UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Diseño e implementación de una interfaz gráfica de usuario en lenguaje Python para el entrenamiento interactivo de redes profundas de segmentación de imagen médica

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Miguel Estevan Moreno

Tutor: José Vicente Manjón Herrera

2018/2019

Resumen

El objetivo de este proyecto es el desarrollo de una aplicación de escritorio para la segmentación interactiva de imágenes RMN cerebral usando técnicas de *Deep Learning*. En clasificación supervisada, uno de los cuellos de botella es la generación de datos etiquetados de calidad.

Este proyecto está orientado a agilizar el proceso de segmentación manual de cualquier tipo de imagen médica, de forma semiautomática mediante las redes neuronales, teniendo en mente siempre al usuario, ya que esta herramienta está pensada para usuarios tanto del campo de la informática como de la medicina.

Con esta herramienta se han obtenido resultados correctos a la hora de crear las segmentaciones, pero algo costosos con respecto al tiempo de creación.

Palabras clave: red neuronal, segmentación.

Abstract

This project aims to develop a desktop application for interactive segmentation of RMI cerebral images using Deep Learning techniques. In labelled classification, a bottle neck is the data labelled qualified creation.

This project has been oriented towards speed up the process of any medical image segmentation in a semiautomatic way by deep learning, always keep in mind the user, then this tool is oriented to users of informatic and medicine field.

With this tool, correct results have been obtained when creating segmentations, but somewhat costly with respect to creation time.

Key words: neural networks, segmentation.

Resum

L'objectiu d'aquest projecte es el desenvolupament d'una aplicació d'escriptori per a la segmentació interactiva d'imatges RMN cerebral usant tècniques de *Deep Learning*. En classificació supervisada, un dels colls de botella es la generació de dades etiquetats de qualitat.

Aquest projecte està orientat per agilitzar i a ajudar en el procés de segmentar manualment qualsevol tipus d'imatge mèdica, de forma semiautomàtica mitjançant les xarxes neuronals, tenint sempre en compte al usuari, ja que aquesta ferramenta ha sigut pensada per usuaris tant del camp de la informàtica com de la medicina.

Amb aquesta aplicació s'han obtingut resultats correctes a l'hora de crear les segmentacions, però quelcom costosos respecte a al temps de creació.

Paraules clau: xarxa neuronal, segmentació.

Tabla de contenido

Índice de figuras

Capítulo 1

Introducción

Se denomina imagen médica al conjunto de técnicas y procesos utilizados para crear imágenes del cuerpo humano, o partes de él, con propósitos clínicos (procedimientos médicos que buscan revelar, diagnosticar o examinar enfermedades) o para la ciencia médica (incluyendo el estudio de la anatomía normal y función).

En esta memoria se va a centrar en la imagen médica obtenida mediante la técnica de resonancia magnética nuclear. La resonancia magnética (RM) consiste en la obtención de imágenes radiológicas de la zona anatómica que se desea estudiar mediante el uso de un campo electromagnético (imán), un emisor/receptor de ondas de radio (escáner) y un ordenador.

Existen distintos tipos de imágenes de RM, dependiendo del proceso físico a estudiar, como, por ejemplo, imágenes potenciadas en T1, T2 o FLAIR (ver Figura 1).

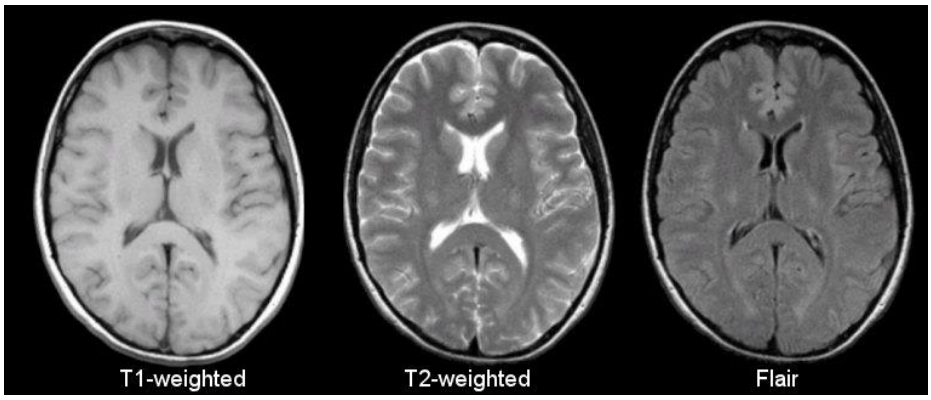


Ilustración 1. Ejemplo de imágenes de RM de distintas potenciaciones o tipos.

En general, el análisis de estas imágenes se basa en la observación por parte del radiólogo de las posibles alteraciones anatómicas que explican una determinada patología. Sin embargo, este análisis cualitativo es, en ocasiones, insuficiente y se requiere de un enfoque más cuantitativo, como, por ejemplo, la evaluación del volumen de distintas estructuras/tejidos o lesiones.

En la actualidad, el desarrollo de algoritmos automáticos que desempeñen dicha tarea es de gran utilidad. Específicamente, el proceso de segmentación de una imagen consiste en dividir una imagen digital en varias partes (grupos de píxeles) u objetos. La segmentación se usa tanto para localizar objetos como para encontrar los límites de estos dentro de una imagen. Es decir, la segmentación de una imagen es el proceso de asignación de una etiqueta a cada uno de los píxeles de la imagen, de forma que los que compartan la misma etiqueta también tendrán ciertas características visuales similares.

Este proceso de segmentación de alta precisión se puede realizar de forma manual, es decir, etiquetar manualmente cada uno de los píxeles de la imagen. Por lo tanto, es un proceso lento, largo y tedioso, ya que las imágenes médicas son tridimensionales, pudiendo tener más de 1000000 píxeles. Por esta razón, es necesario desarrollar algoritmos de segmentación semiautomáticos, precisos y eficientes.

Motivación

El desarrollo de esta herramienta nace de la necesidad de acelerar el proceso de segmentar imágenes médicas. Esto se debe a que la velocidad de creación y almacenamiento de imágenes médicas es mucho mayor que la velocidad de procesado y obtención información de las mismas, por lo que el tratamiento de estas imágenes a mayor velocidad proporcionaría una gran cantidad de información muy útil para los profesionales sanitarios.

El desarrollo de esta herramienta puede resultar útil también para institutos de investigación, como por ejemplo el instituto ITACA [1], ya que en muchos casos invierten mucho tiempo a la hora de segmentar imágenes médicas de forma manual.

Con el desarrollo de esta herramienta se pretende que el alumno profundice su conocimiento en Python, en el campo de la informática médica, y se inicie en la inteligencia artificial, ya que Python es el lenguaje con mayor número de librerías orientadas a la imagen médica, como también el lenguaje con más recursos para trabajar con las redes neuronales convolucionales.

Objetivos

El principal objetivo de este proyecto consiste en desarrollar una herramienta con la que crear y entrenar redes neuronales que sean capaces de generar segmentaciones de imagen médica. Para ello se usarán imágenes con las que se trabajará y editarán para poder entrenar las redes neuronales convolucionales.

En definitiva, el objetivo de este proyecto es crear una herramienta que resulte sencilla para el usuario para la creación y tratamiento de redes neuronales a la hora de trabajar con imagen médica.

Se establecen los siguientes subobjetivos para este proyecto:

- Creación de una interfaz de usuario para la segmentación semiautomática basada en redes neuronales convoluciones.
- Creación y entrenamiento de las redes neuronales.
- Comprobación de la predicción de las redes neuronales.

[1] http://www.itaca.upv.es/

- Capacidad de corrección de las redes neuronales.
- Obtener segmentaciones de las imágenes médicas.

Impacto esperado

Con el desarrollo de esta herramienta se pretende llevar a cabo una herramienta fácil de utilizar tanto por técnicos informáticos como por médicos y sanitarios. Actualmente existen diversos visualizadores de imagen médica, tanto web como herramientas de escritorio, pero estos no permiten segmentar imágenes semiautomáticamente ni tampoco agilizar el proceso de segmentación.

Metodología

Para llevar a cabo los objetivos de este proyecto se ha seguido una metodología iterativa, en contrapartida a las metodologías ágiles. Con respecto a metodologías iterativas, se divide todo el desarrollo del proyecto en distintas fases.

En primer lugar, se desarrolla todo el diseño de la herramienta y su funcionalidad, teniendo siempre claro el enfoque hacia la imagen médica y la inteligencia artificial, a la vez que se busca diseñar una herramienta sencilla e intuitiva para el usuario.

En segundo lugar, más enfocado a la inteligencia artificial, se crean distintos modelos de redes neuronales convolucionales, siempre teniendo en mente que se quiere un proceso interactivo y enfocado a la imagen médica, por lo que los modelos diseñados serán ligeros y entrenables con pocas iteraciones y parametrizados para imágenes tridimensionales.

Estructura

La memoria de este proyecto se encuentra dividida en ocho capítulos. En este **primer capítulo** se realiza una pequeña descripción del proyecto, sus objetivos, la metodología para llevar a cabo estos objetivos, una pequeña explicación de la estructura del documento y las distintas convenciones que se han tenido en cuenta a la hora de redactar este documento.

En el **segundo capítulo** se realiza un análisis del estado del arte con respecto a la segmentación de imágenes convencionales e imágenes médicas de manera automática o interactiva. Para ello se hace un repaso de los proyectos y artículos más relevantes que se han ido publicando a lo largo de los últimos años.

En el **tercer capítulo** se hace un análisis de los distintos aspectos del problema a resolver, como pueden la eficiencia algorítmica, el marco legal o la ética de este y los riegos. Una vez se hayan clarificado los distintos problemas a resolver, se lleva a cabo la identificación y análisis de las soluciones posibles, y de todas las propuestas se elige la más acorde con los requisitos del problema. Por último, se

planifica un plan de trabajo mediante técnicas de planificación y estimación de esfuerzo.

En el **cuarto capítulo** se hace una descripción de la arquitectura del sistema sobre el cual se va a ejecutar la herramienta, el flujo de trabajo de la herramienta y qué tecnologías se han llevado a cabo para el desarrollo de dicha herramienta.

En el **quinto capítulo** se describe el proceso de desarrollo de la herramienta, el en el que se explica de forma detallada como se ha intentado llevar a cabo los objetivos mediante el desarrollo de la aplicación. En este capítulo también se explican todos los problemas que se han ido encontrando a lo largo del proyecto.

En el **sexto capítulo** se presentan los resultados de la herramienta, comprobando su correcto funcionamiento, así como pruebas para conocer sus tiempos, su eficiencia y su nivel de exactitud a la hora de segmentar.

En el **séptimo capítulo** se exponen las conclusiones de forma sencilla, explicando que objetivos se han logrado y que grado de satisfacción se han logrado con los objetivos propuestos.

En el **octavo capítulo** se exponen una serie de futuras mejoras o posibles proyectos que pueden nacer a raíz del desarrollo de este.

Capítulo 2

Estado del arte

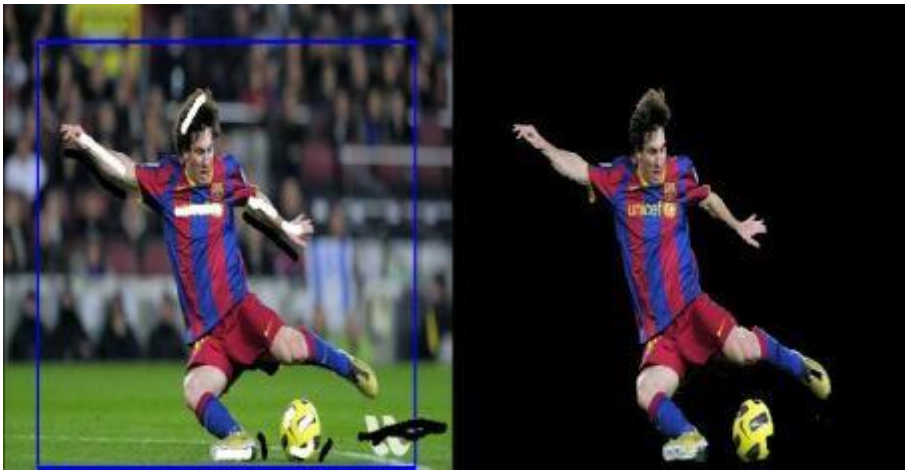
En estos últimos años, con el aumento de la capacidad de cómputo de prácticamente todos los dispositivos, se ha buscado sacar partido de esta potencia que se nos pone al alcance de la mano para casi cualquier propósito.

[1] [2]En 2004, un conjunto de investigadores de la universidad de Cambridge, publicaron un artículo, *"GrabCut": Interactive foreground extraction using iterated graph cuts*. En este articulo explican el algoritmo de *graph-cut* y sus tres mejoras desarrolladas. Primero, una optimización de eficiencia, segundo, una simplificación para el uso del usuario, y tercero, el desarrollo de algoritmo para "*border matting*" para estimar los píxeles del objeto a segmentar y el fondo.

Muchas de las funcionalidades desarrolladas en la mayoría de las librerías orientadas a la segmentación imágenes de prácticamente todos los lenguajes orientados a objetos se basan en este algoritmo.

Ilustración 2. Máscara utilizada en el algoritmo.

[2] https://dl.acm.org/citation.cfm?id=1015720



Como se aprecian en las imágenes, el proceso de a la hora de segmentar la imagen es el mismo que el que se busca en este proyecto. Con indicar en una máscara de la imagen lo que se quiere segmentar de la misma.

[2] [3]En 2016, en el congreso sobre reconocimiento de patrones y la visión por ordenador (CVPR) se presentó un proyecto, el *ScribbleSup*: *Scribble-Supervised Convolutional Networks for Semantic Segmentation,* en el cual se presentaba la segmentación de imágenes de dos dimensiones haciendo uso de redes convolucionales profundas.

Para ello, se crea una máscara sobre el objeto que se quiere segmentar, haciendo distintos garabatos sobre el objeto, u objetos, que se quieren segmentar, con distintos colores para separar unos de otros.

[3] https://arxiv.org/pdf/1604.05144.pdf

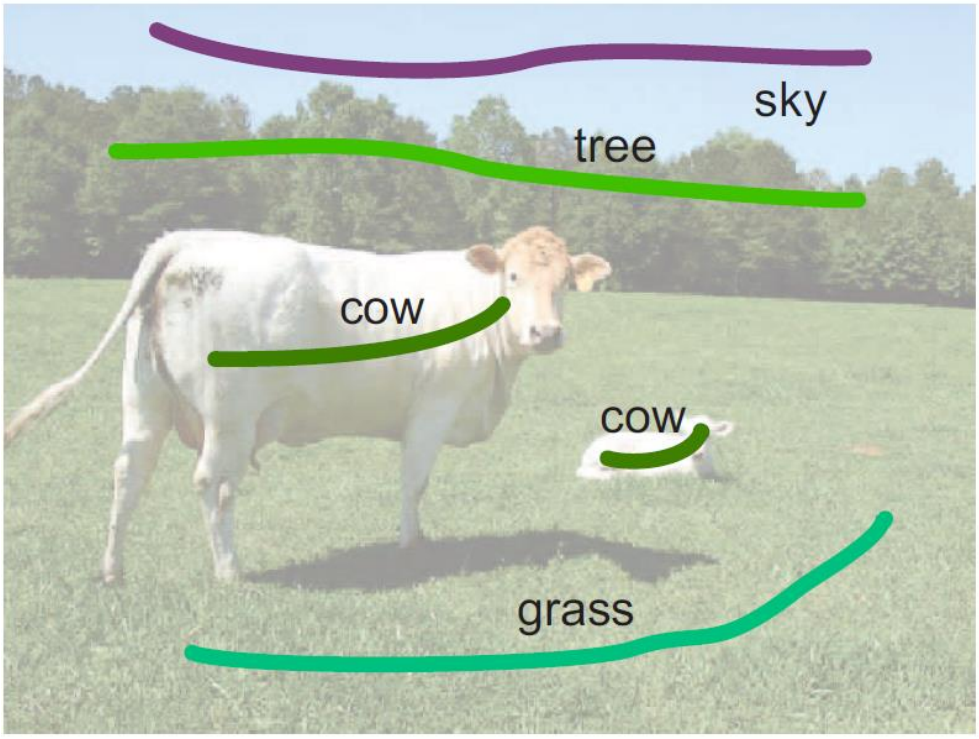


Ilustración 3. Ejemplo de cómo marcar los distintos objetos a segmentar.

Estos garabatos se usan como datos de entrenamiento en lo que se presentó como *Scribble-Supervised Learning,* es decir, aprendizaje mediante el etiquetado de los objetos a partir de garabatos sobre la imagen.

