datos agregados suelen corresponder a un valor constante (ceros por ejemplo) o bien estar relacionados a los datos de la entrada (por ejemplo, realizando una operación de espejado).
- Dilatación (*dilation*): Permite aumentar el campo receptivo del filtro espaciando los elementos que lo componen (esto se logra construyendo un filtro más grande, con ceros en las nuevas posiciones, tal como indica la Figura 2.17).

El tamaño de un mapa de características está dado en cada dimensión por los valores de estos parámetros en la capa convolucional. Para cada dimensión (en este caso i):

$$I_{sal} = \left\lfloor \frac{I_{ent} + 2 \cdot \mathrm{P}[i] - \mathrm{D}[i] \cdot (\mathrm{K}[i] - 1) - 1}{\mathrm{S}[i]} + 1 \right\rfloor \tag{2.13}$$

en donde I_{ent} e I_{sal} son los tamaños de entrada y salida en esa dimensión, P el padding, K el tamaño del kernel, D el tamaño de la dilatación y S el stride elegidos para esa dimensión.

Una ventaja de las redes convolucionales respecto a las totalmente conectadas es que poseen invarianza traslacional, esto es, un mismo patrón podrá ser detectado por el filtro

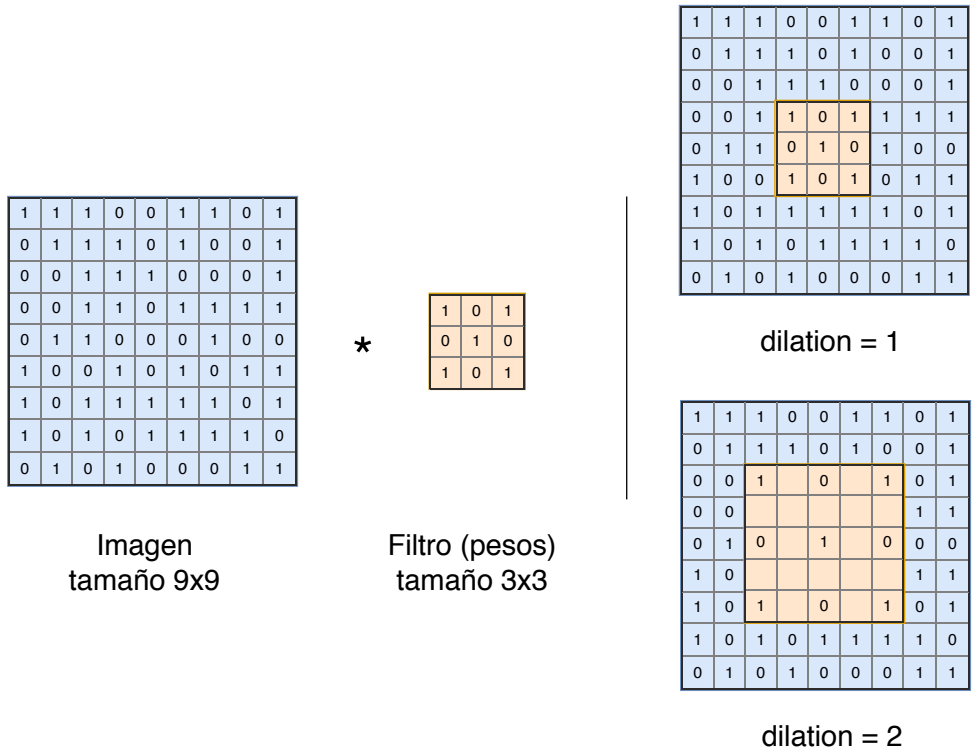

Figura 2.17: Impacto de la dilatación en el campo receptivo de un filtro

independientemente del lugar en que se encuentre. Esto resulta particularmente útil al realizar tareas como la clasificación de imágenes, donde la posición de un determinado patrón no suele ser relevante, sino que el interés radica en la existencia o no de dicho patrón (por ejemplo, al clasificar una imagen con la etiqueta "perro", el interés radica en determinar si existe o no un perro en la imagen, independientemente de su posición).

Otra característica interesante de estas redes es el esquema de compartición de pesos que implementan (o *weight sharing*, en inglés). Al reutilizar el mismo filtro para procesar toda la imagen, la cantidad de pesos a aprender (parámetros del modelo) se reduce drásticamente en comparación con un esquema totalmente conectado, resultando en redes con menor complejidad y capaces de procesar mayores volúmenes de datos.

2.5.2. Conformación de una red neuronal convolucional

Si bien es posible construir un modelo utilizando sólo capas convolucionales, por lo general, en la conformación de una red convolucional estas capas se suelen combinar con otras para aprovechar las particularidades de cada una. Estas pueden ser:

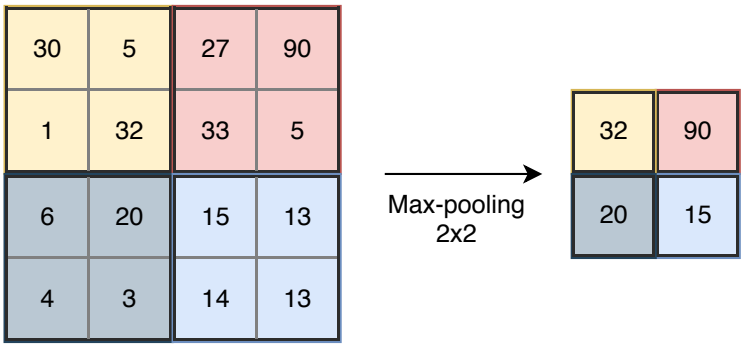

Figura 2.18: Max pooling con un filtro de tamaño 2x2 y stride=2.

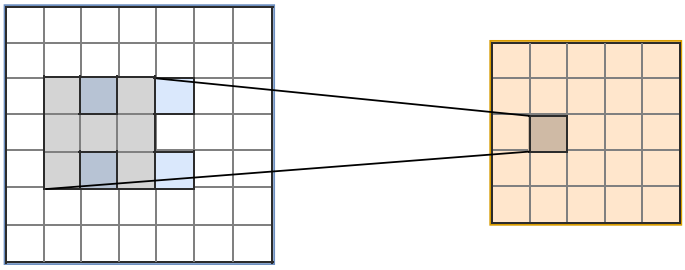

Figura 2.19: Convolución transpuesta. Ejemplo de aplicación en una imagen o mapa de características de tamaño 2x2 (en celeste). La convolución transpuesta puede ser implementada agregando ceros en la imagen de entrada y a continuación efectuando una convolución. El campo receptivo del píxel de salida sombreado en naranja se corresponde sólo con dos píxeles de la imagen de entrada original.

Capas de pooling Las capas de pooling permiten lograr la invarianza del modelo ante transformaciones de la imagen, obtener representaciones más compactas y adquirir robustez ante el ruido (Boureau *et al.*, 2010). Estas capas suelen implementarse mediante un filtro móvil sin parámetros a aprender de tamaño fijo, el cual se desliza por la entrada y aplica algún tipo de operación de agrupamiento (por ejemplo, el *máximo* -ilustrado en la Figura 2.18-, el *promedio* o la *suma*).

Convolución transpuesta También llamada *deconvolución*[5], se puede pensar como la operación opuesta a la convolución, ya que a partir de una representación bajo dimensional, aprende a recuperar mapas de características de dimensiones mayores. Es equivalente a ampliar la resolución de la entrada insertando ceros y, a continuación, realizar una convolución.

[5]Matemáticamente no se considera una deconvolución, aunque trata de imitar el comportamiento que ésta produce.

BatchNorm Existen capas que se pueden agregar para normalizar los datos, esto es, mantenerlos en un rango determinado a través de una transformación. En particular, BatchNorm (Ioffe y Szegedy, 2015) utiliza la media y la varianza de la salida de una capa considerando todos los elementos de un batch de entrenamiento, para que los valores que ésta toma sigan una distribución gaussiana (parametrizada con dos nuevos pesos que son aprendidos durante el proceso de entrenamiento). En general, el uso de BatchNorm resulta en un proceso de aprendizaje mucho más rápido.

Capítulo 3

Desarrollo

Tal como se ha discutido en las secciones anteriores, el problema que se busca resolver en este proyecto consiste en la segmentación automática del cerebro en imágenes cerebrales TAC de cabeza en pacientes con traumatismo de cráneo, poniendo particular atención en el caso de imágenes de pacientes con craniectomía descompresiva.

Existe una gran variedad de métodos que pueden ser utilizados para efectuar dicha segmentación (algunos de ellos explicados en la Sección 2.2.2), tales como los algoritmos de level-sets o crecimiento de regiones. En este trabajo, ambos métodos fueron tomados como base para contrastar posteriormente con el modelo propuesto. En ambos casos (level-sets y crecimiento de regiones) se observa que las segmentaciones que se obtienen resultan erróneas, especialmente en los pacientes con craniectomía descompresiva, donde la falta del cráneo en determinadas zonas del cerebro elimina los bordes bien definidos que permiten a ambos métodos detener el crecimiento de la región segmentada. En estos casos, las intensidades de gris en áreas externas al cerebro suelen confundirse con aquellas correspondientes a la cavidad cerebral, confundiendo a los algoritmos y resultando en una mala segmentación. Un ejemplo de esto se puede ver en la Figura 3.1, en donde se aplicó el algoritmo de crecimiento de regiones, y la segmentación final termina saliéndose del contorno que corresponde al cerebro en las partes donde el cráneo está ausente.

Por lo tanto, el objetivo será obtener un método que permita lograr la segmentación en tomografías con y sin craniectomía descompresiva, teniendo en cuenta que en este último

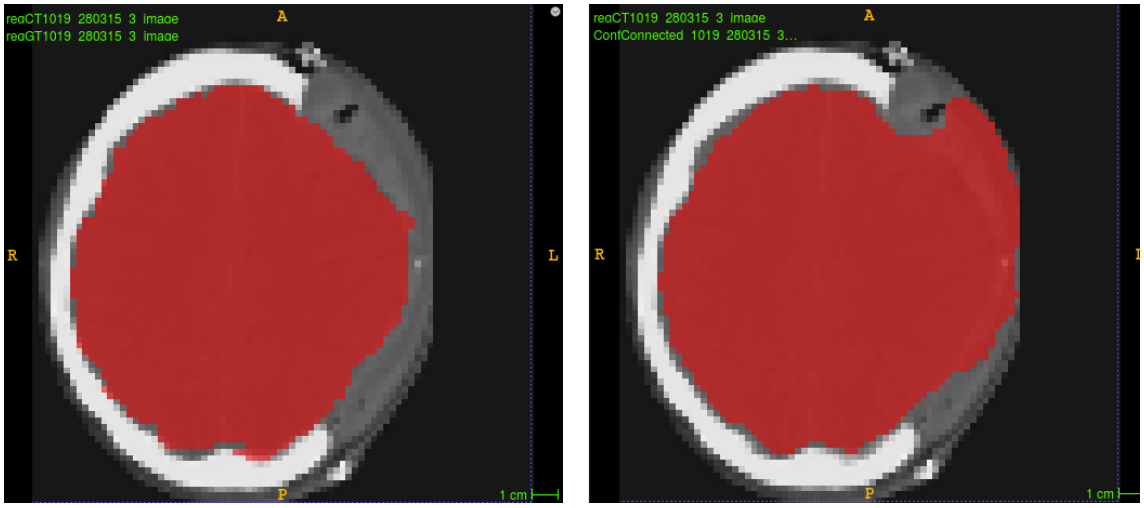

Figura 3.1: Comparación entre el ground truth (izquierda) y la segmentación hecha por el algoritmo de crecimiento de regiones (derecha).

caso los métodos tradicionales de segmentación por lo general fallan en las zonas en donde el cráneo fue removido. La estrategia en este caso fue entrenar un modelo de aprendizaje profundo basado en redes neuronales convolucionales a partir de segmentaciones manuales de la cavidad cerebral.

3.1. Los datos

En este proyecto se trabajó con un dataset provisto por médicos e investigadores de la Universidad de Cambridge en una colaboración con el director de este proyecto, el cual consiste en 92 imágenes de tomografía computarizada de pacientes con traumatismo de cráneo, de las cuales 27 imágenes se corresponden con pacientes que fueron sometidos a una craniectomía descompresiva y 65 poseen el cráneo completo. Además de las imágenes, se cuenta con las respectivas segmentaciones del cerebro realizadas por un especialista, por lo que, al tener un conjunto de segmentaciones *confiable* o *Ground Truth*, se podrán calcular medidas para evaluar la calidad de la segmentaciones generadas por los modelos propuestos en este trabajo.

Las imágenes provistas se encuentran en formato NIfTI[1], con las características indicadas

[1]NIfTI: Formato desarrollado por la Neuroimaging Informatics Technology Initiative, el cual se utiliza para el almacenamiento y visualización de neuroimágenes (imágenes del sistema nervioso central y el

Propiedad	Valor		
	x	y	z
Dimensiones	512	512	29
Spacing	0.4043	0.4043	5
Origen	0	0	0
Rango de Intensidades	-1000 a 2000 HU		
Orientación	RAI[I]		

[I] RAI: sistema de referencia en el cual las coordenadas crecen de derecha a izquierda en la dirección x, y crece de anterior a posterior y z de inferior a superior.

Tabla 3.1: Características de una imagen tipo de la base de datos. Los valores presentan pequeñas variaciones para las distintas imágenes.

en la Tabla 3.1.

La Figura 3.2 muestra un histograma típico de la distribución de intensidades en una CT cerebral. Comparando con la Tabla 2.1, se puede comprobar que los valores más frecuentes de intensidad en escala de Hounsfield se corresponden con el aire (-1000 HU) y materia blanca (20 HU). Los demás valores se corresponden a grasa, agua, sangre, músculo, materia blanca y gris, líquido cefalorraquídeo (entre estos dos picos) y hueso (a partir de 120 HU), entre otros. Cabe destacar que los rangos de intensidad en los que oscilan los distintos tipos de tejidos pueden solaparse, por lo que realizar una segmentación por rango de intensidades no daría un resultado adecuado.

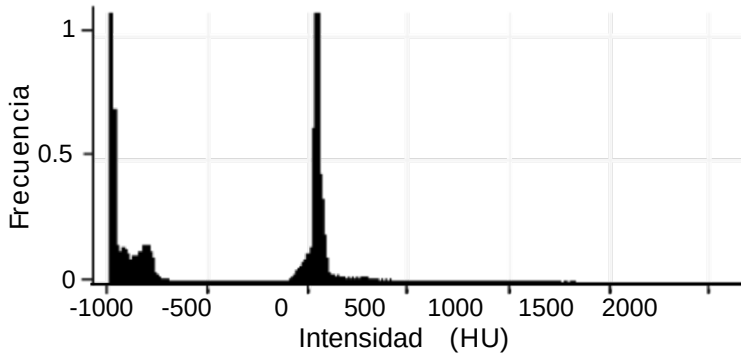

Figura 3.2: Ejemplo de histograma en una de las imágenes de TAC.

Tal como se mencionó en la Sección 2.4.2, es necesario establecer particiones de entrenamiento, validación y prueba a partir de la base de datos original para entrenar los modelos y evaluar su desempeño. A partir de las 92 imágenes, 19 fueron consideradas para prue-

cerebro).

ba (de las cuales 5 poseen craniectomía), mientras que de las restantes se utilizaron 62 imágenes para entrenamiento (20 con craniectomía) y 11 imágenes para validación (2 con craniectomía).

