UNIVERSITAT DE BARCELONA

Trabajo final de grado

GRADO DE INFORMÁTICA

Facultat de Matemàtiques i Informàtica
Universitat de Barcelona

Plugin para segmentación automática en resonáncia magnética de mama mediante Mango Viewer

Autor: Rubén Pacheco Cerro

Director:	Dr. Oliver Díaz Montesdeoca
Realizado en:	Departament de Matemàtiques i Informàtica
Barcelona,	10 de junio de 2024

Resumen

Este trabajo aborda el desarrollo de un plugin para la aplicación Mango Viewer destinado a facilitar el proceso de anotación en imágenes de resonancias magnéticas de mama. Durante el desarrollo del proyecto, se llevó a cabo un estudio exhaustivo de los métodos actuales de segmentación de imágenes y de las herramientas disponibles para el trabajo con imágenes médicas.

Como resultado, se ha creado una herramienta que puede integrarse directamente como un plugin estándar en Mango Viewer. Al activarse, esta herramienta inicia un proceso que culmina en la predicción de una Región de Interés (ROI), la cual se implementa en la imagen seleccionada para su posterior edición y estudio.

Abstract

This work addresses the development of a plugin for the Mango Viewer application to facilitate the annotation process in breast MRI images. During the development of the project, a study was conducted on current image segmentation methods and tools available for working with medical images.

The result is a tool that can be directly integrated like a standard plug-in into Mango Viewer. When activated, this tool initiates a process that culminates in the prediction of a Region of Interest (ROI), which is then implemented in the selected image for further editing and studying.

Resum

Aquest treball adreça el desenvolupament d'un plugin per a l'aplicació Mango Viewer destinat a facilitar el procés d'anotació en imatges de ressonàncies magnètiques de mama. Durant el desenvolupament del projecte, d'ha dut a terme un estudi dels mètodes actuals de segmentació d'imatges i de les eines disponibles per al treball am imatges mèdiques. Com a resultat, s'ha creat una eina que pot integrar-se directament com un plugin estàndard en Mango Viewer. Al activar-se, aquesta eina inicia un procés que culmina en la predicció d'una Regió d'Interès (ROI) la qual s'implementa en la imatge seleccionada per a la seva edició i estudi posterior.

Agradecimientos

Quiero agradecer a mi tutor Oliver por darme la oportunidad de hacer este trabajo y damre soporte durante toda la realización de este. Su ayuda ha sido inestimable y su constante ayuda imprescindible.
También agradecer a Smitri por ayudarme a poder hacer el proyecto y a avanzar con este al igual que a Ioannis y Katerina del grupo FORTH por el apoyo que me han dado a la hora del desarrollo.

Por último quiero agradecer a mi familia por apoyarme durante todo el proceso al igual que a mi amiga Sara por el apoyo moral que me han brindado. Ha sido inestimable y la causa de que este trabajo haya salido adelante.

Índice general

Capítulo 1

Introducción

El concepto de inteligencia artificial (IA) fue acuñado el año 1956 durante la conferencia de Dartmouth (EEUU) [4], sin embargo no ha sido sinó hasta recientemente que las tecnologías de IA han empezado a demostrar su enorme potencial, gracias a técnicas como las redes neuronales o aplicaciones como el reconocimiento de imágenes. De forma resumida, una IA es un concepto amplio que se refiere a un conjunto de algoritmos informáticos que pueden realizar tareas similares al comportamiento humano, como puede ser el aprendizaje o el reconocimiento de objetos. Por otro lado, el aprendizaje automático (o machine learning) es una subárea de la IA, como puede verse en la figura 1.1[1], que abarca las máquinas con capacidad para aprender una tarea determinada aprendiendo de experiencias o datos anteriores, sin necesidad de programación específica.

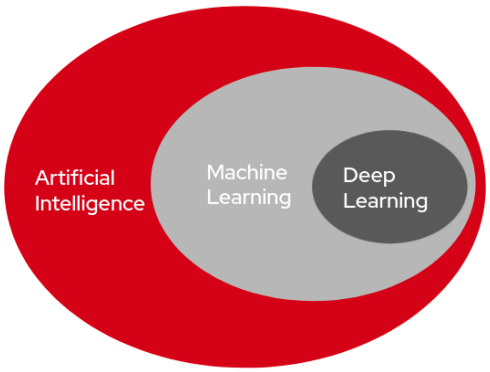

Figura 1.1: Diagrama de la relación entre Inteligencia Artificial, *machine learning* y *deep learning* [41].

En los últimos años y gracias a los avances en el campo del aprendizaje automático, la IA se ha vuelto una herramienta muy prometedora en el sector médico [33], particularmente en el ámbito del radiodiagnóstico y la imágen médica [12]. La

[1]El deep learning es una rama del aprendizaje automático que utiliza redes neuronales profundas para aprender y representar de manera jerárquica patrones complejos y abstractos a partir de datos crudos. Sin embargo este escapa al alcance del proyecto y por lo tanto no será mencionado

rapidez, comodidad y el ahorro de recursos que se gana utilizando métodos que se apoyan en el aprendizaje automático para aislar y segmentar regiones de interés para los especialistas es lo más atractivo del uso de dichas técnicas, sin embargo el uso de éstas acarrea varios problemas consigo.

El proceso actual para obtener *datasets* válidos para el entrenamiento y validación de modelos de aprendizaje automático, redes neuronales y demás arquitecturas de IA es bastante lento, costoso y requiere de un personal especializado y experto que permita garantizar la calidad y precisión de los datos anotados. La creación de estos conjuntos de datos implica no solo la recopilación de imágenes médicas de alta calidad, sino también su correcta etiquetación por parte de radiólogos experimentados, lo que asegura que los algoritmos se entrenen con información fiable y representativa de los escenarios clínicos reales. Este proceso es fundamental para el desarrollo de modelos precisos y robustos, pero también representa una barrera significativa para la rápida implementación y escalabilidad de soluciones de IA en el ámbito del radiodiagnóstico entre otras cosas lo cual puede llevar a una reducción significativa en el uso de este tipo de herramientas en entornos clínicos[45].

Sin embargo, a pesar de las dificultades que existen a la hora de obtener los *datasets*, el uso de la IA en el radiodiagnóstico se trata de un avance importante y se espera que estas tecnologías ayuden a mejorar la calidad y reducir los costes de los procesos médicos que las involucren. Es por esto que surge la necesidad de proporcionar herramientas a los expertos que les permitan tanto generar con una mayor agilidad y precisión los *datasets* necesarios para el entrenamiento de los modelos como aplicar estas tecnologías de manera efectiva en su práctica diaria. Estas herramientas no solo deben facilitar la creación de datos etiquetados de alta calidad al optimizar el proceso de anotación, sino también integrarse de manera fluida en los flujos de trabajo clínicos existentes, permitiendo a los radiólogos aprovechar al máximo las capacidades de la IA sin requerir una formación extensa en informática. Con la implementación de tales soluciones, se podría acelerar significativamente la adopción de la IA en el radiodiagnóstico, mejorando así la eficiencia, precisión y accesibilidad de los diagnósticos médicos.

1.1. Motivación

A día de hoy, el cáncer es la mayor causa de mortalidad a nivel mundial [31] y, dentro de este grupo de enfermedades, el cáncer de mama es a la vez el diagnóstico más común y la principal causa de mortalidad en mujeres [40] de todo el mundo. Sin embargo gracias en gran parte a las mejoras del tratamiento y la globalización del *screening* [2] el número de fallecidos por cáncer de mama ido disminuyendo considerablemente a lo largo de los últimos años. Los programas de *screening* o cribado poblacional consisten en ïdentificar en la población general a personas asintomáticas afectadas por una enfermedad o anomalía que hasta entonces pasaba desapercibida mediante test diagnósticos, exámenes u otras técnicas de aplicación rápida"[32] como lo pueden ser las mamografías en el caso del cáncer de mama; es precisamente en este aspecto en lo que se centrará este trabajo.

El proceso de detección precoz del cáncer requiere de un especialista que pueda entender y analizar los resultados de una prueba para identificar posibles áreas de preocupación. Sin embargo, el análisis manual de imágenes de algunas técnicas de imagen, como lo pueden ser las mamografías o las resonancias magnéticas, es un proceso laborioso, al requerir la revisión de grandes volúmenes de datos tridimensionales, y sujeto a variabilidad entre los radiólogos, lo que puede resultar en diagnósticos inconsistentes y en desacuerdos incluso entre los propios expertos [38]. La segmentación precisa de las imágenes es un paso crucial para identificar y delimitar las áreas sospechosas, pero hacerlo manualmente puede ser laborioso y requerir de mucho tiempo, lo cual puede dar pie a que aparezcan complicaciones en el tiempo en el que toadvía se están calibrando los resultados si el volumen de pacientes es lo suficientemente amplio.

En este contexto, la implementación de tecnologías de IA ofrece una oportunidad para automatizar y mejorar la precisión de la segmentación de imágenes médicas, así como de acelerar el proceso necesario para obtener un resultado. En la figura 1.2 se puede ver el proceso hasta la anotación incluyendo una IA

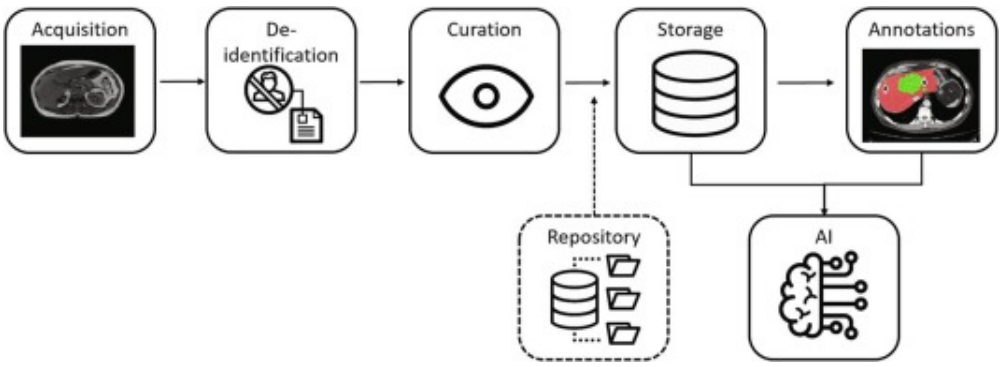

Figura 1.2: Pipeline del proceso [11]

La motivación principal detrás de este proyecto radica en la necesidad urgente de desarrollar herramientas avanzadas que asistan a los profesionales de la salud en la anotación de imágenes. De esta forma, dichos datos pueden ayudar a desarollar y validar algoritmos de IA que mejoren el diagnóstico temprano y tratamientos más eficientes para combatir el cáncer de mama. La integración de un modelo de aprendizaje de *machine learning* en una plataforma de visualización de imágenes médicas facilitaría y aceleraría el análisis de estas.

1.2. Planificación

1.2.1. Definición de objetivos

El objetivo principal de este trabajo es **desarrollar una herramienta de segmentación semántica para imágenes de resonancias magnéticas de la mama e implementarla en la aplicación Mango Viewer, un software espe-**

cializado en el estudio y análisis de imágenes médicas, en forma de plugin. Dicho plugin dará soporte en la anotación de imágenes a radiólogos dentro del proyecto europeo RadioVal (https://radioval.eu/), un proyecto de investigación internacional y multinstitucional cuyo objetivo es desarrollar e implementar una validación integral de soluciones de IA en el tratamiento del cáncer de mama. Este desarrollo pretende añadir un paso adicional al proceso de anotación en el que la IA será responsable de proporcionar un primer esbozo de las regiones de interés, facilitando así la labor del radiólogo encargado del análisis y diagnóstico.

Uno de los objetivos secundarios es la **reducción del tiempo necesario para realizar anotaciones en regiones de interés** (también referida como *ROI* por sus siglas en inglés) en cualquier resonancia magnética. Al automatizar la primera fase del proceso de segmentación, se espera que los radiólogos puedan concentrarse más en la revisión y ajuste final de las imágenes, en lugar de dedicar tiempo a delinear manualmente las áreas de interés. Esta mejora en la eficiencia no solo pretende aumentar la productividad, sino que también tiene como fin permitir a los profesionales de la salud atender a un mayor número de pacientes sin perder calidad en esta.

El objetivo secundario es la **integración de esta herramienta en un entorno clínico real**. Para lograr esto, la herramienta debe ser compatible con las infraestructuras tecnológicas existentes en hospitales y clínicas. La implementación debe considerar aspectos como la interoperabilidad con otros sistemas de información médica, la seguridad de los datos y la facilidad de instalación y mantenimiento del plugin. Este proceso de integración es crucial para asegurar que la herramienta sea adoptada y utilizada de manera efectiva en el ámbito clínico.

El correcto funcionamiento en entornos clínicos es el objetivo final del proyecto que envuelve a este trabajo (RadioVal), y por lo tanto uno de los objetivos que se han tenido en este trabajo ha sido que el plugin sea lo más próximo a un *plug & play*

1.2.2. Planificación

Al iniciar, el tutor de este proyecto propuso la creación de dos *Gantt chart* uno al inicio del trabajo y otro al final. Un *Gantt chart* es una representación visual del cronograma de un proyecto. Utiliza barras horizontales para mostrar las tareas, su duración y la secuencia en la que deben completarse.

El objetivo de contar con dos *Gannt Chart* ha sido poder ver como ha cambiado el foco del proyecto así como poder tener en cuenta cuales han sido los puntos más problemáticos del proceso para, en un futuro en caso de repetir algo similar, poder enfocar más recursos o tiempo en esa área específica y así mejorar la eficiencia y eficacia con la que trabajo

En la figura 1.3 se puede observar la planificación que se hizo del trabajo la primera semana de este. Varias de las tareas se pusieron debido al desconocimiento del proceso y los pasos necesarios para realizar un trabajo así, además de no saber exactamente con que recursos y ayudas contaría a lo largo del proceso.

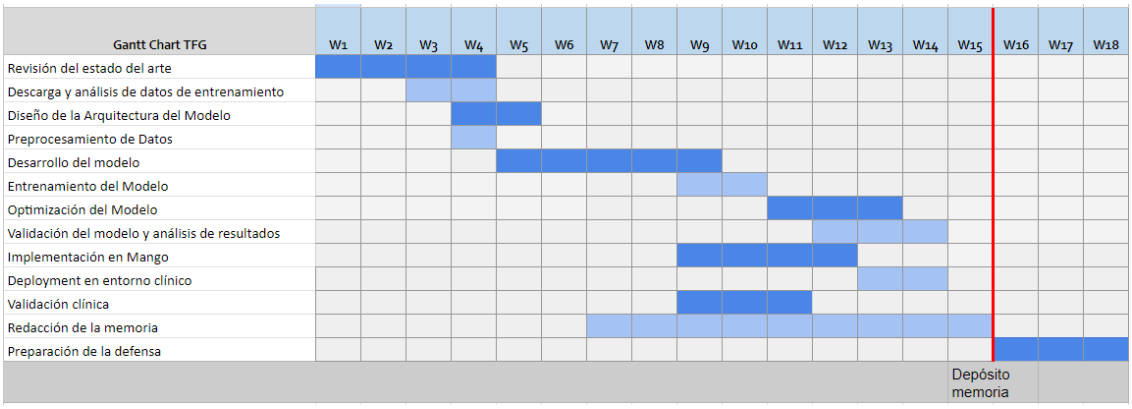

Figura 1.3: *Gannt chart* inicial del trabajo

En la figura 1.4 se puede ver el cronograma final, y por lo tanto real, de todo el proyecto. Se puede apreciar que todo lo relacionado con el entrenamiento de modelo de IA fue lo que más ralentizó el proyecto. El motivo de esto se explica en la sección 5.1 de la memoría.

Además de los problemas que surgieron relacionados con el modelo de IA, muchas de las tareas relacionadas con este resultaron ser al final o bien innecesarias o bien mucho más cortas de lo que en un principio deberían haber sido. El motivo de esto se explica en las secciones 4.2 y 3.2.2 pero el resultado final fue que varias de las tareas que, en un principio, parecían ser importantes terminaron siendo o bien innecesarias para el proyecto o bien se tuvieron que cortar por falta de tiempo o por ser innecesarios para el proyecto final como puede ser el *deployment* en un entorno clínico de la aplicación.

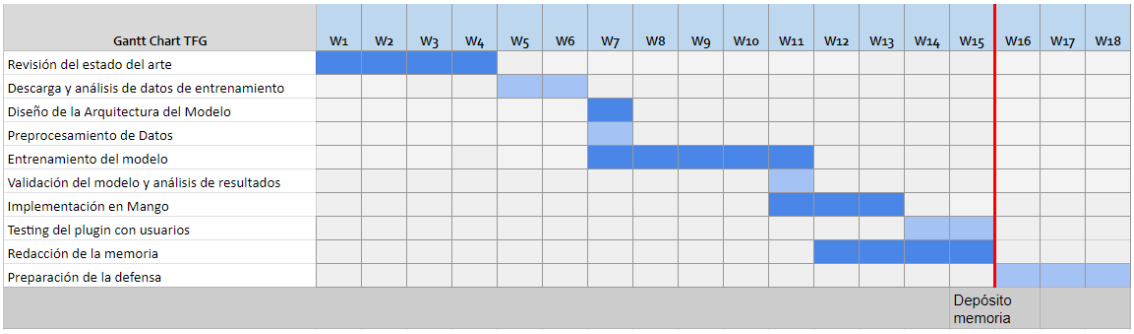

Figura 1.4: *Gantt chart* final del trabajo.

Además del cronograma *Gantt chart*, otra de las partes importantes para la organización del trabajo han sido las reuniones semanales con el tutor. Una vez a la semana se hacía una reunión para hablar con el tutor sobre lo que se había avanzado durante la semana, cualquier duda que hubiera en ese punto del proyecto así como algunas reuniones en las que algunos expertos que han ayudado a desarrollar el proyecto estaban presentes para poder así hacer preguntas o poner en común ideas y sugerencias que contribuyeran al progreso del proyecto.

Capítulo 2

Estado del arte

En este capítulo, se analizará el estado actual de la segmentación, abordando tanto los métodos convencionales como las últimas herramientas de segmentación semántica basadas en modelos de aprendizaje automático. Además, se estudiarán también las herramientas actuales para el estudio de imágenes y sus características.

2.1. Métodos de segmentación

A lo largo de esta sección, se realizará una revisión de los diversos métodos de segmentación de imágenes que han surgido a lo largo de los años. Se detallarán sus principales características y se explicarán los fundamentos teóricos que sustentan su funcionamiento. Primero, se presentarán los métodos que he decidido denominar como çonvencionales". Posteriormente, se explicarán los nuevos métodos de segmentación semántica basados en modelos de *machine learning*, destacando sus ventajas en comparación con los métodos más tradicionales.
La segmentación de imágenes es un proceso fundamental en el análisis de imágenes que consiste en dividir una imagen en partes o regiones significativas. Estas regiones pueden representar objetos, partes de objetos u otras entidades de interés dentro de la imagen. La segmentación permite la extracción de datos específicos de cada una de estas regiones, facilitando su análisis y procesamiento posterior y es fundamental a la hora de analizar imágenes y extraer información de ellas [**EstudioSegmentacion**].
En la imagen 2.1 se puede ver un ejemplo de segmentación en una imagen, dónde diferentes elementos de una escena (coches, peatones, semáforos, carretera, etc) son identificados por un algoritmo y resaltados en diferentes colores.

Figura 2.1: Ejemplo de segmentación de diversos elementos. Cada color representa una clase (coches, peatones, la carretera, semáforos,...) [8]

La segmentación semántica es una técnica avanzada dentro de la segmentación de imágenes tradicional en la que se asigna una etiqueta específica y significativa a cada píxel de una imagen. A diferencia de la segmentación tradicional, que solo divide la imagen en regiones, la segmentación semántica proporciona una comprensión más profunda al identificar y etiquetar con precisión cada elemento (o clase) dentro de la imagen. Esto es crucial para aplicaciones que requieren un reconocimiento detallado y preciso de objetos dentro de una imagen[14].

2.2. Herramientas de segmentación semántica

A la hora de validar la eficacia de los modelos de IA, la anotación de datos es el proceso más importante ya que define en gran parte la calidad, robustez y precisión del resultado final. En términos simples, la anotación de datos implica etiquetar la información para hacerla comprensible para las máquinas. Al anotar los datos, se proporciona contexto y significado a la información "en bruto", permitiendo que los modelos de aprendizaje automático reconozcan patrones, hagan predicciones y realicen tareas complejas. Los ordenadores carecen de la capacidad inherente para procesar y comprender la información visual como lo hacen los humanos, por lo tanto, la anotación de datos sirve como puente entre los datos y los algoritmos de IA, permitiendo que las máquinas hagan predicciones y tomen decisiones informadas.

Para llevar a cabo el proceso de la anotación los expertos suelen contar con herramientas especializadas de visualización que permiten la fácil edición de las imágenes. La elección de la herramienta adecuada para el proceso de anotación es crucial para determinar la eficiencia del trabajo. Una herramienta muy sencilla puede facilitar la anotación, pero podría presentar problemas para el usuario (en este caso, el radiólogo encargado de las anotaciones) debido a la sobre-simplificación de

algunos elementos o la falta de características necesarias en favor de la simplicidad. Por otro lado, una herramienta compleja puede ofrecer una amplia gama de funcionalidades y opciones detalladas, pero puede requerir una curva de aprendizaje más pronunciada, lo que podría ralentizar el proceso inicial y generar frustración entre los usuarios que necesitan dominarla rápidamente.

Es por estos motivos que un estudio de las opciones que existen actualmente en el mercado es importante antes de comenzar a desarrollar el proyecto es vital tanto como para el usuario como para el programador. A continuación se expondrán algunas herramientas para el trabajo con imágenes. La tabla 2.1 muestra un listado de algunas de las herramientas utilizadas para el proceso de anotación de imágenes para la segmentación semántica comunes.