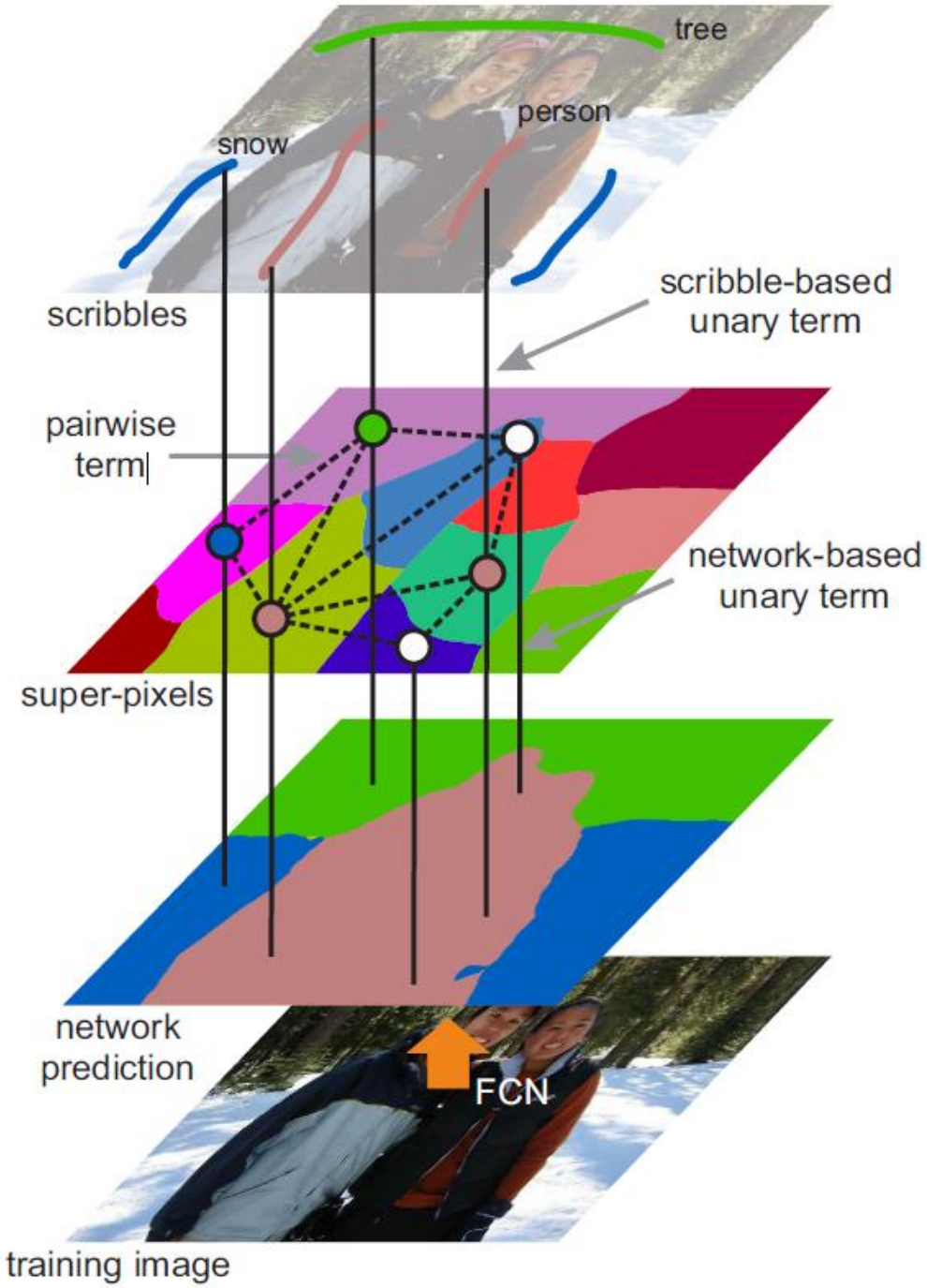


Ilustración 4. Ejemplo de Scribble-Supervised Learning.

[3] [4] En tercer lugar, comentar un artículo publicado el año 2019 por un investigador de Cambridge y otro de la universidad de la República de Corea. En este artículo, se publicaban unos resultados muy positivos en segmentación interactiva de

[4] https://vcg.seas.harvard.edu/publications/interactive-image-segmentation-via-backpropagating-refinement-scheme/paper

imágenes de dos dimensiones con un refinamiento basado en *backpropagation*. Para ello, se basan en las anotaciones del usuario para crear mapas de interacción midiendo la distancia que hay entre cada píxel y las anotaciones del usuario. Una vez se tienen estos mapas, se entrenan las redes neuronales convolucionales con dichos mapas, los cuales han sido refinados con esquema de *backpropagation*, para corregir posibles pixeles mal etiquetados.

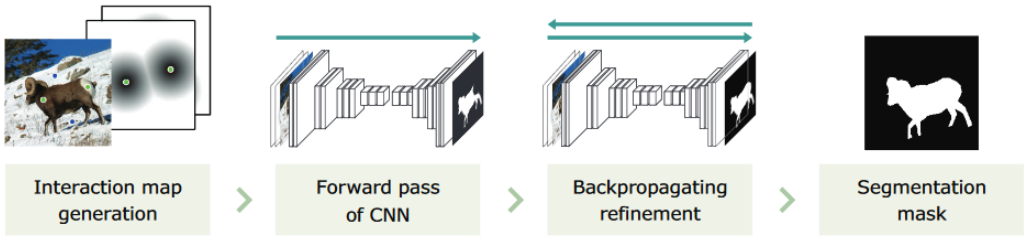


Ilustración 5. Proceso del algoritmo.

Ilustración 6. Resultado de la segmentación con el algoritmo desarrollado.

De nuevo, se puede observar como con las redes neuronales convolucionales se pueden conseguir grandes resultados a la hora de segmentar objetos concretos de imágenes de dos dimensiones.

[4] [5] En cuarto lugar, comentar un artículo publicado en abril de 2019, "*Interactive Full Image Segmentation by Considering All Regions Joinly*", realizado por tres investigadores de la empresa Google. En este artículo, se expone un objetivo interesante, y muy similar al objetivo principal de este proyecto, que es segmentar todas las regiones de la imagen. Para ello, proponen un *framework*, interactivo, y

[5] https://arxiv.org/pdf/1812.01888.pdf

basado en garabatos, y con posibilidad de corrección, que sea capaz de producir una máscara de segmentación de todas las regiones de la imagen, todo esto basado en *machine learning.*

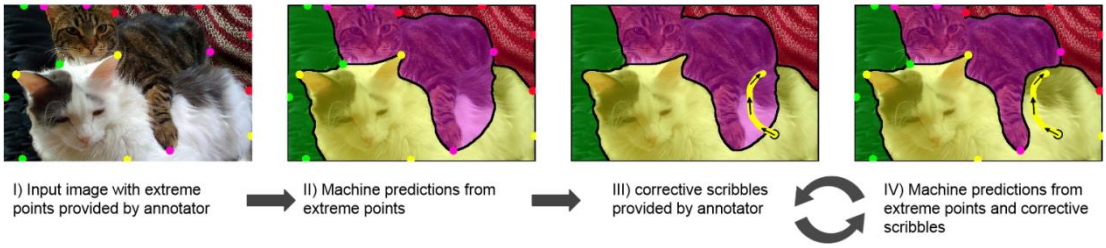


Ilustración 7. Flujo de trabajo de la herramienta propuesta.

[5] [6]Para finalizar, comentar el artículo publicado el año 2019, en el que han participado investigadores de todo el mundo, sobre segmentación de imágenes médicas basadas en redes neuronales *full connected*. Este artículo parte de la premisa del costoso proceso de segmentación manual de las imágenes médicas. Es por ello que desarrollan una herramienta interactiva basada en *Deep learning* para llevar a cabo de forma semiautomática una aproximación de una segmentación de la región de interés de la imagen. Para ello sigue un proceso similar al objetivo de esta herramienta, con unos clics sobre la región de interés, entrenar una red que sea capaz de generar segmentaciones de dicha región.

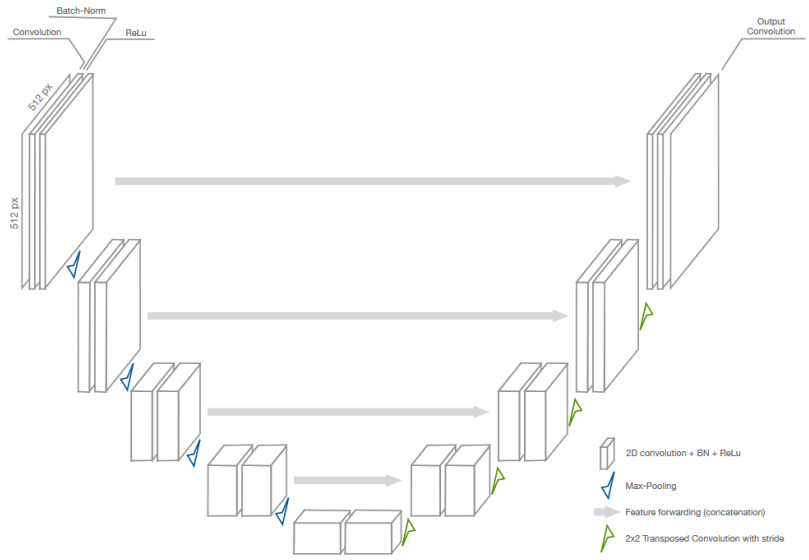


Ilustración 8. Arquitectura de la red.

[6] https://arxiv.org/pdf/1903.08205.pdf

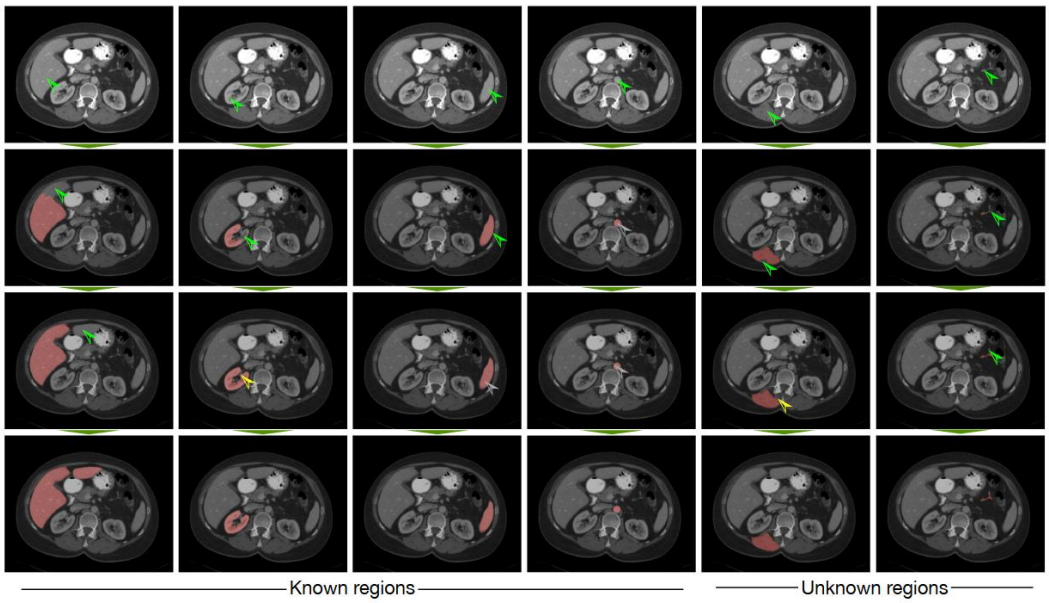


Ilustración 9. Ejemplo de segmentación interactiva.

En la imagen anterior se muestran en las cuatro primeras columnas el resultado de las regiones que se han ido aprendiendo durante el proceso de entrenamiento. En las otras dos restantes, se observan los resultados de las regiones desconocidas.

Crítica del estado del arte

Viendo los resultados obtenidos, se puede comprobar que con el paso del tiempo se ha ido aumentando el uso de la inteligencia artificial, ya que las tarjetas gráficas que permiten agilizar el proceso de entrenamiento de redes son más accesibles económicamente, y más potentes computacionalmente, relanzando de nuevo la inteligencia artificial.

Esto también hace que se hagan grandes avances en el campo de la inteligencia artificial, y se simplifique y acerque a la gran mayoría de programadores, ya que las grandes librerías de inteligencia artificial se han programado en Python, un lenguaje con una sintaxis muy cómoda de leer, y un lenguaje cada vez más utilizado en todo el mundo.

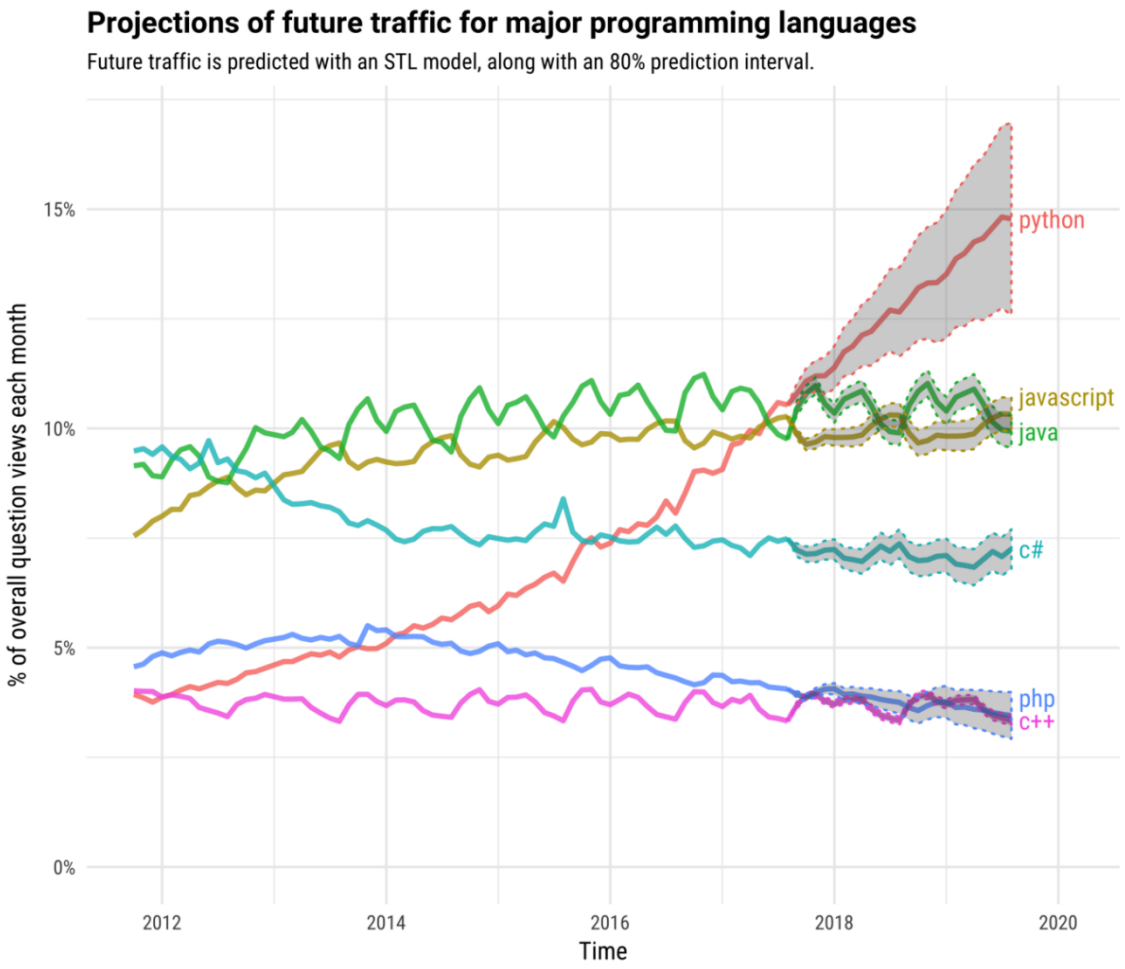


Ilustración 10. Aumento del uso del lenguaje Python en los últimos años.[7]

[7] https://www.paradigmadigital.com/wp-content/uploads/2017/11/grafica2.png

Por lo que el acercamiento de la inteligencia artificial a la comunidad de desarrolladores de todo el mundo ha permitido el desarrollo de grandes proyectos y nuevos ámbitos en los que poder hacer uso de la inteligencia artificial.

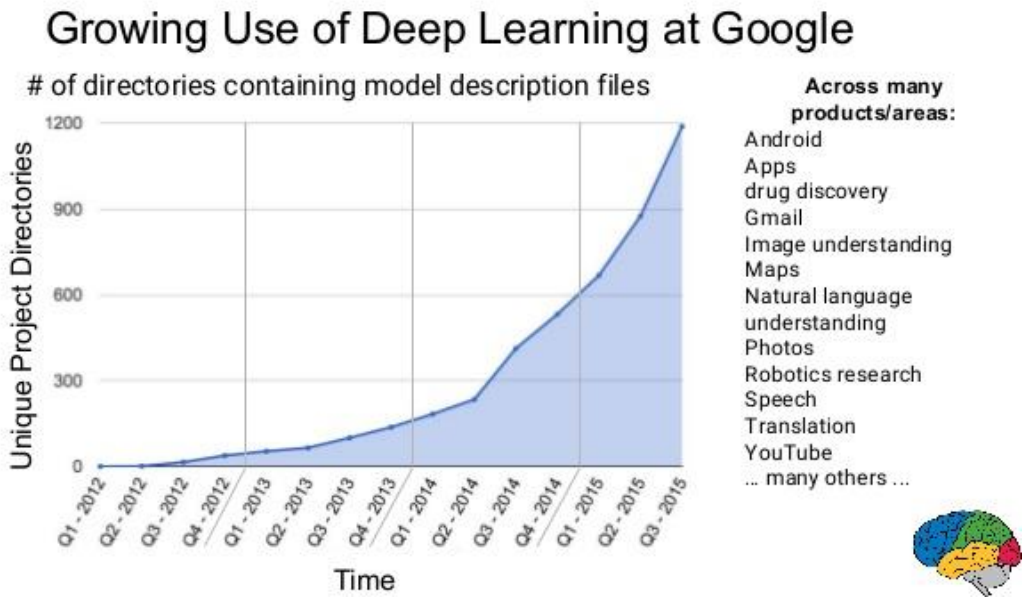


Ilustración 11. Crecimiento del uso del Deep Learning en los últimos años.[8]

Pero a pesar de todo este crecimiento del uso de la inteligencia artificial, no se encuentran proyectos que aprovechen esta gran capacidad de cómputo para la segmentación interactiva de imágenes de tres dimensiones, ya sean imágenes médicas o no.

Por ello, se ha decidido llevar a cabo los primeros pasos en la segmentación de imágenes médicas de manera interactiva con el desarrollo de este proyecto, el cual se va a basar en la inteligencia artificial para crear redes que segmenten partes del cerebro.

Propuesta

Para abordar la solución a este problema se propone una herramienta desarrollada en Python, la cual consiste en una interfaz gráfica de usuario (GUI) que permite la edición de imágenes médicas con formato NIFTI[9].