3.2. Preprocesamiento de los datos

En base a los recursos disponibles (descritos en la Sección 3.6.2) y las características de los datos mencionados en la sección anterior, se estableció una etapa de preprocesamiento con el objetivo de normalizar dichos datos para simplificar su posterior procesamiento.

3.2.1. Normalización de las imágenes

Una de las técnicas que permiten acelerar el entrenamiento de las redes neuronales es la normalización de los datos de entrada (Yang y Karahoca, 2006). Dependiendo del método aplicado, los datos son llevados un mismo rango de intensidades (en el caso de la normalización *min-max*) o las distribuciones de probabilidad de las intensidades son alineadas (normalización por *z-scores*).

Como se mencionó anteriormente, las imágenes del dataset se encuentran en la escala de unidades Hounsfield (ver Tabla 2.1). Dado que el rango de valores correspondiente a los tejidos cerebrales es acotado, se truncaron las intensidades en el intervalo $[-100, 300]$, ya que dicha escala asegura que no existen tejidos cerebrales por afuera del mismo. A las intensidades que se encuentran por afuera de ese rango, les fueron asignadas las intensidades del valor más próximo en el intervalo. Posteriormente se normalizaron las intensidades mediante el método *min-max* al rango $[0, 1]$. Para cada vóxel (x, y, z) de la imagen:

$$V(x, y, z) = \frac{V(x, y, z) - (-100)}{300 - (-100)} = \frac{V(x, y, z) + 100}{400}, \tag{3.1}$$

donde $V(x, y, z)$ es el valor de intensidad en (x, y, z). El resultado de este proceso se puede ver en la Figura 3.3.

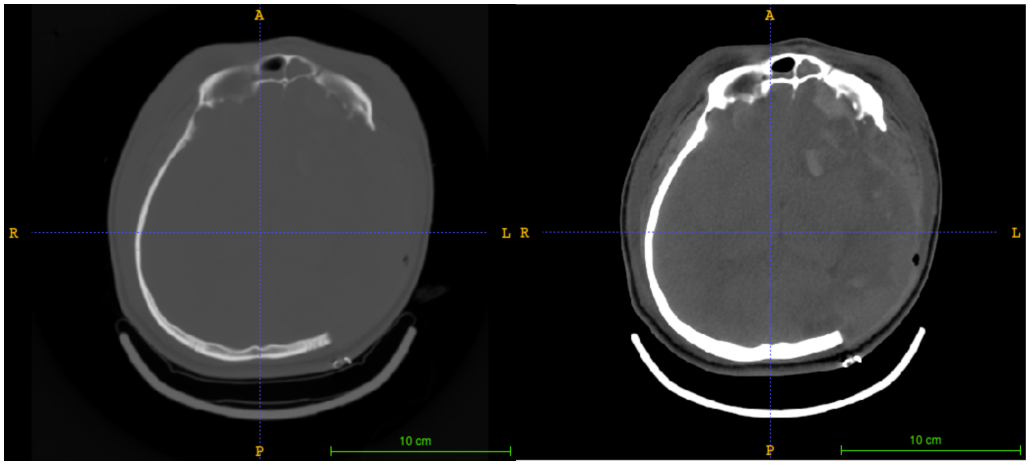

Figura 3.3: Recorte de intensidades y normalización. Izq.: imagen original. Der.: imagen normalizada

3.2.2. Registración de las imágenes

El paso siguiente en el preprocesamiento es la registración (o normalización espacial) de las imágenes, proceso que se realiza para simplificar la tarea de la red convolucional, debido a que las operaciones de convolución que se aplicarán en una etapa posterior no son invariantes ni a la rotación ni al escalado que pueden tener las distintas imágenes (Goodfellow *et al.*, 2016). Este proceso consiste en aplicar una transformación geométrica (como rotación, traslación, etc) sobre una imagen *móvil* para alinearla a una imagen *fija* por medio de la minimización de una medida de dis-similaridad. En este proyecto, utilizaremos un algoritmo de registración para alinear todas las imágenes en un espacio común que simplifique el posterior procesamiento de las mismas. Se utilizaron transformaciones afines para realizar la registración. Esto implica que las imágenes pueden no sólo ser rotadas y trasladadas, sino que también pueden ser deformadas por medio de una transformación lineal que mantiene el paralelismo entre rectas.

El espacio común en que serán alineadas todas las imágenes está dado por un *atlas* (o imagen promedio) provisto en la base de datos, el cual fue generado por medio de la superposición de diversas imágenes TAC co-registradas. Cada imagen es entonces registrada respecto al atlas que se muestra en la Figura 3.4 mediante una extensión de la biblioteca *SimpleITK* llamada *SimpleElastix* (Lowekamp *et al.*, 2013; Marstal *et al.*, 2016), en la cual se define una *imagen móvil* (la imagen que se quiere transformar), una *imagen fija*

(el atlas), un mapa de parámetros (en el cual se definen parámetros de la transformación como el tipo, interpoladores, cantidad de iteraciones, optimizador, entre otros) y da como resultados la imagen transformada y la matriz de la transformación aplicada. Dicha matriz de transformación es útil para deshacer este proceso de registración en una etapa posterior para volver a las dimensiones originales. Además, esta transformación es aplicada a las máscaras de segmentación provistas en la base de datos para trasladarlas al mismo espacio que la imagen preprocesada.

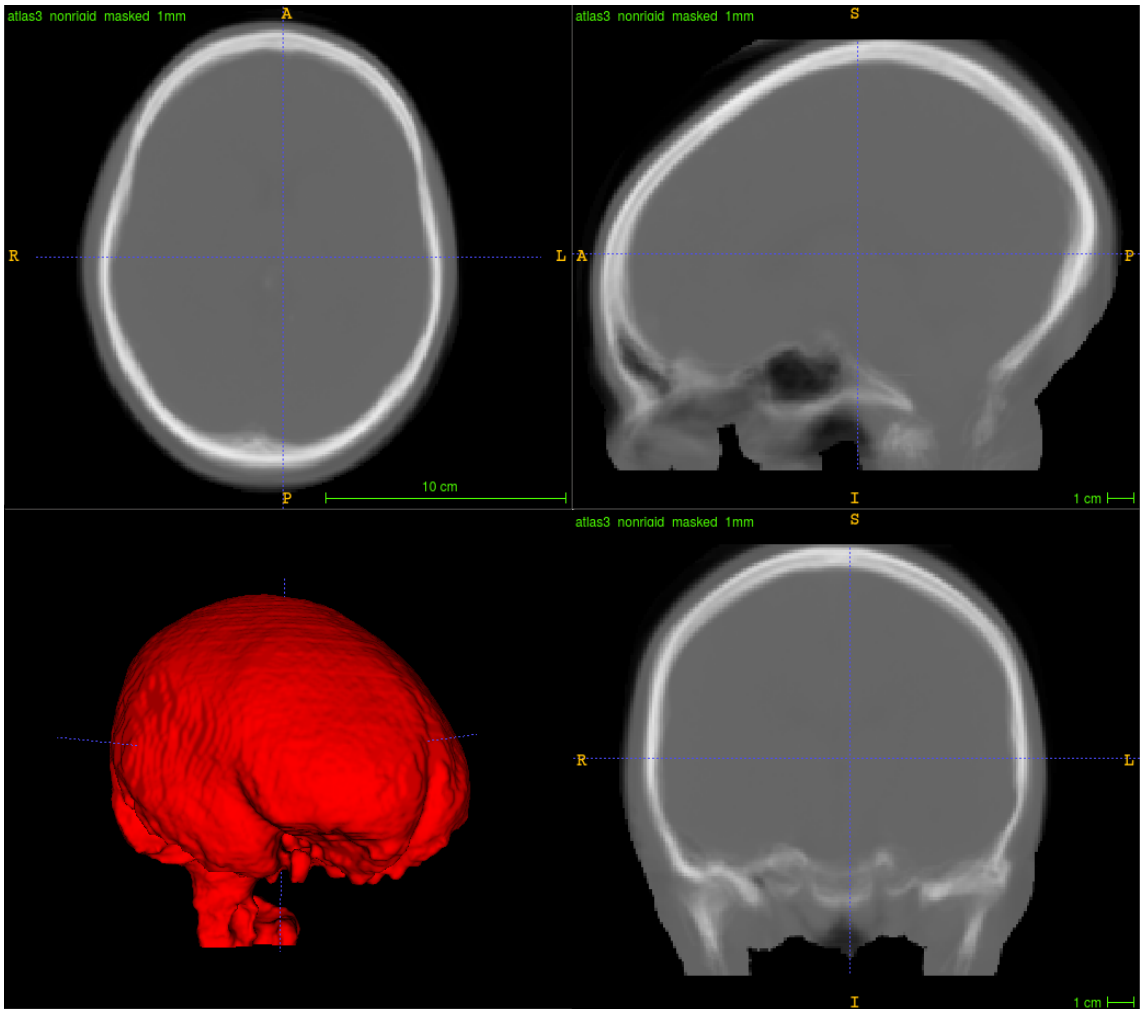

Figura 3.4: Atlas utilizado para la registración con su segmentación (en rojo).

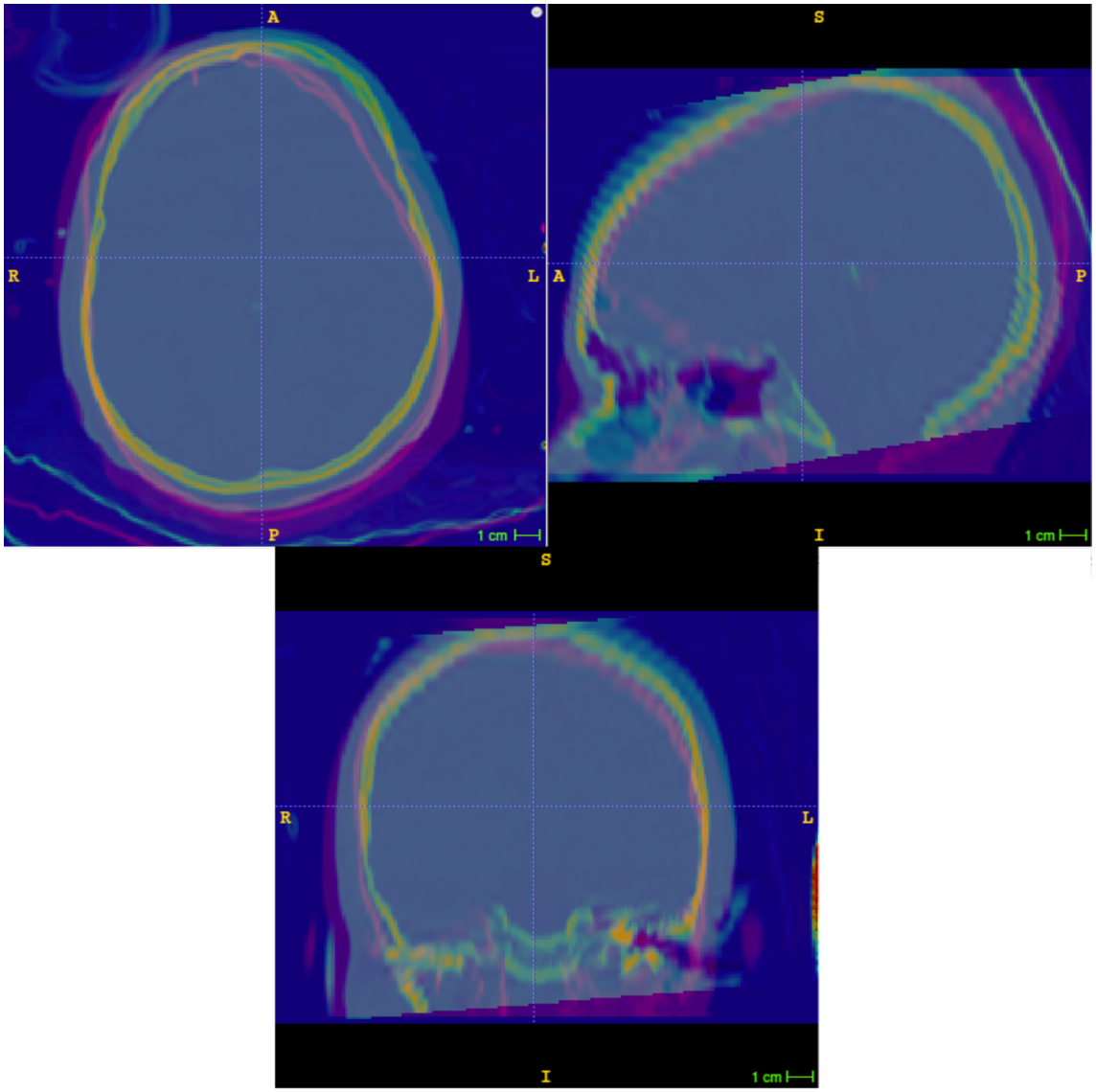

Figura 3.5: Ejemplo de imagen antes (en rojo) y después (en amarillo) de registración

3.2.3. Recorte de las imágenes al envoltorio convexo

Una vez que las imágenes se encuentran registradas, el paso siguiente consiste en recortarlas para descartar los espacios vacíos. Para ello, se calcula el envoltorio convexo que contenga todas las segmentaciones de cerebro en el espacio común obtenido luego de la registración. Para el cálculo de dicho envoltorio, se utilizó el algoritmo *Axis-Aligned Bounding Box* (AA-BB), el cual puede hallarse como los valores mínimos y máximos para cada eje en donde exista algún píxel distinto de cero (Calvo, 2017).

Para lograr esto, se calculó el AA-BB de la suma de todas las segmentaciones del *Ground Truth* de la partición de entrenamiento, al cual se le sumó un valor constante con el objetivo de incluir el cráneo, y finalmente se recortaron todas las imágenes al tamaño de este envoltorio convexo. De esta forma se reduce la posibilidad de tener tejidos o elementos que no introduzcan información de interés en la imagen. En la Figura 3.6 se ilustra este procedimiento.

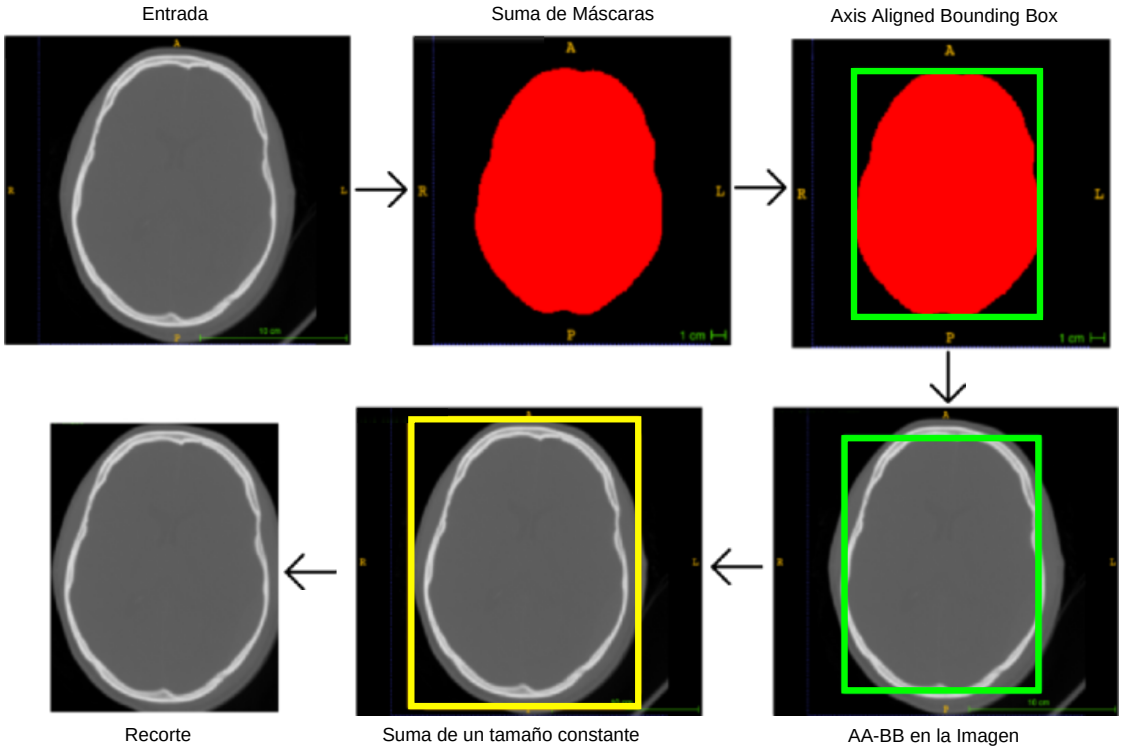

Figura 3.6: Recorte al envoltorio convexo. A partir de una imagen y la suma de todas las segmentaciones del dataset (lo cual es también una imagen binaria) se toma el bounding box de dicha suma y se recorta la imagen en un tamaño levemente superior al bounding box.