Tabla 2.1: Comparación de herramientas de anotación de imágenes. Datos obtenidos parcialmente de [11, 1].

Nombre	¿Open source?	¿Orientado a imagen médica?	Anotaciones	Lenguaje	Referencia
VGG Image Annotator	✓		manual	HTML, CSS, JS	`https://www.robots.ox.ac.uk/~vgg/software/via/`
Ratsnake	✓	✓	manual, automática	Java	`https://is-innovation.eu/ratsnake/index.htm`
Visual object tagging tool			manual, automática	TypeScript	`https://github.com/microsoft/VoTT`
Mask editor	✓		manual	MatLab	`https://es.mathworks.com/help/simulink/gui/mask-editor-overview.html`
Mango Viewer	✓	✓	manual	Java	`https://mangoviewer.com/`
Supervisely			manual, auto	Python	`https://supervisely.com/`
ITK-SNAP	✓	✓	manual, auto	C++	`http://www.itksnap.org`
Labelme	✓		manual	Python, Qt	`https://github.com/labelmeai/labelme`

Tabla 2.1: Comparación de herramientas de anotación de imágenes (continuación)

Nombre	¿Open source?	¿Orientado a imagen médica?	Anotaciones	Lenguaje	Referencia
Computer vision annotation tool	✓		manual, auto	TypeScript, Python	https://docs.cvat.ai/about/
3D-Slicer	✓	✓	manual, auto	C++, Python, Qt	https://www.slicer.org/
ePAD		✓	manual, auto	HTML, CSS, JS	https://epad.stanford.edu
Horos Viewer	✓	✓	manual, semi-auto	Objective-C	https://horosproject.org/
ImageJ/ FIJI	✓		manual, auto	Java	https://fiji.sc/
InVesalius	✓	✓	manual, auto	Python	https://invesalius.github.io/
MedSeg		✓	manual, auto		https://www.medseg.ai/
MeVisLab		✓	manual, auto	C++, Qt	https://www.mevislab.de/
MITK	✓	✓	manual, auto	C++	https://www.mitk.org/
ParaView	✓		manual	C, C++, Python, Fortran	https://www.paraview.org/
Seg3D	✓	✓	manual, semi-auto	C++	https://www.sci.utah.edu/cibc-software/seg3d

2.3. Métodos convencionales de segmentación

La segmentación en imágenes es uno de los campos más importantes dentro de la visión por computador. Tanto es así que, a lo largo de los años, se han desarrollado numerosos métodos y técnicas para esta única tarea.

Distinguimos los siguientes grandes grupos de métodos para segmentar imagenes[20, 18]:

Métodos basados en la intensidad

Uno de los enfoques más sencillos para segmentar una imagen, basada en los niveles de intensidad de los píxeles (o vóxeles), es la aproximación basada en umbrales. Las técnicas basadas en umbrales clasifican la imagen en dos clases (clasificación binaria) y funciona basandose en el principio de que los píxeles pertenecientes a un cierto rango de intensidad representan a una clase mientras que los demás representan a la otra. A los píxeles por encima del valor mínimo definido en el umbral se les asigna el valor 1 mientras que a los píxeles con un nivel de intensidad mas bajo que el del umbral se les asigna el valor 0 y son tratados como píxeles del fondo.

Las técnicas de segmentación basadas en umbrales consumen pocos recursos, son computacionalmente rápidas y, gracias a estas características, pueden ser usadas en aplicaciones en tiempo real con la ayuda de hardware especializado[29]. Los umbrales pueden ser implementados tanto de manera global como local. Dependiendo de la región donde se aplice la segmentación mediante umbrales se pueden distinguir tres tipos [18]:

- **Umbralización global**: Consiste en utilizar cualquier valor apropiado t y aplicarlo de forma constante a toda la imagen de la forma:

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

  donde $g(x,y)$ es la imagen resultante, $i(x,y)$ es la imagen de entrada y t es el valor del umbral.

- **Umbralización variable**: En este tipo de umbralización el valor de t puede variar a lo largo de la imagen. Esto, a su vez, puede tener dos tipos:
  - **Umbralización local**: El valor de t depende de los valores adyacentes de x e y.
  - **Umbralización adaptativa**: El valor de t es una función de x e y.

- **Umbralización múltiple**: Múltiples valores de umbral son utilizados en la misma imagen. La función asociada puede ser vista como:

$$q(x,y) = \begin{cases} m, & \text{si } p(x,y) > T1 \\ n, & \text{si } p(x,y) \leq T1 \\ o, & \text{si } p(x,y) \leq T0 \end{cases}$$

Los valores de los umbrales pueden ser obtenidos mediante el uso de otras técnicas como los histogramas de intensidad de la imagen o algoritmos simples que los computen.

En la tabla 2.2 se puede ver de forma resumida las ventajas y desventajas que conlleva el uso de los métodos basados en intensidad. La ventaja más evidente que este tipo de método proporciona respecto al resto es su simplicidad, sin embargo esta simplicidad es también la que hace que los métodos basados en intensidad no sean deseables en aplicaciones que requieran de un alto nivel de precisión.

Ventajas	**Inconvenientes**
No es necesaria información previa: Estos métodos no requieren información previa sobre la imagen, lo que los hace útiles en situaciones donde la disponibilidad de datos anteriores es limitada o inexistente.	Dependencia del umbral: La selección incorrecta del umbral puede llevar a la exclusión incorrecta de elementos importantes o a una segmentación inexacta.
Simplicidad de implementación: Los métodos basados en la intensidad son relativamente simples de implementar y entender, lo que los hace accesibles para usuarios con poca experiencia en segmentación de imágenes.	Detalles espaciales no considerados: Estos métodos pueden pasar por alto detalles espaciales importantes en la imagen, lo que puede resultar en una segmentación menos precisa, especialmente en imágenes con objetos pequeños o formas complejas.

Tabla 2.2: Ventajas e Inconvenientes de los Métodos Basados en Intensidad

Métodos basados en bordes

Un borde (*edge* en inglés) en una imagen es una transición significativa en la intensidad de los píxeles, que generalmente indica un cambio brusco en la luminosidad o color. Representa la frontera entre dos regiones homogéneas, diferenciando áreas con características distintas en términos de textura, color o intensidad [44]. La detección de bordes o *edge detection* es una herramienta fundamental utilizada en la mayoría de las aplicaciones de procesamiento de imágenes para obtener información mediante la extracción de características o, y el punto de interés de esta sección, la segmentación de objetos.

Mediante este proceso se detectan los contornos de un objeto y las fronteras entre los objetos y el fondo en la imagen. Para llevar a cabo la detección de bordes, se han desarrollado diversas técnicas que permiten identificar y localizar estas transiciones significativas en la intensidad de los píxeles. Cada una de estas técnicas ofrece ventajas específicas y está diseñada para abordar distintos desafíos en el procesamiento de imágenes. A continuación, se presentan algunas de las técnicas más reconocidas y utilizadas en la detección de bordes:

- **Operador Sobel**: Consiste en el uso de dos matrices de convolución con dimensión 3x3, una para detectar cambios de intensidad en la dirección horizontal (G_x) y otra para la dirección vertical (G_y). Dichas matrices se aplican a la imagen mediante el proceso de convolución.

Las matrices de Sobel son las siguientes:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \tag{2.1}$$

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{2.2}$$

Al convolver estas matrices con la imagen original, se obtienen dos imágenes que representan las derivadas aproximadas en las direcciones x e y, conocidas como el gradiente de la imagen. La magnitud del gradiente en cada punto de la imagen se puede calcular como:

$$G = \sqrt{G_x^2 + G_y^2}$$

Mientras que la dirección del gradiente se puede calcular como:

$$\theta = \arctan\left(\frac{G_y}{G_x}\right)$$

La imagen resultante muestra las áreas donde hay cambios abruptos en la intensidad de los píxeles y resalta los bordes y contornos de los objetos en la escena. Esto facilita la identificación de regiones de interés y la separación de objetos del fondo o de otras áreas de la imagen.

- **Operador Prewitt**: Similar en concepto al operador Sobel, el operador Prewitt utiliza exactamente las mismas ecuaciones con la diferencia de no dar énfasis a los píxeles mas cercanos al centro de la máscara sinó dar a todos los píxeles el mismo peso constante c (c = 1 en el ejemplo). Las matrices del operador Prewitt, por tanto, quedarían de la siguiente manera:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \tag{2.3}$$

$$G_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \tag{2.4}$$

Al igual que el operador Sobel, el resultado de $|G_x| + |G_y|$ es un indicador de la intensidad del gradiente en el píxel que se está estudiando [44].

- **Operador Laplaciano**: Al igual que los dos anteriores, el operador Laplaciano consiste en una máscara de convolución que se utiliza para detectar bordes en una imagen. A diferencia de los operadores de gradiente como Sobel y Prewitt, el operador Laplaciano calcula la segunda derivada de la imagen, lo que permite identificar regiones donde la intensidad de los píxeles cambia abruptamente en todas las direcciones. La segunda derivada en una imagen $f(x, y)$ se define de la siguiente forma:

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

Esta técnica es particularmente útil para detectar bordes en imágenes donde los cambios de intensidad no son necesariamente orientados horizontal o verticalmente. Normalmente se utiliza para establecer si un pixel esta en el lado más oscuro o claro de un borde. La máscara de convolución típica para el operador Laplaciano es:

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

Sus usos si bien similares a los de los operadores Sobel y Prewitt, se distinguen por su capacidad para detectar bordes en todas las direcciones con igual precisión, lo que lo hace especialmente útil en situaciones donde los bordes no siguen una orientación predominante. Esta característica permite una mayor flexibilidad y precisión en la segmentación de objetos en imágenes complejas.

- **Detector Canny**: La técnica de Canny [6] es un método fundamental para detectar bordes y realizar la segmentación de objetos en una imagen. Este método se distingue por su capacidad para eliminar el ruido de la imagen antes de identificar los bordes, preservando al mismo tiempo las características distintivas de estos bordes. Posteriormente, se aplica la información obtenida para precisar la detección de los bordes de los objetos en la imagen.

El detector Canny consiste en una serie de técnicas que, combinadas entre ellas, tienen como finalidad el aislamiento y detección de bordes en imágenes. Los pasos algorítmicos para el detector Canny son los siguientes:

Los pasos algorítmicos para la técnica de detección de bordes de Canny son los siguientes:

1. Convolucionar la imagen $f(r, c)$ con una función Gaussiana para obtener una imagen suavizada $f'(r, c)$ donde $f'(r, c) = f(r, c) * G(r, c, \sigma)$

2. Aplicar el operador de gradiente de primera diferencia para calcular la fuerza del borde; la magnitud y la dirección del gradiente se obtienen como se ha mencionado anteriormente.

3. Aplicar la supresión no máxima a la magnitud del gradiente.

4. Aplicar un umbral a la imagen resultante de la supresión no máxima para identificar los bordes finales.

Los dos pasos finales son utilizados para refinar la detección de bordes, eliminando respuestas falsas y asegurando que solo los bordes más significativos sean conservados en la imagen final.

AÑADIR IMAGENES COMPARATIVAS DE [46]

En la tabla 2.3 se listan las ventajas e inconvenientes de los métodos de segmentación basados en bordes. Estos métodos tienen a depender de la calidad de la imagen para su correcto funcionamiento por lo que este es un punto importante a la hora de considerar su uso.

Ventajas	**Inconvenientes**
Simplicidad y velocidad: la segmentación basada en bordes es relativamente sencilla y puede ser más rápida debido a sus operaciones computacionales fundamentalmente menos intensivas.	Dependencia de la calidad de los bordes: existe una gran dependencia de la calidad de la detección de bordes. Cualquier inexactitud en esta etapa puede conducir a una segmentación incompleta o errónea.
Información sobre la forma: dado que esta técnica se centra en los bordes, tiende a preservar mejor la forma de los objetos que otros métodos de segmentación.	Afectado por el ruido: las imágenes ruidosas pueden producir bordes falsos, lo que lleva a una segmentación incorrecta. Se vuelve necesario realizar preprocesamiento para reducir el ruido, lo que agrega complejidad.
Menos sensible a los cambios en brillo y color: las características de los bordes dependen más de los cambios en los valores de píxeles que de sus valores absolutos, lo que hace que los métodos basados en bordes sean menos sensibles a las variaciones en la iluminación o el color.	Dificultad para manejar texturas: las imágenes con texturas complejas pueden ser desafiantes, ya que los cambios de intensidad dentro de las texturas también pueden identificarse como bordes, lo que resulta en una sobresegmentación.

Tabla 2.3: Ventajas e Inconvenientes de los Métodos Basados en Bordes

Métodos basados en regiones

Esta técnica agrupa píxeles según sus propiedades inherentes, como color, intensidad o textura. Al identificar áreas con características similares, la segmentación basada en regiones tiene como objetivo crear segmentos coherentes y significativos dentro de una imagen. Este método es particularmente efectivo cuando los objetos tienen límites distintivos y propiedades bien definidas [9, 13]. Hay dos técnicas básicas basadas en este método:

- **Técnica de *region growing***: el método de crecimiento de regiones (*region growing method*) consiste en dividir una imagen en varias regiones mediante el crecimiento de semillas (*seeds* en inglés) o píxeles iniciales. Estas semillas pueden ser seleccionadas manualmente, utilizando conocimiento previo sobre la imagen, o automáticamente, basándose en características específicas de la aplicación que utilice la técnica.

Una vez seleccionadas las semillas iniciales, el algoritmo de crecimiento de regiones se encarga de expandir estas semillas de manera iterativa. En cada iteración, el algoritmo evalúa la conectividad entre los píxeles adyacentes y decide si un píxel debe ser añadido a la región en crecimiento. Este proceso de expansión continúa hasta que se cumplen ciertas condiciones de detención, como alcanzar un tamaño máximo predefinido para cada región o cuando ya no hay píxeles adyacentes que cumplan con los criterios de crecimiento.

La decisión de agregar un píxel a una región en crecimiento se basa en criterios de similitud entre los píxeles, que pueden incluir la intensidad de color, la textura, la gradiente de la imagen, entre otros. Además, la conectividad entre los píxeles, que determina cómo se propagará el crecimiento de la región, también juega un papel importante en el proceso.

Al finalizar el proceso de crecimiento de regiones, se obtiene una segmentación de la imagen original en varias regiones coherentes, donde cada región representa un conjunto de píxeles con características similares. Este método es especialmente útil para segmentar imágenes donde los objetos tienen bordes bien definidos y propiedades homogéneas dentro de cada región.[34]

Método de *region merging and splitting*: El método de división y fusión (*region merging and splitting* en inglés) es una técnica utilizada en el procesamiento de imágenes para dividir una imagen en regiones uniformes y luego fusionarlas en regiones más grandes[27]. La estructura de representación básica es piramidal, lo que significa que una región cuadrada de tamaño m * m en un nivel de la pirámide tiene 4 subregiones de tamaño $m/2$ * $m/2$ debajo de ella en la pirámide. En la figura 2.2 se puede observar la estructura piramidal y como esta se corresponde con las regiones de la figura a segmentar.

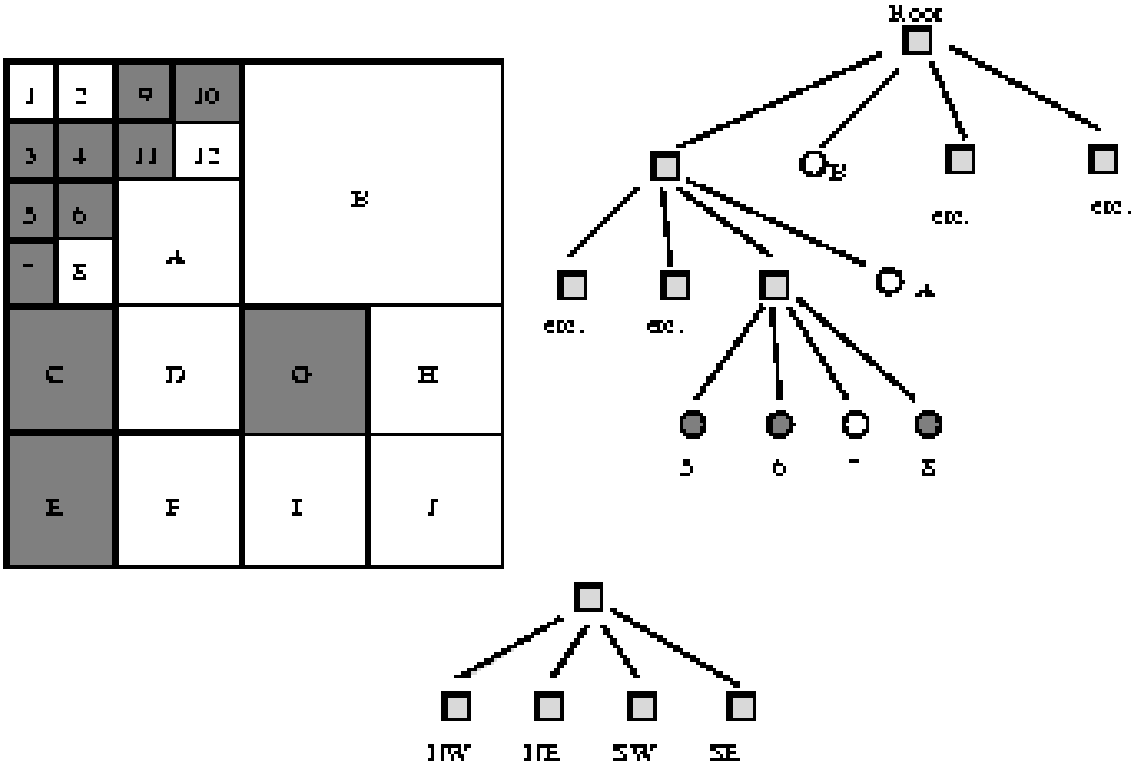

Figura 2.2: Representación visual de un *splitting* cuádruple de una imagen [36]

Normalmente, el algoritmo comienza con la suposición inicial de que toda la imagen es una sola región, luego calcula el criterio de homogeneidad para determinar si es verdadero. Si es falso, la región cuadrada se divide en las cuatro regiones más pequeñas. Este proceso se repite en cada una de las subregiones hasta que no sea

necesario dividir más. Luego, estas pequeñas regiones cuadradas se fusionan si son similares para formar regiones irregulares más grandes.

En la tabla 2.4 se pueden ver, de forma sintetizada, las ventajas y las desventajas del uso de los métodos de segmentación por regiones. Se puede ver que, al igual que los métodos basados en bordes, las desventajas de los métodos basados en regiones también están relacionados con las secciones de los algoritmos en las que se requiere cierta subjetividad por parte de los desarrolladores ya que los parámetros que se utilizan a la hora de segmentar pueden alterar drásticamente el resultado final.

Ventajas	**Inconvenientes**
Alta coherencia espacial: Los segmentos tienden a ser más coherentes en cuanto a las características locales.	Sensibilidad a la inicialización: Los resultados pueden variar según las *seeds* iniciales o los parámetros del algoritmo.
Capacidad para manejar ruido: Los métodos basados en regiones tienden a ser más robustos ante el ruido en comparación con otros enfoques.	Dependencia de los criterios de similitud: La definición de criterios de similitud puede ser subjetiva y afectar la calidad de la segmentación.
Adaptabilidad a diferentes tipos de imágenes: Estos métodos pueden adaptarse a una variedad de imágenes y tipos de objetos.	Costo computacional: Algunos métodos basados en regiones pueden ser computacionalmente costosos, especialmente para imágenes grandes.

Tabla 2.4: Ventajas e Inconvenientes de los Métodos Basados en Regiones

Métodos basados en agrupaciones

Las técnicas basadas en agrupaciones, o clustering based methods en inglés, consisten en dividir una imagen en conjuntos de píxeles con características similares. Estas agrupaciones resultan en la segmentación de objetos en una imagen. El clustering de datos es un método que organiza los elementos de datos de manera que los que pertenecen al mismo conjunto sean más similares entre sí que con los de otros conjuntos. Existen dos categorías básicas de métodos de clustering:

- **Métodos jerárquicos**: utilizan árboles para representar la estructura de las segmentaciones. la raíz del árbol representa todo el conjunto mientras que los nodos representan los conjuntos de píxeles. Se puede ver un ejemplo de la representación de una segmentación jerárquica en la figura 2.3. En esta imagen se aprecia como el elemento de la raiz corresponde a la imagen entera mientras que los nodos van representando cada vez elementos más concretas de la imagen.

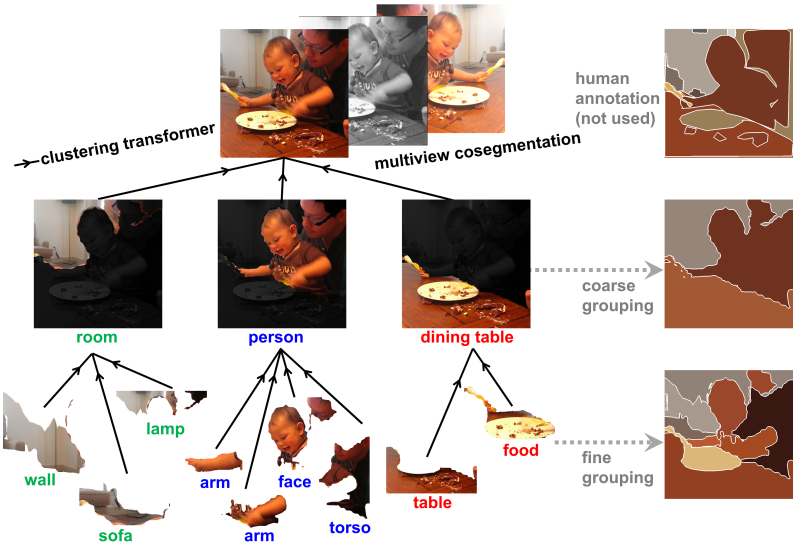

Figura 2.3: Visualización de la jerarquía de una imagen segmentada mediante un método basado en clustering [19]

- **Método basado en particiones**: implican dividir el conjunto de datos en un número predeterminado de grupos o "particiones" de manera que cada dato pertenezca únicamente a un grupo. Luego, estos grupos se ajustan iterativamente para optimizar algún criterio definido por el desarrollador.