En base a estas imágenes, se crearán unas máscaras del mismo tamaño para poder seleccionar con puntos y líneas de distintos colores los objetos que se quieran segmentar. Una vez hecha una primera aproximación, es decir, la máscara de entrenamiento, utilizarla como *dataset* de las redes neuronales, e ir corrigiendo progresivamente las máscaras de segmentación que irá generando la red. Hay que

[8] https://pbs.twimg.com/media/CgfbGpGWQAAA-p9.jpg:large
[9] https://nifti.nimh.nih.gov/nifti-1/

concretar que las correcciones se harán sobre la máscara de segmentación generada, pero los puntos se estarán marcando sobre la máscara de entrenamiento.

Con respecto a los modelos de las redes, serán desarrollarán modelos ligeros, con pocas capas, e intentando adaptar el número de *epochs* para cada modelo, de forma que el flujo de trabajo con la herramienta sea el más rápido, interactivo y ameno posible.

De esta forma se pretende desarrollar una herramienta capaz de entrenar redes neuronales de forma interactiva que segmenten partes concretas del cerebro de la forma más precisa posible.

Capítulo 3

Análisis del problema

Llevar a cabo el desarrollo de una herramienta de estas características conlleva el estudio de las distintas dificultades que pueden surgir en el proceso de desarrollo. En este capítulo se analizarán los distintos problemas que se pueden dar y los distintos requisitos que ha de cumplir dicha herramienta.

Análisis de eficiencia algorítmica

La eficiencia algorítmica no es algo que se pueda descuidar en este proyecto, ya que el hecho de llevar a cabo algoritmos poco depurados ralentizaría el flujo de trabajo de esta herramienta, algo que se debe evitar, ya que trabajar con redes neuronales suele conllevar largas esperas. Es por ello, que además de estudiar la manera de minimizar los tiempos de los algoritmos, se hará uso de complementos y optimizadores de los distintos componentes.

Análisis del marco legal y ético

Análisis de protección de datos

Esta herramienta no almacena datos de usuarios, pero si hace uso de imágenes médicas basadas en resonancias de pacientes, por lo que los ficheros de las imágenes no pueden tener el nombre ni ningún dato de los pacientes. Una buena práctica para evitar estas situaciones es obtener las imágenes de bases de datos y repositorios públicos. Por ejemplo, http://www.aylward.org [10] es una web que indexa una gran cantidad de repositorios de acceso libre de imágenes médicas. Para todas las pruebas y entrenamientos se van a utilizar imágenes médicas de libre acceso.

Identificación y análisis de soluciones posibles

Matlab[11] es una herramienta genial para la visualización y edición de imágenes, ya que es una herramienta esencial a la hora de hacer operaciones con matrices, y una de las mejores opciones a la hora trabajar imágenes, ya que manipular las imágenes como matrices es un proceso muy cómodo, sobre todo cuando se nos ofrecen todas las posibilidades de Matlab. Además, también tiene complementos que permiten el uso de redes neuronales, alguno de ellos adaptados de Keras[12] y/o Tensorflow[13]. La mayor desventaja de Matlab es el requerimiento de espacio en disco,

[10] http://www.aylward.org
[11] https://es.mathworks.com/products/matlab.html
[12] https://keras.io/
[13] https://www.tensorflow.org/

ya que es una herramienta muy pesada de base. Otra de las desventajas, es su apartado de desarrollo de interfaces gráficas, el cual no es nada intuitivo.

Otra opción para llevar a cabo esta herramienta es Java, ya que tiene una gran cantidad de herramientas para el desarrollo de interfaces de usuario, como *frameworks* para diseñar interfaces de usuario de manera muy sencilla, por ejemplo, JavaFX[14]. Pero Java es un lenguaje orientado a objetos cuya ejecución es muy lenta debida su proceso de compilación y ejecución.

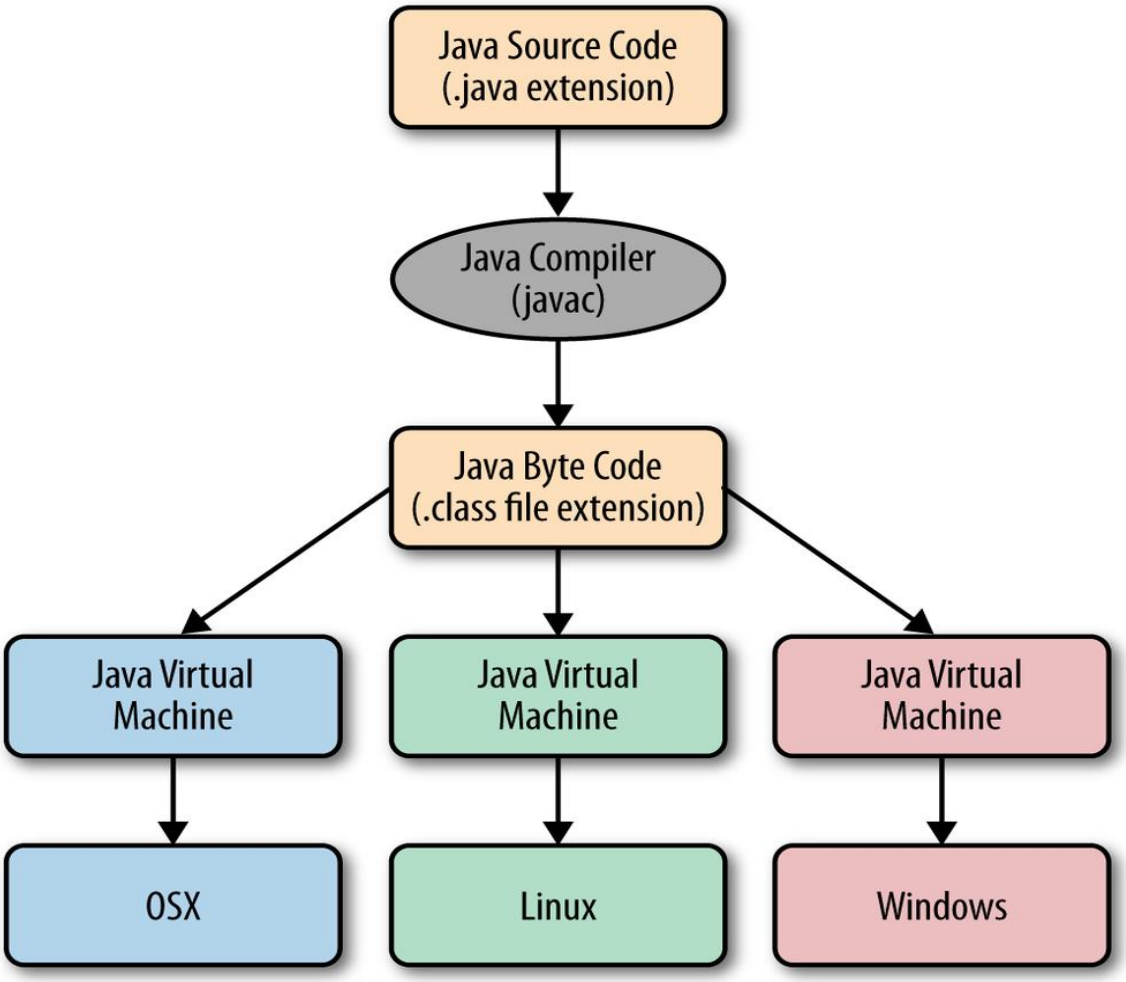


Ilustración 12. Esquema de compilación de Java. [15]

[14] https://openjfx.io/
[15] https://miro.medium.com/max/700/0*sdC9GbNa659Ftywc.png

Otra de las opciones ha sido llevar a cabo la herramienta en C++. Esta era una buena opción, ya que este lenguaje ya cuenta con algunas librerías como "*Image Viewer Cells",*[16] la cual te permite la visualización de imágenes médicas, además de muchas opciones para el desarrollo de interfaces de usuario, como por ejemplo Qt[17]. También cuenta con una API de *Tensorflow,* para poder trabajar con redes neuronales. Pero la desventaja de esta solución es que a la hora de crear los modelos de las redes neuronales se quiere utilizar *Keras,* que no tiene API para esta librería.

También se ha meditado llevar a cabo este proyecto en web, ya que existen visualizadores de imágenes médicas, como por ejemplo Papaya[18], el cual está hecho completamente en JavaScript, lenguaje por excelencia en desarrollo web. El principal problema de llevar a cabo este proyecto de esta forma es que nunca había programado web, y llevaría mucho más tiempo el desarrollo de la herramienta.

Una de las opciones que tenía más peso era desarrollar todo el proyecto en Python. Este es un lenguaje interpretado, por lo que no necesita compilación de código, solo leer un archivo por parte del interprete, por lo que en este paso ya es más rápido que Java a nivel de ejecución, pero un poco más lento que C/C++.

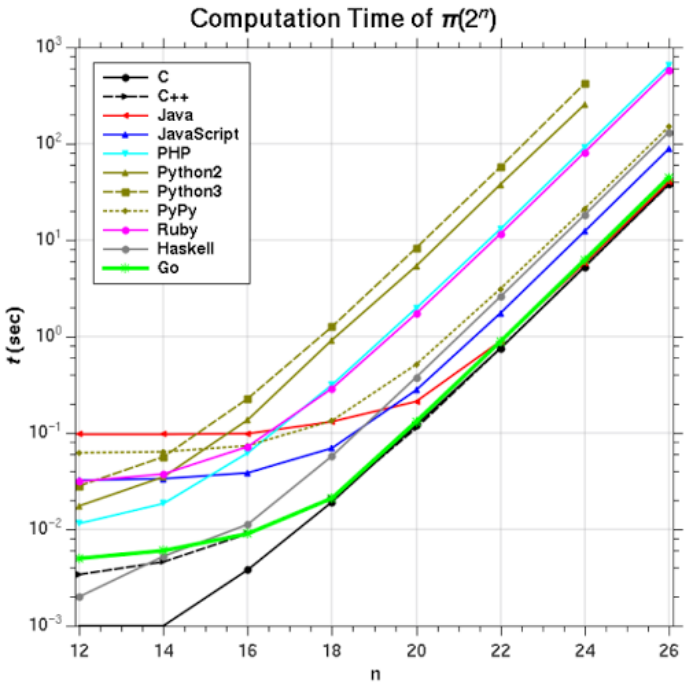


Ilustración 13. Tiempo de cómputo de distintos lenguajes.[19]

Además, cuenta con un gran número de herramientas y librerías para la edición y visualización de imágenes médicas, así como numerosas librerías para el desarrollo

[16] https://www.leadtools.com/help/leadtools/v20/dh/to/image-viewer-cells.html
[17] https://www.qt.io/ui-framework
[18] https://github.com/rii-mango/Papaya
[19] https://qph.fs.quoracdn.net/main-qimg-7ab0a880d83244b15e3e919aa61867b8

de interfaces de usuario. Pero la característica decisiva para elegir este lenguaje es que cuenta con el mayor número de librerías, herramientas y complementos para el desarrollo de redes neuronales.

Solución propuesta

A la hora de desarrollar la herramienta se ha escogido la solución de Python, ya que su única desventaja era, por una parte, la velocidad del propio lenguaje y por otra, la falta de *frameworks,* como JavaFX, para diseñar de manera sencilla la interfaz gráfica de usuario.

A la hora de desarrollar esta herramienta se ha pensado las siguientes fases:

1. Desarrollo de la interfaz gráfica de usuario sin funcionalidad.
2. Desarrollar la funcionalidad de la interfaz gráfica de usuario.
3. Desarrollar el editor de las imágenes médicas.
4. Desarrollar los modelos de las redes neuronales.
5. Integrar la interfaz con los modelos desarrollados.
6. Testeo y corrección.
7. Obtención de resultados.

Capítulo 4

Diseño de la solución

En este capítulo se va a especificar el diseño de la herramienta pensada una vez los requisitos han sido establecidos.

Arquitectura del sistema

La herramienta que se ha desarrollado no requiere un sistema operativo en concreto, ya que Python es multiplataforma[20], con los siguientes requisitos:

- Python 3.5 o superior.
- Tarjeta gráfica NVIDIA.
- Nvidia CUDA 9.0.

A nivel de hardware, será necesario una tarjeta gráfica compatible con Nvidia CUDA para que el flujo de trabajo sea interactivo, debido a que si solo se dispone de procesador en el equipo el proceso de entrenamiento y comprobación de las redes es muy largo. En caso de que no se disponga de ninguna tarjeta gráfica, tanto Google[21] como Nvidia [22] permite llevar tus experimentos con redes neuronales a su nube computacionales.

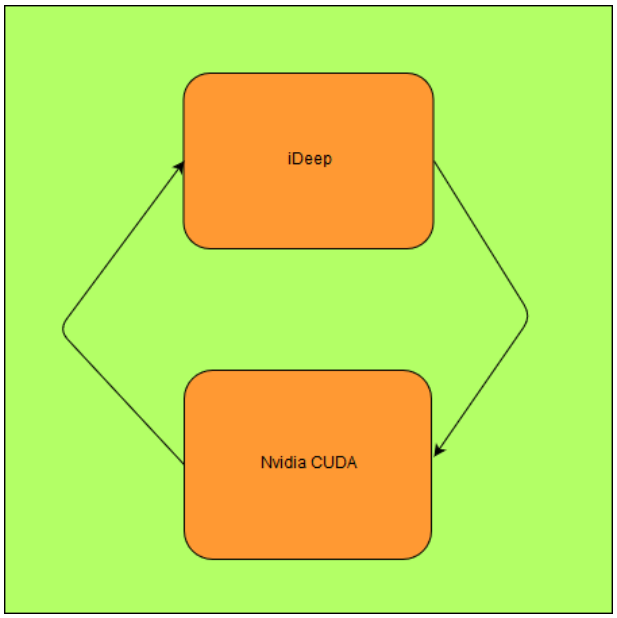


Ilustración 14. Flujo de trabajo.

[20] https://entrenamiento-python-basico.readthedocs.io/es/latest/leccion1/caracteristicas.html
[21] https://cloud.google.com/products/machine-learning/?hl=es-419
[22] https://ngc.nvidia.com/catalog/landing

Diseño detallado

En el apartado anterior se hace un escueto resumen de la arquitectura del sistema, pudiendo profundizar en este apartado aspectos de hardware y software.

Se ha utilizado la versión de Python 3.5. A pesar de que esta versión no es la más rápida, ostentando este puesto la versión 3.6, es la más estable con respecto a todas las librerías y herramientas utilizadas en este proyecto, ya sea la librería *Matplotlib*[23] o CUDA de Nvidia.

Con respecto a las tarjetas gráficas, todas las redes se han entrenado con una Nvidia GTX 1070[24], tanto en versión de ordenador portátil y sobremesa. Dicha tarjeta gráfica, tiene una arquitectura Pascal, propia de Nvidia, una memoria de 8 GB, una frecuencia de memoria de 8 Gbps, y lo más importante para este proyecto, dispone de 1920 núcleos de CUDA, los cuales son una equivalente a los núcleos de los procesadores convencionales, como las CPU de Intel. Estos núcleos están optimizados para ejecutar una gran cantidad de cálculos simultáneamente, algo vital para el procesamiento gráfico actual. Actualmente, existen tarjetas gráficas con una cantidad mucho mayor de núcleos CUDA, como la Nvidia RTX Titan, la cual dispone de 4608 núcleos, pero no es accesible para todos los bolsillos, ya que su precio oscila entre los 2000 €.

Flujo de trabajo

Para llevar a cabo la herramienta, se han pensado los siguientes requerimientos que tiene que cumplir la herramienta en su estado final para cubrir los objetivos propuestos:

1. Abrir una imagen médica con formato NIFTI.
2. Crear una red con los parámetros adecuados acordes a la región/es a segmentar.
3. Etiquetar la imagen para crear un conjunto de datos con los que la red pueda entrenar.
4. Lanzar el proceso para entrenar la red, teniendo en cuenta el flujo de trabajo interactivo.
5. Generar la segmentación en base al entrenamiento, y evaluar si es satisfactoria.
6. Volver al punto tres en caso de que la segmentación no sea satisfactoria.

[23] https://matplotlib.org/
[24] https://www.nvidia.com/es-es/geforce/products/10series/geforce-gtx-1070/

Tecnología utilizada

Para el desarrollo de la herramienta se ha utilizado PyCharm[25], un IDE (entorno de desarrollo integrado) que se especializa en Python. Este entorno de desarrollo simplifica muchos de los procesos de actualización y mantenimiento de las distintas librerías de Python, ya que normalmente estos procesos se hacen desde terminal, lo cual puede ser algo complicado para usuarios inexpertos en el flujo de trabajo desde algún *bash*. Además, integra funcionalidades con Git[26], por lo que también evita el paso por cualquier terminal. También integra un sinfín de complementos, como puede ser el editor de texto inteligente, la adición de numerosos *plugins* o la integración de herramientas científicas, las librerías más utilizadas en desarrollos científicos o la integración con Conda[27]. En definitiva, un entorno de desarrollo muy versátil, que te permite realizar tanto proyectos grandes como pequeños.

A la hora de explicar las librerías, para la interfaz de usuario se ha utilizado en su totalidad Tkinter[28], ya que su uso no es muy complejo e integra perfectamente con el resto de las librerías utilizadas. Para la visualización de las imágenes médicas se utiliza la librería Nibabel, que permite lectura y escritura para los formatos de archivos más comunes en la imagen médica. Para el desarrollo de todos los modelos de las redes neuronales se ha utilizado la librería de *Keras*, la cual se basa en TensorFlow, pero reduciendo su dificultad de uso, facilitando su desarrollo. Para la mayoría de las operaciones que se llevan a cabo durante el flujo de trabajo, se utiliza la librería Numpy, necesaria para poder dar forma a los datos de entrada y salida de las redes neuronales. Por último, hablar de la librería Matplotlib, muy similar a sintácticamente a Matlab. Es una extensión de NumPy[29] para poder mostrar de forma gráfica el contenido de *arrays* y listas, es decir, matrices, además de proporcionarte multitud de operaciones matriciales.