3.2.4. Remuestreo de las imágenes

Debido a que las imágenes del dataset poseen alta resolución en el plano axial, fue necesario reducir el tamaño de las imágenes para simplificar su posterior procesamiento. Por lo tanto, se procedió a remuestrear las imágenes haciendo uso de los métodos provistos en la biblioteca SimpleITK, llevándolas a un spacing (separación entre vóxeles) de 1mm para cada eje, lo que resultó en imágenes cúbicas de 80 vóxeles de lado.

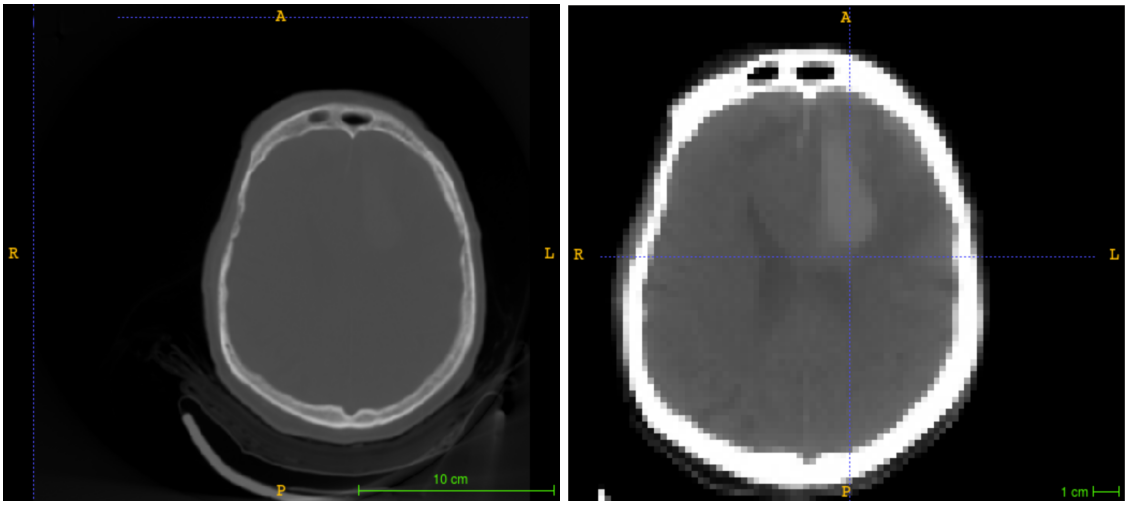

Figura 3.7: Imagen antes y después del preprocesamiento.

En la Figura 3.7 se puede observar cómo queda la imagen luego del preprocesamiento: la superficie ósea ahora se corresponde con las intensidades máximas (cercanas a 1), el fondo se corresponde con las intensidades mínimas (cercanas a 0), debido a la registración las imágenes se encuentran alineadas entre sí, y debido al recorte realizado se logra eliminar parte de la zona que no resulta de interés. Salvo las operaciones relacionadas con las intensidades (como la normalización), se aplicaron las mismas operaciones a las segmentaciones correspondientes, para tener también un Ground-Truth en el mismo espacio que la imagen preprocesada.

3.3. Modelos implementados

En esta sección, se detallarán las arquitecturas de redes neuronales convolucionales que fueron implementadas para realizar la segmentación cerebral, a partir de los datos preprocesados que se definieron en la etapa anterior.

3.3.1. U-Net

U-Net (Ronneberger *et al.*, 2015) es una arquitectura pensada para la segmentación de imágenes biomédicas. A grandes razgos, la red puede ser dividida en dos grandes bloques: En la primera mitad, la imagen de entrada se encoge hasta llegar a un tamaño 16 veces

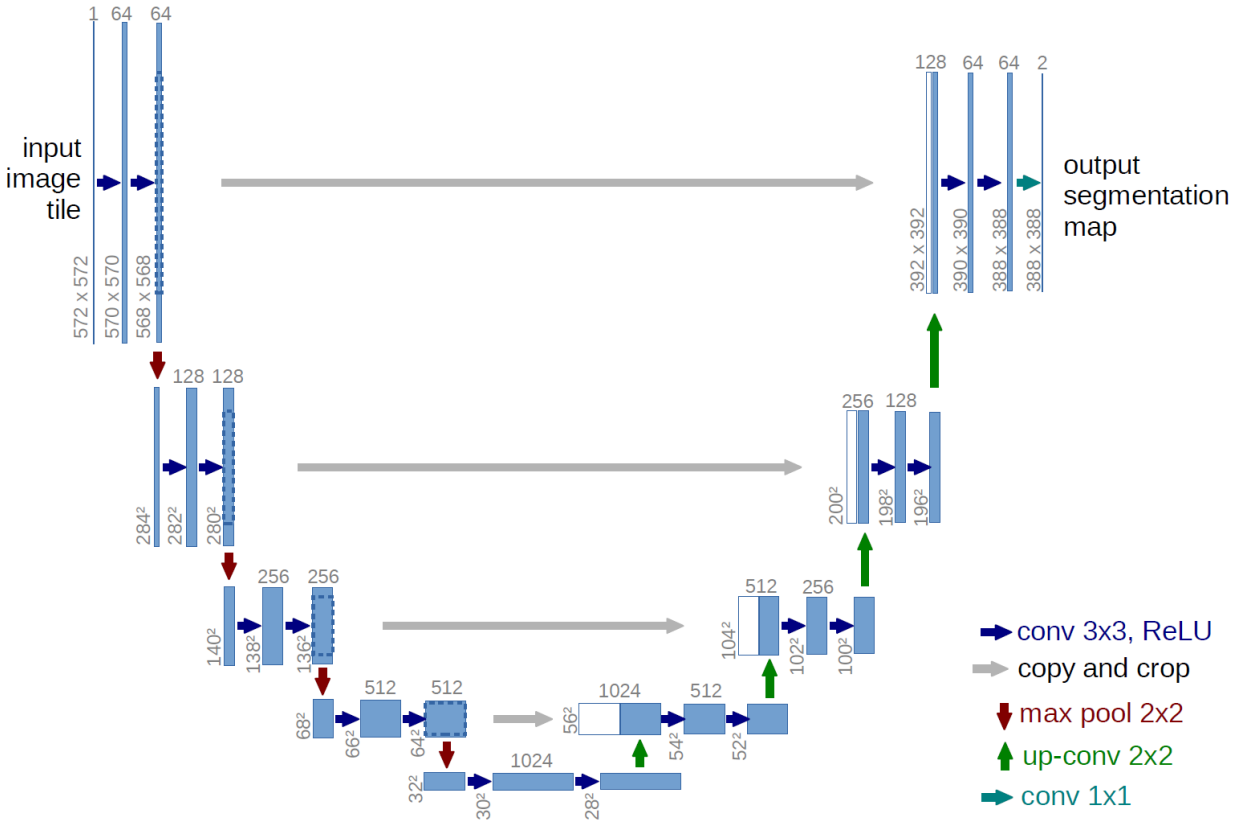

Figura 3.8: Arquitectura del modelo U-Net (Ronneberger *et al.*, 2015).

menor (etapa de codificación), capturando el contexto de la imagen. Luego, en la segunda mitad de la red, a partir de esa representación bajodimensional el modelo aprende a localizar aumentando el tamaño de esta representación hasta llegar al tamaño de la segmentación (etapa de decodificación), comparando la salida de la red con el Ground Truth. El modelo incorpora saltos de conexiones entre los niveles equivalentes de las etapas de codificación y decodificación, concatenando los correspondientes mapas de características, tal como puede verse en la Figura 3.8 (flechas grises).

Para compensar la poca cantidad de ejemplos anotados disponibles, el autor utiliza *aumentación de datos*, proceso por el cual los datos de entrenamiento se alteran efectuándoles transformaciones elásticas (Gabrani y Tretiak, 1996), aumentando así los datos de entrenamiento y reduciendo el *overfitting* (explicado en la Sección 2.4.2).

La arquitectura que propone el autor puede verse en la Figura 3.8.

Para este proyecto, se tomó la arquitectura de este modelo como base, y se le realizaron

una serie de modificaciones:

- Se planteó un modelo que trabaje en tres dimensiones espaciales mas una dimensión adicional para la intensidad (el original está pensado para dos dimensiones). Para esto, se hizo uso de convoluciones 3D en lugar de las clásicas convoluciones 2D.

- La cantidad de canales de salida en la primera capa convolucional se redujo de 64 a 16, modificando en consecuencia la cantidad de canales de las capas sucesivas. La determinación de la cantidad de canales inicial se tuvo experimentalmente, al notar que la calidad de las segmentaciones producidas era prácticamente igual, siendo la única diferencia considerable el tiempo de entrenamiento (que aumentaba con la cantidad de canales en las capas convolucionales). Además, al contar con una mayor cantidad de parámetros, es probable que el modelo aprenda no sólo las características más útiles de los datos de entrenamiento sino que además incorpore detalles que no servirán para los datos de prueba, sobreajustándose de esta manera a los datos de entrenamiento (Kumar, 2019).

- Se incorporó Batch Normalization (ver Sección 2.5.2) en cada capa, de manera de lograr una convergencia más rápida y precisa.

- Se probaron distintas funciones de pérdida y combinaciones de éstas para ver cuál resulta la más apropiada para realizar las segmentaciones. Esto se hizo debido a que es posible que en cada problema, una función de error de mejores resultados que otra (Patravali *et al.*, 2017), por lo que además de la entropía cruzada binaria, se probó la función de pérdida basada en el coeficiente Dice.

- En la estrategia de entrenamiento, no se utilizó la aumentación de datos propuesta por el autor.

3.3.2. Anatomically Constrained Neural Networks (ACNN)

Las redes neuronales anatómicamente restringidas (o ACNN por sus siglas en inglés) fueron propuestas por Oktay *et al.* (2017) como una estrategia de regularización para incorporar

conocimiento previo sobre las estructuras anatómicas de interés durante el proceso de entrenamiento, con el objetivo de mejorar la calidad de las predicciones realizadas.

En este proyecto, ACNN fue implementado con el objetivo de mejorar las segmentaciones cerebrales producidas por el modelo U-Net básico explicado en la sección anterior. En un principio, se planteó como hipótesis que la incorporación de restricciones anatómicas (en este caso, las restricciones estarán dadas por la forma del cerebro) al proceso de segmentación deberían solucionar el problema de sobresegmentación evidenciado en otros métodos (como el caso del algoritmo de crecimiento de regiones incluido en la Figura 3.1), logrando que la forma de las segmentaciones producidas tenga una estructura regular.

Para implementar las restricciones anatómicas, ACNN propone la construcción de un modelo basado en un autocodificador por eliminación de ruido (Vincent *et al.*, 2010), que es entrenado utilizando sólo las máscaras de segmentación, con el objetivo de capturar las características globales de dichas segmentaciones.

Un autocodificador (*Autoencoder* en inglés o AE en forma abreviada) consiste en un modelo de red neuronal que tiene como objetivo comprimir la entrada en un código bajo dimensional que captura las variaciones más importantes en los datos y, a partir de éste, realiza una reconstrucción de dicha entrada. La diferencia entre un autocodificador estándar y uno por eliminación de ruido es que, éste último, agrega algún tipo de ruido aleatorio a las entradas antes de que las mismas sean procesadas, e intenta reconstruir la entrada original a partir de las versiones ruidosas de la misma.

Estos modelos se diseñan de una manera incompleta (*undercomplete*), esto es, se fuerza a que el código sea de un tamaño menor a la entrada, logrando de esta manera que la red capture las características más distintivas de los datos de entrenamiento. Un ejemplo del funcionamiento se puede ver en la Figura 3.9.

El autocodificador implementa una etapa de codificación y una de decodificación al procesar una máscara de entrada. Durante la codificación, el modelo toma las segmentaciones de un conjunto de entrenamiento, les agrega ruido pimienta (ver Figura 3.10) en el borde de las mismas, y genera un código bajo dimensional. Este código ingresa en la etapa de decodificación, donde se reconstruye la segmentación sin ruido a partir de dicho código

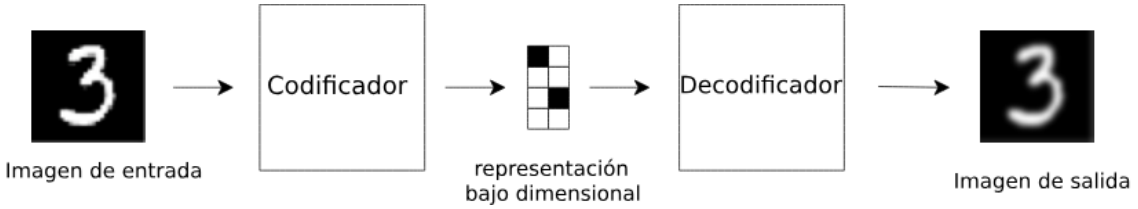

Figura 3.9: Ejemplo de autocodificador. Dada una imagen de entrada, el AE la codifica en una representación bajo dimensional y luego la decodifica, reconstruyendo la imagen de entrada.

bajo dimensional. Un diagrama de la etapa de codificación implementada puede verse en la Figura 3.11 (la etapa de decodificación es idéntica pero realiza los pasos inversos).

En este trabajo, el autocodificador por eliminación de ruido fue entrenado utilizando error cuadrático medio como función de pérdida. En el caso de la arquitectura de una ACNN, la idea es pre-entrenar un autocodificador por eliminación de ruido con las máscaras de segmentación provistas en el dataset de entrenamiento, y luego utilizar los códigos generados por el autocodificador para regularizar el proceso de aprendizaje de la red que aprende a segmentar.

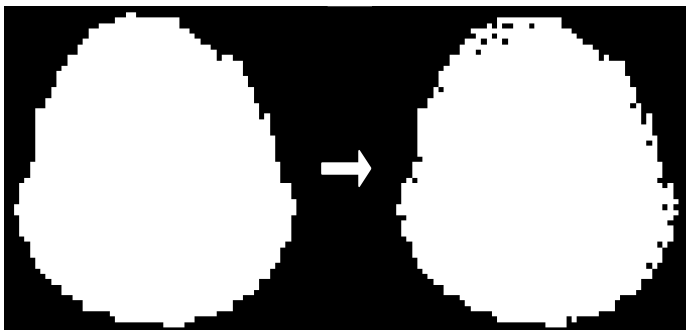

Figura 3.10: Agregado de ruido pimienta en los bordes de la segmentación con probabilidad 0.1. Esta segmentación es utilizada como entrada en el autocodificador por eliminación de ruido.