  En contraste con los métodos jerárquicos que forman una estructura de árbol, los métodos basados en particiones no generan una estructura jerárquica. En cambio, simplemente dividen el conjunto de datos en particiones o grupos distintos como se puede apreciar en la figura 2.4, dónde diferentes regiones de la imagen son divididas sin seguir ningún tipo de progresión.

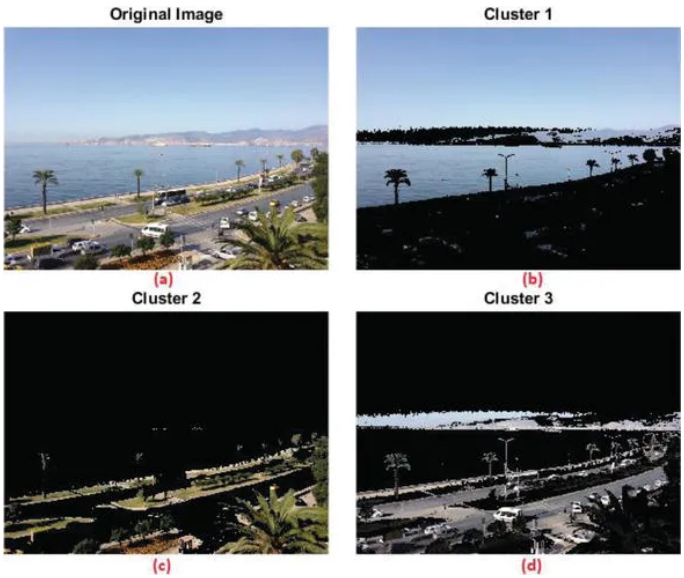

Figura 2.4: Visualización del método basado en particiones. Las 4 agrupaciones de la imagen son independientes entre ellos [3]

Además de las dos categorías a la hora de realizar las agrupaciones también existen dos tipos de *clustering*:

- El ***hard Clustering*** divide la imagen en clusters donde cada píxel pertenece a un solo cluster, utilizando funciones de membresía con valores 1 o 0, es decir, un píxel dado puede pertenecer o no a una agrupación concreta.
- En cambio, el ***soft Clustering*** es más flexible, permitiendo que un píxel pertenezca a múltiples *clusters* con grados de pertenencia definidos por valores de membresía. El *soft clustering*, por lo tanto, es más útil para la segmentación en inmágenes donde la división no debe ser tan estricta.

Un ejemplo de algoritmo de *clustering*, y el más extendido y utilizado para la segmentación semántica mediante métodos de agrupamiento, es el algoritmo *k-means*. Se trata de una técnica iterativa utilizada para dividir una imagen en *k clusters*. En el contexto de segmentación semántica de imágenes, este algoritmo se utiliza para agrupar píxeles en regiones semánticas similares[28, 35]. Funciona de la siguiente manera:

1. **Inicialización de los centroides**: Se eligen K centros de *cluster*, ya sea de forma aleatoria o basados en algún criterio heurístico. Estos k centros (centroides) serán los puntos iniciales desde los que se iniciará el algoritmo.
2. **Asignación de píxeles a *clusters***: Cada píxel en la imagen se asigna al *cluster* cuyo centro tenga la menor distancia al píxel, utilizando alguna medida de distancia como por ejemplo la distancia euclidiana.
3. Actualización de centroides: Se recalculan los centros de los *clusters* tomando el promedio de todos los píxeles asignados a cada agrupación.
4. Iteración: Los pasos 2 y 3 se repiten hasta que se alcance la convergencia, es decir, cuando ningún píxel cambie de *cluster*.

Este proceso busca minimizar la suma de cuadrados dentro de cada cluster (WCSS), donde el centroide de cada cluster es el punto que minimiza la distancia al resto de los puntos en ese cluster.

En la figura 2.5 se puede ver como el algoritmo va aplicándose a un conjunto de puntos aleatorios que representan píxeles de una imagen. Los centroides están representados por

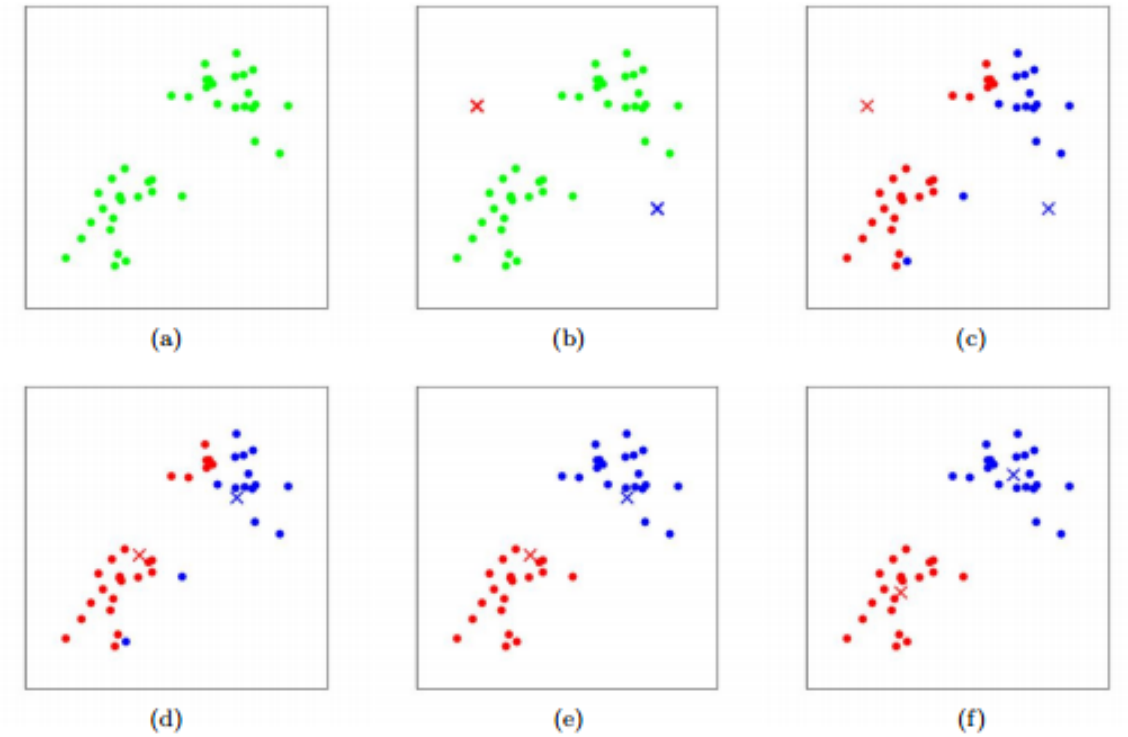

Figura 2.5: Proceso que sigue un conjunto de puntos mediante el algoritmo *k-means* [35]

El algoritmo *k-means* es un ejemplo de un algoritmo perteneciente a la categoría de métodos de agrupación por particiones, ya que las divisiones de los datos no están agrupadas en una estructura en árbol sino que son entidades independientes entre ellas. Asimismo el tipo de *clustering* que se utiliza en *k-means* es el *hard clustering* puesto que los píxeles, a pesar de variar durante el algoritmo, son asignados a un solo *cluster* a la vez. Además los píxeles en el resultado final quedan definidos en únicamente una agrupación y no a múltiples a la vez.

En la tabla 2.5 se puede ver un resumen de las ventajas e inconvenientes de los métodos basados en agrupaciones/*clustering*. Las desventajas de los métodos por agrupaciones están estrechamente relacionadas con el funcionamiento de estos ya que la eficiencia de estas dependerá enteramente de los parámetros iniciales que se utilicen para el algoritmo. Las ventajas, sin embargo, tienen más que ver con la flexibilidad que este tipo de métodos proporciona.

Ventajas	Inconvenientes
Identificación de objetos similares: Los métodos de clustering agrupan píxeles con características similares, lo que facilita la identificación y segmentación de objetos en una imagen.	Sensibilidad a la inicialización: Los resultados del clustering pueden depender en gran medida de la inicialización de los centroides o de otros parámetros, lo que puede afectar la calidad de la segmentación.
Estructura jerárquica: Los métodos jerárquicos representan la estructura de segmentación mediante árboles, lo que puede proporcionar una comprensión más profunda de la relación entre las regiones segmentadas.	Funciones de membresía: Determinar las funciones de membrería de los algoritmos puede ser una tarea complicada e incluso pequeñas variciones pueden llevar a resultados muy diversos
Flexibilidad en la definición de criterios de agrupamiento: Los métodos de clustering permiten definir criterios de agrupamiento personalizados según las necesidades específicas de la aplicación.	Interpretación de los resultados: La interpretación de los clusters puede ser subjetiva y dependiente del usuario, especialmente en métodos basados en particiones donde la determinación del número óptimo de clusters puede ser difícil.

Tabla 2.5: Ventajas e Inconvenientes de los Métodos de Clustering

2.4. Métodos de ML

Los métodos de segmentación tradicionales como los que se han visto en el apartado **??** se basan en propiedades de las imágenes y sus objetos como los bordes, las regiones o los colores para identificar y separar objetos de interés dentro de una imagen. Sin embargo, con los avances en el campo del aprendizaje automático, han surgido enfoques más sofisticados y poderosos para abordar este problema.

El aprendizaje automático (*machine learning* o ML para abreviar) es una rama de la inteligencia artificial (IA) y que se centra en el uso de datos y algoritmos para permitir que la IA imite la forma en que aprenden los humanos, mejorando gradualmente su precisión[15]. El *machine learning* se ha convertido en un pilar fundamental en la segmentación de imágenes, ofreciendo una alternativa a los métodos convencionales. A diferencia de los enfoques tradicionales que se basan en reglas específicas y criterios predefinidos, el machine learning permite que los algoritmos aprendan automáticamente patrones y características relevantes directamente de los datos de entrenamiento. Esto lleva a una segmentación más adaptativa y precisa, capaz de manejar una amplia gama de escenarios y condiciones de imagen.

En comparación con los métodos convencionales de segmentación vistos anteriormente, el enfoque del *machine learning* presenta varias diferencias clave. Mientras que los métodos convencionales dependen en gran medida de reglas y parámetros predeterminados, el aprendizaje automático permite que los algoritmos se adapten

y mejoren continuamente a medida que se exponen a más datos. Además, los métodos de *machine learning* tienden a ser más flexibles y generales, capaces de manejar una variedad más amplia de imágenes y condiciones.

Sin embargo, esta flexibilidad suele venir acompañada de una importante carga computacional por lo que este tipo de métodos suelen requerir de dispositivos potentes e incluso de algunos dispositivos especializados para poder realizar las segmentaciones de grandes volúmenes de imágenes en un tiempo relativamente corto.

2.4.1. Redes neuronales Convolucionales (CNN)

Las redes neuronales convolucionales (CNN por sus siglas en inglés) son una arquitectura de redes neuronales profundas que se ha destacado en el campo del aprendizaje automático y la visión por computadora, especialmente en tareas de segmentación de imágenes. Estas redes están compuestas por múltiples capas diseñadas específicamente para procesar datos de imágenes.

Una "capa" en este contexto hace referencia a una estructura de datos fundamental que realiza operaciones específicas en los datos de entrada para transformarlos en características más abstractas y útiles para la tarea de segmentación de imágenes. Cada capa toma datos de entrada, realiza cálculos o transformaciones en estos datos, y produce una salida que sirve como entrada para la siguiente capa.

En una CNN, hay varios tipos capas. Cada una de estas capas desempeña un papel específico en el procesamiento de las imágenes:

- Capa convolucional: es la primera y más importante capa de una CNN, donde ocurre la mayor parte del cálculo. Requiere datos de entrada, un filtro y un mapa de características. La imagen de entrada se representa como una matriz tridimensional de píxeles (alto, ancho y profundidad para RGB). Un detector de características (filtro) se desplaza por la imagen, realizando el producto escalar (*dot product*) entre los píxeles de entrada y el filtro para crear un mapa de características. Este proceso se conoce como convolución.

  Después de cada operación de convolución, se aplica una transformación de Unidad Lineal Rectificada (ReLU) al mapa de características, introduciendo no linealidad en el modelo. Un ReLU es una función de activación que convierte todos los valores negativos en cero y deja los valores positivos sin cambios.

  En la figura 2.6 se puede apreciar el comportamiento de una convolución dada una matriz (como puede ser una imagen).

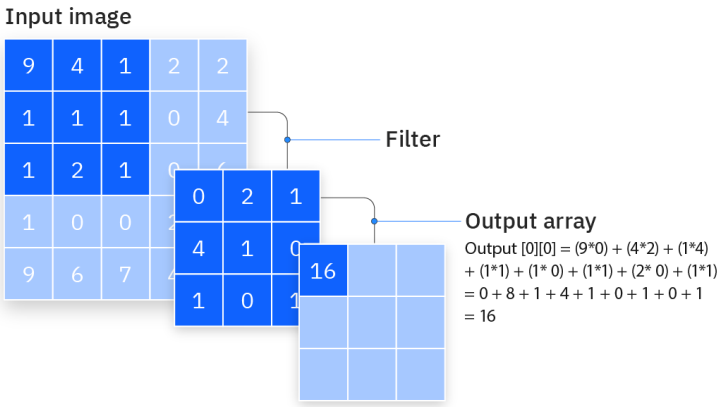

Figura 2.6: Visualización del proceso de convolución de una CNN. El filtro se desplaza por la imagen de entrada, realizando productos punto entre los valores de la imagen y los valores del filtro, generando el array de salida[47].

- **Capas convolucionales adicionales**: estas capas pueden seguir a la inicial, formando una estructura jerárquica que permite reconocer patrones más complejos en la imagen, desde componentes simples hasta objetos completos.
- **Capa de *pooling***: también conocida como *downsampling*, esta capa reduce la dimensión y la cantidad de parámetros. Se utiliza un filtro que aplica una función de agregación (máximo o promedio) en los valores dentro del campo receptivo, seleccionando el valor máximo o el promedio para el *array* de salida. Aunque se pierde información, esto mejora la eficiencia y reduce el riesgo del *overfitting*, un ajuste demasiado preciso a los datos de entrenamiento que puede causar una disminución en el rendimiento del modelo en datos nuevos y no vistos. Un ejemplo visual del overfitting puede observarse en la figura 2.7 dónde el modelo se ha adaptado demasiado a los datos proporcionados y esto ha generado que el aprendizaje futuro del modelo se vea comprometido.

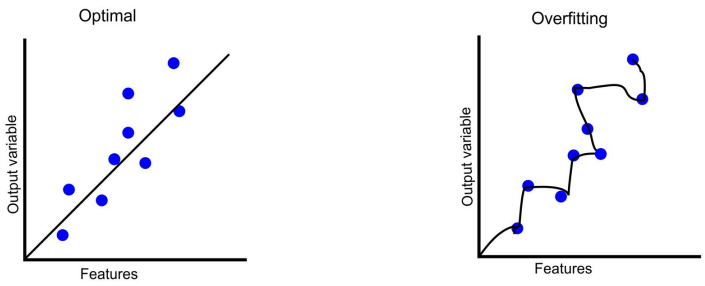

Figura 2.7: Visualización del concepto de *overfitting* [30]

- **Capa Totalmente Conectada** En esta capa, cada nodo de la capa de salida se conecta directamente a un nodo de la capa anterior. Realiza la tarea de clasificación basada en las características extraídas por las capas anteriores. Normalmente se utiliza una función de activación *softmax* (2.5 para clasificar las entradas, produciendo una probabilidad de 0 a 1.

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}} \tag{2.5}$$

La característica distintiva de las CNN es su capacidad para aprender características jerárquicas y complejas directamente de los datos de entrada. A través de capas convolucionales, la red puede extraer automáticamente patrones y características significativas de las imágenes, como bordes, texturas y formas. Posteriormente, estas características se utilizan para realizar la segmentación de las imágenes, donde se asigna una etiqueta a cada píxel según la clase a la que pertenece.

Algunas de las ventajas y desventajas con las que cuentan las redes neuronales convolucionales son las siguientes [7]:

- **Ventajas**
  - Alto rendimiento en diversas aplicaciones: Las CNN han demostrado resultados notables en una amplia gama de aplicaciones, incluyendo reconocimiento facial[23], detección de objetos en movimiento[5] y clasificación de imágenes[10].
  - Eficiencia en tareas de clasificación y segmentación de imágenes: Son altamente efectivas en tareas como clasificación y segmentación de imágenes, lo que las hace adecuadas para aplicaciones como el procesamiento de imágenes médicas y sistemas de reconocimiento automático [10].
  - Capacidad para aprender características automáticamente: A diferencia de los métodos tradicionales que requieren la extracción manual de características, las CNN pueden aprender automáticamente las características relevantes de las imágenes durante el entrenamiento.
  - Adaptabilidad a diferentes tipos de datos: Son capaces de manejar diferentes tipos de datos, incluyendo imágenes de diversas dimensiones y tipos de contenido[26].
- **Desventajas**:
  - Tiempo de entrenamiento prolongado: El entrenamiento de CNN puede ser costoso en términos de tiempo, especialmente cuando se trabaja con conjuntos de datos grandes y durante múltiples épocas[22].
  - Requisitos computacionales elevados: La implementación y entrenamiento de CNN pueden requerir recursos computacionales significativos, lo que puede ser una limitación en entornos con recursos limitados.

- Posibilidad de sobreajuste: Existe el riesgo de sobreajuste, especialmente cuando se entrenan con conjuntos de datos pequeños o ruidosos, lo que puede comprometer la generalización del modelo a nuevos datos[21].
- Interpretación de resultados: A menudo, las CNN se consideran como çajas negras"debido a la complejidad de sus arquitecturas, lo que dificulta la interpretación de cómo se llega a ciertas predicciones.

Dentro de el gran campo que son las redes neuronales convolucionales, se pueden encontrar diversas arquitecturas. Las arquitecturas de una red neuronal hacen referencia a la estructura y organización de sus capas, así como la manera en que estas capas están conectadas entre sí. La elección y diseño de estas arquitecturas determinan en gran medida el rendimiento y la eficiencia de la red para tareas específicas. A continuación se expondrá la arquitectura más utilizada actualmente para la segmentación de imágenes médicas.

2.4.2. U-net

La *U-Net*[42] es una arquitectura de red neuronal convolucional diseñada específicamente para tareas de segmentación de imágenes biomédicas.

Se compone de un camino de contracción para capturar contexto y un camino de expansión para una localización precisa, lo que permite un entrenamiento eficiente con un número limitado de muestras anotadas.

Es esta capacidad para obtener resultados prometedores a partir de un número limitado de anotaciones lo que hace de la *U-net* la herramienta predilecta para el análisis de imágenes médicas, ya que uno de los principales problemas a los que actualmente se enfrentan los expertos es la falta de datos para el entrenamiento de los modelos.

La arquitectura de la U-Net se distingue por su diseño en forma de U, resultado de la forma en la que las diferentes partes de la arquitectura están conectadas entre sí, que consta de dos vías principales: la vía de la contracción (o codificador/*encoder*) y la vía de expansión (o descodificador/*decoder*) simétrica. Cada una de estas vías desempeña un papel crucial en el proceso de segmentación de imágenes y permite que la U-Net capture tanto el contexto global como los detalles locales de la imagen.

Vía de contracción (*encoder*)

Esta vía se encarga de capturar el contexto de la imagen a través de la extracción de características. Está compuesta por una serie de capas de convolución seguidas de capas de *max pooling* que reducen la resolución espacial de la imagen conservando solo las características más relevantes al tomar el valor máximo dentro de cada región de la imagen.

Las capas de convolución aplican filtros a la imagen de entrada para detectar

características locales y patrones relevantes. Estas características se van volviendo más abstractas a medida que se profundiza en la red.

Por otro lado, las capas de *max pooling* reducen progresivamente la resolución espacial de la imagen, disminuyendo su tamaño y, por ende, la cantidad de parámetros de la red. Esto ayuda a capturar el contexto global de la imagen y a reducir la carga computacional.

Vía de expansión (*decoder*)

La vía de expansión simétrica o *decoder* tiene como objetivo reconstruir la segmentación de la imagen a partir de las características extraídas por el *encoder*. Utiliza capas de convolución transpuesta (o deconvolución) que aumenta la resolución espacial de la representación de características, de forma inversa a como la capa de *max pooling* reducía la resolución en la vía de contracción. Al igual que en el codificador, se emplean capas de convolución para procesar las características y refinar la segmentación.

El motivo detrás de las conexiones en forma de U de la arquitectura radica en su capacidad para combinar la capacidad de capturar características de alto nivel con la preservación de detalles espaciales finos. Al utilizar una vía de contracción para capturar el contexto y una vía de expansión para la localización precisa, la U-Net logra producir segmentaciones precisas incluso en imágenes con detalles sutiles. Además, la utilización de conexiones residuales entre las vías de contracción y expansión ayuda a mitigar el problema de la pérdida de información durante el proceso de codificación y decodificación, lo que resulta en una mejor preservación de los detalles y una mayor precisión en la segmentación de imágenes. Estas características sumado a la capacidad de obtener resultados con el uso de pocos datos hacen de la U-net la arquitectura idónea para trabajar con el tipo de imágenes que suelen utilizarse en ámbitos médicos donde el nivel de detalle es elevado y la precisión es crucial para diagnósticos precisos y decisiones clínicas informadas.

En la figura 2.8 se puede ver un diagrama de la arquitectura U-net dónde se aprecian de forma visual las conexiones en forma de U entre el codificador y el decodificador (figura 2.8).