Comentar también que para este proyecto es fundamental tener disponibilidad de una tarjeta gráfica Nvidia para que el proceso de interactividad sea real, ya que con un procesador únicamente como único elemento de computo, el flujo de trabajo resultaría ser muy lento y tedioso. Por ese motivo, en este proyecto se va a hacer uso de una tarjeta gráfica Nvidia GTX 1070, gracias a la herramienta que ha desarrollado Nvidia, CUDA. CUDA es una arquitectura de cálculo paralelo que aprovecha la potencia de la GPU (unidad de procesamiento gráfico) para proporcionar un incremento extraordinario del rendimiento del sistema.

Los sistemas informáticos están pasando de realizar “procesamiento central” en el procesador a realizar “coprocesamiento” repartido entre la CPU y la GPU. Para posibilitar este nuevo paradigma computacional, Nvidia ha inventado la nueva arquitectura de cálculo paralelo CUDA.

[25] https://www.jetbrains.com/pycharm/
[26] https://git-scm.com/
[27] https://docs.conda.io/projects/conda/en/latest/
[28] https://wiki.python.org/moin/TkInter
[29] https://www.numpy.org/

Gracias a este aumento de rendimiento del sistema se han podido llevar a cabo muchas aplicaciones prácticas en campos como el procesamiento de imagen y vídeo, biología y química computacional, simulación de la dinámica de fluidos o la reconstrucción de imágenes de TC, entre otros.

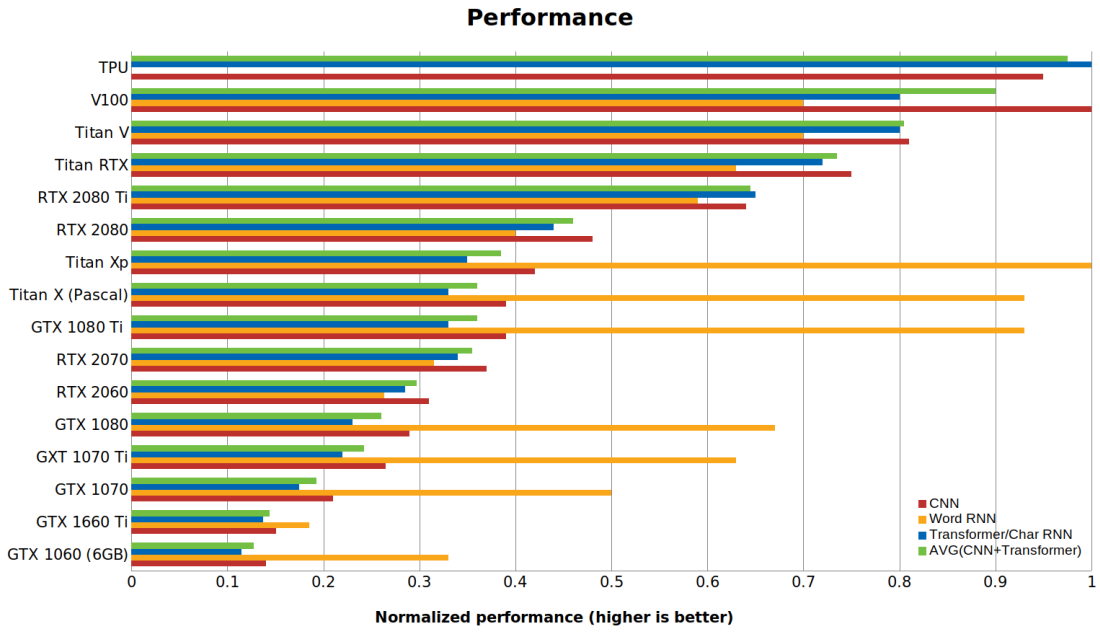


Ilustración 15. Gráfico potencia por tarjeta gráfica aplicada a Deep Learning.[30]

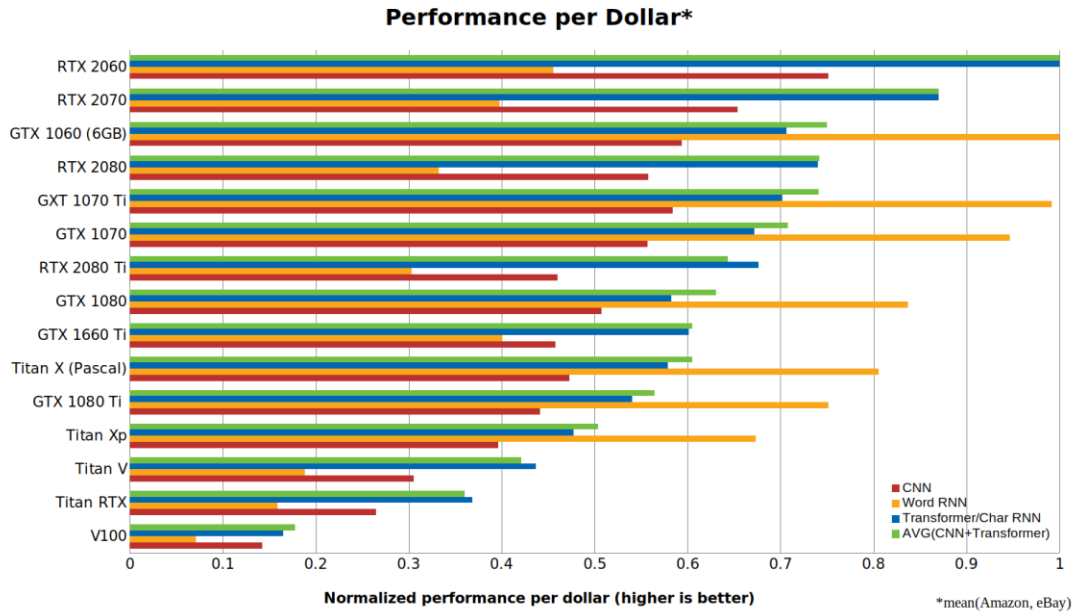


Ilustración 16. Gráfico de las GPU más rentables por unidad monetaria.[31]

30 https://i1.wp.com/timdettmers.com/wp-content/uploads/2019/04/performance_RTX.png?resize=1024%2C598&ssl=1

Introducción al *Deep Learning*

En este apartado se explicarán los conceptos básicos sobre *Deep Learning*, redes neuronales convolucionales y todo lo necesario para poder conocerlas y trabajar con ellas.

[6] En primer lugar, definir las redes neuronales. Una red neuronal es un sistema de procesamiento de información que tiene en común ciertas características con las redes neuronales biológicas. Los cuales, a través de un proceso de entrenamiento mediante ejemplos "prototipo", almacenan conocimiento de tipo experiencial y lo hacen disponible para su uso. Las redes neuronales artificiales (denominadas habitualmente RNA) son un paradigma de aprendizaje y procesamiento automático inspirado en la forma en que funciona el sistema nervioso. Se trata de un sistema de interconexión de neuronas que colaboran entre sí para producir un estímulo de salida. Las redes neuronales han sido desarrolladas como generalizadores de modelos matemáticos de cognición humana o biológica neuronal basadas en suposiciones:

1. El procesamiento de la información ocurre en muchos elementos simples llamados neuronas.
2. Las señales se pasan entre neuronas sobre enlaces de conexión.
3. Cada enlace de conexión tiene un peso asociado, el cual, en una típica red neuronal, multiplica la señal transmitida.
4. Cada red neuronal aplica una función de activación (normalmente no lineal) a su aporte neto (la suma del aporte neto ponderado) para determinar su señal de salida.

[7] En este proyecto se van a hacer uso de redes neuronales convolucionales. La principal ventaja de este tipo de redes neuronales es que cada parte de la red se le entrena para realizar una tarea, eso reduce significativamente el número de capas ocultas, es decir, aquellas capas que se encuentran entre la capa de entrada y la de salida, por lo que el entrenamiento es mucho más rápido. Además, presenta invarianza a la traslación de los patrones a identificar.

Las redes neuronales convolucionales son muy potentes para todo lo que tiene que ver el análisis de imágenes, debido a que son capaces de detectar características simples como por ejemplo detención de bordes, líneas, etc, y componer en características más complejas hasta detectar lo que se busca.

La arquitectura de una red neuronal convolucional es una red multicapa que consta de capas convolucionales y de reducción alternadas, y al finalmente tiene capas de conexión total como una red perceptrón multicapa.

[31] https://i1.wp.com/timdettmers.com/wp-content/uploads/2019/04/performance_per_dollar_RTX.png?resize=1024%2C598&ssl=1

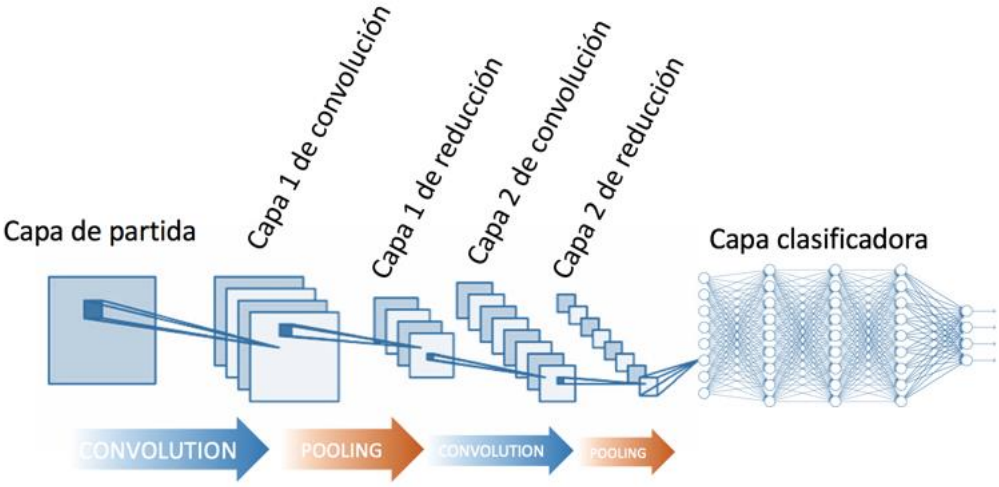


Ilustración 17. Ejemplo de arquitectura CNN.[32]

Durante las distintas convoluciones se realizan las operaciones de productos y sumas entre la capa de entrada y los n filtros que genera un mapa de características. Las características extraídas corresponden a cada posible ubicación del filtro en la imagen original.

Después de aplicar la convolución se aplica a los mapas de características una función de activación. Dichas funciones se encargan de devolver una salida a partir de un valor de entrada, normalmente, el conjunto de valores de salida en un rango determinado.

[8] Las funciones más utilizadas son la sigmoide y la *rectified linear unit* o **ReLU**. En el desarrollo de este proyecto se va a hacer uso principalmente de la función ReLU, la cual se define de la siguiente manera:

$$R(z) = \max(0, z)$$

Es decir, este tipo de funciones permiten el paso de todos los valores positivos sin modificarlos, pero asigna todos los valores negativos a cero. Las principales características de este tipo de funciones son:

- Activación *Sparse* – solo se activa si son positivos.
- No está acotada.
- Se pueden morir demasiadas neuronas.
- Se comporta bien con imágenes.

32 http://www.diegocalvo.es/wp-content/uploads/2017/07/red-neuronal-convolucional-arquitectura.png

- Buen desempeño en redes convolucionales.

En las capas de reducción o *pooling* se disminuye la cantidad de parámetros al quedarse con las características más comunes.

La forma de reducir parámetros se realiza mediante la extracción de estadísticas como el promedio o el máximo de una región fija del mapa de características, al reducir características el método pierde precisión, aunque mejora su compatibilidad.

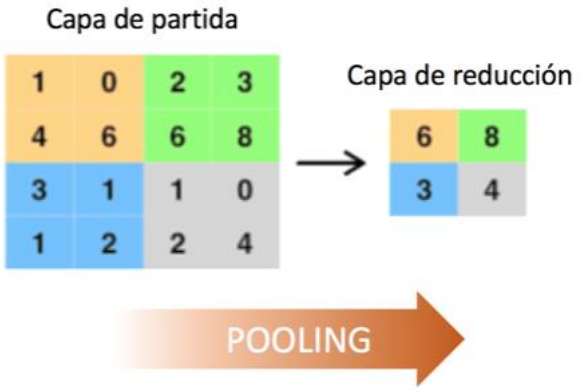


Ilustración 18. Ejemplo de pooling.[33]

[9] Por último, explicar que son las funciones ***loss***. Las funciones *loss* son un tipo de funciones que mapean un evento o valores de una o más variables en una representación de coste de forma intuitiva de dicho evento. En *Deep Learning*, se utilizan para medir la cuan dista el resultado que se obtiene con respecto con el que se quiere obtener. En este proyecto se hará uso de la función *categorical crossentropy,* la cual tiene la siguiente forma:

$$L(y, \hat{y}) = \sum_{j=0}^{M} \sum_{i=0}^{N} (y_{ij} * log\,(\hat{y}ij))$$

Esta función compara la distribución de las predicciones con la distribución dada, donde la distribución de la clase “verdadera” varía entre cero y uno.

[33] http://www.diegocalvo.es/wp-content/uploads/2017/07/reducci%C3%B3n.png

Capítulo 5

Desarrollo de la solución propuesta

En este capítulo se llevará a cabo un análisis de todo el proceso de desarrollo de la herramienta, desde sus primeras vistas hasta su estado final.

Desarrollo de la interfaz de usuario

Desarrollo de la interfaz de usuario principal

Como se ha explicado en el capítulo 3 - Análisis del problema, se ha decidido empezar el desarrollo implementando la interfaz gráfica de usuario de la herramienta.

Las primeras ideas han sido intentar adaptar la distribución de las distintas opciones conforme están en el programa *ITK-Snap*, un programa de software libre que permite la visualización y navegación de las imágenes médicas.

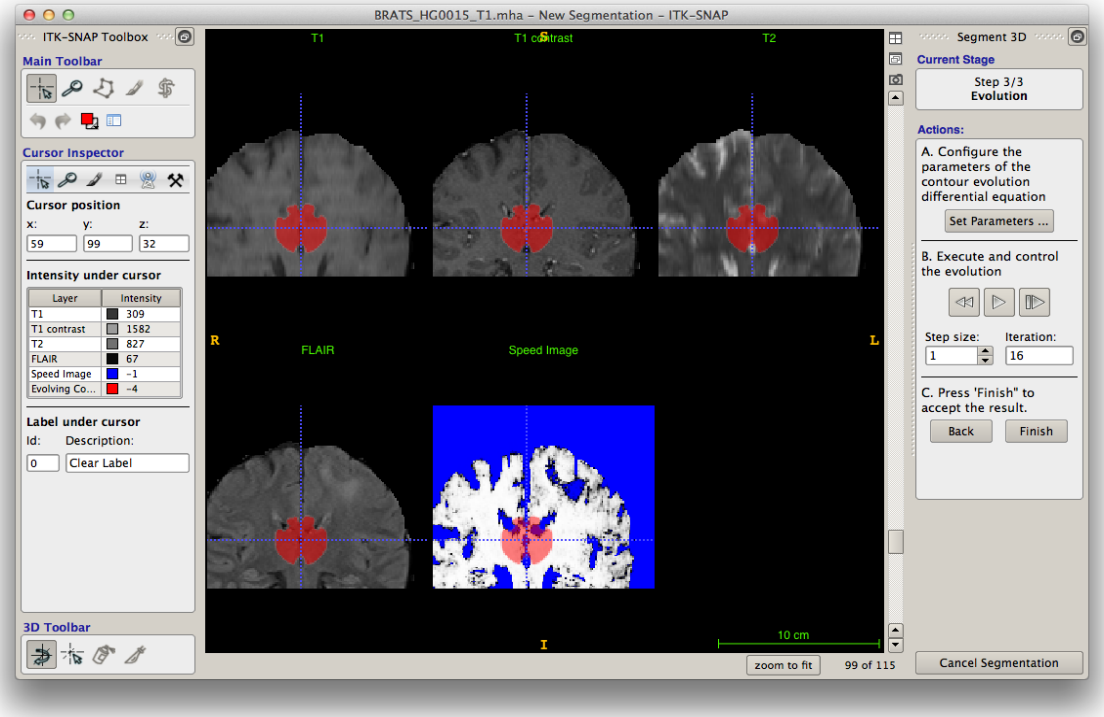


Ilustración 19. Ejemplo de la vista principal del programa ITK-SNAP. [34]

34 https://sourceforge.net/p/itk-snap/screenshot/Screen%20Shot%202014-06-12%20at%2012.28.33%20PM.png

Para ello, se desarrolla una interfaz de usuario mediante la librería Tkinter, ya que es la que mejor integra los *canvas* en los cuales se han de mostrar las tres vistas del plano anatómico.

En primer lugar, se definen las agrupaciones de los botones, se crean los distintos botones para cada una de las funcionalidades y se establece el fondo negro para poder visualizar correctamente las tres vistas.

Ilustración 20. Interfaz de usuario con intensidades.

1. **Botón para guardar la máscara:** Guarda la máscara de segmentación en la carpeta seleccionada en el diálogo de ficheros del sistema.
2. **Botón para la creación de la red:** Despliega la interfaz para parametrizar la nueva red.
3. **Botón para carga una red:** Botón que abre un diálogo del sistema para cargar una red y sus respectivos datos.
4. **Botón para colocar puntos:** Permite colocar puntos al seleccionar un corte de alguna de las tres vistas.
5. **Botón para finalizar la colocación de puntos:** Finaliza la colocación de puntos al seleccionar algún punto de la imagen.
6. **Parámetro alfa:** Consiste en el nivel de opacidad de la máscara que se visualiza junto a la imagen médica, siendo uno totalmente visible, y cero invisible.
7. **Tamaño de puntero:** Cantidad de píxeles que se por cada clic en la imagen. Por defecto su valor es uno, pero se puede aumentar el radio del puntero para abarcar más píxeles.
8. **Intensidad mínima:** Intensidad mínima para el correcto visionado de la imagen.
9. **Intensidad máxima:** Intensidad máxima para el correcto visionado de la imagen.