En la Figura 3.12 se muestra la estrategia propuesta en el trabajo original: a partir de una imagen de entrada y un modelo de segmentación (en este caso la arquitectura U-Net), dicho modelo se entrena utilizando una función de pérdida que se compone de dos medidas de error: por un lado, $L_{\mathcal{X}}$ que mide el error en la capa de salida (por ejemplo, pudiendo utilizar funciones como la *entropía cruzada* o la función de pérdida Dice definidas en la Sección 3.4) dando un error "pixel a pixel", y por otro lado L_{ACNN}, que busca que la

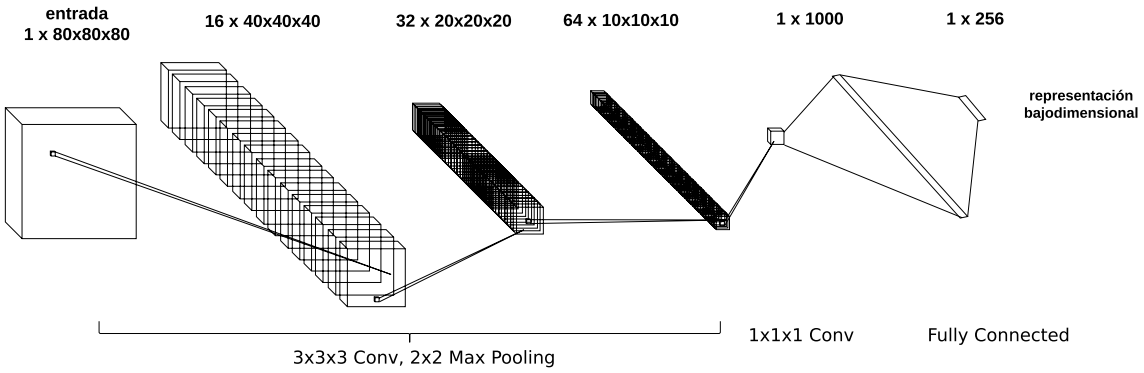

Figura 3.11: Etapa de codificación en el autocodificador implementado.

segmentación generada y el Ground Truth sean similares en un espacio bajo dimensional al comparar las codificaciones de ambas, incorporando así las restricciones anatómicas al proceso de aprendizaje:

$$L_{ACNN}(x, y) = \|h(\phi(x)) - h(y)\|^2, \tag{3.2}$$

donde x es la imagen de entrada, y es la segmentación ground truth, ϕ es la predicción realizada por la red neuronal y h es la codificación generada por el autocodificador. Al entrenar un modelo con restricciones anatómicas, la función de pérdida a minimizar queda entonces definida por:

$$L(x, y) = L_{\mathcal{X}}(x, y) + \lambda_1 \cdot L_{ACNN}(x, y) \tag{3.3}$$

siendo λ_1 un peso para este nuevo término de error.

3.4. Funciones de pérdida y modelos evaluados

En base a los modelos presentados en la sección anterior, para la segmentación se implementó la arquitectura U-Net con las modificaciones planteadas, agregando también la posibilidad de incorporar restricciones anatómicas y de entrenarla tanto con la función de pérdida basada en entropía cruzada binaria (definida en la ecuación 3.5) como con la

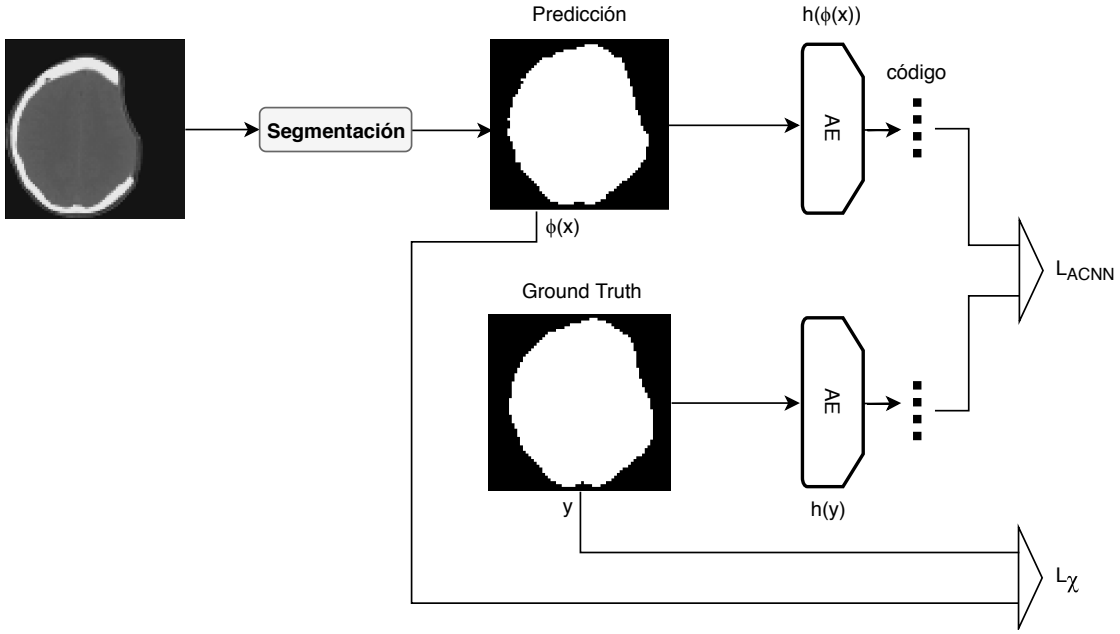

Figura 3.12: Estrategia de entrenamiento ACNN (Oktay *et al.*, 2017). A partir de una imagen, la red convolucional produce una segmentación $\phi(x)$ que es comparada de dos formas diferentes con el ground truth y. Por un lado, $\phi(x)$ ingresa a un autocodificador que produce un código bajo-dimensional $h(\phi(x))$. Dicho código es comparado con el código del ground truth $h(y)$ mediante el error cuadrático medio, formando L_{ACNN}. Por otro lado, $\phi(x)$ e y son comparados píxel a píxel mediante otra medida de error $L_{\mathcal{X}}$ como el error de entropía cruzada binaria o la función Dice.

función de pérdida *Dice* (Liu *et al.*, 2018). Para el caso de un problema de segmentación binario (tal como el de la segmentación de cerebro), esta última función se define como:

$$L_{Dice}(x, y) = 1 - 2\frac{\sum_j \phi^j(x) y^j + \epsilon}{\sum_j \phi^j(x) + y^j + \epsilon}, \tag{3.4}$$

siendo j una posición en la imagen, y la máscara binaria que representa el cerebro a segmentar (es decir, el ground-truth), $\phi(x)$ la predicción realizada por la red neuronal dada la entrada x, y ϵ es un factor de suavizado que resulta de utilidad para evitar la división por cero en el caso que tanto la predicción como el ground truth sean nulos.

La función de pérdida Dice[2] es una función ampliamente utilizada en tareas de segmentación, ya que mide la similaridad o solapamiento entre las segmentaciones generadas y el *Ground Truth*. Como en este caso esta función se desea minimizar, al valer cero indica un total solapamiento entre las segmentaciones, y al valer uno indica que las segmentaciones no poseen vóxeles en común.

Adicionalmente, en el contexto de la segmentación binaria de imágenes, si ϕ^j representa la probabilidad de que el pixel j corresponda a la clase foreground, entonces es posible definir una función de pérdida basada en la entropía cruzada binaria dada por:

$$L_{BCE}(x, y) = -\sum_j y^j \log(\phi^j(x)), \tag{3.5}$$

Siguiendo la estrategia presentada en (Patravali *et al.*, 2017), se considerará un modelo adicional donde ambas funciones de pérdida (Dice y entropía cruzada binaria) son combinadas en una única función.

Para el entrenamiento, en base a las ecuaciones 3.5, 3.2 y 3.4, se elaboró una única función de pérdida para entrenar la arquitectura implementada, la cual permite activar o desactivar

[2]Una función idéntica pero numéricamente opuesta a esta es el coeficiente Dice, el cual se utiliza como métrica para evaluar la calidad de la segmentación (ver Sección 3.7.1), indicando que existe un solapamiento cuando el coeficiente vale uno.

Parámetro	BCE	Dice	BCE+Dice	BCE+ACNN	Dice+ACNN	BCE+Dice+ACNN
λ_1	1	0	1	1	0	1
λ_2	0	1	1	0	1	1
λ_3	0	0	0	0.1	0.1	0.1

Tabla 3.2: Ponderación de cada función de pérdida para cada combinación posible.

cada uno de los términos por medio de los pesos $\lambda_{1,2,3}$:

$$L(x, y) = \lambda_1 \cdot L_{BCE}(x, y) + \lambda_2 \cdot L_{dice}(x, y) + \lambda_3 \cdot L_{ACNN}(x, y), \tag{3.6}$$

donde L_{BCE}, L_{dice} y L_{ACNN} son las funciones de pérdida de entropía cruzada, Dice y error cuadrático medio desarrolladas en las ecuaciones antes descriptas y λ_1, λ_2 y λ_3 la ponderación de cada una de estas funciones de error en la suma.

Para determinar cuál es la combinación de λ más adecuada para obtener el modelo de segmentación buscado, y dada la cantidad de combinaciones posibles de funciones de pérdida, se entrenaron por separado seis modelos distintos, cada uno utilizando una combinación de funciones de pérdida distintas, incorporando (o no) de esta manera la regularización anatómica y el error Dice en el modelo U-Net.

En la Tabla 3.2 se muestran los valores de λ utilizados para cada combinación de funciones de pérdida, y en la Tabla 3.3 los hiperparámetros utilizados para todos los modelos. Dichos λ fueron establecidos mediante la variación de los mismos y observando el comportamiento de la curva de error.

Para el caso del autocodificador utilizado para realizar la regularización por restricciones anatómicas, la única función de pérdida utilizada es el error cuadrático medio, descripto en la ecuación 2.7.

3.5. Detalles de la implementación y búsqueda de hiperparámetros

Para la elaboración de la herramienta propiamente dicha, se implementó un proyecto en Python 2.7 utilizando la biblioteca PyTorch para la construcción de las arquitecturas de redes neuronales (ver Sección 3.6.2 para una descripción completa de los recursos utilizados), cuyos módulos[3] se pueden observar en la Figura 3.13, y se describen brevemente a continuación:

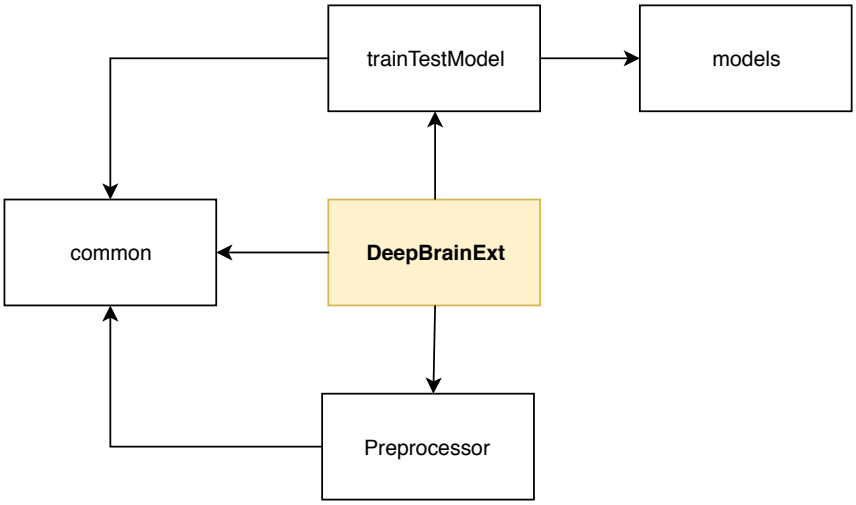

Figura 3.13: Módulos del proyecto. Las flechas indican la inclusión de un módulo. En amarillo, el módulo que implementa la herramienta por línea de comandos.

- DeepBrainExt: Este módulo implementa la herramienta por línea de comandos, y contiene la clase *controller*, la cual funciona como intermediario entre la herramienta y las clases que implementan sus funcionalidades.
- trainTestModel: Contiene la clase *modelLoader*, la cual se encarga de entrenar los modelos implementados y posteriormente cargarlos para generar predicciones. También posee la clase BrainSegmentationDataset, la cual define cómo se abren los archivos que se cargarán en la red neuronal y define si es necesario transformaciones que se aplicarán a los mismos.
- Preprocessor: Posee la clase que se encarga de efectuar el preprocesamiento de las

[3]En Python, un módulo es un archivo que contiene definiciones y declaraciones de Python.

imágenes, descritos en la Sección 3.2.

- common: Funciones de uso general que son compartidas por los demás módulos.
- models: Posee las clases que implementan los modelos convolucionales descriptos en la Sección 3.3:
  - U-Net (UNetF en el diagrama de clases): Utilizado para efectuar la segmentación cerebral. Como entrada recibe una imagen de TAC y produce en su salida una imagen binaria (la segmentación).
  - Autocodificador ACNN (aeACNN en el diagrama de clases): autocodificador utilizado para efectuar la regularizarización del modelo U-Net, siguiendo la estrategia de entrenamiento de Oktay *et al.* (2017). Recibe como entrada una imagen binaria (segmentación con ruido sal y pimienta) y produce como salida otra imagen binaria (segmentación sin ruido).

En la Figura 3.14 se muestra un diagrama de clases que incluye las mencionadas anteriormente.

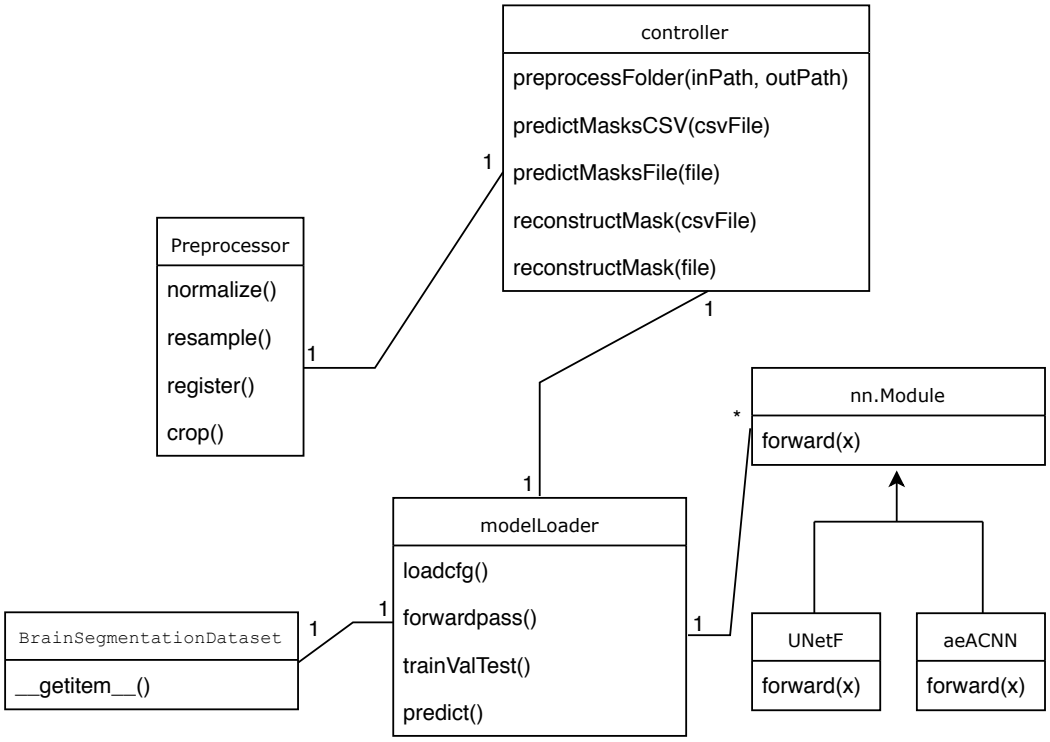

Figura 3.14: Diagrama de las clases del modelo.