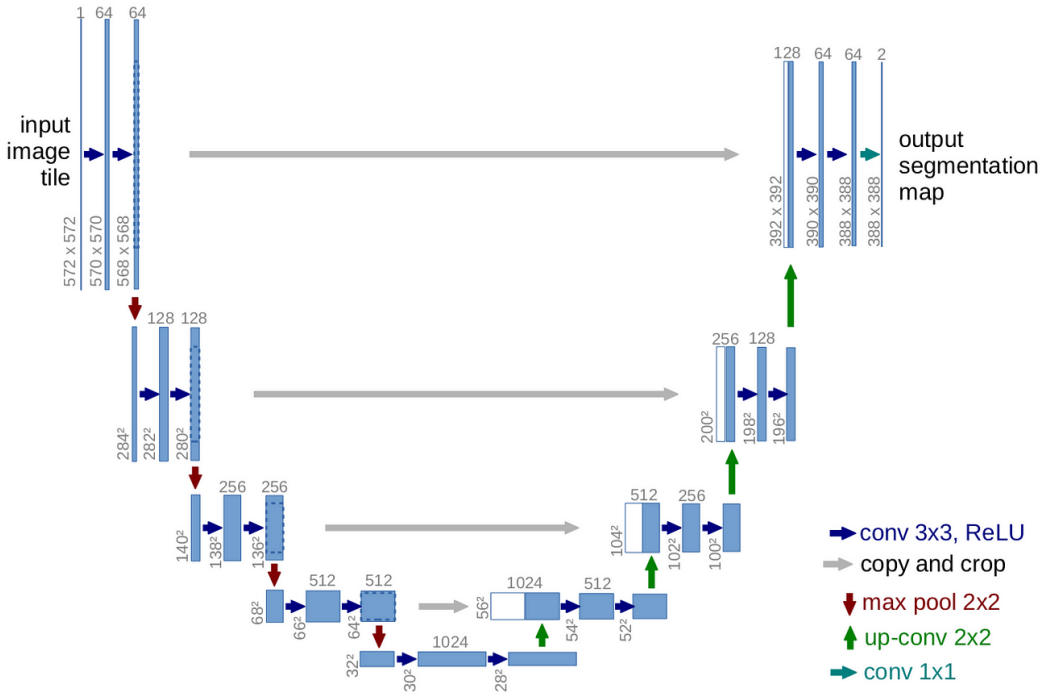

Figura 2.8: Arquitectura de la U-net [42].

Al utilizar técnicas de aumento de datos como deformaciones elásticas, la U-Net puede lograr un alto rendimiento con un número mínimo de imágenes anotadas y tiempos de entrenamiento razonables. La arquitectura ha demostrado un rendimiento excepcional en diversas aplicaciones de segmentación biomédica[42], lo que demuestra su versatilidad y eficacia en diferentes tareas. La U-Net se ha aplicado con éxito en tareas como la segmentación de estructuras cerebrales [37], nódulos pulmonares[48] y, como se ha hecho en este trabajo, tumores de mama.

Capítulo 3

Materiales y metodología

3.1. Materiales

En esta sección se hablará sobre los diferentes recursos y materiales que se han usado a lo largo de la elaboración del proyecto.

3.1.1. nnU-net v2

El enfoque de este proyecto ha sido el desarrollo del plugin y no el del modelo de *machine learning* encargado de realizar la segmentación automática dada una imagen. Esto se ha debido principalmente a una limitación de hardware a la hora del entrenamiento, por lo que en este proyecto se ha utilizado un modelo pre-entrenado de segmentación de lesiones en MRI basado en nnU-net v2[17] desarrollado previamente dentro del grupo de investigación BCN-AIM de la Universidad de Barcelona.

NnU-net v2 es un método de segmentación semántica que se adapta automáticamente a un *dataset* dado. Analiza los casos de entrenamiento proporcionados y configura automáticamente una *pipeline* de segmentación basada en una U-Net correspondiente a dicho *dataset*. Esto incluye la selección de hiperparámetros óptimos, ajustes en la arquitectura del modelo y la implementación de estrategias de preprocesamiento y posprocesamiento específicas para maximizar el rendimiento de la segmentación.

Las principales características de nnU-net v2 son:

- Auto-configuración: nnU-net v2 ajusta automáticamente la arquitectura del modelo y los hiperparámetros según el *dataset* de entrenamiento, eliminando la necesidad de ajustes manuales.
- Pipeline completa: Proporciona una solución integral que abarca todas las etapas del proceso de segmentación, desde el preprocesamiento hasta el posprocesamiento.
- Adaptabilidad: Es capaz de adaptarse a diferentes tipos de datos y tareas de

segmentación, lo que lo hace extremadamente versátil y útil en una amplia variedad de aplicaciones.

- Rendimiento robusto: Ha demostrado un rendimiento sobresaliente en numerosos desafíos de segmentación de imágenes biomédicas, destacándose por su precisión y fiabilidad.

Dado un nuevo conjunto de datos, nnU-Net analiza sistemáticamente los casos de entrenamiento proporcionados y crea una "huella digital del dataset"(*dataset fingerprint*) en un archivo JSON. A partir de esta información nnU-Net genera varias configuraciones de U-Net para cada dataset:

- **2D**: una U-Net 2D (para conjuntos de datos 2D y 3D).
- **3d_fullres**: una U-Net 3D que opera a una alta resolución de imagen (exclusiva para conjuntos de datos 3D).
- **3d_lowres → 3d_cascade_fullres**: Una cascada de U-nets 3D donde primero una U-net 3D opera en imágenes de baja resolución y luego una segunda U-net 3D de alta resolución refina las predicciones de la anterior (exclusiva para *datasets* con imágenes grandes).

Es importante añadir que no todas las configuraciones de U-Net son creadas para todos los datasets. En conjuntos de datos con imágenes de pequeño tamaño, la cascada de U-Nets (y con ella la configuración 3d_lowres) es omitida porque el tamaño de la U-Net a resolución completa ya cubre una gran parte de las imágenes de entrada.

nnU-Net configura sus *pipelines* de segmentación basándose en tres factores:

1. Los parámetros fijos no se adaptan. Durante el desarrollo de nnU-net v2 se identificó una configuración interna robusta (referente a ciertas propiedades de entrenamiento y de la arquitectura) que pueden ser usados siempre. Esto incluye, por ejemplo, la función de pérdia de la nnU-net, la mayor parte de la estrategia del aumento de datos y la tasa de aprendizaje entre otros.
2. Los parámetros basados en reglas utilizan la huella del *dataset* para adaptar ciertas propiedades de la *pipeline* de la segmentación utilizando unas reglas heurísticas predefinidas, Por ejemplo, la topología de la red (comportamiento del *pooling* y la profundidad de la arquitectura de la red) es adaptada al tamaño del *patch*.

2. **Parámetros basados en reglas**: Usan la huella digital del dataset para adaptar ciertas propiedades del *pipeline* de segmentación siguiendo reglas heurísticas predefinidas. Por ejemplo, la topología de la red (comportamiento del *pooling* y la profundidad de la arquitectura de la red) se adapta al tamaño del parche; el tamaño del parche, la topología de la red y el tamaño del lote son optimizados conjuntamente considerando las limitaciones de memoria de la GPU (unidad de procesado

de gráficos). 3. **Parámetros empíricos**: Se basan esencialmente en prueba y error. Por ejemplo, la selección de la mejor configuración de U-Net para el dataset dado (2D, 3D de resolución completa, 3D de baja resolución, cascada 3D) y la optimización de la estrategia de posprocesamiento.

En este proyecto, el uso de nnU-net v2 ha permitido centrarme en el desarrollo del plugin, aprovechando las capacidades de un modelo de segmentación pre-entrenado y altamente eficiente sobre el cual se puede hacer inferencia sin necesidad de conocer la arquitectura interna.

3.1.2. Mango Viewer

Multi-image Analysis GUI o Mango para acortar es una herramienta de visualización de imágenes orientado a la investigación de imágenes médicas. Mango pone a la disposición del usuario una serie de herramientas de análisis así como una interfaz de usuario para navegar entre volumenes de imágenes.

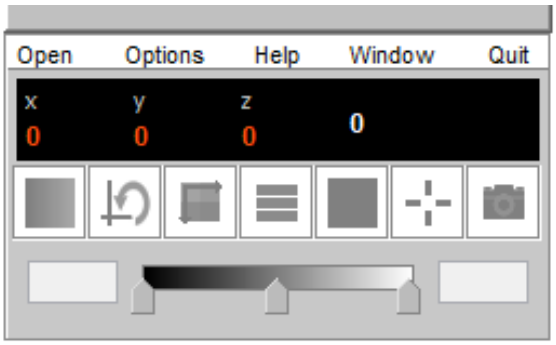

Figura 3.1: GUI principal de Mango Viewer

Mango ha sido la herramienta elegida para el desarrollo del plugin debido a varios factores que la hacen ideal para este proyecto.

En primer lugar, la afinidad con el proyecto RadioVal ha sido un factor determinante. Este proyecto busca promover la medicina de precisión con herramientas que ayudarán a los clínicos a realizar tratamientos más precisos e individualizados según las necesidades del paciente y para conseguir eso trabajan con diferentes herramientas de segmentación, siendo una de ellas Mango Viewer. La posibilidad de contar con expertos para consultas en este trabajo ha sido uno de los principales motivos de la elección de Mango como la herramienta de segmentación de preferencia.

Además de la posibilidad de recibir apoyo, Mango ofrece una Interfaz de Programación de Aplicaciones (*API* por sus siglas en inglés) clara y bien documentada, lo que facilita enormemente el desarrollo de nuevos plugins y extensiones. La claridad de la API permite un rápido entendimiento sobre como trabajar con la herramienta, acelerando el proceso de desarrollo y reduciendo la posibilidad de errores. Esto ha demostrado ser crucial debido al margen de tiempo tan ajustado del trabajo.

Por último, Mango está desarrollado con Java y utiliza Swing para la creación

de la interfaz gráfica. Ambas herramientas han sido utilizadas anteriormente en la carrera y la capacidad de poder trabajar directamente sin la necesidad de tener que aprender un nuevo lenguaje de programación o librería desde 0 ha sido vital para el correcto desarrollo del plugin, tanto en dificultad como en rapidez.

La documentación referente a Swing puede encontrarse en **docs.oracle.com/javase/6/docs/tec**

3.1.3. Lenguajes de programación y librerías

El plugin está dividido en dos grandes partes: la red neuronal, desarrollada en Python, y el plugin de Mango, desarrollado en Java.

Python

Para desarrollar el programa encargado de activar la red neuronal así como todo aquello relacionado con la inferencia se ha utilizado Python 3.9 junto con el IDE de PyCharm versión 2023.2.3 y las librerías que se muestran a continuación.

Librería	**Versión**
numpy	1.24.3
pytorch	2.2.2+cu118
nnunetv2	2.4.2
os	librería nativa de python
argparse	librería nativa de python

Tabla 3.1: Librerías utilizadas en el proyecto

Además de las librerías anteriormente mencionadas el entorno virtual con el que se ha desarrollado la parte de Python de la aplicación cuenta con más librerías de apoyo como scikit que, si bien están dentro del proyecto para que otras librerías (especialmente la librería nnunetv2) funcionen, no han sido utilizadas explícitamente por lo que no es necesario entrar en detalle sobre estas.

Java

Para desarrollar el plugin que interactúa con la aplicación de Mango Viewer directamente he utilizado IntelliJ en su versión 2021.1.1 junto con el SDK de Java coretto-1.8.0_412 en el nivel de lenguaje 6. Estos requisitos no son arbitrarios puesto que el uso de otros SDK para crear plugins hace que estos no sean detectados por la aplicación al intentar abrirlos o que estos provoquen cierres inesperados en la propia aplicación. La versión de Java 1.6 también es admitida por Mango Viewer pero para el desarrollo del plugin he decidido usar Java 1.8 al ser una versión un poco mas nueva y con la que estoy mas familiarizado.

Debido a como está hecho Mango Viewer he tenido también que utilizar *Swing* para trabajar con las interfaces gráficas, concretamente con el selector de archivos que permite al usuario seleccionar que imagen quiere segmentar y visualizar.

Además de *Swing* el uso de la clase *ProcessBuilder* de Java ha sido necesario para poder comunicar la aplicación de Python que genera las predicciones y la aplicación de Java que se encarga de gestionar todo lo relacionado al plugin y a la visualización de los datos.

3.1.4. Datos médicos

Para obtener los datos médicos utilizados para el entrenamiento del modelo de segmentación se han utilizado las imagenes proveïdas por el *National Cancer Institute*, concretamente el Duke-Breast-Cancer-MRI *dataset* [43] que contiene 922 imágenes de pacientes con cáncer de mama confirmados mediante biopsia invasiva recopilados a lo largo de 10 años. El *dataset* incluye los siguientes componentes sobre los datos:

- Datos demográficos, clínicos, patológicos, de tratamiento, resultados y genómicos.
- Las resonancias magnéticas dinámicas con contraste (DCE-MRI) preoperatorias en formato DICOM.
- Las segmentaciones de las lesiones en resonancia magnética dinámica con contraste (*DCE-MRI* por sus siglas en inglés) en forma de anotaciones en las imágenes de DCE-MRI realizadas por radiólogos.
- Características radionómicas: un conjunto de 529 características extraídas mediante ordenador sobre las imágenes recopiladas. Estas características representan una gran variedad de atributos de las imágenes como pueden ser el tamaño, forma, textura y realce tanto como del tumor como del tejido que lo rodea.

3.2. Metodología

A continuación se expondrán las metodologías que han sido utilizadas o relevantes a lo largo del proyecto. Si bien no todas han sido aplicadas de forma explícita si que han sido importantes para la realización del proyecto de alguna u otra forma.

3.2.1. API de Mango Viewer

Mango Viewer cuenta con APIs públicas para desarrolladores que permiten la fácil personalización de la aplicación para adaptarse a las necesidades específicas que se requieran de esta. Actualmente, Mango cuenta con una API de Java y una de Python.

La API de Python permite implementar (o "grabar"mediante la GUI de Mango) *scripts* que posteriormente pueden ser ejecutados de nuevo mediante la herramienta

Script Manager. Esta API, aunque útil a nivel usuario, no ha sido utilizada en el proyecto puesto que no permite modificar directamente el funcionamiento de Mango Viewer sinó que permite implementar código más bien parecido al de una *macro*.

La segunda API, la de Java, es la que se ha utilizado en este proyecto. Un plugin de Mango realmente es una implementación de una de las múltiples interfaces de las que dispone la API de Mango; a continuación se listan las interfaces con las que cuenta la API de Mango Viewer:

- **Atlas**: se utiliza para mapear coordenadas a etiquetas. Dichas etiquetas pueden aparecer en las *herramientas* y se actualizan conforme el usuario mueve el cursor del ratón en el visor de imágenes.
- **EditableHeader**: utilizado para dar soporte a la opción de *Edit Header* de los plugins que trabajan con los metadatos de una imagen.
- **ImageLoader**: proporciona acceso a una biblioteca de archivos basada en la web y puede aparecer en los cuadros de diálogo de carga de imágenes y de carga de Regiones de interés (*ROI* por sus siglas en inglés) en línea en Mango Viewer.
- **MangoPlugin**: un plugin de uso general que proporciona acceso a los datos de las imágenes y la interfaz gráfica de control. Puede usarse para hacer cálculos, crear ROI, generar gráficos, escribir archivos, etc..
- **ReadableHeader**: puede usarse para cargar imagenes con un formato desconocido para la aplicación.
- **SurfaceFormat**: facilita el guardado y la carga de datos de superficie. Estos datos pueden ser utilizados después en otras aplicaciones de renderizado de superficies.
- **SurfacePlugin**: es un plugin de propósito general que brinda acceso a datos de superficie y control de la interfaz de usuario del visor de superficies.
- **WritableHeader**: una extensión de *ReadableHeader* que permite leer y escribir en formatos de encabezado desconocidos.

En la Figura 3.2 se muestra el diagrama de clases en formato UML estándar. Este diagrama representa la estructura estática de las clases y sus relaciones en la aplicación. El diagrama está hecho en formato UML estándar. UML, o Lenguaje de Modelado Unificado, es un lenguaje de modelado estándar utilizado en ingeniería de software para visualizar, especificar, construir y documentar los artefactos de un sistema de software. En este caso concreto nos muestra como se relacionan las diferentes clases de Mango entre ellas.

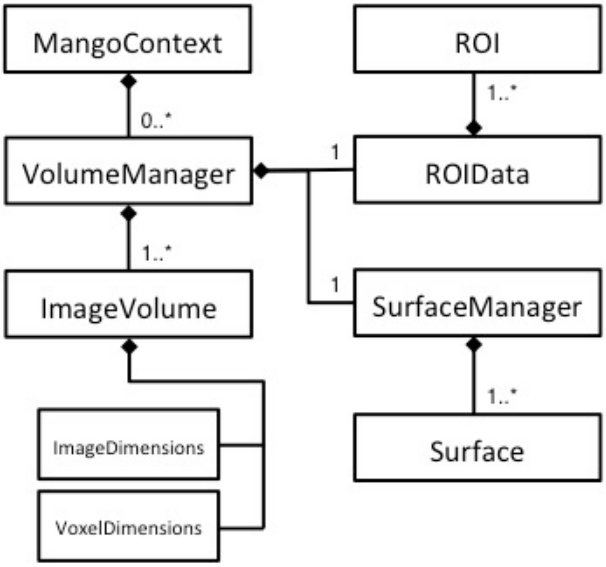

Figura 3.2: Diagrama de clases de la API de Mango Viewer [39]

El uso de estas interfaces permite un control mas preciso sobre los diferentes aspectos que constituyen Mango Viewer y para el desarrollo del proyecto se han utilizado las interfaces de *MangoPlugin* para el control de los aspectos generales del plugin como pueden ser los protocolos de comunicación entre otras apliaciones y los nuevos componentes de GUI, e *ImageLoader* para la carga de las ROI (regiones de interés) que se generan mediante el uso del modelo de IA. Más adelante, en la sección Desarrollo se entrará en más detalle sobre el proceso de creación del plugin.

3.2.2. Segmentación mediante nnU-net

La segmentación mediante el uso de la nnU-net v2 es la parte más importante del plugin ya que es lo que proporciona la funcionalidad. Sin embargo al estar hecha en Python esta se encuentra aislada del resto del plugin por lo que la forma de desarrollarlo ha seguido otro proceso que el del plugin de Mango. Como se ha mencionado anteriormente, el modelo que se ha utilizado para el desarrollo del plugin ha sido pre-entrenado con datos del un *dataset* público (*Duke-Breast-Cancer-MRI*) por lo que en esta memoria no se entrará en mucho detalle en el proceso de entrenamiento del modelo. Sin embargo si que es importante tocar ciertos puntos sobre el modelo en y el proceso que se sigue para entrenarlo utilizando nnU-net v2.

Como se ha comentado en la parte de Materiales, nnU-net v2 cuenta con 3 modos en los que se puede entrenar un modelo: *2D*, *3D-fullres* y *3D lowres* $\rightarrow$ *3D cascade fullres*. Al estar trabajando con mamografías hechas por resonancia magnética se ha utilizado la configuración de 3D-fullres puesto que las imágenes por MRI son datos tridimensionales generados a partir de la unión de varias capas de imágenes 2D. Para entrenar un modelo utilizando la nnU-net v2 primero es necesario tener en cuenta el hardware que se va a utilizar. La documentación de nnU-net v2 [16] recomienda

el uso de los siguientes componentes como mínimos para el entrenamiento de una nnU-net:

- GPU: Se recomienda encarecidamente usar una GPU o unidad de procesado de gráficos, ya que el entrenamiento en CPU o MPS (Apple M1/M2) tomará mucho tiempo en caso contrario. Para el entrenamiento, se requiere una GPU con al menos 10 GB de memoria. Algunas opciones recomendadas son la RTX 2080ti, la RTX 3080/3090 o la RTX 4080/4090

- CPU: Así como se recomienda el uso de una GPU, también se recomienda una CPU (unidad central de procesamiento) potente para acompañar la GPU. El mínimo recomendado es de 6 núcleos (12 hilos). Los requisitos de la CPU están relacionados con los de la GPU puesto que esta tiene que escalar con la GPU para evitar posibles cuellos de botella.

Para la inferencia, si bien se recomienda el uso de una GPU para reducir considerablemente los tiempos, no es necesario disponer de tanta potencia de cómputo paralelo como para el entrenamiento. Una GPU con 4GB de RAM (libre) se considera como el mínimo para el correcto funcionamiento de la inferencia utilizando nnU-net v2.

Para poder utilizar nnU-net v2 para el entrenamiento es necesario contar con PyTorch instalado. Para una mayor eficiencia a la hora de entrenar el modelo es recomendable también instalar PyTorch junto con CUDA (*Compute Unified Device Architecture*) para así poder utilizar una GPU para el entrenamiento y la inferencia de los modelos, sin embargo es posible utilizar únicamente la CPU del sistema para ambas cosas aunque la velocidad se verá considerablemente reducida.

nnU-net v2 se instala como cualquier otra librería de Python (*pip install nnunetv2* si solo se quiere utilizar como un algoritmo de segmentación básico para el entrenamiento y la inferencia) una vez cumplidos los requisitos previos de instalación y se sirve de comandos por consola para ejecutar las diferentes funcionalidades que ofrece. También es necesario especificar mediante el uso de variables de entorno dónde se guardará la *raw data* que se utilizará para el proceso de entrenamiento. Los comandos más importantes que nnU-net v2 proporciona, y los que se han utilizado en este proyecto, són los siguientes:

- **nnUNetv2_plan_and_preprocess**: extrae propiedades del conjunto de datos (el *data fingerprint* anteriormente mencionado), planifica experimentos con tres configuraciones de U-Net, y realiza el preprocesamiento del conjunto de datos, verificando su integridad.

- **nnUNetv2_train**: inicia el proceso de entrenamiento de un modelo de segmentación. El comando permite poner configuraciones específicas, número de epochs mínimos para hacer un *checkpoint*, especificar el dispositivo que se quiere utilizar y más opciones relacionadas con el entrenamiento de un modelo.