10. **Botón para entrenar:** Empieza el proceso de entrenamiento con la máscara de entrenamiento.
11. **Botón para testear**: Empieza el proceso de creación de la máscara de segmentación.
12. **Botón para cargar imagen**: Abre un dialogo de archivos para seleccionar la imagen.

En la barra superior está el desplegable *file.* En el desplegable de *file* se encuentran las mismas opciones, con la excepción de que aquí se encuentra la opción para cargar una máscara.

En la versión final se ha decidido suprimir las dos barras de la intensidad, ya que se esa funcionalidad se puede implementar editando el navegador de las imágenes médicas vaciando el menú de opciones para evitar la saturación del usuario.

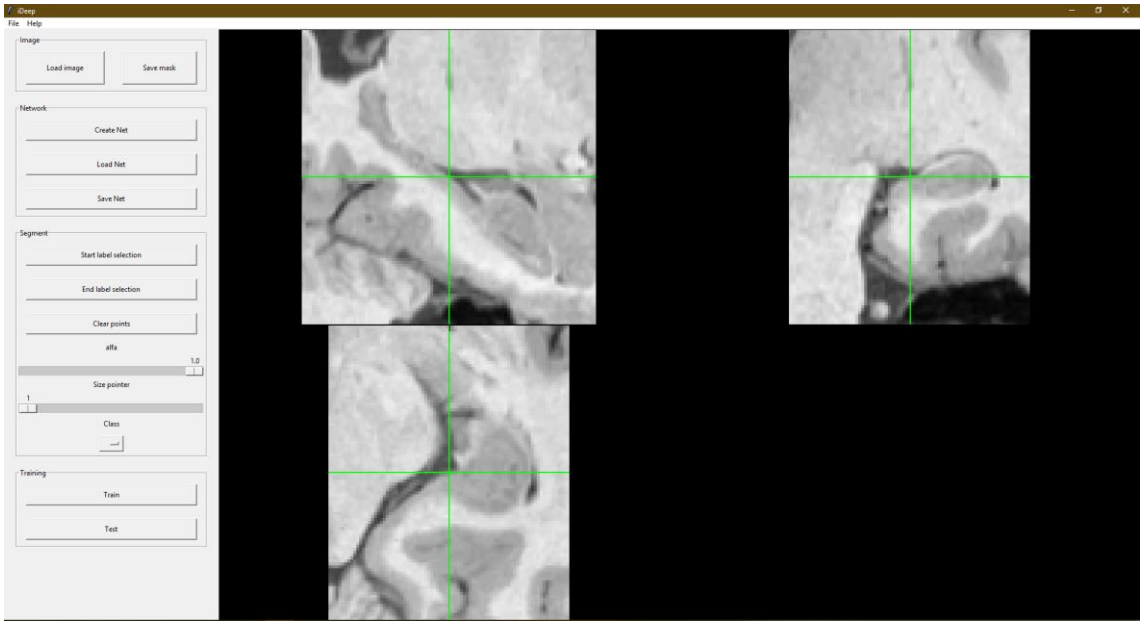


Ilustración 21. Versión final de la interfaz.

Cuando se quiera crear una nueva red, se selecciona el botón de "*Create new net*", y se nos abre una nueva ventana, en la que se nos pide una serie de parámetros para la red.

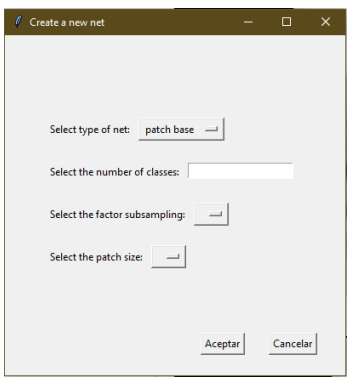


Ilustración 22. Ventana de creación de red.

Antes de continuar con el desarrollo de la funcionalidad de la herramienta, recordar de forma escueta la funcionalidad de esta:

1. Una vez se abre una imagen médica para segmentar, se crea una máscara de segmentación.

2. Una vez hecho esto, se puede cargar una red para seguir entrenándola o crear una nueva red. En caso de cargar una red para seguir entrenándola, si se ha guardado previamente con esta herramienta, también se cargarán los puntos con los que haya entrenado, la máscara, y en caso de que la red se base en *patches*, también se cargarán.

3. La opción de introducción estará disponible cuando la herramienta tenga una red que entrenar.

4. Una vez se tengan los puntos seleccionados, se puede entrenar la red seleccionando el botón de "*train*". Esto pondrá en marcha el proceso de entrenamiento, el cual finalizará cuando la ventana emergente de "training" se cierre.

5. Para comprobar los resultados se selecciona el botón de "test" para generar la máscara de segmentación y se evalúa la segmentación generada.

6. En caso de que la segmentación generada no sea satisfactoria, volver al punto 3 para la corrección de puntos.

Una vez se tiene el diseño completado de la interfaz de usuario se pasa a desarrollar las funcionalidades restantes y refactorizar aquellas implementadas durante el desarrollo de la interfaz de usuario.

Desarrollo de la funcionalidad

Para empezar, se revisa la implementación del botón de carga de la imagen. Para su correcto funcionamiento, primero se lee la imagen, se convierte en matriz y se le pasa como parámetro al objeto controlador de las vistas. Para poder abrir la imagen se realiza mediante un dialogo de archivos nativo del sistema. Luego, se inicializa la máscara en función de las dimensiones de la imagen y se inicializa las tres variables que guardan las coordenadas actuales.

Una vez refactorizado el botón de carga, se centra el desarrollo en el método "*onclick*" para la selección de los puntos en las imágenes. Para ello, se quiere resolver primero el problema de cómo "fusionar" la máscara de segmentación y la imagen original.

Para ello se decide editar el navegador utilizado de la biblioteca de Nibabel para poder pasar como parámetro una máscara con la que poder trabajar. Entonces, se edita el método de la clase del navegador en la que se trata la imagen, aplicando la siguiente fórmula en la que interviene la imagen original, la máscara y el valor alfa.

$$I' = \Big((m > 0) * \big(m * alfa + I * (1 - alfa)\big)\Big) + ((m == 0) * I)$$

Siendo m la máscara e I la imagen original.

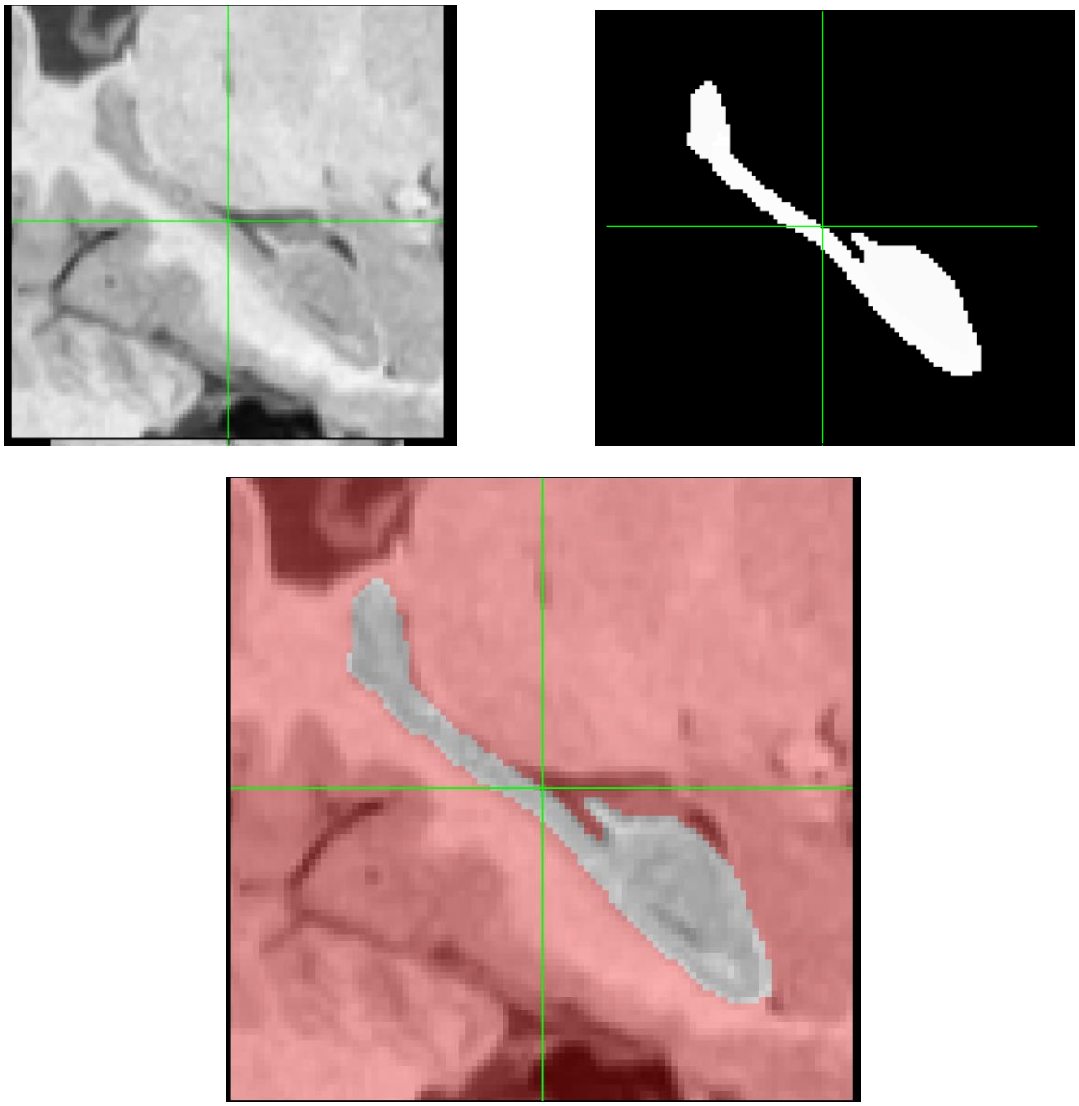


Ilustración 23. Imagen resultada de fusionar la original con su máscara.

Los dos primeros operandos de las dos partes de la ecuación son los puntos mayores que cero y los puntos equivalentes a cero respectivamente.

Llevar a cabo esta fórmula al lenguaje de Python es muy sencillo, quedando de la siguiente manera.

$$\text{I'} = ((\text{mask} > 0)*(\text{mask} * \text{alfa} + \text{ima}*(1\text{-alfa}))) + ((\text{mask} == 0)* \text{ima})$$

El siguiente problema que ha surgido a raíz de resolver este último, es que el visor de la biblioteca Nibabel no permite dibujar colores, es decir, es monocromático, y esto es un problema, ya que en este estado no se puede diferenciar los puntos de las clases.

Para solucionar este problema, se decide modificar el tratamiento que hace el visor de Nibabel, añadiéndole una dimensión extra para poder hacer uso de los canales rojo, verde y azul, y de esta manera combinarlos y crear una paleta de colores para las clases.

Para ello, se amplía la matriz de la imagen y de la máscara de segmentación con una dimensión extra, en la que se podrá seleccionar el canal RGB que se quiera. Una vez hecho esto, se hace la combinación lineal de la imagen original, el valor alfa y la máscara aplicando la formula anterior para disponer de la imagen final para poder trabajar sobre ella.

Una vez solucionados estos problemas, se puede continuar con la implementación de la principal funcionalidad de la herramienta.

Para ello, por cada clic en cada una de las tres vistas de la imagen, se marca del color correspondiente a la clase tanta cantidad de pixeles como nos indique el valor del puntero seleccionado.

```
for x in range(-radio, radio+1):
  for y in range(-radio, radio+1):
    for z in range(-radio, radio+1):
      if(x * x + y * y + z * z) < radio:
        self._mask[int(round(event.xdata+x)), int(round(event.ydata+y)),
        int(round(self._last_z+z))] = clase_actual

        self._seg_view[int(round(event.xdata+x)),
       int(round(event.ydata+y)), int(round(self._last_z+z))] = clase_actual
```

Una vez se aplica este código para las tres vistas, se actualizan las máscaras de segmentación y visualización.

Redes neuronales y sus modelos

En primer lugar, comentar que para esta herramienta se hacen uso de dos tipos de redes neuronales, las basadas en volúmenes, es decir, que entrenan con toda la imagen tridimensional, y aquellas que se basan en *patches 3D*, es decir, aquellas que entrenan con subsecciones más pequeñas de la imagen original de tamaño fijo.

Todos los modelos desarrollados son modelos relativamente simples, con pocas convoluciones y capas, ya que lo que se busca es un entrenamiento rápido e interactivo.

Una vez que la selección de puntos está implementada, se pasa a la parte de preprocesado de los datos para que las redes puedan trabajar con ellos. Para empezar, las redes neuronales trabajan con *arrays* o “tensores” que necesitan dos dimensiones extras, una colocada al principio de esta, y otra al final. La primera

dimensión consiste en la cantidad de datos con los que va a entrenar, las tres siguientes corresponden con las dimensiones de la imagen, y la última dimensión es el canal.

Para ello, se implementan cuatro métodos: uno para preprocesar la imagen y la máscara que va a ser procesada por un modelo basado en volúmenes, un método para preprocesar los *patches* de la máscara, otro para los *patches* de la imagen, y otro para procesar la imagen en caso de que el factor de *subsampling* sea mayor que uno.

Para el preprocesamiento de la imagen y la máscara para el modelo volumétrico, primero se aplica la siguiente fórmula:

$$I' = \frac{I - mean(I)}{\sigma(I)}$$

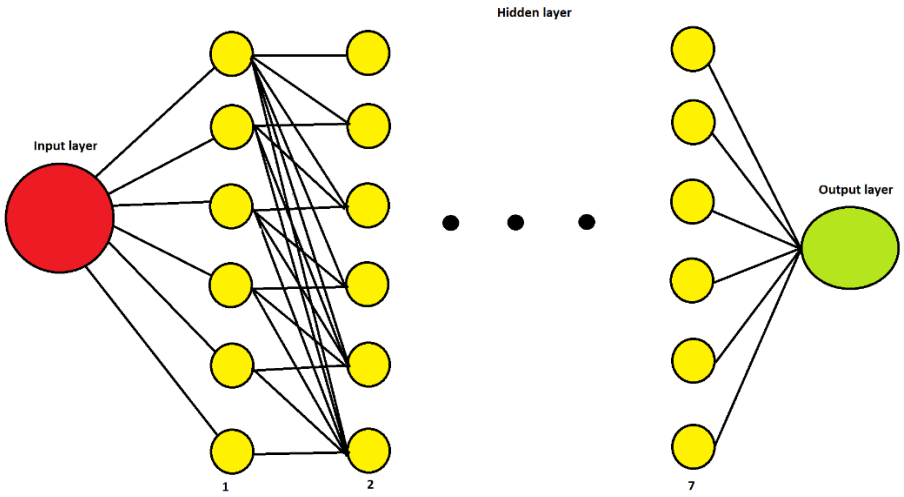
Siendo $mean$ la media de la imagen y σ la desviación típica.

Una vez hecho esto, se aplica un *padding* a la imagen para que su tamaño sea múltiplo de ocho, ya que, como se verá más adelante, la primera red volumétrica tiene ocho capas, y como las redes van reduciendo el tamaño de la imagen a la mitad conforme el proceso de entrenamiento avanza, se necesita este tamaño para que no se dé el caso tener una dimensión con tamaño impar al reducir a la mitad el tamaño de la imagen. Por último, se añaden las dos dimensiones extra a la imagen. En este caso, la primera dimensión será un uno, ya que solo va a trabajar con la imagen entera, y la dimensión del canal, corresponderá a uno.

Para preprocesar la máscara, se procede de la misma forma que para el caso de la imagen, a excepción, de que, en este caso, la dimensión del canal será tres.

El preproceso de los *patches*, ya sean procedentes de la máscara o de la imagen, es un método más sencillo. Basta con añadir las dos dimensiones que faltan, la primera, la cantidad de *patches*, y la segunda, el canal, que su valor es uno. Para calcular la cantidad de *patches,* basta con calcular los valores de la máscara mayores que uno.

Respecto a las redes basadas en volúmenes, se han desarrollado dos modelos. El primero algo más pesado, con ocho capas y tres convoluciones por capa. La segunda red, más pequeña, está formada por dos capas y dos convoluciones por capa. Ambos modelos tienen como parámetro de entrada 32 filtros, un *pool-size* equivalente a dos por cada una de las dimensiones de la imagen y ambos modelos con la función ReLU como función de activación.



A continuación, se explicarán los distintos tamaños de *patches* y los modelos basados en estos. En este proyecto se va a trabajar con *patches* de tamaño de 4x4x4, 8x8x8, 16x16x16 y 32x32x32, ya que cada uno de los distintos *patches* tienen sus ventajas e inconvenientes. Los *patches* más grandes son más pesados a la hora de ser procesados por la red, pero aportan más información a esta, por lo que el número de interacciones sería menor, teóricamente, pero más largas en el tiempo. En contrapartida, los *patches* más pequeños son más fáciles de procesar por la red, pero aportan menos información, por lo que el número de interacciones sería mayor, pero menores en el tiempo.

Una vez explicados los *patches*, se procede a explicar sus modelos. Se han llevado a cabo ocho modelos de redes, los cuales se pueden agrupar en dos tipos, unos modelos básicos y otros con información extra. El primer tipo de modelos tienen la misma estructura que los modelos volumétricos, la única diferencia es el número de capas, la cual disminuye conforme disminuye el tamaño del patch, ya que, al entrenar con *patches* de 4x4x4 no es eficiente hacer un *pool* a tamaños más pequeños.

El segundo tipo de modelos es muy similar, la única peculiaridad que tiene es que en la última capa se le concatena un *array* con los puntos que forman los *patches*, añadiendo así información espacial, con la finalidad de acortar el tiempo de entrenamiento.