3.5.1. Entrenamiento del modelo

Una vez finalizada la etapa de implementación de los modelos, el paso siguiente consiste en entrenarlos y realizar la búsqueda de hiper-parámetros (estos son aquellos parámetros que determinan cómo se va a realizar el entrenamiento, pero no forman parte del modelo, por ejemplo: la tasa de aprendizaje, el tamaño del batch, etc.). Estos parámetros no son aprendidos durante el proceso de entrenamiento de la red neuronal (como los pesos), sino que se definen a través de un proceso de exploración.

Para facilitar el cambio de hiperparámetros y opciones relacionadas con el entrenamiento de los modelos implementados, se decidió cargarlos en un archivo de configuración, el cual es definido para cada modelo entrenado. Un ejemplo de archivo de configuración puede verse en la Figura 3.15.

```
[MODEL]
Type = unet
name = u_BCE_dice_ACNN
device = cuda
Epochs = 300
resumeTraining = models/u_BCE_dice_ACNN_ep100.pt
optimizer = sgd
learningRate = 0.001
momentum = 0.99
weightDecay = 0
ACNNlambda = 1
dicelambda = 2
bce_lambda = 1
batch_size = 4

[PATHS]
modelsPath = models
ACNNmodel = models/a_AE.pt
plots = img
trainData = ../scans/bboxandclip/all/files_train.csv
validationData = ../scans/bboxandclip/all/files_validation.csv
testData = ../scans/bboxandclip/all/files_test.csv

[MISC]
imgPrefix = image
maskPrefix = head_mask
plotUpdateInterval = 10
autosaveEpochs = 100
saveDicePlots = True
```

Figura 3.15: Ejemplo de archivo de configuración.

Los parámetros que se pueden cambiar desde este archivo son:

- Model: Hiperparámetros relativos al entrenamiento.

- type: Tipo de red neuronal convolucional a utilizar. Valores posibles: “unet” (para generar segmentaciones), “acnn” (autoencoder de segmentaciones), “aerec” (autoencoder de imágenes TAC).
- name: Nombre del archivo correspondiente al modelo entrenado. Si no se proporciona, se construye con el tipo y la fecha.
- device: Establece si el modelo se ejecuta en la GPU (“cuda”) o en la CPU (“cpu”).
- Epochs: Cantidad de épocas de entrenamiento.
- resumeTraining: Permite cargar un modelo de PyTorch previamente entrenado y continuar su entrenamiento, ingresando su ruta.
- optimizer: Permite seleccionar el optimizador (“adam” o “sgd”).
- learningRate: Establece la tasa de aprendizaje.
- momentum: Establece el término de momento en el optimizador.
- weightDecay: Incorpora regularización L2, también llamada weight decay.
- ACNNlambda: Si es mayor a cero, incorpora la regularización del método ACNN (en el caso de que se provea un autocodificador entrenado). Se corresponde con λ_3 de la ecuación 3.6.
- dicelambda: Si es mayor a cero, incorpora a la función de pérdida el error Dice entre la salida del modelo y el ground truth, ponderando esta función de pérdida con el valor establecido en este parámetro. Se corresponde con λ_2 de la ecuación 3.6.
- bce_lambda: Si es mayor a cero, incorpora a la función de pérdida el error por entropía cruzada binaria, ponderando esta función de pérdida con el valor establecido en este parámetro. Se corresponde con λ_1 de la ecuación 3.6.
- batch_size: Determina el tamaño del batch utilizado durante el entrenamiento.

- Paths: Variables relacionadas a rutas.
  - modelsPath: Carpeta donde guardar el modelo entrenado.

- ACNNmodel: Si se incorpora la regularización por ACNN, mediante este parámetro se establece la ruta del autoencoder entrenado para realizar esta regularización.
- plots: Carpeta en donde se guardarán las gráficas (boxplots y curvas de error).
- trainData, validationData, testData: Rutas de los archivos csv que contienen las rutas de los archivos correspondientes a los splits de entrenamiento, validación y test respectivamente.

▪ Misc: Parámetros adicionales.

- imgPrefix: string que identifica a la imagen en el nombre de un archivo al guardarlo.
- maskPrefix: string que identifica a la segmentación en el nombre de un archivo al guardarlo.
- plotUpdateInterval: Establece cada cuántas épocas se actualizan los archivos que poseen las gráficas de las curvas de error en las particiones de entrenamiento y validación.
- autosaveEpochs: Permite guardar el modelo entrenado luego de tantas épocas como indique este parámetro, junto con las curvas de error en ese instante.
- saveDicePlots: Si es "True", además de las curvas de error guarda también una curva en la que se muestra la evolución del coeficiente Dice.

La Tabla 3.3 detalla cuáles resultaron los hiperparámetros utilizados para el entrenamiento de los seis modelos de segmentación mencionados anteriormente. Los hiperparámetros utilizados para entrenar el autocodificador propuesto en el modelo ACNN se muestran en la Tabla 3.4. En ambos casos, se realizó una búsqueda de grilla para determinar los parámetros, lo cual consiste en especificar manualmente un conjunto de valores para cada hiperparámetro para evaluar la precisión del modelo en las particiones de entrenamiento y validación con cada combinación posible de parámetros, y escogiendo aquella combinación

Parámetro	Valor
Cantidad de epocas	500
Tamaño del batch	10
Optimizador	SGD[I]
Tasa de aprendizaje	0.0001
Momentum	0.99
Weight Decay	0.0005

[I] SGD: Gradiente Descendiente Estocástico

Tabla 3.3: Hiperparámetros utilizados durante el entrenamiento de todos los modelos de segmentación.

Parámetro	Valor
Cantidad de epocas	1000
Tamaño del batch	8
Optimizador	SGD[I]
Tasa de aprendizaje	0.0001
Momentum	0.99
Weight Decay	0
Probabilidad de ruido	0.1
Relación Sal vs Pimienta	0.01
Profundidad del ruido	0.1

[I] SGD: Gradiente Descendiente Estocástico

Tabla 3.4: Hiperparámetros para el entrenamiento del autocodificador utilizado en el regularizador ACNN.

que muestre mayor capacidad de generalización (Goodfellow *et al.*, 2016).

3.5.2. Herramienta por línea de comandos

Para la implementación de la herramienta por línea de comandos, se utilizó el módulo de Python *argparse*, que está incluido en Python 2.7 y permite escribir interfaces de este tipo con la incorporación de argumentos, ayudas, mensajes de uso y error cuando le son proporcionados parámetros incorrectos (Sneeringer, 2015).

Si bien para el modelo de segmentación se probaron varias combinaciones de funciones de prueba, para el modelo incluido en la herramienta de uso final se eligió como función de pérdida aquella que brinda un mejor desempeño con los datos de prueba para un conjunto de métricas establecido (ver Sección 3.7).

La herramienta permite:

- Preprocesar una carpeta y generar un archivo csv de la salida, dadas las rutas de la carpeta de entrada y salida.
- Predecir máscara(s):
  - de una sola imagen, dada su ruta.
  - de un conjunto de imágenes listadas en un archivo csv, dada su ruta.
- Predecir la(s) reconstrucción(es) del cráneo en casos de craniectomía descompresiva (esta funcionalidad es considerada en los trabajos futuros descriptos en la Sección 4.2.1):.
  - de una sola imagen, dada su ruta.
  - de un conjunto de imágenes listadas en un archivo csv, dada su ruta.

El formato de los archivos csv aquí mencionados es el siguiente (las rutas pueden ser tanto absolutas como relativas):

```
image,segmentation
datos/0001_image.nii.gz,datos/0001_segmentation.nii.gz
datos/0002_image.nii.gz,datos/0002_segmentation.nii.gz
...
```

En el caso de existir el archivo indicado como `segmentation`, se mostrará el error obtenido dependiendo del modelo que se esté utilizando, de lo contrario se ignorará esta ruta y sólo se generará la predicción. Este formato es el utilizado durante el entrenamiento para cargar las imágenes y sus respectivas segmentaciones.

3.5.3. Repositorio de la herramienta

La herramienta desarrollada para este proyecto final de carrera y su respectivo código fuente se encuentran disponibles en el siguiente respositorio: `https://gitlab.com/matzkin/deep-brain-extractor`. Una captura del sitio se muestra en la Figura 3.17.

```
(venv) fmatzkin@pc49:~/pfc/deep-brain-extractor$ python DeepBrainExt.py --help

usage: DeepBrainExt.py [-h] [-p PREP_FOLDERS PREP_FOLDERS] [-csv PREDICT_CSV]

Deep Brain Extraction tool.

optional arguments:
  -h, --help            show this help message and exit
  -p PREP_FOLDERS PREP_FOLDERS
                        Preprocess: Paths of the input and output folder.
  -csv PREDICT_CSV      Predict segmentation: csv with the files for making
                        predictions (they must be preprocessed)
```

Figura 3.16: Ayuda de la herramienta.

3.6. Recursos utilizados

En esta sección se detallarán cuáles fueron los recursos (físicos y tecnologías) utilizados para la implementación de la herramienta desarrollada en este Proyecto Final de Carrera.

3.6.1. Tecnologías

La implementación de esta herramienta se realizó en Python 2.7, utilizando la biblioteca de aprendizaje profundo PyTorch 0.4.0 (Paszke *et al.*, 2017) para la implementación y prueba de los modelos de redes neuronales propuestos. Esta biblioteca brinda flexibilidad y facilidad en el prototipado sin tener que preocuparse demasiado en al complejidad del framework (Subramanian y Agarwal, 2018), con la posibilidad de poder depurar con detalle los modelos planteados (en comparación con otros frameworks como Keras) lo cual permite tener un modelo funcionando en poco tiempo (Migdal, 2018).

Para el preprocesamiento y registración se utilizó SimpleITK (Lowekamp *et al.*, 2013), una herramienta de código abierto para Python que actúa como una capa simplificada de ITK (Ibanez *et al.*, 2003). ITK es un toolkit originalmente implementado en lenguaje C, que se utiliza para desarrollar programas de segmentación y registración de imágenes médicas. Además, se utilizó SimpleElastix (Marstal *et al.*, 2016) para efectuar la registración de imágenes.

Para llevar a cabo el desarrollo se utilizó el entorno (IDE) PyCharm[4], el cual facilita la

[4] `https://www.jetbrains.com/pycharm/`

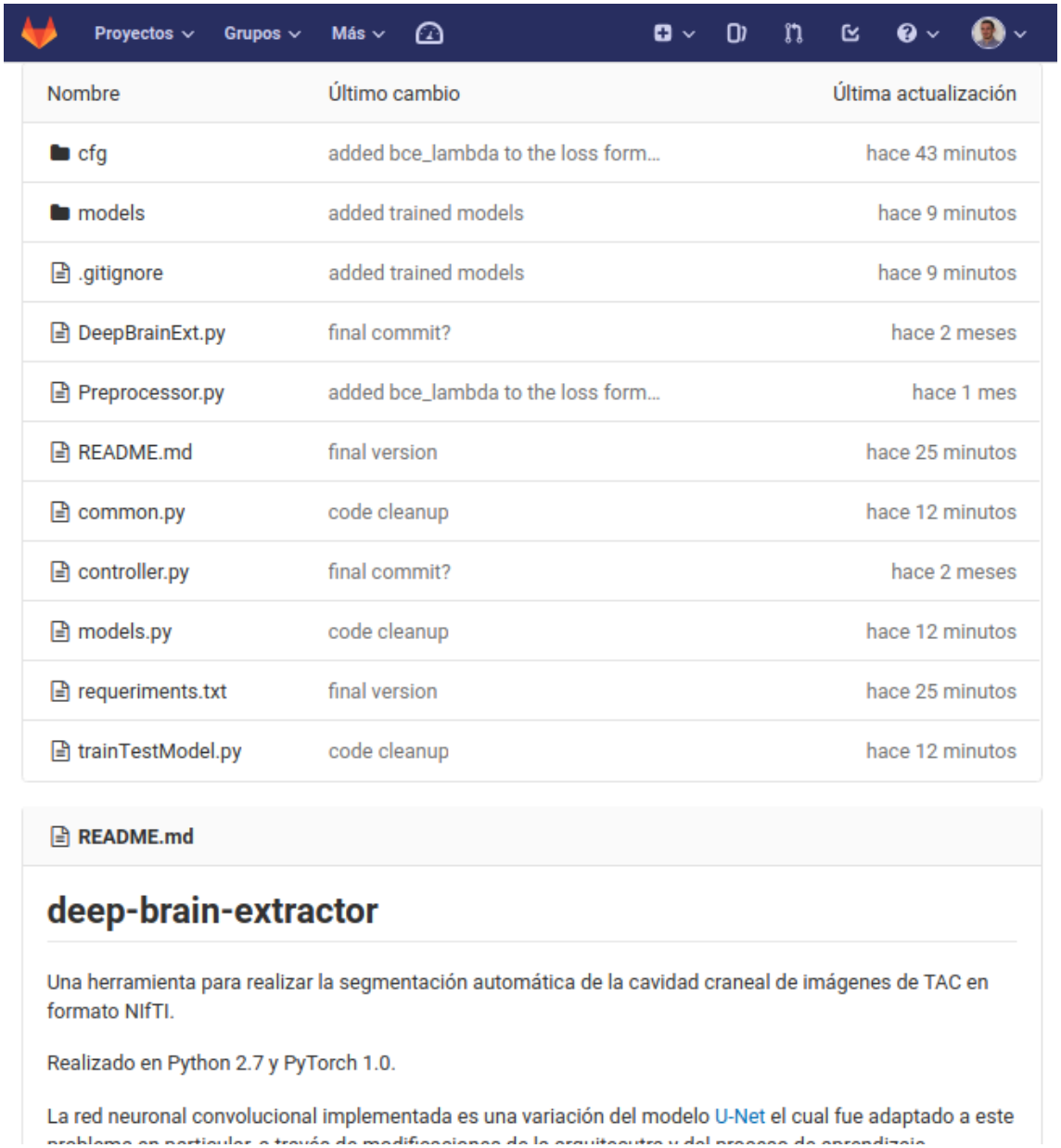

Figura 3.17: Vista del repositorio donde está alojado el proyecto (`https://gitlab.com/matzkin/deep-brain-extractor`).