- **nnUNetv2_predict**: dado un checkpoint, el comando, como bien indica el nombre, realiza predicciones utilizando el modelo seleccionado. Al igual que *train* es posible utilizar parámetros para indicar que dispositivos usar, de que carpetas obtener las imágenes, etc.

3.2.3. Comunicación entre programas

Al estar el programa dividido en dos grandes partes (el plugin y el modelo) y haber sido estas realizadas en dos lenguajes de programación diferentes (Java y Python respectivamente) ha sido necesaria la impementación de algun tipo de sistema que permita la comunicación y sincronización entre los dos programas. Con la ayuda de FORTH-Hellas (*Foundation for Research and Technology*), un grupo de investigación griego miembro del consorcio de RadioVal y con experiencia previa en el desarrollo de plugins con Mango Viewer se plantearon algunas posibilidades sobre como resolver este problema:

- **Creación de una interfaz en Python**: la solución menos compleja a nivel comunicación interprogramaria era, simplemente, no requerir de tal comunicación. Crear tanto el modelo como el visor con Python mediante el uso de librerias como matplotlib y scikit evitaría la necesidad de comunicar dos programas completamente ajenos.

- **Servidor interno**: utilizando los sockets del propio ordenador se puede establecer una comunicación entre los programas Java y Python. Esto implica que los programas actúen como clientes y servidores, enviando y recibiendo datos entre sí a través de una conexión de red interna.

- **RESTful API**: se puede resolver el problema mediante la creación de un servidor REST en uno de los lenguajes (Java o Python) y haciendo que el otro programa actúe como cliente. De esta manera, puedes enviar solicitudes HTTP entre los programas para intercambiar datos[24]. Similar en concepto al uso de sockets TCP/IP.

- **Comunicación a través de la salida y entrada estándar**: es posible utilizar la línea de comandos y tanto *process builder* en Java como la librería *os* en Python para comunicar ambos programas a través de la línea de comandos y la capacidad de acceder al mismo directorio de ambos programas.

Capítulo 4

Implementación y resultados

En este capítulo se hablará sobre el desarrollo del plugin, todo lo que se ha tenido en cuenta, los requisitos para su funcionamiento y demás consideraciones que han habido para el funcionamiento de este.

4.1. *Overview* de la aplicación

Como se ha mencionado anteriormente, el plugin cuenta de dos partes: el plugin de Mango, un archivo JAR que se comunica directamente con la aplicación, y el modelo en Python, un modelo de ML que ha sido entrenado mediante el uso de la nnU-net v2.
A la hora de desarrollar el plugin ha sido necesario tener en cuenta el hecho de que ambas aplicaciones al haber sido desarrolladas en lenguajes de programación diferentes requerirían de alguna forma para comunicarse y sincronizarse entre ellas por lo que el uso de la línea de comandos (entrada y salida estándar) ha sido la opción final que se ha utilizado. Esta decisión ha sido tomada en base tanto a los objetivos de la aplicación, el contexto en el que se va a utilizar y la experiencia personal. La creación de un servidor TCP/IP o una API que comunique mediante solicitudes HTTP podría generar conflictos con otras aplicaciones médicas que requieran el uso de puertos para compartir información. Además al estar pensado para un entorno clínico lo ideal sería que la aplicación fuese lo más *plug & play* posible y el uso de servidores internos podría complicar las cosas. No solo eso si no que la complejidad que implicaría el uso de RESTful API no habría merecido la pena en este proyecto en particular puesto que la cantidad de datos que se deben compartir entre programas es mínima (únicamente segmentaciones almacenadas en una carpeta). El uso de RESTful API habría sido completamente nuevo y, por ende, habrían supuesto un retraso en el desarrollo del plugin. Esta ralentización del proceso de desarrollo sumado con la posibilidad de fallos tanto de programación como en la propia aplicación en un supuesto uso real han hecho que el uso de sockets o APIs no sea atractivo. Si bien la creación de una GUI propia que acompañe al modelo es atractiva, uno de los objetivos de este proyecto es que el plugin sea de fácil integración en un entorno médico. Mango Viewer es una aplicación diseñada

específicamente para el estudio de imágenes médicas y está preparada y adaptada para el uso de los especialistas del campo. Una nueva GUI habría supuesto no utilizar una herramienta ya integrada en el campo y, por lo tanto, habría implicado un proceso de aprendizaje adicional para los usuarios, así como una posible resistencia al cambio. Al integrar el plugin en Mango Viewer, se aprovecha la familiaridad de los usuarios con la interfaz de esta aplicación, lo que facilita su adopción y uso en entornos médicos. Además, al utilizar una herramienta ya establecida en el campo de la medicina, se garantiza una mayor confianza en la precisión y la fiabilidad de los resultados obtenidos con el plugin.

4.2. Desarrollo Python: modelo de ML

La parte de desarrollo en Python ha sido la más sencilla y corta del proyecto puesto que, al haber sido préviamente desarrollado el modelo de segmentación automático por un tercero, no ha sido necesario trabajar con otra cosa aparte de los comandos necesarios para hacer inferencia del modelo mediante el uso de los comandos anteriormente mencionados de nnU-net v2 en el apartado 3.2.2.

El código desarrollado no es más que una serie de comprobaciones que se hacen antes de ejecutar el comando necesario para activar la función de predicción de la nnU-net v2, el cúal ya viene integrado al descargar el módulo (*nnunetv2_predict*).

El código utiliza la librería argparse para ir montando el comando *nnunetv2_predict* en función de los resultados de las comprobaciones. El comando base que se utiliza para hacer la predicción es:

```
nnUNetv2\_predict -i \{\} -o prediction\_output -d \{\} -c 3d\_fullres
-chk checkpoint\_best.pth -f 0
```

Los espacios con llaves corresponden a argumentos que indican la id del *dataset* que se quiere utilizar (001 es la id de la carpeta que se debe usar por defecto) y la ruta de la carpeta de la cual se quieren utilizar las imágenes para segmentar (la letra p es utilizada por la palabra *path*, ruta en inglés) respectivamente. Estos argumentos son definidos en variables a las cuales se les da valor mediante el comando que se utiliza al ejecutar el programa.

Además de la ruta necesaria y del id que se utilizará para reconocer el *dataset* que se utilizará para segmentar es importante también que el programa aproveche al máximo las capacidades del dispositivo en el que se está ejecutando. Para esto, y como se ha comentado en el apartado 3.2.2, se puede usar la GPU o tarjeta gráfica en caso de que el dispositivo disponga de una además del software necesario para utilizarlo. Mediante el uso de la librería PyTorch el programa detecta si CUDA[1] está disponible en el sistema. En caso de que una tarjeta gráfica esté disponible se añadirá el argumento **-device cuda** al comando base, en caso contrario se añadirá al comando **-device cpu** y el modelo utilizará únicamente la CPU del dispositivo.

[1]Plataforma de computación paralela desarrollada por NVIDIA que permite utilizar la potencia de las GPU para acelerar aplicaciones computacionales

En la imagen 4.1 se puede ver de forma gráfica las opciones que existen a la hora de generar el comando de entrenamiento.

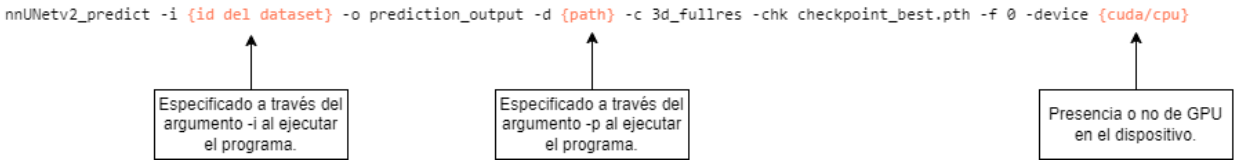

Figura 4.1: Resumen sobre el comando generado

Durante el desarrollo se han hecho pruebas con ambas opciones, los resultados de estas se pueden ver en 4.5.

Una vez el modelo termina el proceso de predicción, el *output* de este consiste en un fichero **nifti** (extensión nii.gz) con una máscara binaria (es decir, valores de 0 o 1 en cada píxel) que representa la sección que el modelo ha predicho como región de interés, es decir, aquellas secciones de la imagen donde el modelo considere que se encuentran los tumores.

En la figura 4.2 se puede ver el resultado de una predicción del modelo de ML. Al utilizar la imagen que se quiere estudiar (izquierda) el resultado generado es una máscara como la que se puede ver en la imagen resultante (derecha).

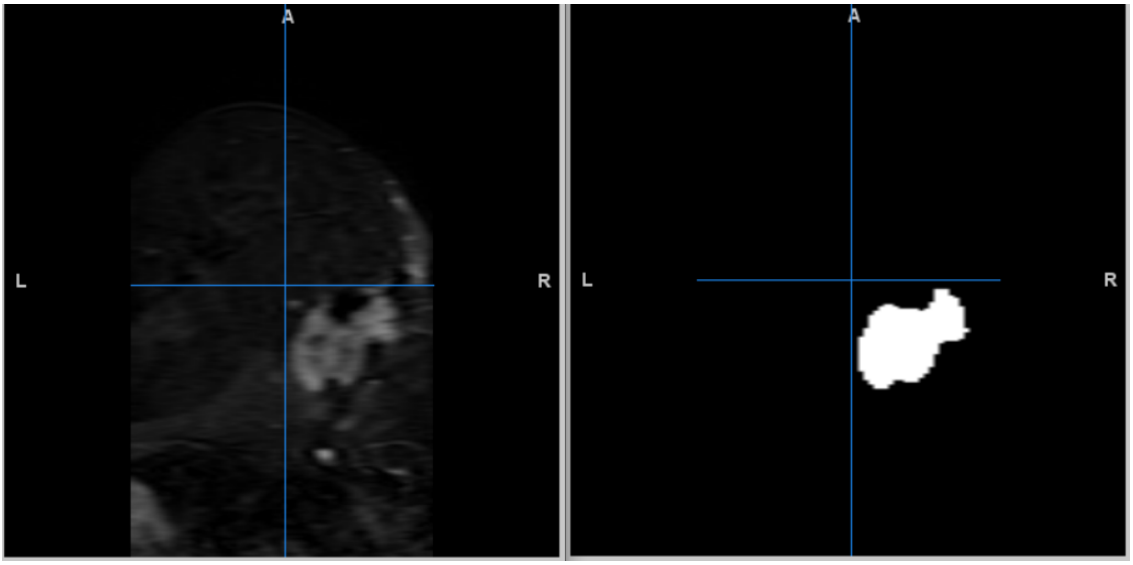

Figura 4.2: Comparación entre una imagen y su predicción

Dichas máscaras siempre se almacenarán en la carpeta **prediction_output** dentro de la carpeta **autosegmentation_utils** y será la ubicación desde la cual se cargarán las imágenes resultantes de las predicciones como ROI.

4.3. Desarrollo Java: Plugin de Mango

El desarrollo del plugin de Mango se benefició significativamente del acceso a ejemplos prácticos proporcionados por Mango Viewer a través de su página oficial. Los recursos disponibles en **mangoviewer.com/develop.html** han servido como herramienta de aprendizaje fundamental durante todo el proceso, acelerando significativamente el desarrollo. Estos recursos incluyen una variedad de ejemplos simples que se descargan junto con las herramientas de desarrollo, los cuales resultaron ser indispensables para comprender y aplicar los conceptos necesarios en la programación del plugin.

Además de los ejemplos proveidos por Mango, la documentación de la API de Mango Viewer ha sido también un documento de referéncia a lo largo del proyecto y varias de las explicaciones de los métodos de la API vienen dadas por esta. Se puede encontrar todo lo referente a la documentación en **mangoviewer.com/api**.

Entrando ya en el desarrollo del plugin, este consta de dos clases de Java.

La primera clase, llamada AutoSegmentation, podría considerarse la clase *main* del plugin. En esta clase se encuentra la "entrada" al resto del código desde Mango y es la encargada de la comunicación entre el propio plugin y el resto de programas ajenos a este. La clase hereda dos interfaces diferentes: *MangoPlugin* y *ConsoleLogger* ambas parte de la API de Mango. *MangoPlugin* es la interfaz que se encarga de dar una puerta de acceso a cualquier tipo de plugin o *add-on* que se quiera desarrollar. El único método de la interzaz es *doOperation* que tiene como parámetros de la función un objeto de tipo *MangoContext* llamado *mango* y otro de tipo *VolumeManager* llamado *viewer*. El método *doOperation* se ejecuta una única vez durante todo el ciclo: cuando el plugin es cargado por primera vez en Mango. Esto permite hacer todas las operaciones necesarias de inicialización nada más cargar el plugin sin necesidad de utilizar otros *listeners* o *event handlers*. En el caso del plugin nada más iniciarse se realizan las siguientes operaciones:

- Revisar si el sistema tiene instalada la versión correcta de Python.
- Crear una instáncia de la clase Predictor (más detalles en 4.4).
- Filtrar los archivos nii.gz (extensión para los archivos nifti) que corresponden a las imágenes para segmentar.
- Cargar la ROI generada en el visor.

Estos pasos son, resumidamente, lo que hace la función doOperation y, por ende, lo que hace el plugin nada más abrirse. En la figura 4.3 se puede ver de forma esquematizada todo el proceso el plugin ejecuta al cargarse desde el menú de "plugins".

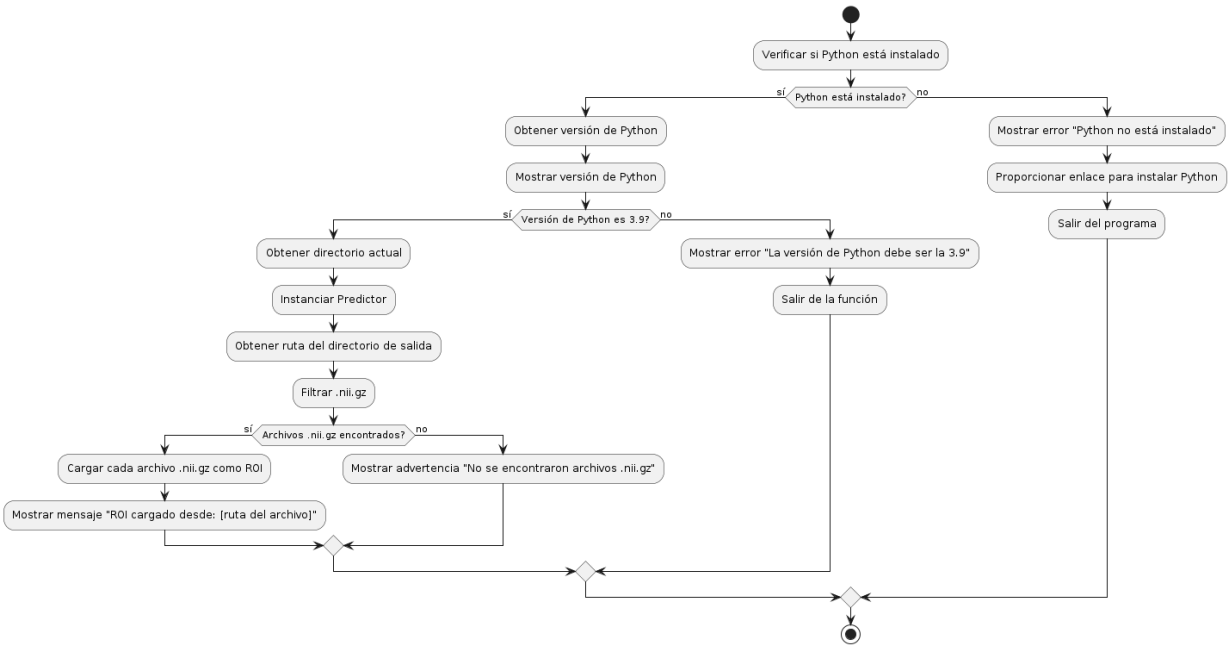

Figura 4.3: Diagrama de la función *doOperation*

Además de los dos elementos dados por parámetro en la función *doOperation*, también ha sido necesario el uso del *ProcessBuilder* de Java para poder ejecutar comandos por la línea de comandos del sistema operativo. Se verá el motivo y la forma de su implementación en la sección 4.4.

La otra función de soporte que se utiliza en la clase *AutoSegmentation*, *checkIfPython*, simplemente utiliza la clase de Java *process builder* para revisar si el sistema tiene instalado Python. Las comprobaciones del sistema de Python se hacen en la función doOperation por lo que checkIfPython solo tiene como cometido dar a la función la versión de Python que el sistema está utilizando.

La otra clase que se ha creado para el plugin se llama *Predictor*. Esta clase es la encargada de implementar la lógica necesaria para activar el modelo de segmentación automática hecho en Python de que se ha hablado anteriormente. Para hacer esto se han utilizado, de nuevo, la clase *ProcessBuilder* de Java así como algunas funcionalidades de *Swing* para generar los elementos gráficos que permiten al usuario interactuar con la aplicación. La clase *Predictor* se ha desarrollado de forma que lo único necesario para que utilizarse las funcionalidades de predicción sea instanciar la clase una única vez. Esto se debe a que las funciones auxiliares que activan el modelo de predicción se encuentran en el constructor de la clase, al cual hay que pasarle como argumento únicamente la ruta absoluta en la que el plugin se encuentra. Se hablará más a fondo de la clase *Predictor* en la sección 4.4 puesto que esta clase resulta la pieza clave para la correcta comunicación entre los dos programas independientes que conforman la herramienta completa.

Por último, la compilación del programa ha requerido del uso de SDK coretto-1.8.0_412 con un nivel de lenguaje 6 al haber sido *Mango Viewer* compilado en una versión compatible con esta. Cualquier intento de generar un fichero JAR utilizando

otras versiones de SDK dan como resultado que el plugin no sea detectado por Mango Viewer y que no se pueda ejecutar las funcionalidades.

4.4. Interoperabilidad entre Python y Java

De forma reiterada se ha mencionado a lo largo de esta memória que la comunicación entre el programa de Java y el modelo de Python es la parte más importante para el correcto funcionamiento de la aplicación en conjunto. Debido a las necesidades de cada aplicación en cada momento de un ciclo de ejecución normal es imperativo que ambos programas operen de forma sincronizada puesto que la ausencia de un recurso en un momento dado puede impedir el correcto funcionamiento del plugin.

Para solucionar el problema con la comunicación entre las dos aplicaciones he decidido utilizar la línea de comandos para poder comunicar ambos programas entre ellos. Al utilizar la línea de comandos es posible activar programas utilizando el propio sistema operativo, así como sincronizar los procesos para poder controlar cuando continuar una ejecución y así no tener problemas relacionados con los recursos necesarios.

La clase encargada de la comunicación entre aplicaciones es, como se ha mencionado en la sección 4.3, la clase *Predictor*.

Mediante el uso de la línea de comandos (gracias a la clase *ProcessBuilder* de Java) ha sido posible tanto la activación del modelo de Python a través de un programa de Java como la comprobación de que los elementos necesarios para el correcto funcionamiento del programa se se encuentren correctamente instalados en el dispositivo dónde se va a utilizar la aplicación.

La clase *Predictor* se trata de un objeto que, nada más instanciarse, intenta generar una predicción mediante el modelo de ML sin la necesidad de llamar a otras funciones en el código. Esto es debido a que el constructor de la clase es la única función pública con la que cuenta *Predictor*, siendo los otros dos métodos de la clase privados y de apoyo para el constructor.

La primera función auxiliar a la que se llama cuando se genera una instáncia de *Predictor* es *checkIfDirectoryExist*. Se trata de una función privada booleana que simplemente devuelve **true** en caso que el *path* en el que se están intentando buscar las imágenes para predecir y false en caso contrario. La comprovación de la ruta se realiza mediante el uso de la clase de Java *File* y su función asociada *directoryExists*.

La otra función auxiliar es la encargada de ejecutar el script de Python expuesto en 4.2: *executePythonProgram*.

Primero, la función define el comando que se debe ejecutar. Este comando incluye la ruta al intérprete de Python dentro del entorno virtual y el script Python que se va a ejecutar junto con sus argumentos. En este caso, el comando es:

```
autosegmentation_utils\\venv\\Scripts\\
python\\autosegmentation_utils\\predict.py -p autosegmentation_utils\\
nnUnet_raw\\ Dataset001_Predict\\imagesTs -i 001
```

Éste ejecuta el script predict.py con la ruta de las imagenes y el id correspondiente a la carpeta donde se almacenarán.

Luego, se utiliza la clase *ProcessBuilder* para crear un proceso que ejecutará el comando en una consola de comandos (cmd.exe en Windows). El comando se pasa como un argumento a cmd.exe con la opción /c para que se ejecute y luego la consola se cierre. Se configura *ProcessBuilder* para redirigir el flujo de error estándar (stderr) al flujo de salida estándar (stdout), lo que permite combinar ambas salidas y leerlas juntas sin tener que diferenciar entre errores y salidas de texto normales.

El siguiente paso es iniciar el proceso con *processBuilder.start()*. Esto lanza el comando en una nueva instancia de la consola. Para capturar la salida del proceso, se crea un BufferedReader que lee el flujo de entrada del proceso (*process.getInputStream()*). Se leen todas las líneas de salida del proceso y se registran utilizando *AppLogger.info* para poder mostrarlas por la consola de Mango Viewer.

Una vez que se han leído todas las líneas, se asegura de cerrar el *BufferedReader* en un bloque *finally* asociado al *try* que envuelve la creación del *buffer* para evitar fugas de recursos. Además, se maneja cualquier posible excepción que ocurra durante el cierre del *BufferedReader*.

Después, la función espera a que el proceso termine con *process.waitFor()*, lo que devuelve el código de salida del proceso. Este código de salida se utiliza para determinar si el comando se ejecutó exitosamente o si hubo algún error. Este paso es crucial y es lo que permite el correcto funcionamiento de la aplicación en conjunto ya que sin esta instrucción el plugin de Mango avanzaría al mismo tiempo que el modelo de predicción se ejecuta, lo que provocaría que al intentar cargar la ROI esta no se encuentre disponible debido a la diferencia que hay entre el tiempo en que se ejecutan las dos instrucciones encargadas de ejecutar el modelo y cargar la máscara generada y el tiempo que tarda en generarse dicha máscara. Si el modelo de Python se ejecuta de forma exitosa entonces la ejecución en Java continúa y se muestra un cuadro de texto que informa al usuario que el programa se ha ejecutado exitosamente, en caso contrario se muestra un mensaje de error con el código de salida correspondiente.