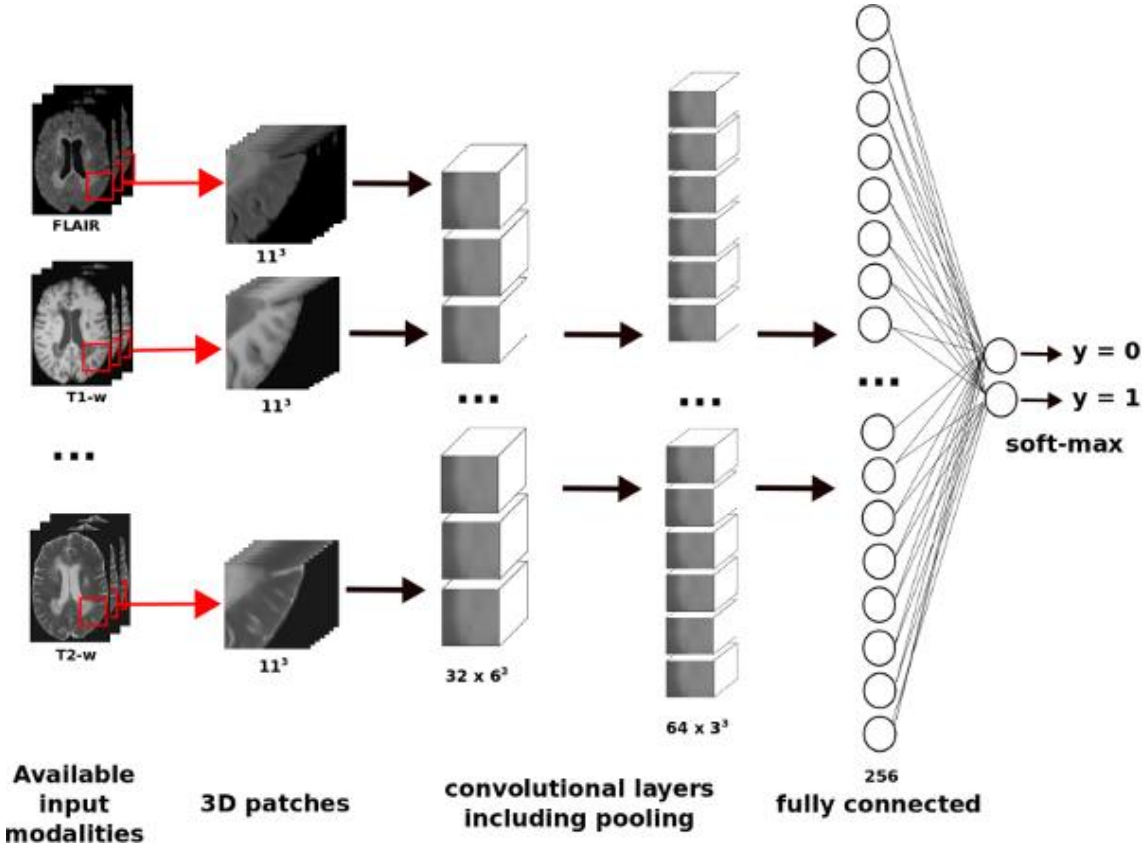


Ilustración 24. Ejemplo de red basada en patches.[35]

Una vez explicados los tipos de modelos que se van a utilizar, se procede a explicar el factor de *subsampling*, opcional al crear un nuevo modelo de red. Este parámetro se decide introducir ya que:

- Las imágenes médicas cuentan con una dimensión extra, por lo que el tiempo de procesado es mayor.
- Normalmente, las imágenes médicas tienen un tamaño considerable.
- Las tarjetas gráficas potentes no son accesibles para todos los usuarios.

Es por ello por lo que se introduce un factor de *subsampling*, es decir, reducir el tamaño de la imagen en función de este antes de testear la red con la imagen, ya que para testear es necesario, en caso de un modelo volumétrico, procesar toda la imagen, y en caso del modelo basado en *patches*, procesar el patch de cada uno de los puntos de la imagen Es por ello que el proceso de testeo requiere más tiempo que el proceso de entrenamiento, ya que durante este proceso, solo se computan aquellos valores en los que la máscara es mayor que uno.

[35] https://sergivalverde.github.io/img/cnn.png

Capítulo 6

Pruebas

En este capítulo se van a mostrar los resultados de las pruebas con los distintos modelos de las redes. En todas las pruebas que se van a llevar a cabo, el objetivo va a ser conseguir una segmentación del hipocampo, en color verde se va a segmentar el objeto y en rojo el fondo.

El hipocampo es un pequeño órgano situado dentro del lóbulo temporal intermedio del cerebro y crea a una parte importante del sistema límbico, la región que regula las emociones.

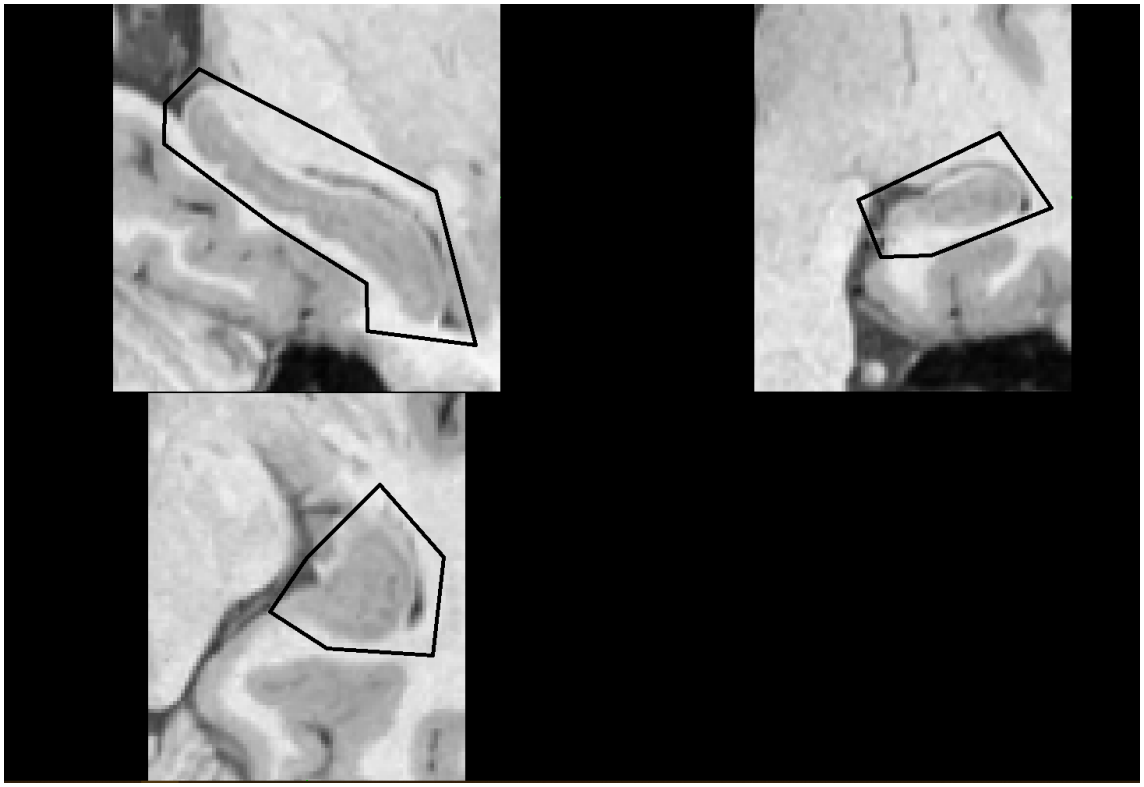


Ilustración 25. Indicaciones de la región del hipocampo.

Todos los tiempos que se miden en todas las pruebas con cada uno de los modelos está medido en segundos.

Pruebas de redes volumétricas

UNET3D

Las primeras pruebas se llevan a cabo con el modelo de red basada en volumen de ocho capas, el más grande de los dos. Se va a entrenar con imágenes de tamaño 82x100x100, sin factor de *subsampling*, ya que al entrenar con la imagen entera no se desea perder calidad en esta, además de entrenar con 20 *epochs.*

	Repeticiones					
	1	2	3	4		TOTAL
Train	60,49	47,06	46,77	44,3		198,62
Tess	1,31	0,46	0,44	0,45		2,66

Ilustración 26. Resultados UNET 3D.

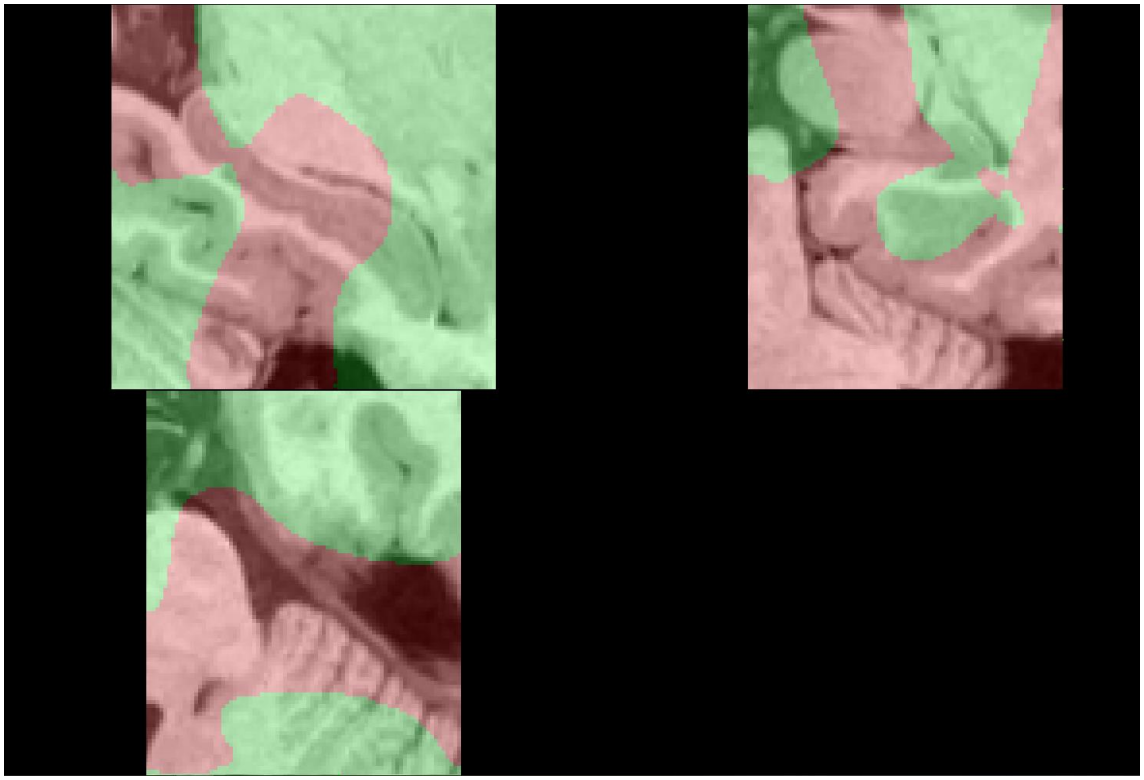


Ilustración 27. Modelo volumétrico, primera prueba.

Con este modelo de red, las primeras correcciones hay un 50% de puntos objeto y otro 50% de fondo, y con cada corrección no se aprecian prácticamente mejoras, ya que siempre está oscilando entre el 40% y 60% entre fondo y objeto, con muchos falsos positivos y falsos negativos, por lo que no es un buen modelo de red para la segmentación.

NET3D

A continuación, se expondrán las pruebas con el modelo volumétrico de dos capas. Como este es un modelo más ligero, se van a aumentar el número de *epochs* a 40.

	Repeticiones					
	1	2	3	4		TOTAL
Train	28,49	20,96	22,29	20		91,74
Test	0,46	0,29	0,29	0,26		1,3

Ilustración 28. Resultados NET3D.

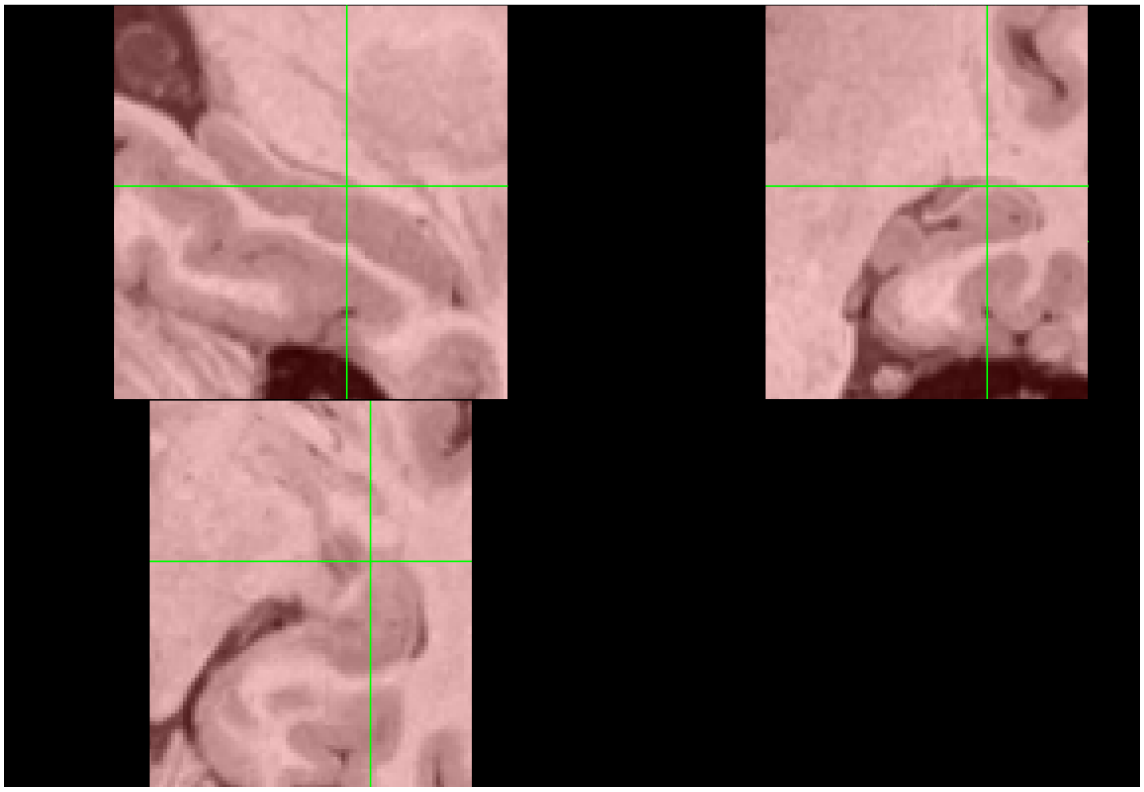


Ilustración 29. Resultado de la red volumétrica pequeña.

Como se puede observar en la imagen, la segunda red basada en volumen tampoco es un buen modelo de red para la segmentación de imágenes médicas, ya que predice que toda la imagen es fondo y los pocos puntos de objeto son falsos positivos. Las primeras predicciones del modelo son relativamente buenas, pero con el paso de las correcciones el modelo pierde calidad, acercándose a la predicción errónea de todo fondo.

Pruebas de redes basadas en *patches*

En este apartado se van a desarrollar las pruebas de los modelos basados en *patches*, los cuales van a ser siete en total, los modelos de *patches* de 16 y 32 voxels, con y sin concatenación y con y sin *subsampling*.

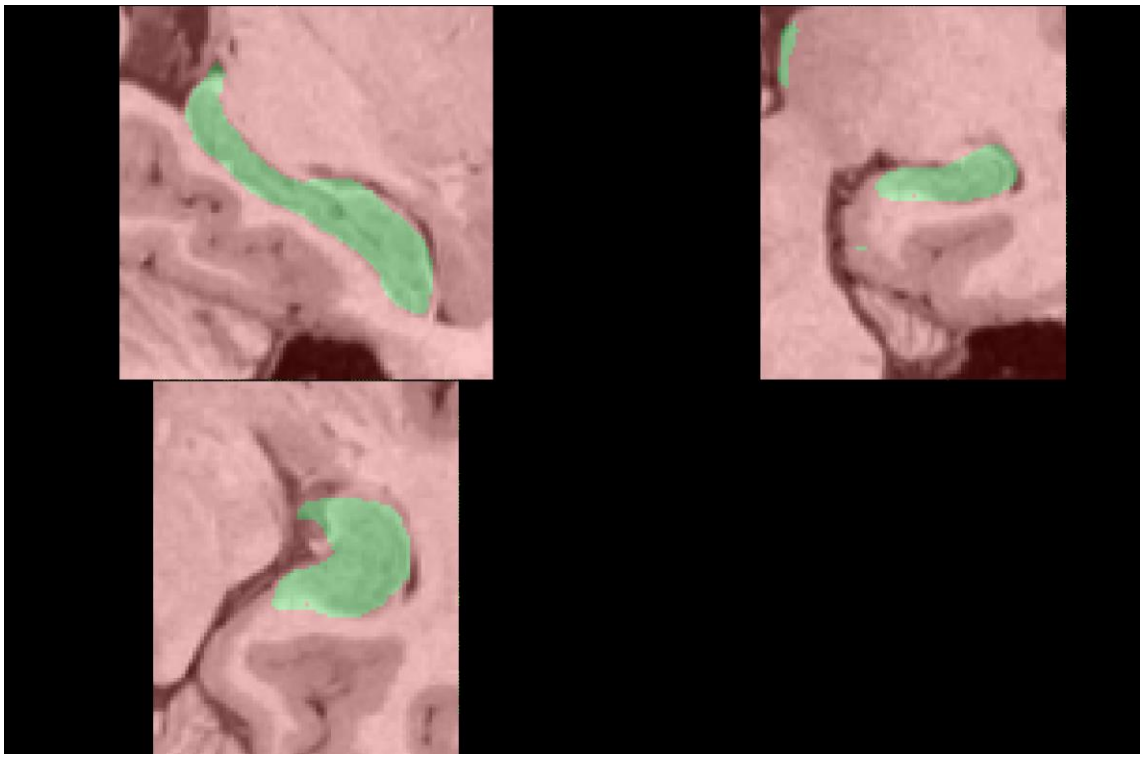
Para el entrenamiento de las redes basadas en *patches* se han añadido dos *callbacks*. La primera se dedica a hacer guardados periódicos de los pesos de la red durante el entrenamiento en la carpeta seleccionada y la otra se encarga de monitorizar que la red aprenda lo suficiente durante el entrenamiento con los datos dados, en caso de no ser así detiene el entrenamiento.