Figura 3.18: Algunas tecnologías utilizadas durante el desarrollo.

escritura de código, la depuración, control de versiones y desarrollo remoto. Si bien parte del proyecto fue desarrollado y probado de manera local en una PC de escritorio sin placa GPU, al momento de tener que implementar las redes convolucionales se continuó trabajando en un servidor con GPU ubicado en el Instituto de Señales, Sistemas e Inteligencia computacional, *sinc(i)*, de manera remota. Por esta razón, para la ejecución de los scripts de entrenamiento de los modelos, se mantuvo una carpeta con el proyecto en dicho servidor, y la ejecución del código se realizó de forma remota media ssh. Para conservar las sesiones activas en el servidor remoto se utilizó tmux (*terminal multiplexer*), el cual permitió en este caso, a partir de una sesión en el servidor, crear un conjunto de terminales que puedan ser visibles de manera remota con la posibilidad de desconectarse, seguir ejecutándose en el servidor y eventualmente volver a establecer una conexión para acceder a las terminales sin que los procesos que están ejecutando fueran interrumpidos. Sin tmux, cuando la terminal se cierra de manera remota, el proceso en ejecución se interrumpe inmediatamente. Esta aplicación resultó de mucha utilidad debido a que el entrenamiento de los modelos suele llevar varias horas, por lo que era posible conectarse mediante cualquier terminal con ssh (incluso desde un celular mediante la aplicación JuiceSSH) para monitorear el estado de la ejecución. Al tener también acceso al servidor Pirayú, también se utilizó el sistema

de manejo de clusters y scheduling slurm para procesar trabajos en dicho servidor.

Para alojar el proyecto y gestionar los cambios se utilizó GitLab, un servicio web de control de versiones basado en Git el cual brinda la posibilidad de tener repositorios privados de manera gratuita. Esto resultó de utilidad durante el desarrollo de este proyecto.

Para visualizar las imágenes tridimensionales se utilizó la aplicación ITK-SNAP, una herramienta gratuita, de código abierto y multiplataforma útil para la visualización y segmentación de estructuras tridimensionales.

Paquetes de Python

Además de SimpleITK, otros paquetes relevantes de Python utilizados en este proyecto fueron Matplotlib, MedPy, Nibabel, Numpy, Pandas, Pillow, pip, Scipy y Torchvision.

3.6.2. Recursos físicos

Para el desarrollo del presente trabajo se utilizaron recursos computacionales del Instituto de Investigación en Señales, Sistemas e Inteligencia Computacional sinc(i), ubicado en la Facultad de Ingeniería y Ciencias Hídricas de la Universidad Nacional del Litoral, utilizando para el entrenamiento de los modelos una GPU NVIDIA Titan XP, donada mediante el programa *GPU Grant* de *NVIDIA Corporation*.

Además se utilizó el cluster Pirayú, adquirido con fondos de la Agencia Santafesina de Ciencia, Tecnología e Innovación (ASACTEI), Gobierno de la Provincia de Santa Fe, mediante el Proyecto AC-00010-18, Resolución No117/14, el cual forma parte del Sistema Nacional de Computación de Alto Desempeño de la Secretaría de Ciencia, Tecnología e Innovación Productiva de la República Argentina.

3.7. Resultados

En esta sección se detallarán los resultados obtenidos producto de la implementación de los modelos desarrollados en el apartado anterior, contrastándolos con dos de los métodos

mencionados anteriormente que no están basados en redes neuronales: Conjuntos de nivel y crecimiento de regiones. Para la implementación de dichos algoritmos se utilizaron los métodos `GeodesicActiveContourLevelSetImageFilter` (para conjuntos de nivel) y `ConfidenceConnectedImageFilter` (para crecimiento de regiones) provistos en la biblioteca SimpleITK.

3.7.1. Métricas para la evaluación de los resultados

Para comprobar la precisión de los métodos entrenados, utilizaron tres métricas estándar para la comparación de las segmentaciones producidas por los distintos algoritmos (S_A) con respecto a las segmentaciones provistas por el dataset (S_{GT}, por *ground-truth*): el coeficiente Dice, la distancia de Hausdorff y la distancia media promedio.

Coeficiente Dice

Esta métrica es la más utilizada para validar segmentaciones volumétricas (Taha y Hanbury, 2015), dando una medida del solapamiento entre las segmentaciones comparadas. Este coeficiente puede tomar valores en el rango [0,1]. Cuanto más se solapen o coincidan las segmentaciones, más cercano a 1 será el coeficiente.

Matemáticamente se puede definir como:

$$D(S_A, S_{GT}) = \frac{2|S_A \cap S_{GT}|}{|S_A| + |S_{GT}|}, \tag{3.7}$$

donde || indica la cantidad de elementos marcados como foreground en la máscara de segmentación, y $\cap$ es la operación de intersección entre las máscaras.

Distancia de Hausdorff

La distancia de Hausdorff se define como la distancia máxima entre las segmentaciones. Debido a que se busca que dichas segmentaciones sean similares, un valor bajo en esta

métrica indicará mayor calidad en las segmentaciones obtenidas. Esta distancia se mide en las mismas unidades que separan los vóxeles de la imagen (en el caso particular de este trabajo, en mm). Esta medida resulta muy sensible al ruido que pueda tener la segmentación, por lo que será de gran utilidad combinada con otras métricas para asegurar la precisión de un método.

Matemáticamente se define como:

$$d_H(S_A, S_{GT}) = \text{máx}\{ \sup_{x \in S_A} \inf_{y \in S_{GT}} d(x,y), \sup_{y \in S_{GT}} \inf_{x \in S_A} d(x,y) \} \tag{3.8}$$

donde x e y son puntos en las superficies de las segmentaciones, y $d(x, y)$ una medida de la distancia entre esos puntos.

Distancia Media Promedio

Esta es una métrica similar pero más estable que la anterior, ya que es menos sensible a valores atípicos (es decir, es menos sensible a segmentaciones espurias). Consiste en calcular el promedio de las distancias entre las superficies de ambas segmentaciones.

Matemáticamente, se define como

$$d_A(S_A, S_{GT}) = \text{máx}\{ \frac{1}{N} \sum_{x \in S_A} \min_{y \in S_{GT}} d(x,y), \frac{1}{N} \sum_{x \in S_{GT}} \min_{y \in S_A} d(x,y) \} \tag{3.9}$$

donde x e y son puntos en las superficies de las segmentaciones, N es la cantiadad de pares de puntos evaluados y $d(x, y)$ una medida de la distancia entre esos puntos.

3.7.2. Análisis de los resultados

A continuación, se incluyen los resultados experimentales obtenidos en los datos de prueba utilizando los diferentes modelos propuestos (siguiendo el particionamiento en datos de entrenamiento, validación y prueba descripto en la Sección 3.1). En la Figura 3.19 pueden verse los diagramas de caja y bigote de las métricas calculadas para cada uno de los

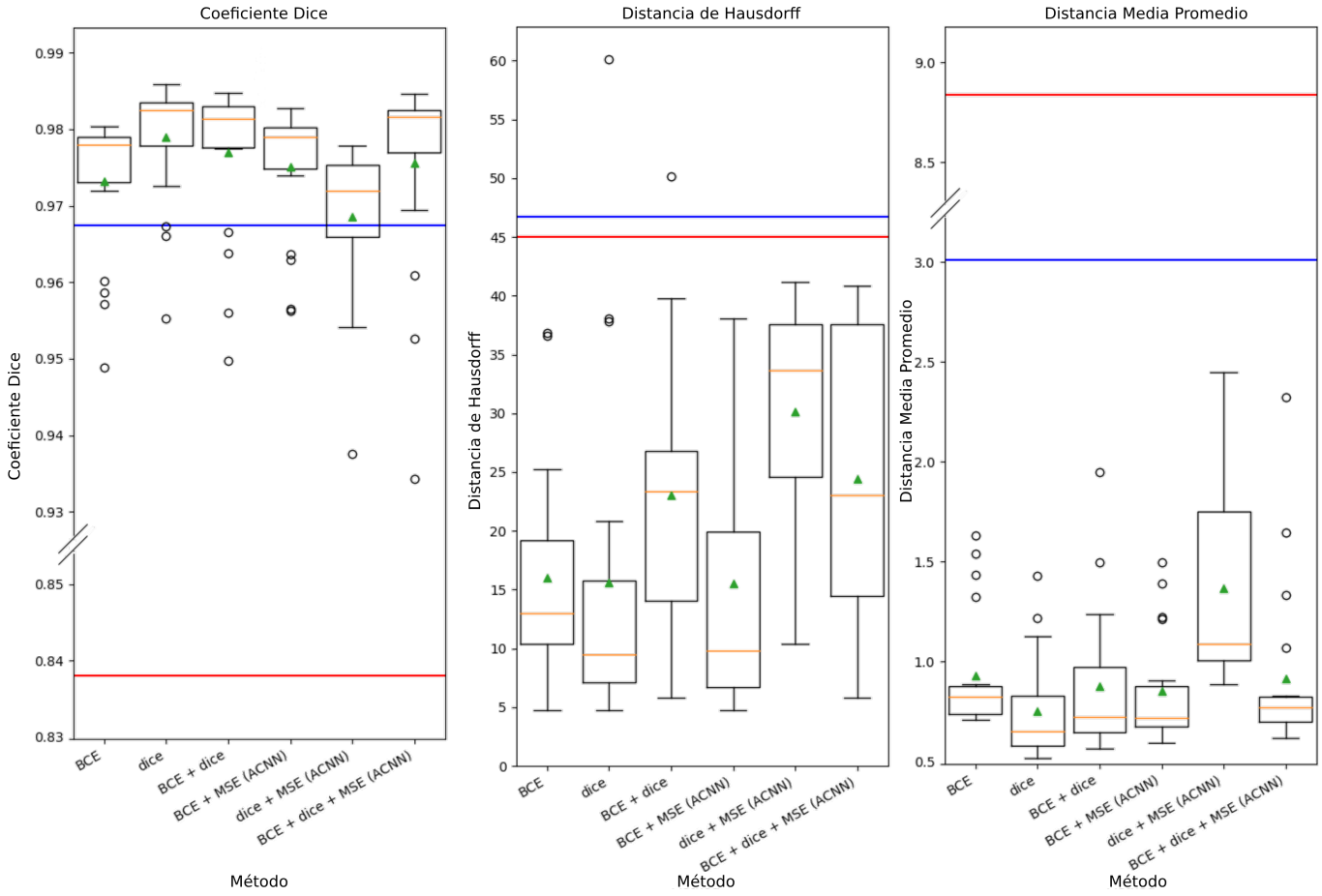

Figura 3.19: Diagramas de caja y bigote de las métricas calculadas en la partición de prueba, para los diferentes modelos propuestos. La línea roja representa la media de cada métrica obtenida con el método *Crecimiento de regiones* y la azul la media obtenida con *Conjuntos de Nivel*.

métodos en los datos de prueba incluyendo cabezas tanto con craniectomía como sin este procedimiento, mientras que en la Figura 3.20, se muestran estos mismos diagramas pero observando solamente las cabezas con craniectomía de la partición de prueba.

Haciendo un análisis de estos diagramas, se puede observar que todos los métodos basados en redes neuronales convolucionales resultaron significativamente mejores que los métodos del estado del arte basados en level-sets y crecimiento de regiones. Sin embargo, al comparar las distintas variantes con y sin restricciones anatómicas, la hipótesis inicial de que la regularización por el método ACNN mejoraría la calidad de las segmentaciones generadas respecto a otros métodos sin esta etapa no se cumplió. Esto se puede ver en la distancia de Hausdorff y en la distancia media promedio (observando aquellos que tie-

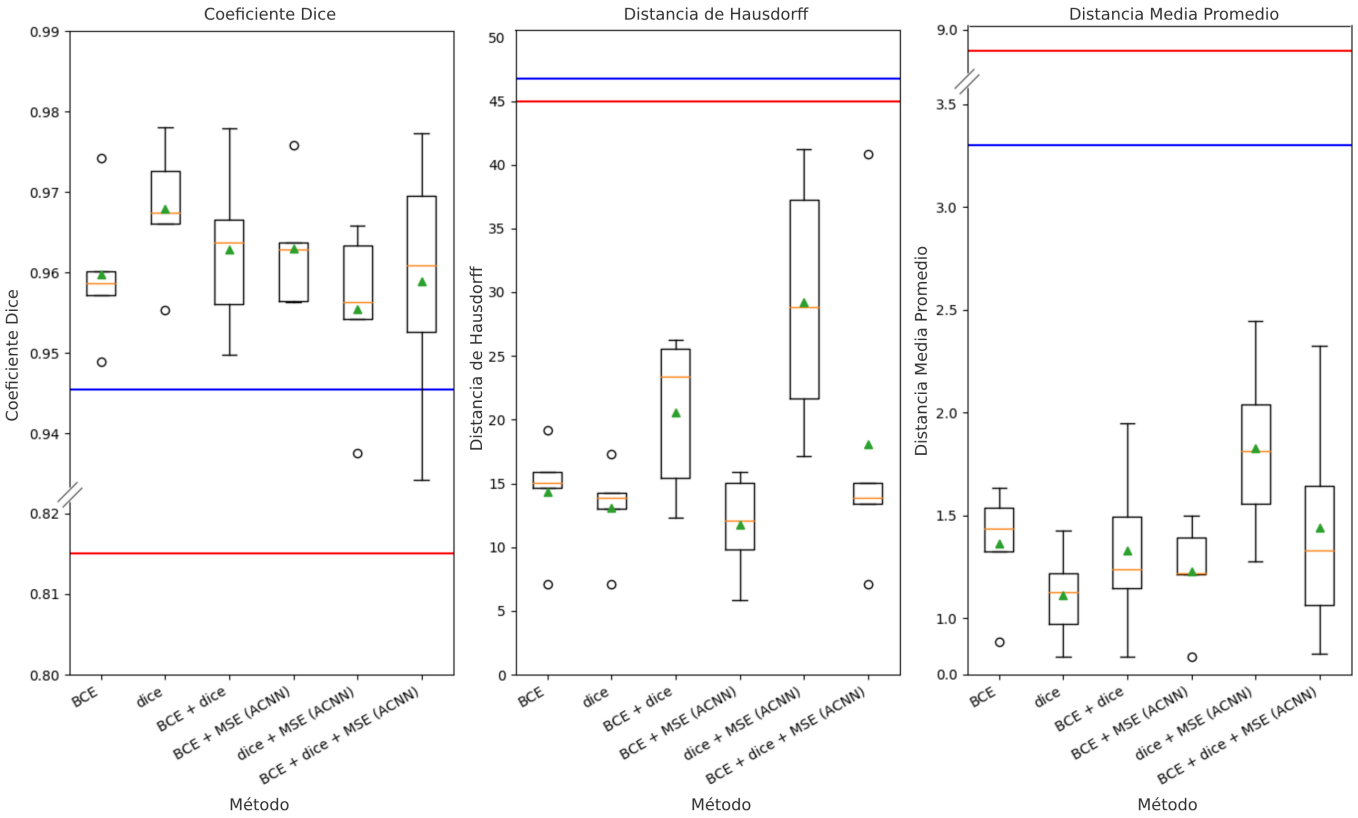

Figura 3.20: Diagramas de caja y bigote de las métricas calculadas en las imágenes con craniectomía de la partición de prueba, para los diferentes modelos propuestos. La línea roja representa la media de cada métrica obtenida con el método *Crecimiento de regiones* y la azul la media obtenida con *conjunto de nivel*.

nen el término MSE(ACNN)), en donde las medidas son similares o incluso mayores a los métodos sin regularización. Se intuye que esto ocurrió porque el autocodificador entrenado para efectuar la regularización fue entrenado con un dataset cuya partición de entrenamiento poseía una mayor proporción de segmentaciones de cabezas sin craniectomía, por lo que la penalización producida al incorporar la regularización obligaba a las segmentaciones a seguir formas más regulares, esto es, segmentaciones que se parezcan más a las de un cerebro sin craniectomía, evitando salirse demasiado de esa forma, y conservando las simetrías. Dado que en los casos sometidos a craniectomía el cerebro suele colapsar y deformarse respecto a su estado original, la incorporación de restricciones anatómicas resulta demasiado restrictiva disminuyendo así el rendimiento del método. En la Figura 3.21 puede observarse un ejemplo de comparación entre una segmentación obtenida por medio de una red convolucional con restricciones anatómicas, y el ground truth, ilustrando este problema.