En la figura 4.4 se puede ver un pequeño diagrama de como funcionaría una ejecución normal del plugin. En la imagen se puede apreciar que parte de la aplicación se encarga de que, como se sincronizan y comunican y que hace cada una de las partes en cada momento. Se puede apreciar como en ningún momento el usuario que está utilizando el plugin interactua de ninguna forma con el modelo de segmentación y que es el propio plugin de Java el que se encarga de la transmisión de información entre lo que ve el usuario y lo que el modelo genera.

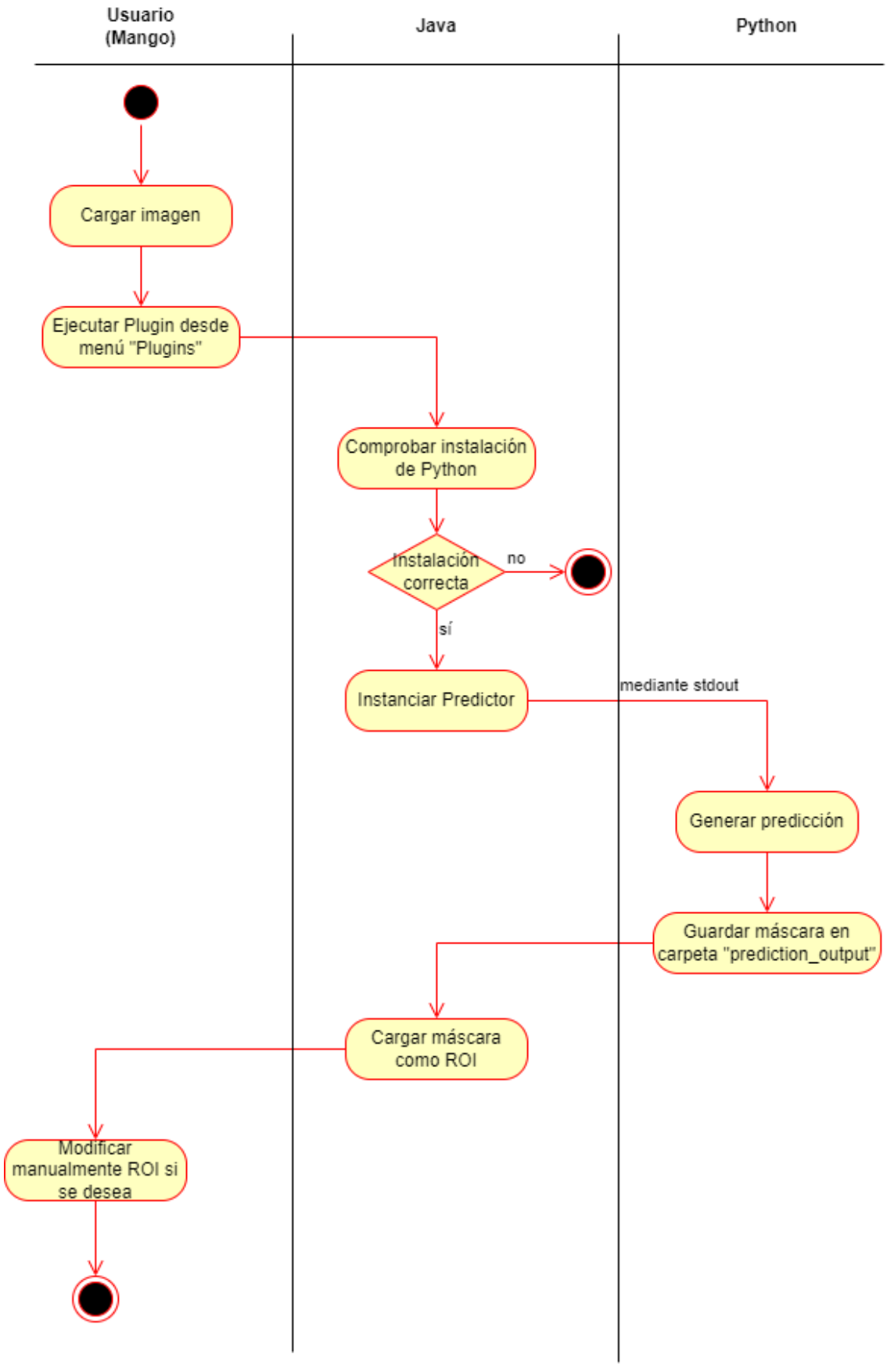

Figura 4.4: Diagrama de flujo de la ejecución del plugin

4.5. Pruebas y resultados

4.5.1. Ejecución estándar del programa

Para ilustrar los resultados se utilizará la imagen 1104 del dataset I-SPY [25].

Para presentar los resultados del proyecto, se seguirá el proceso estándar que un usuario experimentaría al interactuar con la aplicación. Se incluirán imágenes ilustrativas para proporcionar una representación visual de cómo se vería la aplicación en uso.

La primera funcionalidad que se ha incluido es la capacidad de abrir imágenes desde la aplicación. En la figura 4.5 se puede ver la pantalla que aparece cuando un usuario quiere abrir una imagen. Recalcar que para que la imagen pueda detectarse para la segmentación esta debe estar dentro de la carpeta de *imagesTs* como se ha comentado en 4.1 para que la aplicación funcione correctamente y el modelo pueda encontrar las predicciones.

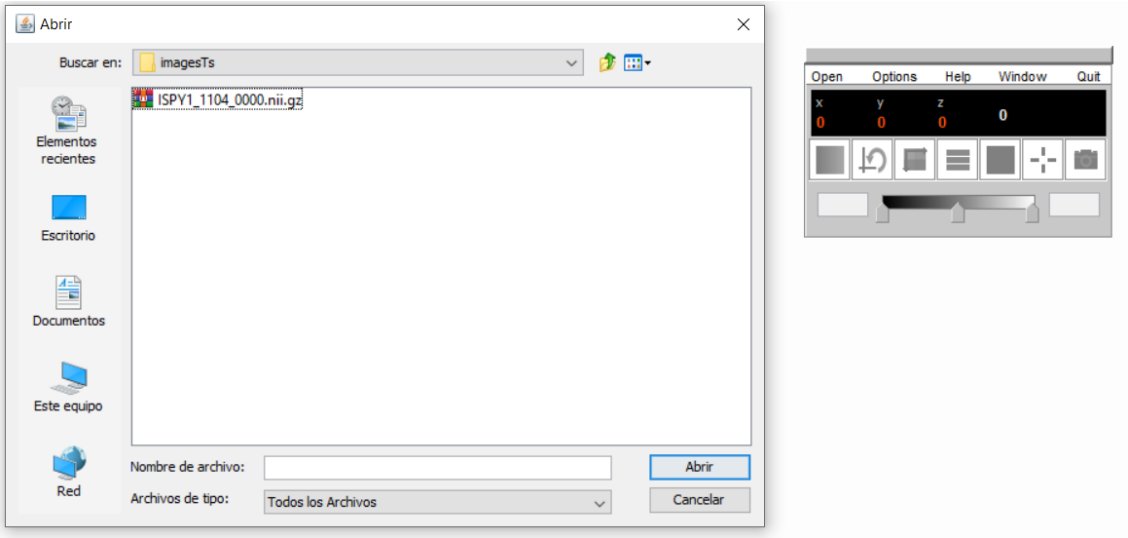

Figura 4.5: Selector de imagenes del plugin

El programa abre las imágenes sin problema cuando se seleccionan. En la figura 4.6 se puede ver como aparece la imagen base, sin ROI ni alteraciones, en el visor de Mango.

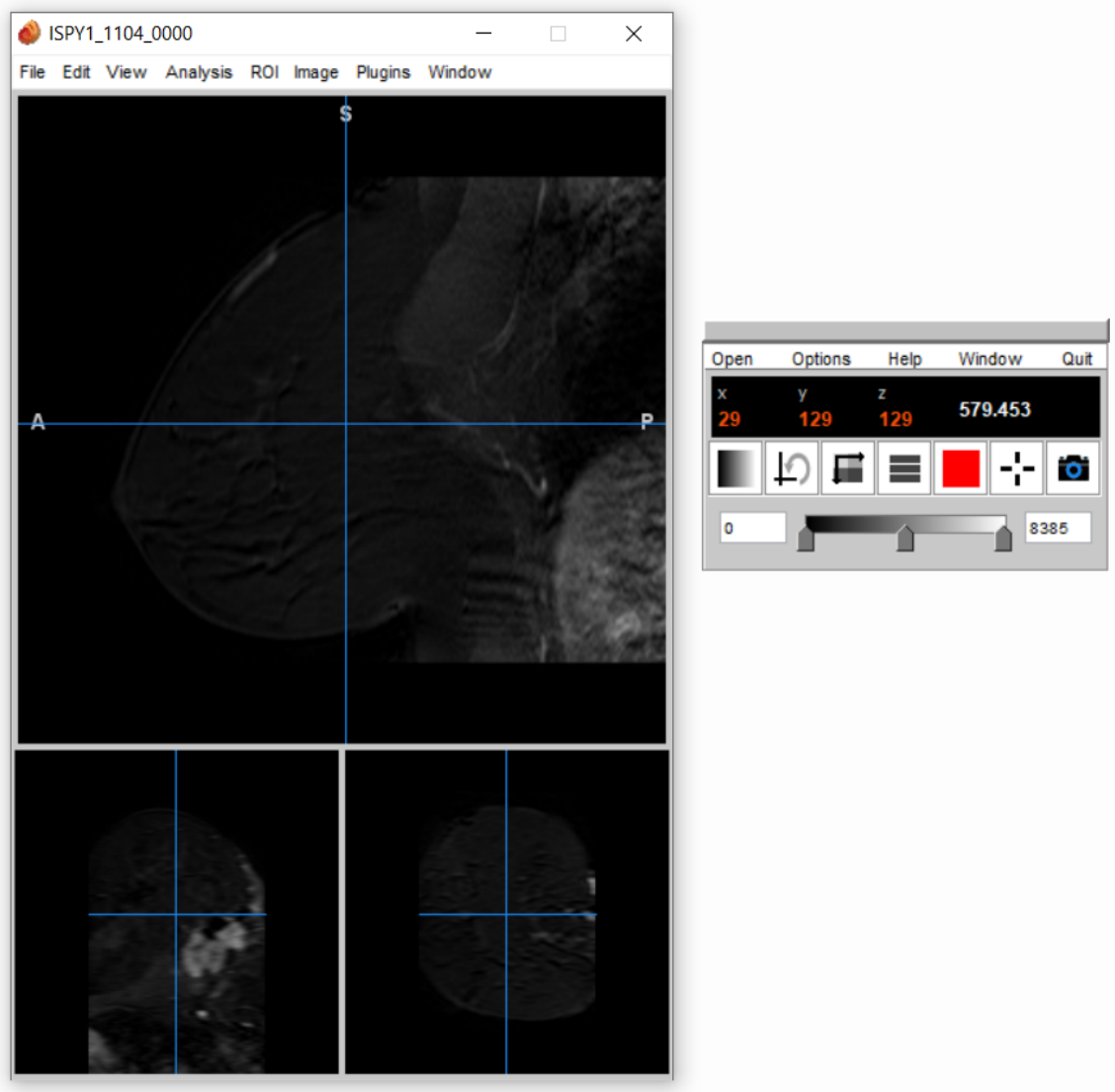

Figura 4.6: Imagen base sin ninguna ROI superpuesta. Esto es lo que el usuario ve al abrir una imagen cualquiera.

Para abrir el plugin, una vez está cargado, lo único que se debe hacer es hacer click a la opción de *Plugin* dentro del visor de imágenes. El nombre con el que se puede identificar al plugin es *Automatic semantic segmentation plugin*. Una vez se aprieta este recuadro todo el proceso explicado a lo largo del apartado 4.5 se pone en marcha y toda la lógica del plugin se empieza a ejecutar.

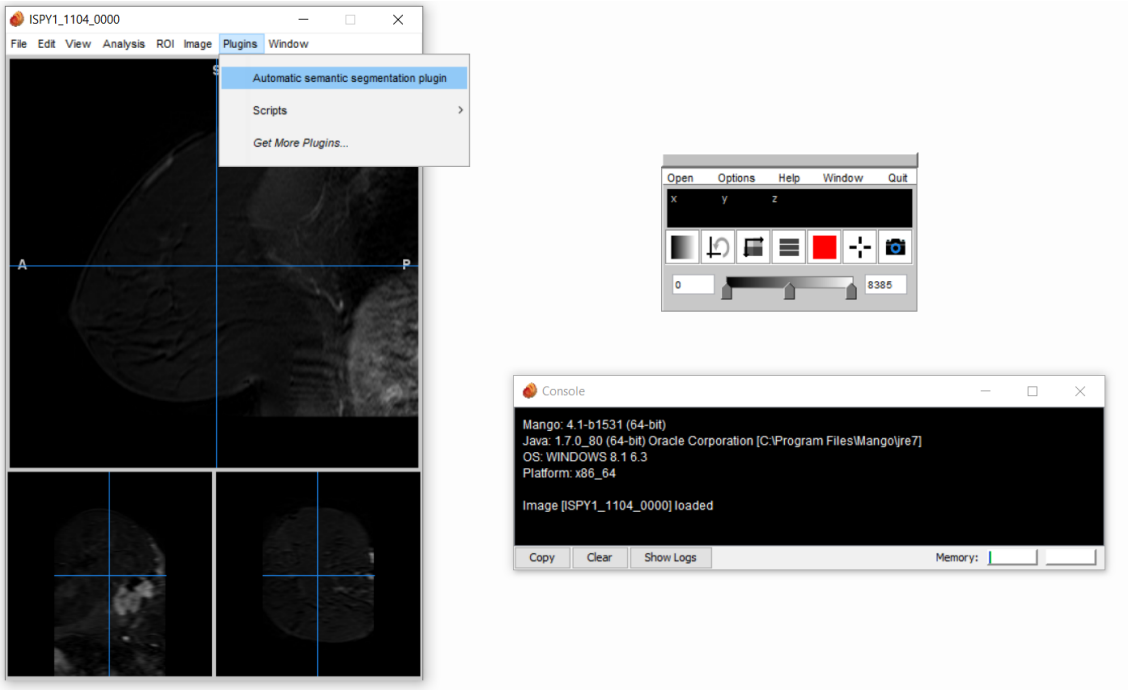

Figura 4.7: Opción que se debe seleccionar para abrir el plugin.

Durante las imágenes mostradas a lo largo de esta sección también se mostrará la consola de Mango. La consola de Mango, a pesar de haber sido un elemento importante a la hora de poder hacer *debug* del código y el sitio escogido para mostrar los errores y avisos del código conforme este se va ejecutando, no se abre por defecto al abrir la aplicación sino que se debe abrir activamente desde el propio menú de *Window* → *Show Console*. Al darle a la opción de la consola se abrirá una terminal como la que se puede ver en la figura 4.8. He decidido poner el output del modelo a través de la consola por lo que el progreso, los errores y la información referente al programa de Python que se ha explicado en la sección 4.2 se puede ver a través de esta.

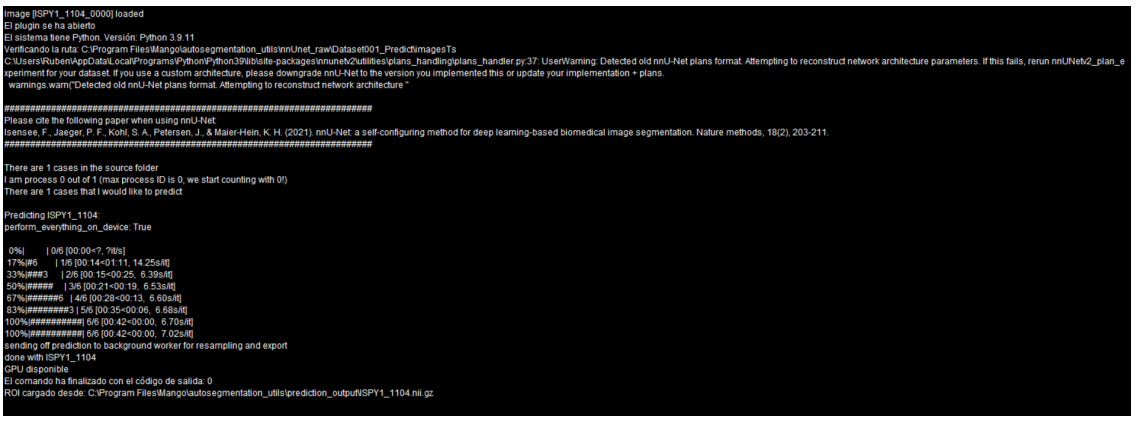

Figura 4.8: *Logs* de una ejecución correcta. Se pueden apreciar algunos mensajes de debug como el de "GPU disponible"pertenecientes al programa de Python.

Una vez el modelo ha finalizado, el programa crea un cuadro de texto informativo como de la figura 4.9 donde se informa al usuario de que la ejecución del modelo ha terminado. Una vez se cierra el cuadro de texto la ROI generada se carga en la imagen.

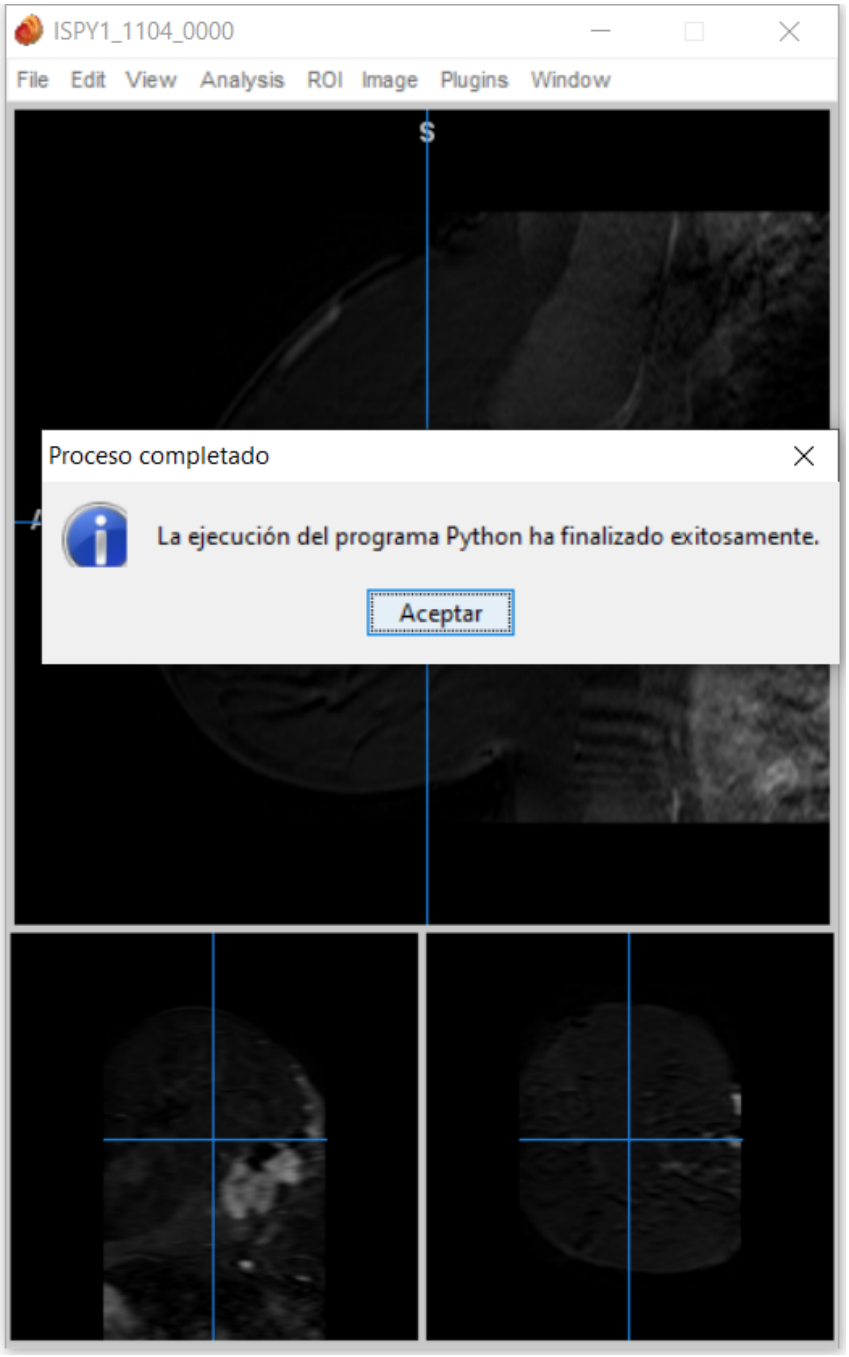

Figura 4.9: Aviso de finalización del modelo.