Patches de 16 sin concatenación ni *subsampling*

En este caso se va a evaluar la red que hace uso de *patches* de tamaño 16, sin *subsampling* ni concatenación de puntos.

	Repeticiones					
	1	2	3	4		TOTAL
Train	15,01	13,87	45,88	57,38		132,14
Test	319,88	292,76	271,87	287,64		1172,15

Ilustración 30. Patches de 16 sin concatenación ni subsampling.



Se observa que con esta primera red basada en patches hay una gran mejoría, ya que se tiene una buena aproximación del hipocampo con cuatro correcciones, con algunos falsos positivos en las fronteras del objeto, y algunos falsos positivos en zonas

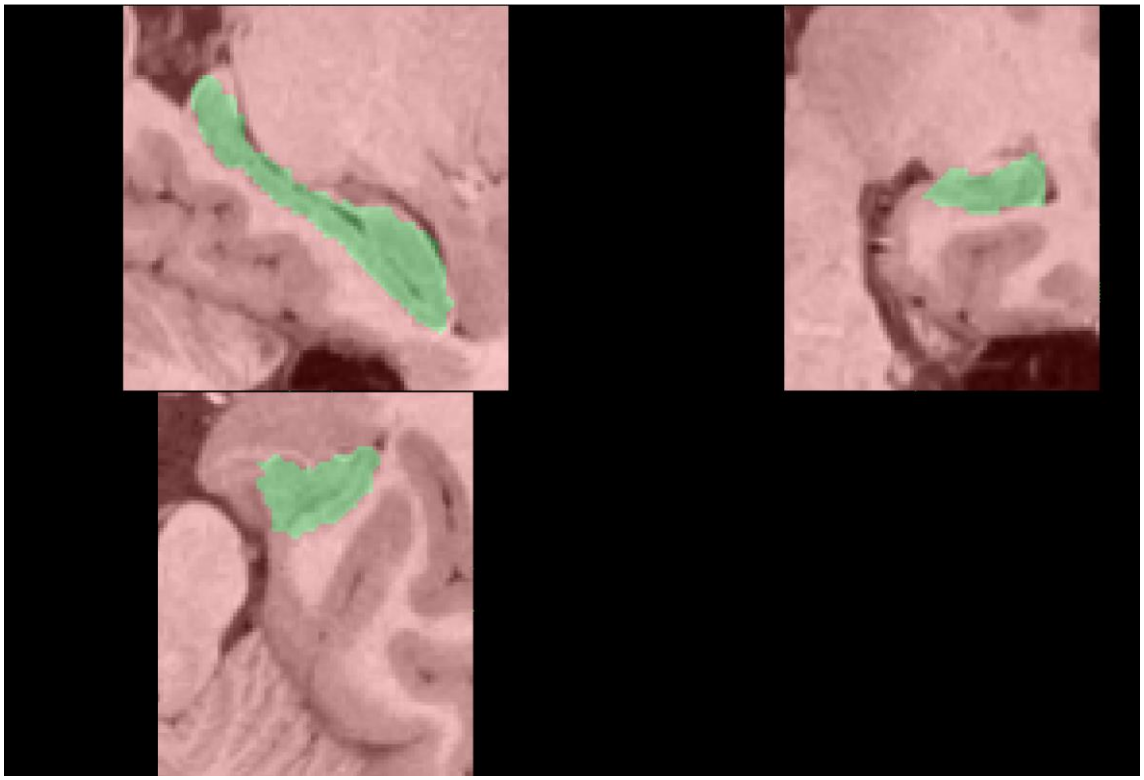
muy similares al hipocampo. El único cuello de botella es el tiempo de creación de máscara, ya que tiene que construir el *patch* de cada uno de los puntos de la imagen y evaluarlo con la red, por lo que se introduce el parámetro de *subsampling*.

Patches de 16 sin concatenación y con *subsampling*

En este caso se va a evaluar la misma red que en el caso anterior, pero introduciendo un *subsampling* a la imagen a la hora de testear, además de aplicar este valor también a los *patches*.

	Repeticiones					
	1	2	3	4	5	6
Train	9,38	8,8	17,14	35,52	16,06	28,86
Test	4,5	4,44	4,49	5,32	4,46	4,5
	TOTAL					
Train	115,76					
Test	27,71					

Ilustración 31. Patches de 16 sin concatenación y con subsampling.



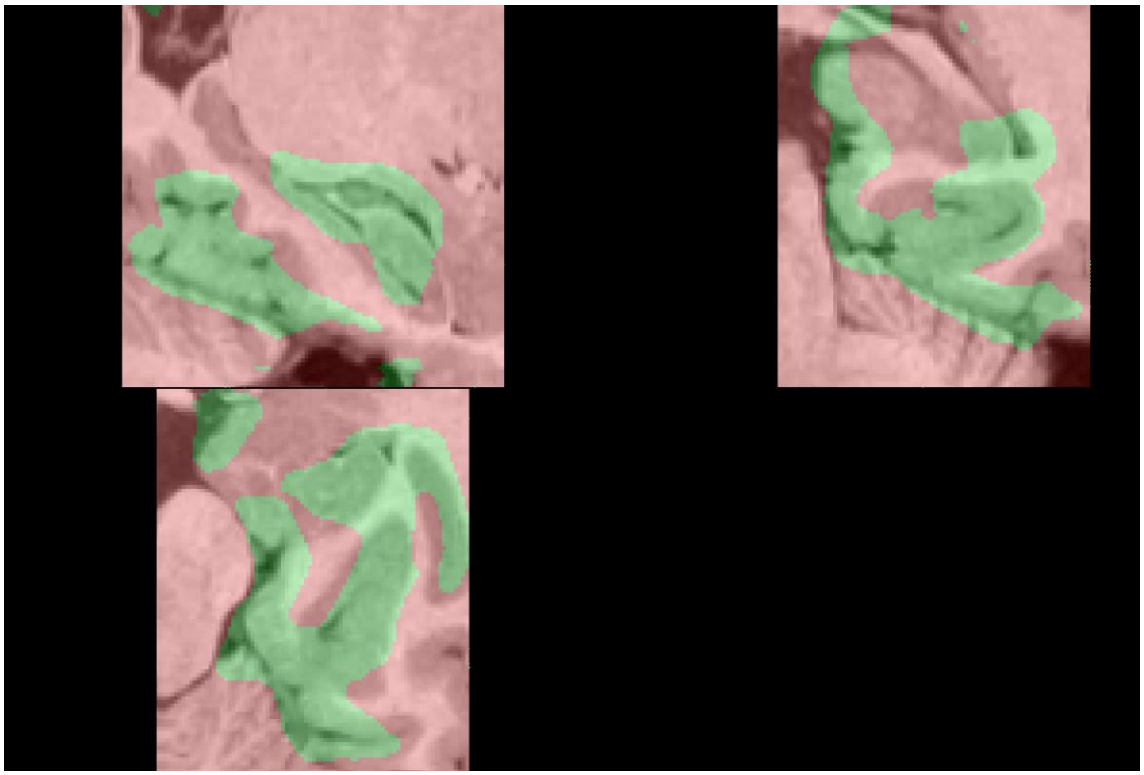
En este caso, se puede observar un resultado similar a la anterior prueba, con algo menos de precisión, debido a la reducción del tamaño, pero con un tiempo de testeo mucho menor que en el caso anterior, y con más correcciones para llegar a un resultado similar.

Patches de 16 con concatenación y sin *subsampling*

En este caso se va a hacer uso de la red en la cual se concatenan los puntos de los *patches* al final, sin aplicar el factor de *subsampling*.

	Repeticiones					
	1	2	3	4	5	TOTAL
Train	9,92	16,97	20,08	22,73	38,83	108,53
Test	276,34	285,19	286,64	281,21	305,05	1434,43

Ilustración 32. Patches de 16 con concatenación y sin subsampling.



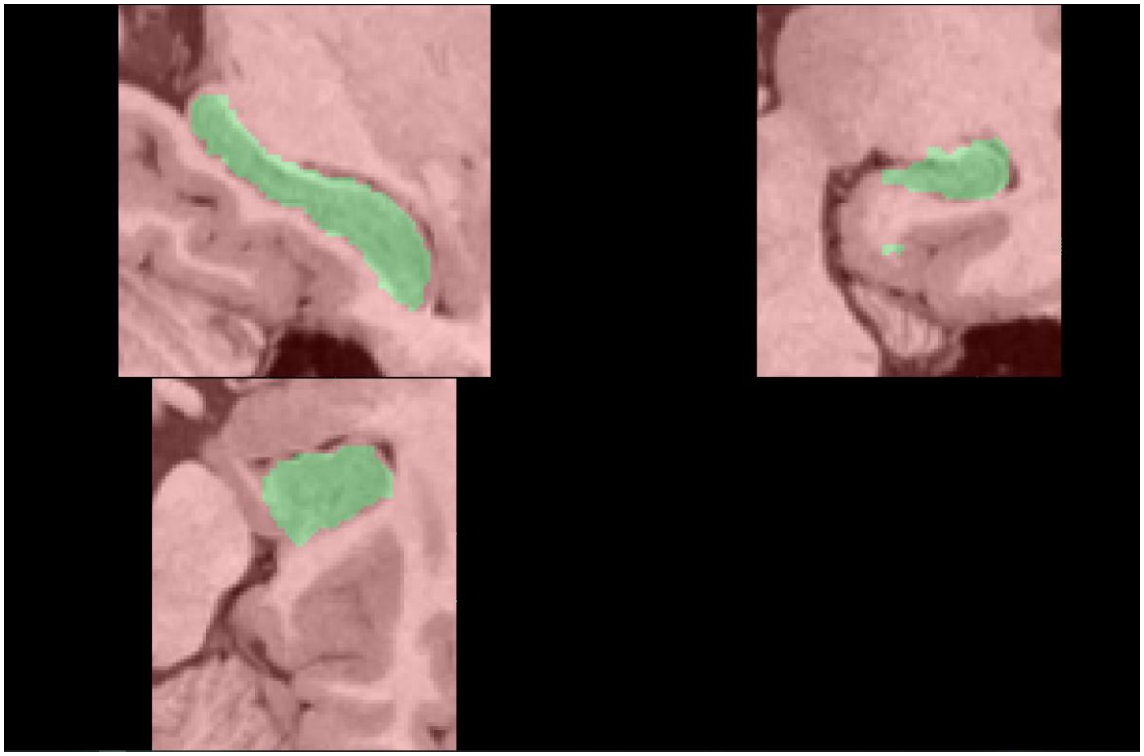
Se puede comprobar que con este modelo de red la segmentación es muy irregular, con muchos falsos positivos y muchos falsos negativos, además que el tiempo de test es del orden de diez veces mayor que el tiempo de entrenamiento, lo que dificulta el proceso interactivo.

Patches de 16 con concatenación y con *subsampling*

En este caso se va a evaluar la red que hace uso de *patches* de 16 y concatenación de puntos, con un factor de *subsampling* de dos.

	Repeticiones					
	1	2	3	4	5	6
Train	9,49	21,71	25,45	42,23	15,08	22,72
Test	4,96	4,71	4,79	4,82	4,8	4,86
	TOTAL					
Train	136,68					
Test	28,94					

Ilustración 33. Patches de 16 con concatenación y con subsampling.



En esta segmentación se puede observar un resultado muy parecido a la evaluación de la red sin concatenación de puntos, con bordes menos redondeados por la aplicación del factor de subsampling, y con un número equivalente en las correcciones.

Patches de 32 sin concatenación y sin *subsampling*

Para esta prueba se va a hacer uso de la red que hace uso de *patches* de 32, sin concatenación de puntos ni factor de *subsampling*.

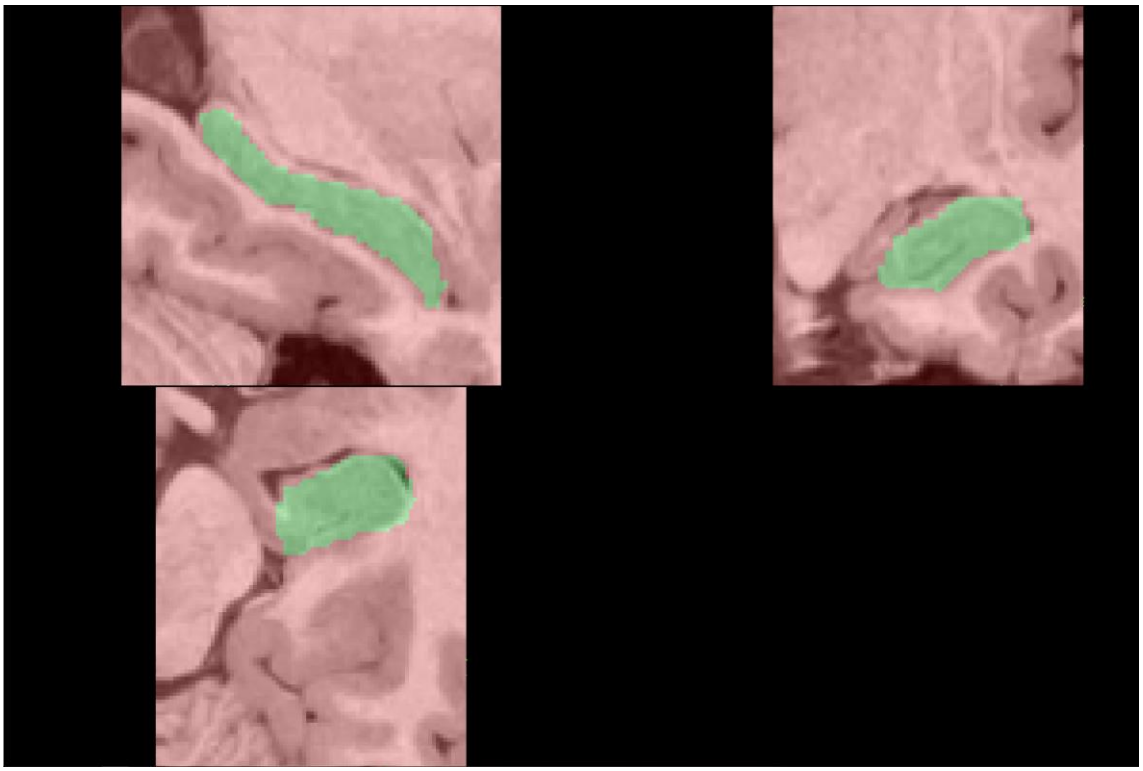
Hacer uso de *patches* de 32 sin factor de *subsampling* conlleva esperar una media de 15 minutos por cada evaluación de test, además, el requerimiento de memoria RAM es superior a 16 GB. Es por esto que se va a descartar en un futuro hacer pruebas con modelos con patches de 32 sin factor de *subsampling*, ya que la espera hace que el flujo de trabajo no sea interactivo.

Patches de 32 sin concatenación y con *subsampling*

Para llevar a cabo esta prueba se va a hacer uso del modelo de red que utiliza *patches* de 32, sin concatenación de puntos en la última capa, y con *subsampling*.

	Repeticiones					
	1	2	3	4	5	TOTAL
Train	30,15	52,7	84,77	123	91,65	382,27
Test	30,33	30,39	30,67	28,54	30,18	150,11

Ilustración 34. Patches de 32 sin concatenación y con subsampling.



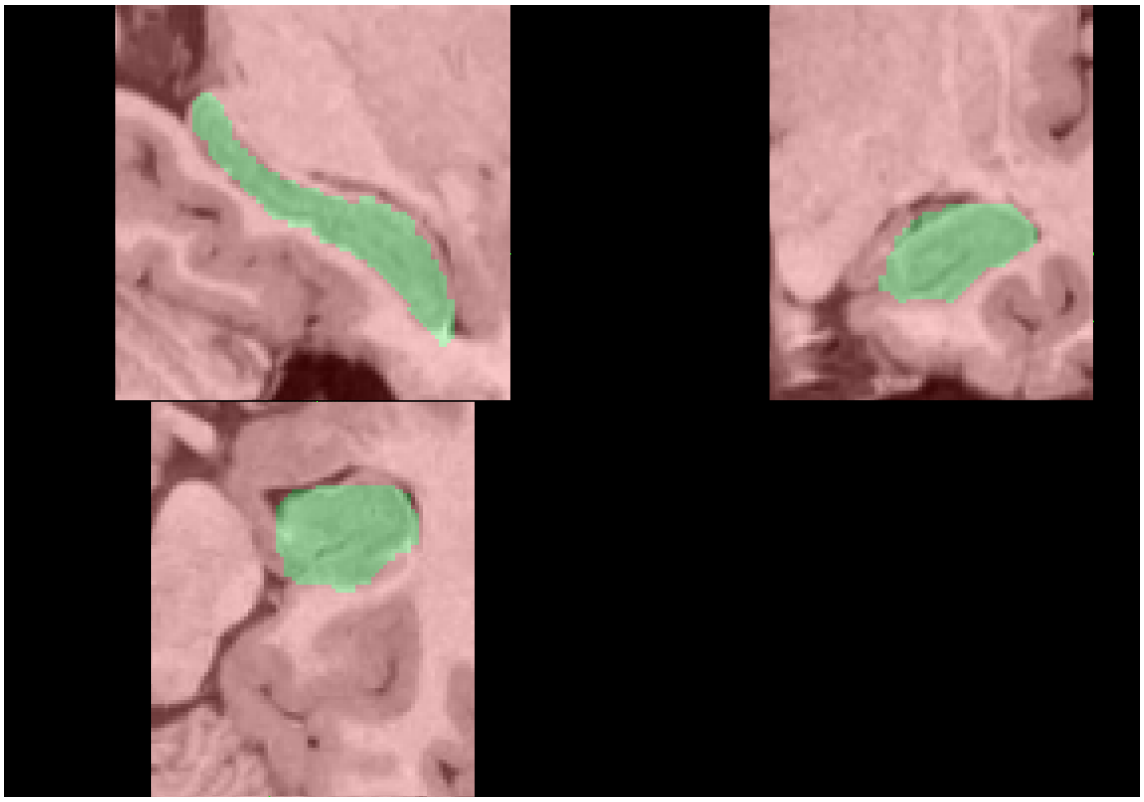
Como se puede observar, se presenta un resultado parecido a las pruebas anteriores, con un tiempo de entrenamiento mayor, pero un número menor de correcciones.