Por otro lado, las segmentaciones producidas por los modelos que no incorporaban la regularización de ACNN (Dice o entropía cruzada binaria, o una combinación entre ambos) resultaron de una calidad mayor, ya que al no imponer condiciones sobre la forma que puede tener la estructura, los modelos aprendidos tendieron a generar segmentaciones lo más similares "pixel a píxel" posible, optimizando en cada caso una función de pérdida diferente. Una comparación de los distintos métodos en una imagen con craniectomía puede verse en la Figura 3.22.

Si se observa qué ocurrió en el caso particular de las imágenes con craniectomía (Figura 3.20) respecto al caso general, se puede notar que si bien la similaridad obtenida (coeficiente Dice) entre las segmentaciones es levemente inferior en promedio, el análisis realizado para el caso general es el mismo: en los métodos que incorporan restricciones anatómicas, el hecho de forzar las segmentaciones a seguir estructuras simétricas o regulares logró que éstas fueran más similares a las de una imagen de una cabeza sin craniectomía.

Si bien el método de regularización por restricciones anatómicas propuesto no funcionó como se esperaba para este caso, es posible que esta técnica cumpla con su objetivo si eventualmente se logra conseguir un dataset que contenga más imágenes con craniectomía

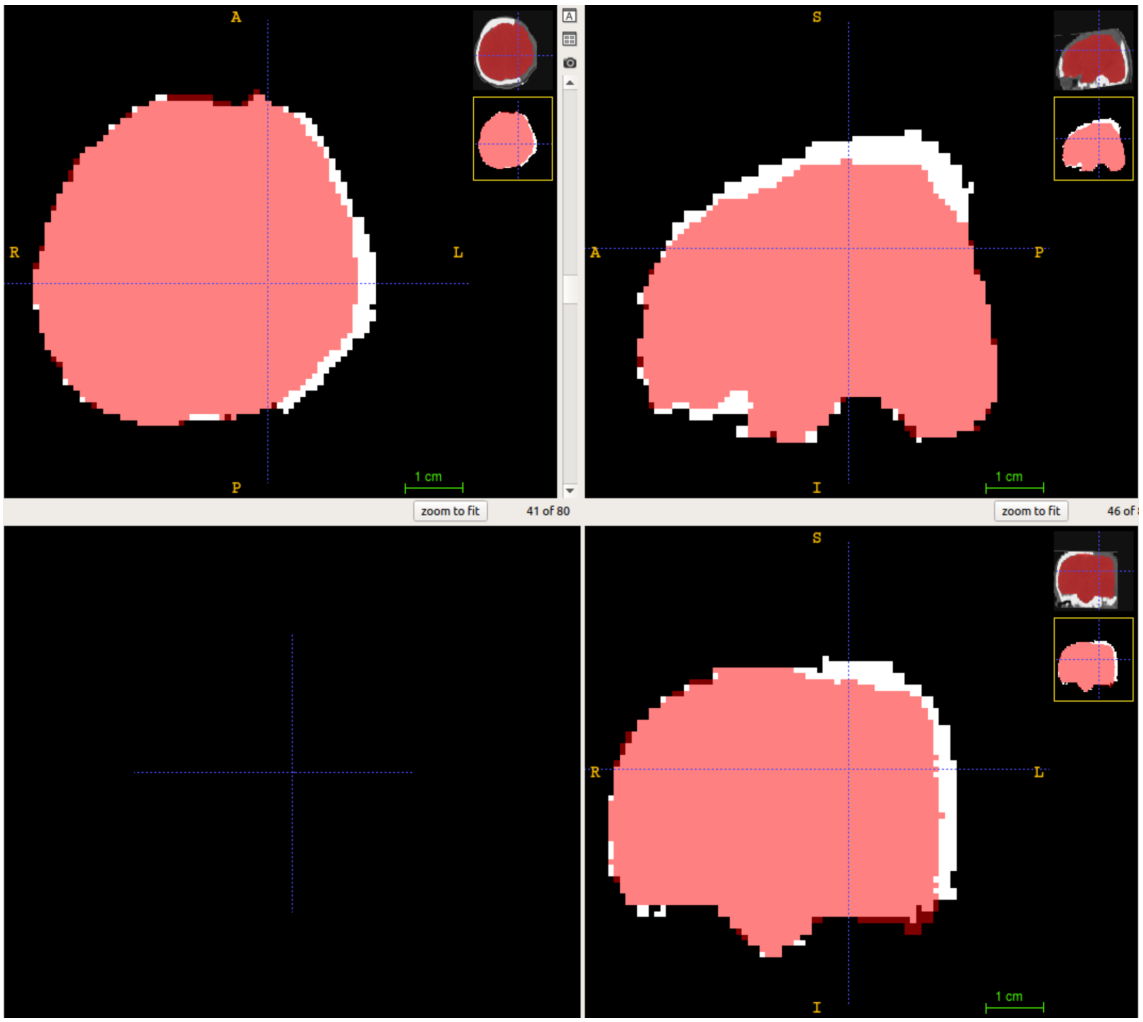

Figura 3.21: Comparación entre el ground truth de una cabeza con craniectomía (en blanco) y una segmentación generada (en rojo) utilizando la regularización del método ACNN. Se puede observar que la segmentación generada busca seguir formas más regulares.

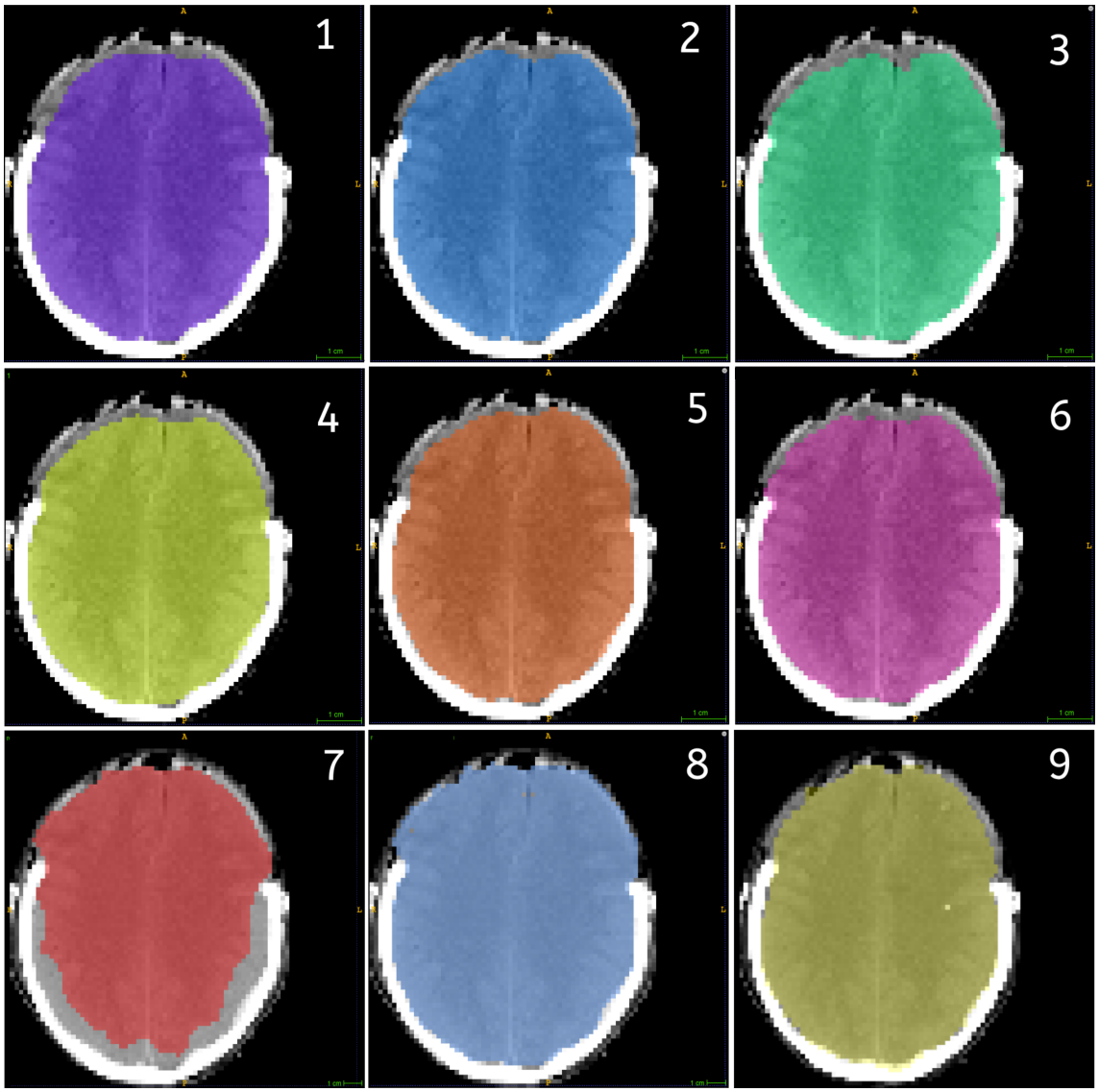

Figura 3.22: Comparación de los métodos implementados con el ground truth (9) en una imagen de un paciente con craniectomía. (1) BCE, (2) Dice, (3) BCE + Dice, (4) BCE + ACNN, (5) Dice + ACNN, (6) BCE + Dice + ACNN, (7) Crecimiento de regiones, (8) Conjuntos de Nivel. Si bien en este plano axial los métodos convolucionales otorgan un resultado similar, los demás métodos logran una segmentación defectuosa: por un lado Conjuntos de Nivel sobresegmenta tejidos que no se corresponden con el cerebro (en los bordes) mientras que crecimiento de regiones no logra segmentar correctamente la parte opuesta a la craniectomía.

para así poder entrenar con más imágenes de este tipo, o bien si se consigue aumentar los datos mediante transformaciones que deformen espacialmente las imágenes actuales del dataset, produciendo así más imágenes y segmentaciones de este tipo para aumentar la partición de entrenamiento. Debido a cuestiones de tiempo y el alcance de este proyecto, esta última alternativa no fue explorada exhaustivamente, por lo que es propuesta como trabajo futuro (ver Sección 4.2).

En base a los resultados obtenidos, y observando que el modelo que presentó mayor desempeño en todos los casos fue el entrenado utilizando la función de pérdida basada en Dice (considerando las tres medidas analizadas), se tomó como modelo entrenado para utilizar en la herramienta aquel que incorpora esta última función de pérdida.

Capítulo 4

Conclusiones

4.1. Conclusiones

La realización de este proyecto final permitió que el autor del mismo profundice en temáticas introducidas en materias como Inteligencia Computacional y Procesamiento Digital de Imágenes, y de esa manera tener un acercamiento a las tareas de investigación, ya que fue necesario explorar publicaciones y trabajos que abordan las temáticas de este proyecto.

Asimismo, fue necesario aprender tecnologías como Python, PyTorch, herramientas de manipulación de imágenes médicas como SimpleITK e itk-SNAP. También śe profundizó el manejo de sistemas operativos Linux, especialmente el manejo por consola mediante ssh y la ejecución de los diferentes componentes de este proyecto de manera remota y en GPU.

Asociado a la herramienta aquí presentada, se obtuvo una herramienta que cumple con los objetivos planteados en el anteproyecto, esto es, producir segmentaciones de cerebro en imágenes TAC cerebrales de pacientes con traumatismo de cráneo. Si bien la hipótesis de que la incorporación de restricciones anatómicas mejoraría el proceso de segmentación no se cumplió, se obtuvieron excelentes resultados a partir de una variación del modelo U-Net, que mejoran los alcanzados con métodos del estado del arte (tales como crecimiento de regiones y level-sets).

4.2. Trabajos Futuros

En esta sección se mencionarán diversos trabajos futuros que quedaron por fuera del alcance de este proyecto, y serán abordados en una etapa posterior.

4.2.1. Propuesta adicional: reconstrucción digital del cráneo

Al analizar las alternativas para la resolución de los problemas de los algoritmos tradicionales en la segmentación de imágenes de pacientes con craniectomía descompresiva (explicados en la Sección 3), se pensó como alternativa implementar un modelo convolucional neuronal el cual sea capaz de reconstruir la parte de cráneo faltante ocasionada por este procedimiento, para en una etapa posterior realizar la segmentación de los tejidos cerebrales, un problema que resulta menos complejo en imágenes que tienen el cráneo completo.

Para ello, junto al desarrollo de modelos convolucionales implementados en este proyecto, se implementó también un autocodificador, similar al que se utiliza en la etapa de regularización de ACNN, que, en lugar de ser entrenado con segmentaciones, fue entrenado sólo con imágenes de tomografía de pacientes sin craniectomía, aprendiendo de esta forma a reconstruir imágenes de TAC de cráneos completos. La hipótesis es que a partir de este modelo entrenado, si al mismo le es proporcionada una imagen con craniectomía, el autocodificador completará la región de cráneo faltante.

Si bien la arquitectura de la red y los hiperparámetros no fueron extensamente estudiados, se obtuvieron resultados preliminares muy prometedores. En la Figura 4.1 puede observarse cómo el cráneo es reconstruido al utilizar esta técnica. Como se puede observar, los bordes de la predicción no están lo suficientemente definidos, por lo que habría que estudiar estrategias (cambios en la arquitectura, función de pérdida o en la estrategia de entrenamiento) para evitar este problema.