Al cargarse la máscara como una ROI como la que se puede ver en la figura 4.10 en lugar de como una imagen solapada encima de la que se ha abierto originalmente el usuario puede modificar y editar la segmentación hecha por el modelo de forma

completamente libre a través de la sección *ROI* del visor de Mango

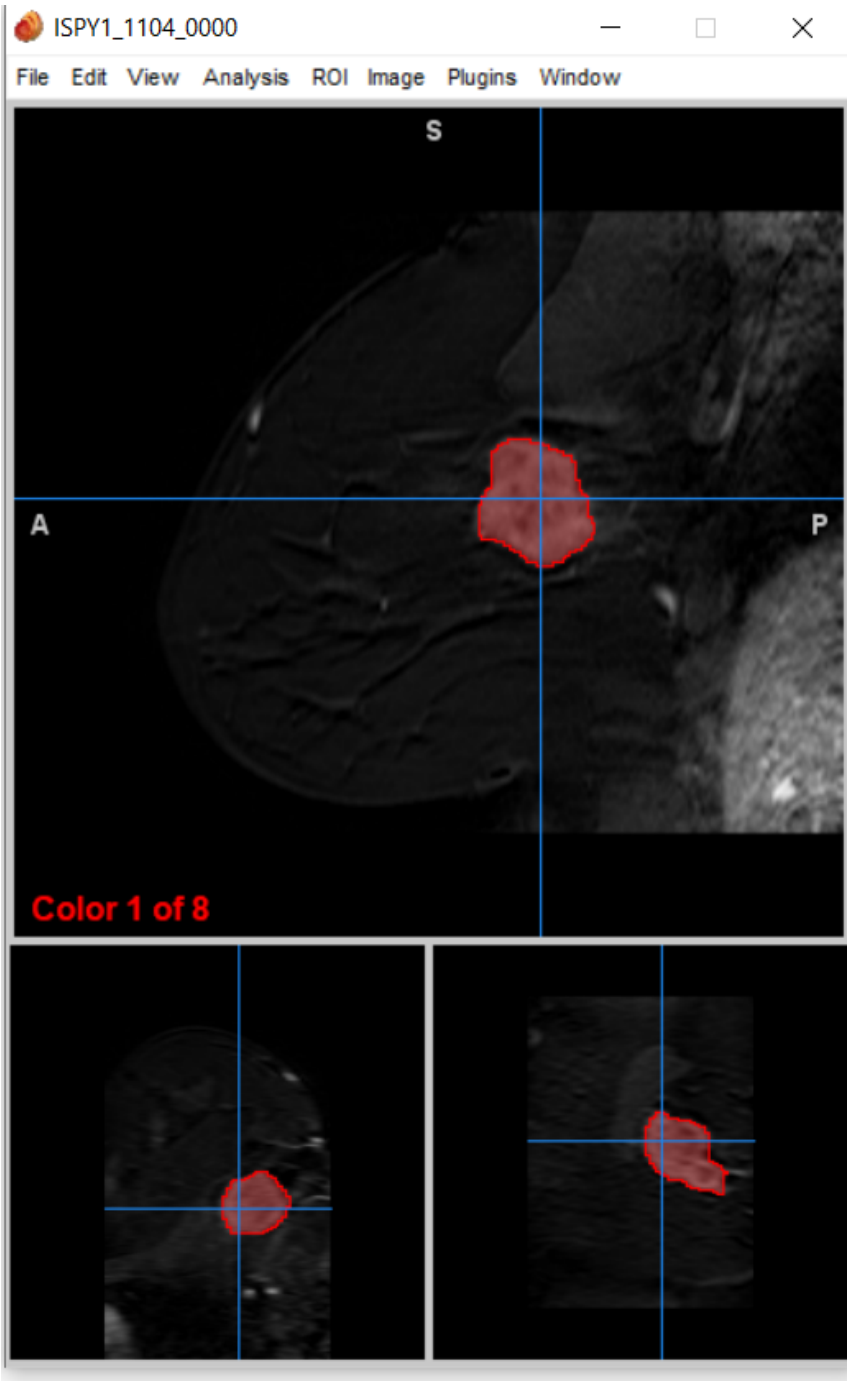

Figura 4.10: Resultado final de una ejecución normal.

4.5.2. Resultados

A continuación se mostrarán algunos resultados de aplicar el plugin a varias de las imágenes del *Duke-Breast-Cancer-MRI dataset*. En el pie de foto se mostrará el número de la imagen junto con el tiempo total y el tiempo que le toma al modelo

terminar cada una de las iteraciones necesarias para hacer la predicción.

Estos datos han sido obtenidos de la consola mostrada en el apartado 4.5 por lo que el tiempo total se empieza a contar desde el momento en el que se aprieta el botón que activa la segmentación hasta el momento en el que aparece el cuadro de texto informando de la finalización del código.

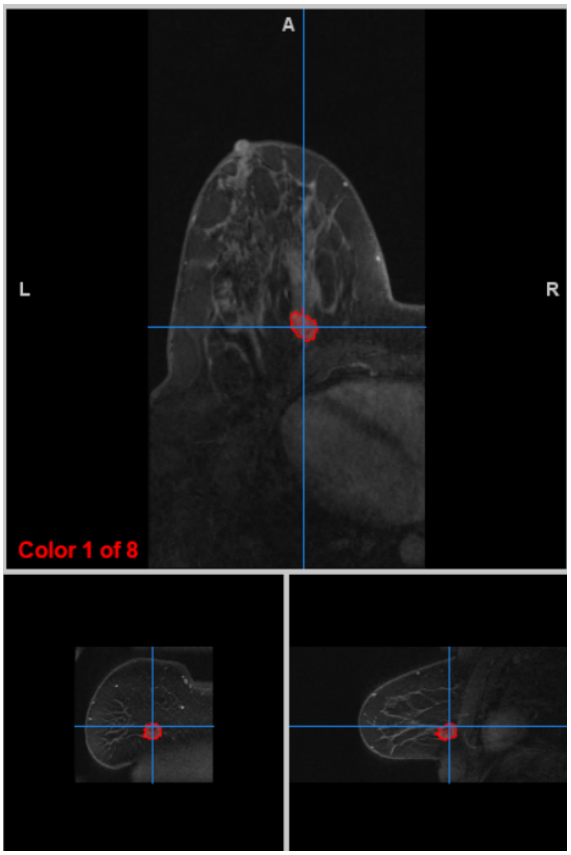

Figura 4.11: Imagen 1: Tiempo por iteración:1.41s, Tiempo total 38s

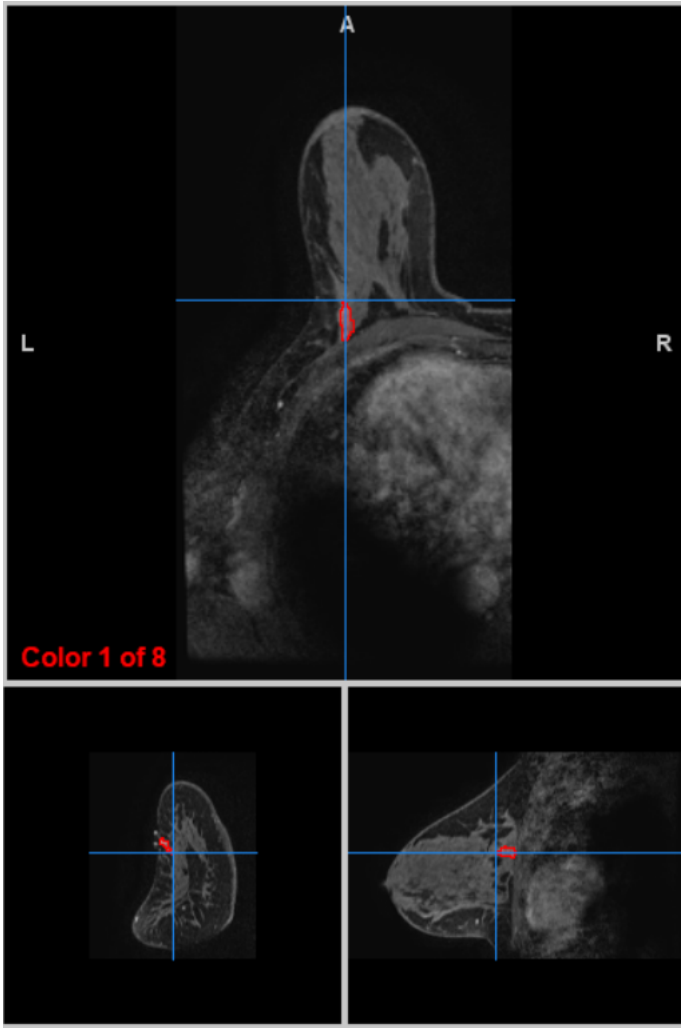

Figura 4.12: Imagen 2: Tiempo por iteración:1.43s, Tiempo total 17s

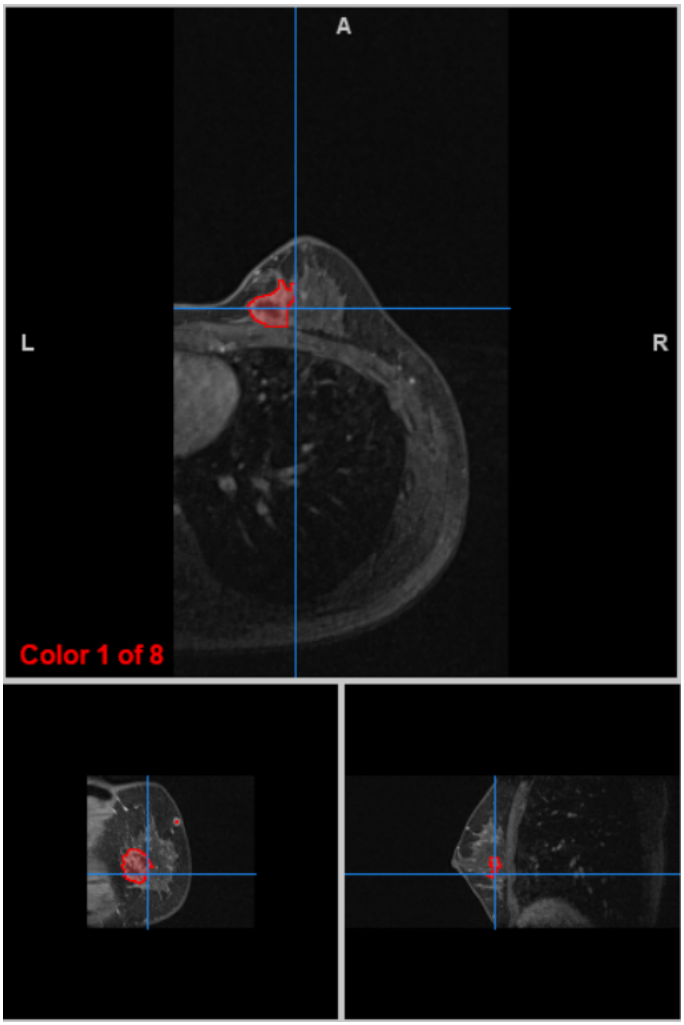

Figura 4.13: Imagen 5: Tiempo por iteración:1.33s, Tiempo total 36s

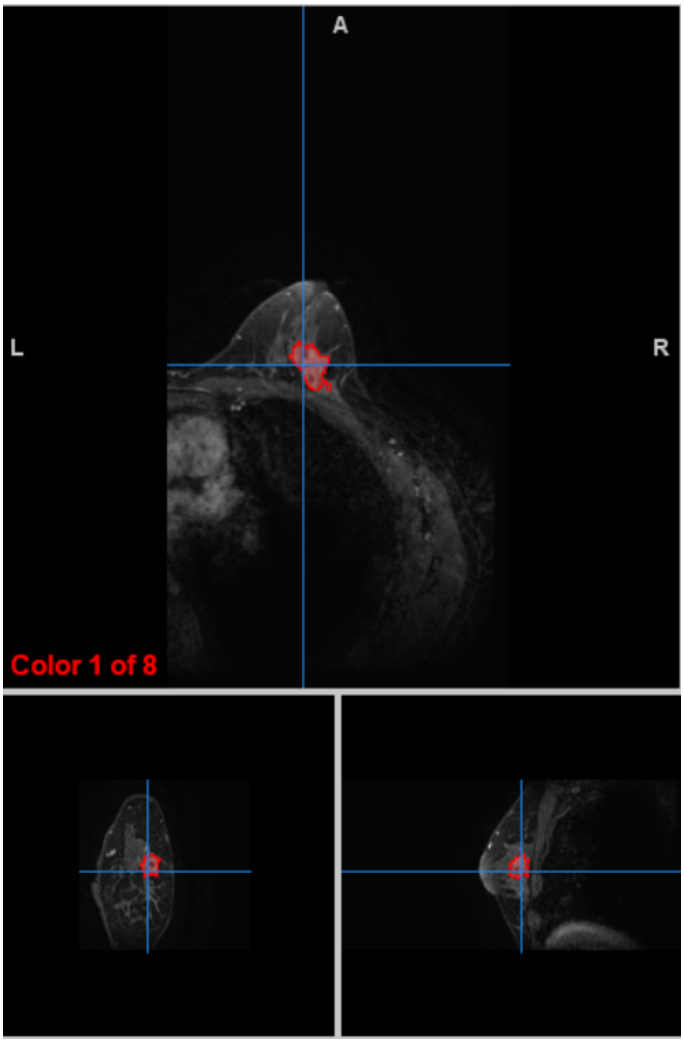

Figura 4.14: Imagen 9: Tiempo por iteración:1.33s, Tiempo total 35s

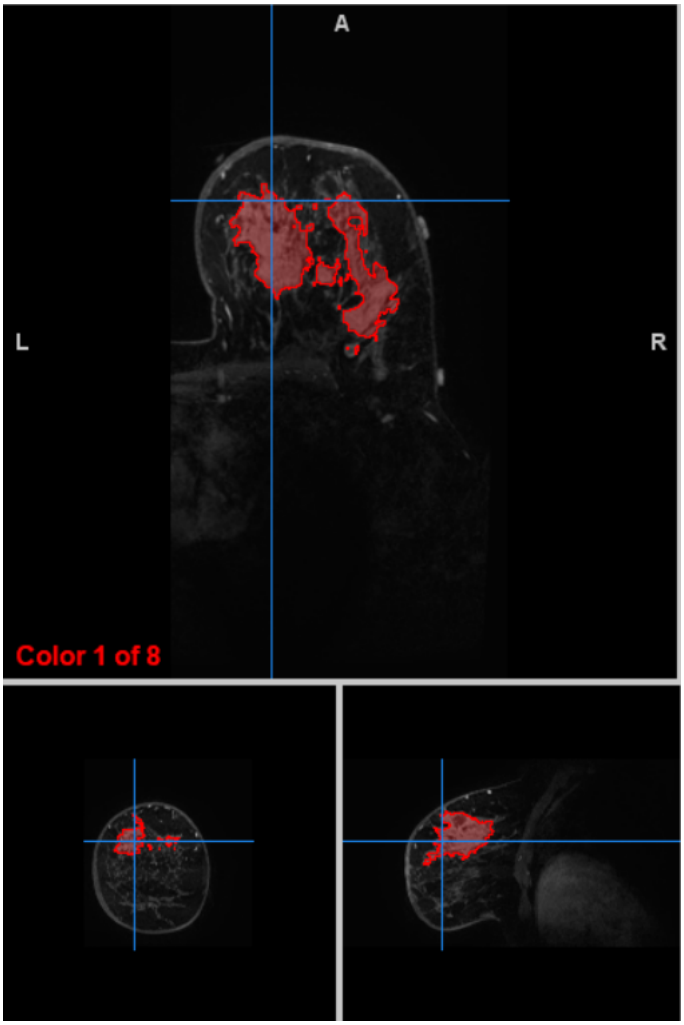

Figura 4.15: Imagen 12: Tiempo por iteración:1.32s, Tiempo total 47s

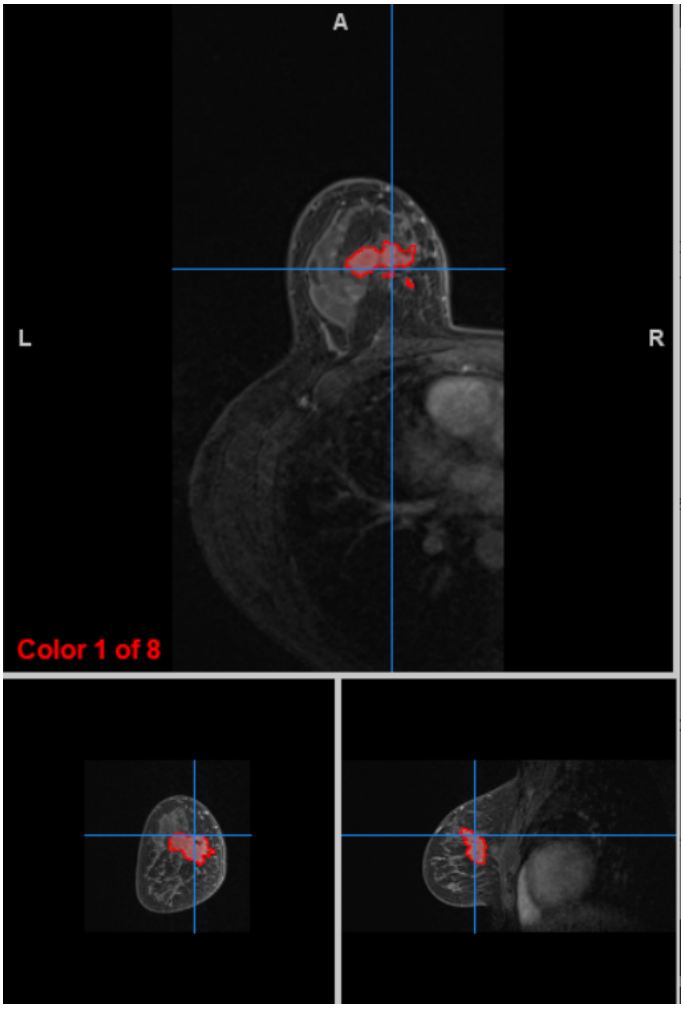

Figura 4.16: Imagen 19: Tiempo por iteración:1.33s, Tiempo total 36s

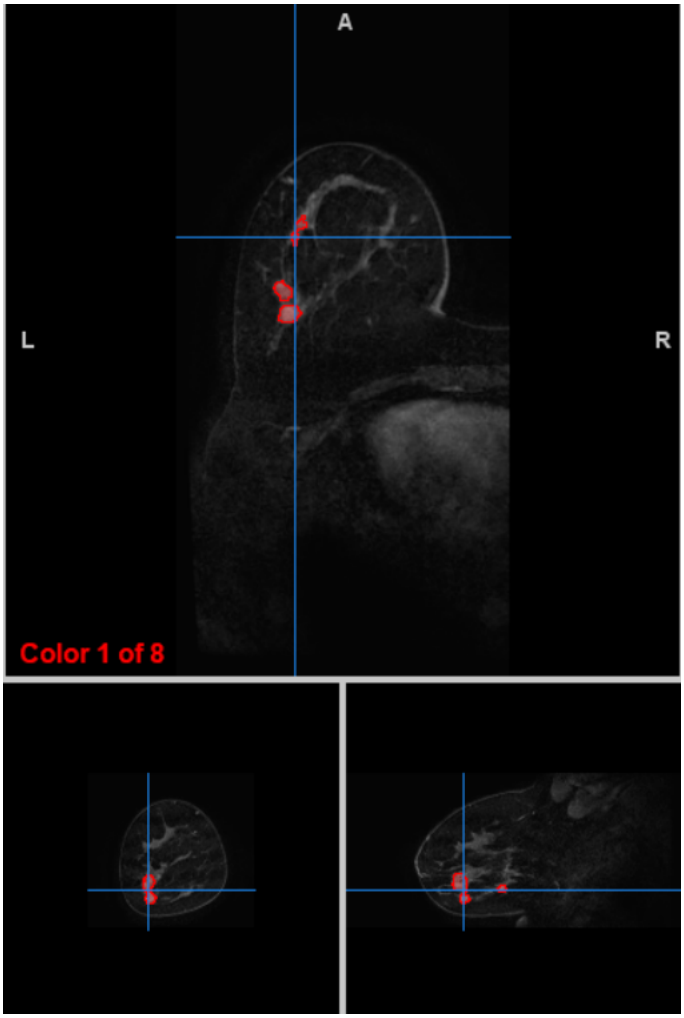

Figura 4.17: Imagen 21: Tiempo por iteración:1.34s, Tiempo total 36s

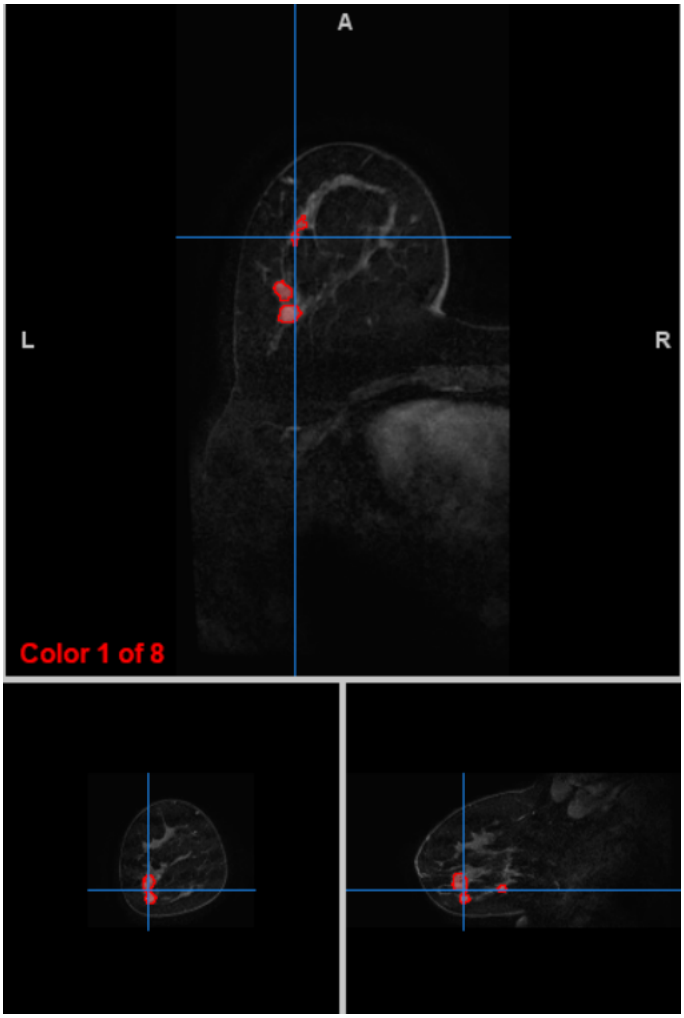

Figura 4.18: Imagen 21: Tiempo por iteración:1.34s, Tiempo total 36s

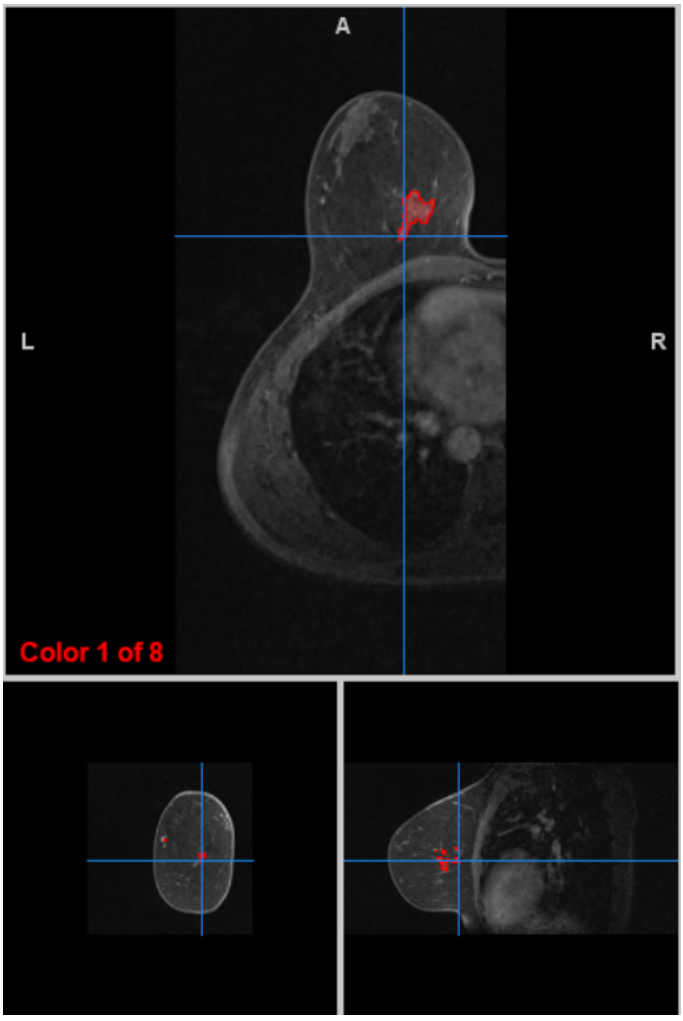

Figura 4.19: Imagen 48: Tiempo por iteración:1.34s, Tiempo total 36s

Los resultados de las ejecuciones muestran una mediana de tiempo de aproximadamente 36 segundos. Esto es una reducción del tiempo considerable en comparación a lo que suele tardar el proceso regular realizado por humanos que suele medirse en minutos.

A partir de estos resultados se puede confirmar, además, que el plugin funciona correctamente además en diversos tipos de imágenes e independientemente de las formas que presentan las mamas que se deben segmentar.

4.5.3. Ejecución GPU y CPU

Como se ha comentado en las secciones 3.2.2 y 4.2, el uso de una GPU es altamente recomendado al trabajar con la nnU-net v2, debido a las significativas mejoras en rendimiento y eficiencia que proporciona. Aunque el uso de una GPU es opcional y cualquier modelo puede funcionar con una CPU, las diferencias en los tiempos de procesamiento y el rendimiento general son notables.

Para las siguientes pruebas se ha utilizado la imagen 356 del *dataset* Duke-Breast-Cancer-MRI[43] el cúal ha sido también utilizado como se ha mencionado en la sección 3.2.2. El número de programas abiertos mientras se ejecutaban ambas pruebas era el mismo, se ha utilizado el mismo equipo para ambas y entre pruebas se ha eliminado la memoria caché del equipo mediante un reinicio para evitar cualquier tipo de interferencia o influencia externa en los resultados.

A pesar de que las *CPU* son perfectamente capaces de ejecutar modelos de machine learning incluyendo la nnU-net v2, dicha capacidad viene acompañada de una considerable lentitud en el procesamiento. Para este proyecto el equipo con el que se contaba disponía de un procesador *11th Gen Intel(R) Core(TM) i5-11400H @ 2.70GHz*. Al utilizar esta CPU específica, el tiempo total de procesamiento para la ejecución completa del proceso de predicción es de 14 minutos y 46 segundos. Este tiempo puede ser una limitación importante si, como en este proyecto, el objetivo de implementar un modelo de ML es ahorrar tiempos o acelerar procesos. En la figura 4.20 se puede ver el resultado de hacer una predicción de la imagen utilizada para ambos ejemplos.

```
Predicting Breast_MRI_356:
perform_everything_on_device: False
100%|██████████| 27/27 [14:46<00:00, 32.85s/it]
sending off prediction to background worker for resampling and export
done with Breast_MRI_356
```

Figura 4.20: Tiempo de predicción utilizando una CPU

En comparación, al utilizar una GPU el tiempo necesario para completar la predicción se reduce drásticamente. En el equipo utilizado contaba con una Nvidia GeForce RTX 3050 laptop GPU, una tarjeta gráfica que, a pesar de ser de gama media-baja, completa la predicción en tan solo 37 segundos como puede apreciarse en la figura 4.21. Esta diferencia es debida a la arquitectura de las GPUs, que está optimizada para realizar cálculos paralelos masivos, lo cual es ideal para las operaciones requeridas en los modelos de machine learning como lo es el de este proyecto.

```
Predicting Breast_MRI_356:
perform_everything_on_device: True
100%|██████████| 27/27 [00:37<00:00,  1.39s/it]
sending off prediction to background worker for resampling and export
done with Breast_MRI_356
```

Figura 4.21: Tiempo de predicción utilizando una GPU

Esta drástica reducción en el tiempo de procesamiento se refleja también en el tiempo medio por iteración. Con la CPU, cada iteración toma aproximadamente

32.85 segundos, mientras que con la GPU, este tiempo se reduce a solo 1.39 segundos.

Además, trabajar con una GPU como la Nvidia GeForce RTX 3050 no solo mejora el tiempo de inferencia, sino que también mejora considerablemente el tiempo que tarda un modelo de ML en ser entrenado.

Las conclusiones que se sacan de esta comparación en cuanto al proceso tanto de desarrollo como al propio uso del plugin son evidentes: aunque es posible utilizar una CPU para trabajar con la nnU-net v2, el uso de una GPU es claramente beneficioso y recomendado. La significativa mejora en los tiempos de procesamiento y en la eficiencia general del modelo justifica la inversión en hardware más especializado, especialmente cuando se busca aumentar la productividad y velocidad a la cual se realizan las anotaciones.

Capítulo 5

Conclusiones

Después de completar el proyecto y realizar múltiples pruebas con diversas imágenes, se puede afirmar que se ha logrado el desarrollo e implementación exitosa de una herramienta "plug & play"para la aplicación Mango Viewer, permitiendo la segmentación semántica automática de tumores de mama.

Además, debido al breve tiempo requerido por la aplicación para llevar a cabo la segmentación, también se ha conseguido reducir significativamente el tiempo necesario para realizar anotaciones en imágenes médicas y seleccionar las regiones de interés (ROI).

Aunque no se ha podido confirmar la facilidad de integración de la aplicación en un entorno clínico real debido a la falta de pruebas con usuarios, se ha alcanzado el objetivo principal del proyecto de manera satisfactoria. Asimismo, dos de los objetivos secundarios, la reducción del tiempo de anotación y la facilidad de uso del plugin, han sido cumplidos exitosamente.

5.1. Problemas encontrados

Durante el desarrrollo del proyecto han habido varios problemas asociados, especialmente, a la creación del modelo de predicción.

El problema más evidente durante el desarrollo del trabajo ha sido el equipo con el que contaba. Actualmente cuento con un portatil con una tarjeta gráfica RX 3050 Laptop de 4GB de RAM asociada. Esta pieza no es lo suficientemente potente ni tiene la suficiente memoria como para poder entrenar un modelo de predicción mediante la *nnU-net v2* en un tiempo razonable debido al reducido margen de tiempo en el que se tenía que desarrollar el proyecto.

Para solucionar esto se ha optado por utilizar un modelo pre-entrenado,como se ha mencionado anteriormente, cedido por Smitri Joshi del grupo de investigación BCN-AIM.

Sin embargo cabe destacar que el entrenamiento de un modelo propio era una parte planificada del proyecto. El problema con esto, como se puede ver en la figura

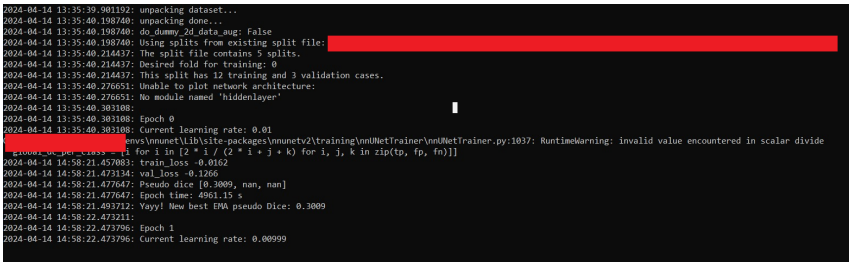

Figura 5.1: Se puede ver que el tiempo de un ciclo de entrenamiento del modelo ha sido de 4961.15 esgundos o alrededor de 1 hora y 22 minutos.

5.1, fue la gran cantidad de tiempo que esto habría requerido dados mis recursos. Se estuvieron barajando opciones como adquirir un VPS (servidor virtual privado) para poder entrenar el modelo de IA en la nube pero el alto coste de este tipo de servicios en caso de requerir tarjetas gráficas terminó en la decisión de utilizar un modelo ya existente.

La gran cantidad de tiempo que supuso el no poder entrenar el modelo y estar .[a]tascado.[en] esa parte del proyecto desencadenó en una considerable reducción del tiempo disponible para el desarrollo del plugin en sí. Esto resultó en que varias funcionalidades deseadas terminasen no siendo implementadas, lo que impactó en la funcionalidad y completitud del proyecto final. Sin embargo, y por suerte, esta falta de tiempo no afectó al cumplimiento del objetivo principal.

5.2. Trabajo futuro

Como trabajo a futuro en caso de continuar el desarrollo del plugin se podría considerar la mejora tanto del modelo de predicción como la interacción del usuario con el plugin.

Ofrecer la posibilidad de segmentar varias a la vez pero solamente seleccionar una para la visualización e implementar más podría aumentar la eficiencia del proceso de anotación.

Además la posibilidad de probar el plugin con usuarios podría dar pie a cumplir el objetivo de implementar el plugin en un entorno cl

Referencias

[1] M. Aljabri et al. "Towards a better understanding of annotation tools for medical imaging: a survey". En: *Multimedia Tools and Applications* 81.18 (2022), págs. 25877-25911. DOI: 10.1007/s11042-022-12100-1. URL: https://doi.org/10.1007/s11042-022-12100-1.

[2] Estela Vega Alonso. *Mamografías: Prevención del Cáncer de Mama*. https://www.contraelcancer.es/es/todo-sobre-cancer/tipos-cancer/cancer-mama/prevencion/mamografias. Revisado en septiembre de 2023, Unidad de cáncer de mama, Centro integral Oncológico HM Clara Campal, Madrid. Accessed: 2024-06-08. 2023.

[3] Divyanshu Anand. *Partitional Clustering*. Published in Analytics Vidhya, July 4, 2020. 2020. URL: https://www.analyticsvidhya.com/blog/2020/07/partitional-clustering/.

[4] Scott L Andresen. "John McCarthy: father of AI". En: *IEEE Intelligent Systems* 17.5 (2002), págs. 84-85.

[5] Carlo Migel Bautista et al. "Convolutional neural network for vehicle detection in low resolution traffic videos". En: *2016 IEEE Region 10 Symposium (TENSYMP)*. 2016, págs. 277-281. DOI: 10.1109/TENCONSpring.2016.7519418.

[6] John Canny. "A computational approach to edge detection". En: *IEEE Transactions on pattern analysis and machine intelligence* 6 (1986), págs. 679-698.

[7] Anshumaan Chauhan, Siddhartha Bhattacharyya y S. Vadivel. *DQNAS: Neural Architecture Search using Reinforcement Learning*. 2023. arXiv: 2301.06687 [cs.LG].

[8] *Cityscapes Dataset*. https://www.cityscapes-dataset.com/examples/. 2024.

[9] Cloudinary. *Region-based Segmentation*. https://cloudinary.com/glossary/region-based-segmentation. Last accessed: April 2024.

[10] Patel Dhruv y Subham Naskar. "Image Classification Using Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN): A Review". En: *Machine Learning and Information Processing*. Ed. por Debabala Swain, Prasant Kumar Pattnaik y Pradeep K. Gupta. Singapore: Springer Singapore, 2020, págs. 367-381. ISBN: 978-981-15-1884-3.

[11] Oliver Diaz et al. "Data preparation for artificial intelligence in medical imaging: A comprehensive guide to open-access platforms and tools". En: *Physica Medica* 83 (2021), págs. 25-37. ISSN: 1120-1797. DOI: https://doi.org/10.1016/j.ejmp.2021.02.007. URL: https://www.sciencedirect.com/science/article/pii/S1120179721000958.

[12] Bradley J Erickson et al. "Machine learning for medical imaging". En: *radiographics* 37.2 (2017), págs. 505-515.

[13] J. Freixenet et al. "Yet Another Survey on Image Segmentation: Region and Boundary Information Integration". En: *Computer Vision — ECCV 2002*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, págs. 408-422. ISBN: 978-3-540-47977-2.

[14] Shijie Hao, Yuan Zhou y Yanrong Guo. "A Brief Survey on Semantic Segmentation with Deep Learning". En: *Neurocomputing* 406 (2020), págs. 302-321. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2019.11.118. URL: https://www.sciencedirect.com/science/article/pii/S0925231220305476.

[15] IBM. *Machine Learning*. https://www.ibm.com/topics/machine-learning. Consultado en junio de 2024. 2021.

[16] Fabian Isensee. *nnU-Net: Installation Instructions*. https://github.com/MIC-DKFZ/nnUNet/blob/master/documentation/installation_instructions.md. Last edited 8 months ago. 2023. (Visitado 06-2023).

[17] Fabian Isensee et al. "nnU-Net: a self-configuring method for deep learning-based biomedical image segmentation". En: *Nature Methods* 18.2 (2021), págs. 203-211.

[18] Dilpreet Kaur y Yadwinder Kaur. "Various image segmentation techniques: a review". En: *International Journal of Computer Science and Mobile Computing* 3.5 (2014), págs. 809-814.

[19] Tsung-Wei Ke et al. "Unsupervised Hierarchical Semantic Segmentation with Multiview Cosegmentation and Clustering Transformers". En: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022.

[20] A. M. Khan y Ravi. S. "Image Segmentation Methods: A Comparative Study". En: *International Journal of Soft Computing and Engineering (IJSCE)* 3.4 (sep. de 2013). Manuscript received on August 02, 2013. Revised Manuscript received on August 25, 2013. Manuscript published on September 05, 2013., págs. 84-93. ISSN: 2231-2307. URL: http://www.ijsce.org/.

[21] Steve Lawrence, C Lee Giles y Ah Chung Tsoi. "Lessons in neural network training: Overfitting may be harder than expected". En: *Aaai/iaai*. 1997, págs. 540-545.

[22] RICH LEE e ING-YI CHEN. "The Time Complexity Analysis of Neural Network Model Configurations". En: *2020 International Conference on Mathematics and Computers in Science and Engineering (MACISE)*. 2020, págs. 178-183. DOI: 10.1109/MACISE49704.2020.00039.

[23] Jiaxing Li et al. "Facial Expression Recognition with Faster R-CNN". En: *Procedia Computer Science* 107 (2017). Advances in Information and Communication Technology: Proceedings of 7th International Congress of Information and Communication Technology (ICICT2017), págs. 135-140. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2017.03.069. URL: https://www.sciencedirect.com/science/article/pii/S1877050917303447.

[24] Li Li et al. "Design Patterns and Extensibility of REST API for Networking Applications". En: *IEEE Transactions on Network and Service Management* 13.1 (2016), págs. 154-167. DOI: 10.1109/TNSM.2016.2516946.

[25] Wen Li et al. *I-SPY 2 Breast Dynamic Contrast Enhanced MRI Trial (ISPY2) (Version 1) [Data set]*. https://doi.org/10.7937/TCIA.D8Z0-9T85. The Cancer Imaging Archive. 2022. DOI: 10.7937/TCIA.D8Z0-9T85.

[26] Morgan Lynch. "Neural Networks with Multiple Data Sources: How to design a neural network with inputs from multiple data sources using Tensorflow". En: *Towards Data Science* (ene. de 2023). Published in Towards Data Science, 5 min read. URL: https://towardsdatascience.com/neural-networks-with-multiple-data-sources-ef91d7b4ad5a.

[27] I.N. Manousakas et al. "Split-and-Merge Segmentation of Magnetic Resonance Medical Images: Performance Evaluation and Extension to Three Dimensions". En: *Computers and Biomedical Research* 31.6 (1998), págs. 393-412. ISSN: 0010-4809. DOI: https://doi.org/10.1006/cbmr.1998.1489. URL: https://www.sciencedirect.com/science/article/pii/S0010480998914896.

[28] Hossam M. Moftah et al. "Adaptive k-means clustering algorithm for MR breast image segmentation". En: *Neural Computing and Applications* 24.7 (2014), págs. 1917-1928. ISSN: 1433-3058. DOI: 10.1007/s00521-013-1437-4. URL: https://doi.org/10.1007/s00521-013-1437-4.

[29] S Nagabhushana. *Computer vision and image processing*. New Age International, 2005.

[30] Mene-Ejegi Ogbemi. *What is Overfitting in Machine Learning?* https://www.freecodecamp.org/news/what-is-overfitting-machine-learning/. Oct. de 2023.

[31] World Health Organization. *Cancer*. https://www.who.int/es/news-room/fact-sheets/detail/cancer. 2022.

[32] World Health Organization. *Cancer - Screening and Early Detection*. https://www.who.int/europe/news-room/fact-sheets/item/cancer-screening-and-early-detection-of-cancer. 2010.

[33] Vimla L. Patel et al. "The coming of age of artificial intelligence in medicine". En: *Artificial Intelligence in Medicine* 46.1 (2009). Artificial Intelligence in Medicine AIME' 07, págs. 5-17. ISSN: 0933-3657. DOI: https://doi.org/10.1016/j.artmed.2008.07.017. URL: https://www.sciencedirect.com/science/article/pii/S0933365708000961.

[34] T. Pavlidis e Y.-T. Liow. "Integrating region growing and edge detection". En: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 12.3 (1990), págs. 225-233. DOI: 10.1109/34.49050.

[35] Chris Piech y Andrew Ng. *K Means*. Written by Chris Piech. Based on a handout by Andrew Ng. © Stanford 2013 — Designed by Chris. Inspired by Niels and Percy. 2013. URL: https://stanford.edu/~cpiech/cs221/handouts/kmeans.html.

[36] Sarah Price. *Image from Segment and Split Medium Section*. https://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/MARBLE/medium/segment/split.htm.

[37] Saqib Qamar et al. "A variant form of 3D-UNet for infant brain segmentation". En: *Future Generation Computer Systems* 108 (2020), págs. 613-623. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2019.11.021. URL: https://www.sciencedirect.com/science/article/pii/S0167739X18332291.

[38] A Redondo et al. "Inter- and intraradiologist variability in the BI-RADS assessment and breast density categories for screening mammograms". En: *The British Journal of Radiology* 85.1019 (nov. de 2012), págs. 1465-1470. ISSN: 0007-1285. DOI: 10.1259/bjr/21256379. URL: https://doi.org/10.1259/bjr/21256379.

[39] Research Imaging Institute, University of Texas Health Science Center. *UML Diagram of Mango Viewer*. https://mangoviewer.com/images/uml.png. 2006–2024.

[40] International Agency for Research on Cancer. *Cancer TODAY*. https://gco.iarc.who.int. Data version: Globocan 2022 (version 1.1) - 08.02.2024, ©All Rights Reserved 2024. Accessed: 2024-06-08. 2024.

[41] Deb Richardson. "What is AI/ML and Why Does it Matter for Your Business". En: *Red Hat Blog* (2023). Consultado: 23 de junio de 2023. URL: https://www.redhat.com/es/blog/what-aiml-and-why-does-it-matter-your-business.

[42] Olaf Ronneberger, Philipp Fischer y Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. arXiv: 1505.04597 [cs.CV].

[43] Ashirbani Saha et al. "A machine learning approach to radiogenomics of breast cancer: a study of 922 subjects and 529 DCE-MRI features". En: *British journal of cancer* 119.4 (2018), págs. 508-516.

[44] Khamitkar S.D. Salem Saleh Al-amri N.V. Kalyankar. "Image segmentation by using edge detection". En: *International Journal on Computer Science and Engineering* 2 (mayo de 2010).

[45] Nima Tajbakhsh et al. "Embracing imperfect datasets: A review of deep learning solutions for medical image segmentation". En: *Medical Image Analysis* 63 (2020), pág. 101693. ISSN: 1361-8415. DOI: https://doi.org/10.1016/j.media.2020.101693. URL: https://www.sciencedirect.com/science/article/pii/S136184152030058X.

[46] Nika Tsankashvili. "Comparing Edge Detection Methods". En: *Medium* (ene. de 2018). 5 min read. URL: https://medium.com/@nikatsanka/comparing-edge-detection-methods-638a2919476e.

[47] *What are convolutional neural networks?* https://www.ibm.com/topics/convolutional-neural-networks. 2021.

[48] Zhitao Xiao et al. "Segmentation of Lung Nodules Using Improved 3D-UNet Neural Network". En: *Symmetry* 12.11 (2020). ISSN: 2073-8994. DOI: 10.3390/sym12111787. URL: https://www.mdpi.com/2073-8994/12/11/1787.