En este contexto, la reconstrucción del cráneo en este tipo de imágenes resulta de utilidad no sólo como una forma de simplificar el problema de segmentación del cerebro, sino que también puede ser utilizado en otras tareas, tales como el cálculo del volumen, superficie

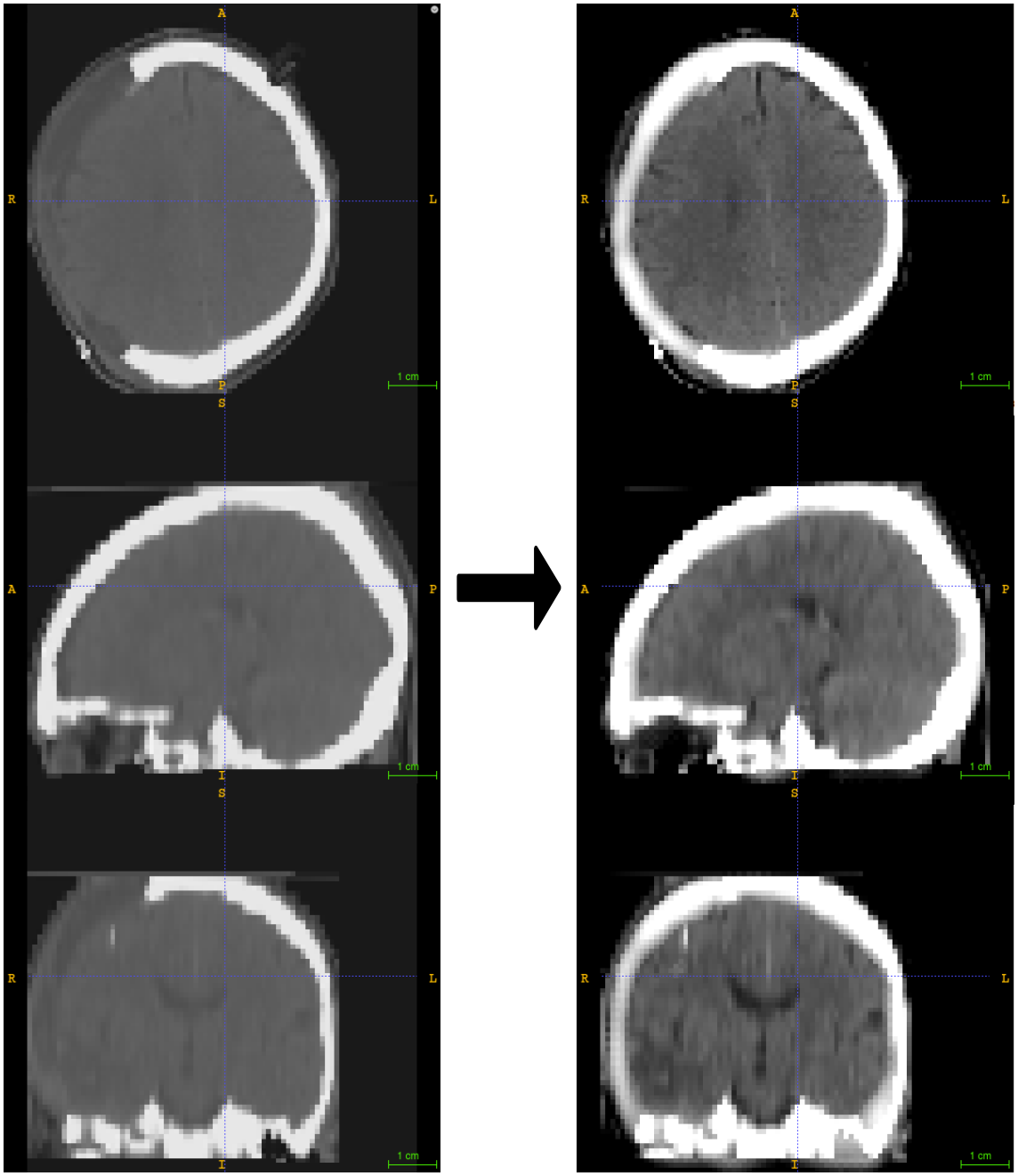

Figura 4.1: Reconstrucción del cráneo en una imagen de TAC de un paciente con craniectomía descompresiva. Izquierda: imagen original. Derecha: Imagen con cráneo reconstruido.

y otras medidas del trozo de cráneo extraído por el procedimiento quirúrgico. En la actualidad, estos indicadores son estimados a partir de modelos matemáticos extremadamente simples como el propuesto por Xiao *et al.* (2012). Contar con una herramienta para la estimación precisa de dichos indicadores podría resultar de mucho valor en la práctica e investigación clínica, donde el volumen de cráneo extraído (y su relación con el volumen del cerebro colapsado (Liao *et al.*, 2015)) puede ser incorporado a otras características del paciente en la construcción de modelos para predecir la evolución de la patología.

4.2.2. Tareas asociadas a la herramienta desarrollada

Algunas características de la herramienta que no fueron implementadas debido al alcance de este proyecto y que lo serán en una etapa posterior son:

- Etapa de posprocesamiento: Una vez que se logran las segmentaciones, estudiar alternativas para volver a las dimensiones originales de la imagen, de manera que los usuarios de la misma no dependan de las propiedades impuestas en la etapa de preprocesamiento.
- Aumentación de datos para mejorar la regularización por restricciones anatómicas: Tal como se mencionó en la Sección 3.7.2, queda pendiente estudiar e implementar distintas técnicas de aumentación de datos para el dataset actual, logrando así mejorar la regularización obtenida experimentalmente para este proyecto.

Bibliografía

Alpaydin, E. (2014). *Introduction to Machine Learning*. Adaptive Computation and Machine Learning series. MIT Press.

ASNR (2013). Traumatic brain injury (tbi) and concussion. Disponible en https://www.asnr.org/patientinfo/conditions/tbi.shtml. Fecha de consulta: 27/03/2018.

Badrinarayanan, V., Kendall, A., y Cipolla, R. (2017). Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Beucher, S. (1994). Watershed, hierarchical segmentation and waterfall algorithm. En *Mathematical morphology and its applications to image processing*, pp. 69–76. Springer.

Boureau, Y., Ponce, J., y LeCun, Y. (2010). A theoretical analysis of feature pooling in visual recognition. En *International Conference on Machine Learning*, pp. 111–118.

Calvo, N. (2017). Intersecciones y ordenamiento espacial. Apuntes de la cátedra Computación Gráfica. Disponible en el sitio web que se indica en: `http://fich.unl.edu.ar/planificaciones/planificacion.php?id=344&carrera=3.`.

Caverly, R. H. (2015). Mri fundamentals: Rf aspects of magnetic resonance imaging (mri). *IEEE Microwave Magazine*, 16(6):20–33.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314.

De Silva, M. J., Roberts, I., Perel, P., Edwards, P., Kenward, M. G., Fernandes, J., Shakur, H., y Patel, V. (2009). Patient outcome after traumatic brain injury in high-, middle-

and low-income countries: analysis of data on 8927 patients in 46 countries. *International Journal of Epidemiology*, 38(2):452–458.

Díaz, I. R. R. (2014). Imágenes diagnósticas: Conceptos y generalidades. *Revista de la Facultad de Ciencias Médicas*.

Engelbrecht, A. (2007). *Computational Intelligence: An Introduction*. Wiley.

Freyschlag, C. F., Gruber, R., Bauer, M., Grams, A. E., y Thomé, C. (2018). Routine postoperative computed tomography is not helpful after elective craniotomy. *World Neurosurgery*.

Gabrani, M. y Tretiak, O. J. (1996). Elastic transformations. En *Conference Record of The Thirtieth Asilomar Conference on Signals, Systems and Computers*, volumen 1, pp. 501–505 vol.1.

Galgano, M., Toshkezi, G., Qiu, X., Russell, T., Chin, L., y Zhao, L.-R. (2017). Traumatic brain injury: Current treatment strategies and future endeavors. *Cell Transplantation*, 26(7):1118–1130. PMID: 28933211.

Goldman, L. W. (2007). Principles of CT and CT technology. *Journal of Nuclear Medicine Technology*, 35(3):115–128.

Gonzalez, R. C. y Woods, R. E. (2006). *Digital Image Processing (3rd Edition)*. Pearson/Prentice Hall, Upper Saddle River, NJ, USA.

Goodfellow, I., Bengio, Y., y Courville, A. (2016). *Deep Learning*. Adaptive Computation and Machine Learning series. MIT Press.

Herzog, P. y Rieger, C. T. (2004). Risk of cancer from diagnostic x-rays. *The Lancet*, 363(9406):340–341.

Hubel, D. H. y Wiesel, T. N. (1959). Receptive fields of single neurones in the cats striate cortex. *The Journal of Physiology*, 148(3):574–591.

Ibanez, L., Schroeder, W., Ng, L., y Cates, J. (2003). *The ITK Software Guide*. Kitware, Inc., first edición. ISBN 1-930934-10-6.

Ioffe, S. y Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167.

Kalavathi, P. y Prasath, V. B. (2016). Methods on Skull Stripping of MRI Head Scan Images-a Review. *J Digit Imaging*, 29(3):365–379.

Kirk, M. (2017). *Thoughtful Machine Learning with Python: A Test-Driven Approach*. O'Reilly Media.

Kohli, M. y Rosenman, M. (2013). Indiana university chest x-ray collection. Disponible en `https://openi.nlm.nih.gov/detailedresult.php?img=CXR111_IM-0076-1001&req=4`.

Kumar, R. (2019). *Machine Learning Quick Reference: Quick and essential machine learning hacks for training smart data models*. Packt Publishing.

LeCun, Y., Bengio, Y., y Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

LeCun, Y., Haffner, P., Bottou, L., y Bengio, Y. (1999). Object recognition with gradient-based learning. En *Shape, Contour and Grouping in Computer Vision*, pp. 319–, London, UK, UK. Springer-Verlag.

Lee, B. y Newberg, A. (2005). Neuroimaging in traumatic brain imaging. *NeuroRx*, 2(2):372–383.

Liao, C.-C., Tsai, Y.-H., Chen, Y.-L., Huang, K.-C., Chiang, I.-J., Wong, J.-M., y Xiao, F. (2015). Transcalvarial brain herniation volume after decompressive craniectomy is the difference between two spherical caps. *Medical Hypotheses*, 84(3):183 – 188.

Litjens, G., Kooi, T., Bejnordi, B. E., Setio, A., Ciompi, F., Ghafoorian, M., van der Laak, J. A., van Ginneken, B., y Sánchez, C. I. (2017). A survey on deep learning in medical image analysis. *Medical Image Analysis*, 42:60 – 88.

Liu, Q., Fu, M., Jiang, H., y Gong, X. (2018). Volumetric densely dilated spatial pooling connets for prostate segmentation. *CoRR*, abs/1801.10517.

Lowekamp, B., Chen, D., Ibanez, L., y Blezek, D. (2013). The design of simpleitk. *Frontiers in Neuroinformatics*, 7:45.

Maas, A. I. R. y Menon, D. K. e. a. (2017). Traumatic brain injury: integrated approaches to improve prevention, clinical care, and research. *The Lancet Neurology*, 16(12):987–1048.

Marstal, K., Berendsen, F., Staring, M., y Klein, S. (2016). Simpleelastix: A user-friendly, multi-lingual library for medical image registration. En *2016 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 574–582.

McCorduck, P. (2004). *Machines who think: a personal inquiry into the history and prospects of artificial intelligence*. Ak Peters Series. A.K. Peters.

Menon, D. K. y Zahed, C. (2009). Prediction of outcome in severe traumatic brain injury. *Current Opinion in Critical Care*, 15(5):437–441.

Migdal, P. (2018). Keras or pytorch as your first deep learning framework. Disponible en `https://deepsense.ai/keras-or-pytorch/`.

Monostori, L. (2003). Ai and machine learning techniques for managing complexity, changes and uncertainties in manufacturing. *Engineering applications of artificial intelligence*, 16(4):277–291.

Muschelli, J., Ullman, N. L., Mould, W. A., Vespa, P., Hanley, D. F., y Crainiceanu, C. M. (2015). Validated automatic brain extraction of head CT images. *Neuroimage*, 114:379–385.

Narvaez, M., Merello, E., Toribio, C., y Jose, B. (2009). Diagnóstico por la imagen (2009): Estudio de prospectiva. *Fundación OPTI Observatorio de Prospectiva Tecnológica Industrial y FENIN, Federación de Empresas de Tecnología Sanitaria, Madrid*.

Oktay, O., Ferrante, E., Kamnitsas, K., Heinrich, M. P., Bai, W., Caballero, J., Guerrero, R., Cook, S. A., de Marvao, A., Dawes, T., O'Regan, D. P., Kainz, B., Glocker, B., y Rueckert, D. (2017). Anatomically constrained neural networks (ACNN): application to cardiac image enhancement and segmentation. *CoRR*, abs/1705.08302.

Osher, S. y Paragios, N. (2003). *Geometric level set methods in imaging, vision, and graphics*. Springer Science & Business Media.

Otsu, N. (1979). A threshold selection method from gray-level histograms. *Systems, Man and Cybernetics, IEEE Transactions on.*

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., y Lerer, A. (2017). Automatic differentiation in pytorch.

Patel, A., van Ginneken, B., Meijer, F. J. A., van Dijk, E. J., Prokop, M., y Manniesing, R. (2017). Robust cranial cavity segmentation in CT and CT perfusion images of trauma and suspected stroke patients. *Medical Image Analysis*, 36:216–228.

Patravali, J., Jain, S., y Chilamkurthy, S. (2017). 2d-3d fully convolutional neural networks for cardiac MR segmentation. *CoRR*, abs/1707.09813.

Preim, B. y Bartz, D. (2007). *Visualization in Medicine: Theory, Algorithms, and Applications.* The Morgan Kaufmann Series in Computer Graphics. Elsevier Science.

Raudys, S. (1998). Evolution and generalization of a single neurone: I. single-layer perceptron as seven statistical classifiers. *Neural networks : the official journal of the International Neural Network Society*, 11(2):283—296.

Rawat, V., Jain, A., y Shrimali, V. (2018). Automated Techniques for the Interpretation of Fetal Abnormalities: A Review. *Appl Bionics Biomech*, 2018:6452050.

Ren, X., Luo, Y., Gao, N., Niu, H., y Tang, J. (2016). Common ultrasound and contrast-enhanced ultrasonography in the diagnosis of hepatic artery pseudoaneurysm after liver transplantation. *Exp Ther Med*, 12(2):1029–1033.

Ronneberger, O., Fischer, P., y Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597.

Russell, S. y Norvig, P. (2002). *Artificial Intelligence: A Modern Approach.* Prentice Hall.

Sarangi, S. y Sharma, P. (2018). *Artificial Intelligence: Evolution, Ethics and Public Policy.* Taylor & Francis.

Sneeringer, L. (2015). *Professional Python.* Wiley.

Subramanian, V. y Agarwal, M. (2018). *Deep learning with PyTorch: a practical approach to building neural network models using PyTorch.* Packt Publishing.

Sun, D., Sui, P., Zhang, W., Zhang, L., y Xu, H. (2018). Cerebral air embolism during percutaneous computed tomography scan-guided liver biopsy. *Journal of Cancer Research and Therapeutics*, 14(7):1650–1654.

Szeliski, R. (2010). *Computer Vision: Algorithms and Applications*. Texts in Computer Science. Springer London.

Taha, A. A. y Hanbury, A. (2015). Metrics for evaluating 3D medical image segmentation: analysis, selection, and tool. *BMC Med Imaging*, 15:29.

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., y Manzagol, P.-A. (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. Mach. Learn. Res.*, 11:3371–3408.

Wolbarst, A. y Cook, G. (1999). *Looking Within: How X-Ray, CT, MRI, Ultrasound, and Other Medical Images Are Created, and How They Help Physicians Save Lives*. University of California Press.

Wyss, P. O., Hock, A., y Kollias, S. (2017). The application of human spinal cord magnetic resonance spectroscopy to clinical studies: A review. *Seminars in Ultrasound, CT and MRI*, 38(2):153 – 162. Spinal Cord Imaging, Part 2.

Xiao, F., Chiang, I.-J., Hsieh, T. M.-H., Huang, K.-C., Tsai, Y.-H., Wong, J.-M., Ting, H.-W., y Liao, C.-C. (2012). Estimating postoperative skull defect volume from ct images using the abc method. *Clinical Neurology and Neurosurgery*, 114(3):205 – 210.

Yang, Z. y Karahoca, A. (2006). An anomaly intrusion detection approach using cellular neural networks. En Levi, A., Savaş, E., Yenigün, H., Balcısoy, S., y Saygın, Y., editores, *Computer and Information Sciences – ISCIS 2006*, pp. 908–917, Berlin, Heidelberg. Springer Berlin Heidelberg.

Yuh, E. L., Cooper, S. R., Ferguson, A. R., y Manley, G. T. (2012). Quantitative CT improves outcome prediction in acute traumatic brain injury. *J. Neurotrauma*, 29(5):735–746.