Patches de 32 con concatenación y con *subsampling*

Para llevar a cabo esta prueba se va a hacer uso del modelo de red que hace uso de los *patches* de 32, con concatenación de puntos en la última capa y con *subsampling.*

	Repeticiones					
	1	2	3	4	5	TOTAL
Train	12,77	20,63	26,48	60,06	84,17	204,11
Test	31,16	30,61	30,46	31,32	31,93	155,48

Ilustración 35. Patches de 32 con concatenación y con subsampling.



Este modelo de red sea aquel que mejor se comporta, ya que desde las primeras comprobaciones conseguía unas muy buenas aproximaciones del hipocampo, con muy pocos falsos positivos y falsos negativos.

En primer lugar, comentar que las correcciones se tienen que hacer de forma homogénea en las tres vistas, ya que, si se centran en alguna de las tres vistas casi exclusivamente a la hora se colocar puntos, la red no aprende de forma correcta y los resultados no son muy satisfactorios.

En segundo lugar, se va a comentar los resultados de los modelos de las distintas redes. Como se ha podido comprobar, las redes volumétricas no eran la mejor opción para la segmentación de imagen médica, ya que sus resultados eran de muy baja calidad y precisión, ya que, en las primeras correcciones de la segmentación, la red se sobreentrenaba e indicaba todo como fondo u objeto.

En tercer lugar, comentar que con los modelos de redes basadas en *patches* se obtenía un mejor resultado. Se ha podido comprobar como el tiempo de evaluación aumenta con el tamaño del *patch*, además, los modelos de redes con concatenación de puntos en la última capa obtenían un mejor resultado en un tiempo menor. Comentar también que los tiempos de entrenamiento son variables en todos los casos, aunque también aumentan con el tamaño de *patch*, el tiempo era más variable, debido a la *callback* "*EarlyStopping*", que detiene el proceso de entrenamiento en caso de que no haya mejora notable entre un número dado de *epochs*.

Para finalizar, hay que comentar que los mejores resultados se han obtenido con los modelos de redes con mayor tamaño de *patch* y con concatenación de los puntos en la última capa de las redes. También cabe destacar que es un software que requiere unos componentes de alta gama para conseguir unos resultados medianamente buenos, y, sobre todo, que el flujo de trabajo sea interactivo, ya que con tarjetas gráficas de dos generaciones por detrás a la presente, disponen de muy pocos núcleos CUDA, por lo que el tiempo en estos casos, o en el caso en que no se disponga de tarjeta gráfica, el tiempo se vería aumentado significativamente, lo que rompería el flujo de trabajo interactivo, por lo que se estaría dejando de lado el principal objetivo de este proyecto.

Capítulo 7

Conclusiones

En primer lugar, durante el desarrollo de este proyecto se han ido aprendido conceptos y tecnologías nuevos, como el desarrollo de una interfaz de usuario de manera “clásica”, ya que el proceso ha sido sin un *framework* de diseño. Además, se han aprendido conceptos sobre imagen médica y tratamiento complementarios a los vistos en la asignatura de Informática médica. También el uso de librerías para trabajar con este tipo de imagen, y principalmente, una muy buena introducción al mundo de la inteligencia artificial con el desarrollo de las redes, sus modelos, y su comportamiento con las distintas imágenes médicas.

También se ha familiarizado con el entorno de desarrollo integrado PyCharm, además de conocer el funcionamiento básico de los navegadores dedicados a la imagen médica, como por ejemplo ITK-Snap, además de obtener una buena resolución con las distintas librerías científicas, como Matplotlib, Numpy, Tkinter, Nibabel o Keras.

Comentar que todos los objetivos propuestos se han llevado a cabo de una manera satisfactoria en mayor o menor medida, ya que desde un principio se esperaba un mejor resultado de las predicciones de las redes, pero se han limitaciones por los requerimientos de hardware.

A raíz de los resultados obtenidos con los modelos de las redes basadas en volumen, un nuevo objetivo podría ser llevar a cabo un desarrollo de modelos de redes volumétricas que tuvieran un comportamiento similar al de las redes basadas en *patches*, ya que los tiempos de entrenamiento y comprobación en este tipo de modelos eran más que aceptables.

La parte más difícil del proyecto fue la parte en la que se tuvo que dar funcionalidad a la herramienta, ya que era código que nunca se había visto, además de ser complejo, ya que se trataba de hacer simple un proceso que a priori me era muy complejo. Otro apartado de dificultad fue cuando se tuvo que editar el navegador de imágenes médicas, ya que no había prácticamente comentarios y era código muy depurado y difícil de entender, además de ser muchas líneas de código.

Relación del trabajo desarrollado con los estudios cursados

Durante el desarrollo de este proyecto se han tenido en cuenta los conocimientos de las asignaturas de desarrollo de interfaces, aunque es poca medida, ya que estas se desarrollaban con diseñadores interactivos, y sus lenguajes eran Java y C#. Luego, al haber cursado la rama de computación, Python es el lenguaje predilecto. También comentar todo el aprendizaje obtenido con Matlab durante toda la rama de computación, esencial a la hora de trabajar con las imágenes en forma de

matriz, ya que Matlab y la librería Matplotlib de Python son prácticamente idénticas. Luego, y visto en menor medida, todo el refuerzo que ha conllevado el desarrollo de este proyecto de las asignaturas optativas de cuarto, bioinformática e informática médica, por el uso de imágenes médicas.

Para finalizar comentar, que es proyecto ya se llevó a cabo, en menor escala, como proyecto final de la asignatura de informática médica, el cual consistió en realizar también una interfaz en Matlab para la segmentación de imágenes médicas de dos dimensiones.

Capítulo 8

Trabajos futuros

En este capítulo se van a comentar posibles mejoras, proyectos que puedan derivar o ideas surgidas en el desarrollo de este proyecto que quedarían excluidas del alcance de este por distanciarse de este o por ser muy ambiciosos.

En primer lugar, una buena ampliación de este proyecto podría ser el estudiar qué modelos de redes basadas en volúmenes son las más acertadas a la hora de segmentar imágenes médicas, su estructura, su forma de aprender de la máscara, sus parámetros, etc.

En segundo lugar, estudiar la manera para que los requerimientos de hardware para poder trabajar con este tipo de herramientas no fueran tan altos, ya que en muchos casos los requerimientos pueden llegar a ser privativos, debido a que las tarjetas gráficas de NVIDIA tienen un alto coste.

En tercer lugar, podría adaptarse dicho proyecto para que, sin perder mucha precisión de segmentación, poder ejecutar esta herramienta en equipos en los que no se cuenta con una tarjeta gráfica.

En cuarto lugar, adaptar esta herramienta para trabajar con imagen médica descompuesta en dos dimensiones intentar un flujo de trabajo más interactivo disminuyendo el tiempo de espera durante el entrenamiento y creación de segmentaciones.

Por último, una pequeña ampliación para este proyecto podría ser el estudio de nuevos modelos de redes que puedan llegar a ser útiles a la hora de trabajar con segmentaciones de imágenes médicas.

Bibliografía

[1] Carsten Rother, Vladimir Kolmogorov, Andrew Blake, “GrabCut”: interactive foreground extraction using iterated graph cuts, 2004, https://dl.acm.org/citation.cfm?id=1015720

[2] Di Lin, Jifeng Dai, Jiaya Jia, Kaiming He, Jian Sun, ScribbleSup: Scribble-Supervised Convolutional Networks for Semantic Segmentation, 2016, https://arxiv.org/pdf/1604.05144.pdf

[3] Won-Dong Jang, Chang-Su Kim, Interactive Image Segmentation via Backpropagating Refinement Scheme ,2019, https://vcg.seas.harvard.edu/publications/interactive-image-segmentation-via-backpropagating-refinement-scheme/paper

[4] Eirikur Agustsson, Jasper R. R. Uijlings, Vittorio Ferrari, Interactive Full Image Segmentation by Considering All Regions Jointly, 2019, https://arxiv.org/pdf/1812.01888.pdf

[5] Tomas Sakinis, Fausto Milletari, Holger Roth, Panagiotis Korfiatis, Petro Kostandy, Kenneth Philbrick, Zeynettin Akkus, Ziyue Xu, Daguang Xu, Bradley J. Erickson, Interactive segmentation of medical images through fully convolutional neural networks, 2019, https://arxiv.org/pdf/1903.08205.pdf

[6] Definición de red neuronal. http://redes-neuronales.wikidot.com/introduccion-definicion-redes-neuronales

[7] Red neuronal convolucional. http://www.diegocalvo.es/red-neuronal-convolucional/

[8] Función de activación. http://www.diegocalvo.es/funcion-de-activacion-redes-neuronales/

[9] Función *loss.* *https://gombru.github.io/2018/05/23/cross_entropy_loss/*

Apéndice A
GLOSARIO

En este glosario se van a explicar los distintos términos que el lector pueda desconocer a la hora de leer la memoria.

- **Red neuronal:** Las redes neuronales son un modelo computacional inspirado en el comportamiento observado en su homólogo biológico. Consiste en un conjunto de nodos, llamados neuronas conectadas entre sí para transmitir señales. La información de entrada pasa a través de la red de neuronas produciendo unos valores de salida.

- **Segmentación:** La segmentación en el campo de la visión artificial es el proceso de dividir una imagen digital en varias partes (grupos de píxeles) u objetos.

- **NIFTI:** Es un formato utilizado en imagen médica, el cual significa *Neuroimaging Informatics Technology Initiative.*

- **Python:** Lenguaje multiparadigma, debido a que soporta orientación a objetos, programación imperativa y en menor medida programación funcional. Es interpretado, de tipado dinámico y multiplataforma.

- **Metodología ágil:** Método de desarrollo para llevar a cabo productos y servicios de calidad que respondan a las necesidades de los clientes cuyas prioridades cambian a una velocidad cambian cada vez a mayor velocidad.

- ***Clustering*:** Consiste en dividir la población en un número de grupos en los cuales los puntos en un mismo grupo son similares entre sí, y distintos de los puntos del resto de grupos.

- **GIF:** Un GIF (*Graphics Interchange Format*) es un tipo de formato de compresión de imagen limitado a 256 colores, que permite fusionar unas imágenes para realizar un video sin audio de entre tres y cinco segundos.

- **RAM:** Siglas de *Random Access Memory* (memoria de acceso aleatorio), la cual forma la memoria principal de un ordenador, donde residen la ejecución de los programas y sus datos, sobre la que se pueden hacer operaciones de lectura y escritura.

- **NVIDIA:** Es una empresa multinacional especializada en el desarrollo de unidades de procesamiento gráfico y tecnologías de circuitos integrados para estaciones de trabajo, ordenadores personales y dispositivos móviles.

- **Keras:** Keras es una biblioteca de redes neuronales de código abierto escrita al completo en Python.

- **Tensorflow:** Tensorflow es una biblioteca de código abierto para aprendizaje automático a través de un rango de tareas, y desarrollado por Google para satisfacer sus necesidades de sistemas capaces de construir y entrenar redes neuronales para detectar y descifrar patrones y correlaciones, análogos al aprendizaje y razonamiento usados por los humanos.

- **API:** Las API son un conjunto de comandos, funciones y protocolos informáticos que permiten a los desarrolladores crear programas específicos para ciertos sistemas operativos y lenguajes.

- **Java:** Java es un lenguaje de programación orientado a objetos y una plataforma informática.

- **C:** C es un lenguaje de programación orientado a la implementación de sistemas operativos.

- ***Plugins*:** Un *plugin* es un aplicación o programa informático que se relaciona con otro para agregarle nuevas funcionalidades.

- **Conda:** Conda es un gestor de paquetes y de entornos virtuales multiplataforma que nos permite instalar, ejecutar y actualizar los paquetes de Python y sus dependencias.

- **Convolución:** Una convolución es un conjunto de operaciones de sumas y multiplicaciones que se realizan entre la capa de entrada de la red y las capas ocultas, lo que genera un mapa de características.

- ***Pooling*:** El concepto de *pooling* consiste en una reducción de la imagen en base a un factor, por ejemplo, un *pooling* de dos, significa reducir a la mitad las dimensiones de la imagen.

- **Función de activación:** Una función de activación es aquella que utiliza la misma suma ponderada de la entrada anterior y la transforma una vez más como salida.

- ***Epoch*:** Una *epoch* en Deep Learning es un hiperparametro definido antes del entrenamiento. Una *epoch* se da cuando todo el *dataset* ha iterado por completo por toda la red neuronal, es decir, que la red ha hecho una pasada sobre todos los datos de entrada.

- **Función *loss:*** Las funciones *loss* funciones se utilizan para medir la calidad del modelo con respecto a los datos dados.

- ***Backend*:** Dentro de un proyecto de *software* es la capa que permite el acceso a los datos y su manipulación.

- **Tensor:** Los tensores son la estructura de datos que utiliza la librería Tensorflow.

- ***Padding*:** Concepto informático que consiste en aumentar el tamaño de una imagen en función de un factor.
- ***Patch***: Un *patch* es una subsección de la imagen de tamaño fijo, la cual tiene información relevante para segmentar el objeto en cuestión.
- ***Subsample***: Concepto informático, el cual consiste en reducir el tamaño de una imagen, dividiendo el tamaño de esta por un parámetro dado.
- ***Callback***: Una devolución de llamada o *callback* es una función “A” que se usa como argumento de otra función “B”.

Apéndice B
Manual de usuario de la herramienta

Abrir imagen médica

En primer lugar, una vez arrancada la herramienta, se selecciona el botón de "*Load image"* para cargar una imagen, ya sea para continuar con el entrenamiento de una red, o para crear un nuevo tipo de red.

Crear red

Una vez se tiene la imagen que se quiere segmentar, se tiene que crear la red con los parámetros adecuados para la región a segmentar. Para ello, se selecciona el botón "*Create net*", con lo que aparece una ventana en la que se establecen los parámetros de la red.

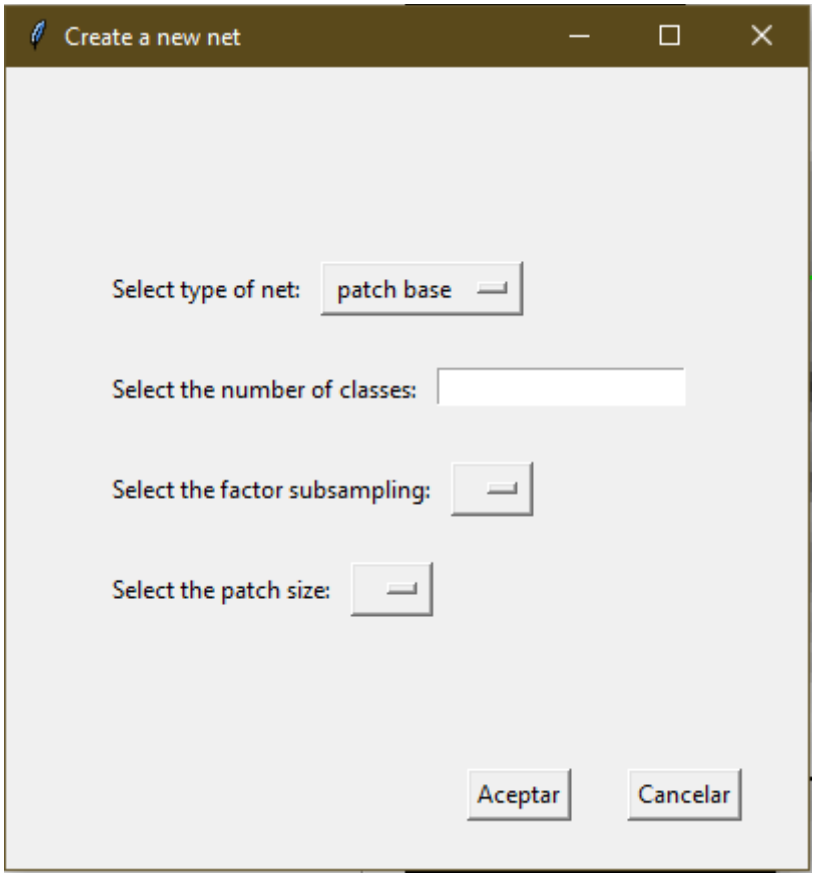


Ilustración 36. Ventana para configurar red.

Una vez se tenga parametrizada la red, al seleccionar el botón de aceptar, aparecerá un diálogo de archivos, el cual permite seleccionar el fichero en el que se

guardan los puntos y sus clases, los puntos de los *patches*, la red y la máscara en caso de que se seleccione el botón de "*Save mask*".

Introducir puntos

Una vez se tiene la red creada, se puede proceder a introducir los puntos en la máscara. Para ello, se selecciona la clase de la que se quieren introducir los puntos con el desplegable "c*lass*", y se selecciona el botón de "*Start label selection*" para que cada clic en una de las vistas marque dicho punto de la clase correspondiente.

Para corregir cualquier punto, basta con hacer seleccionarlo con su clase correspondiente en cualquiera de las tres vistas.

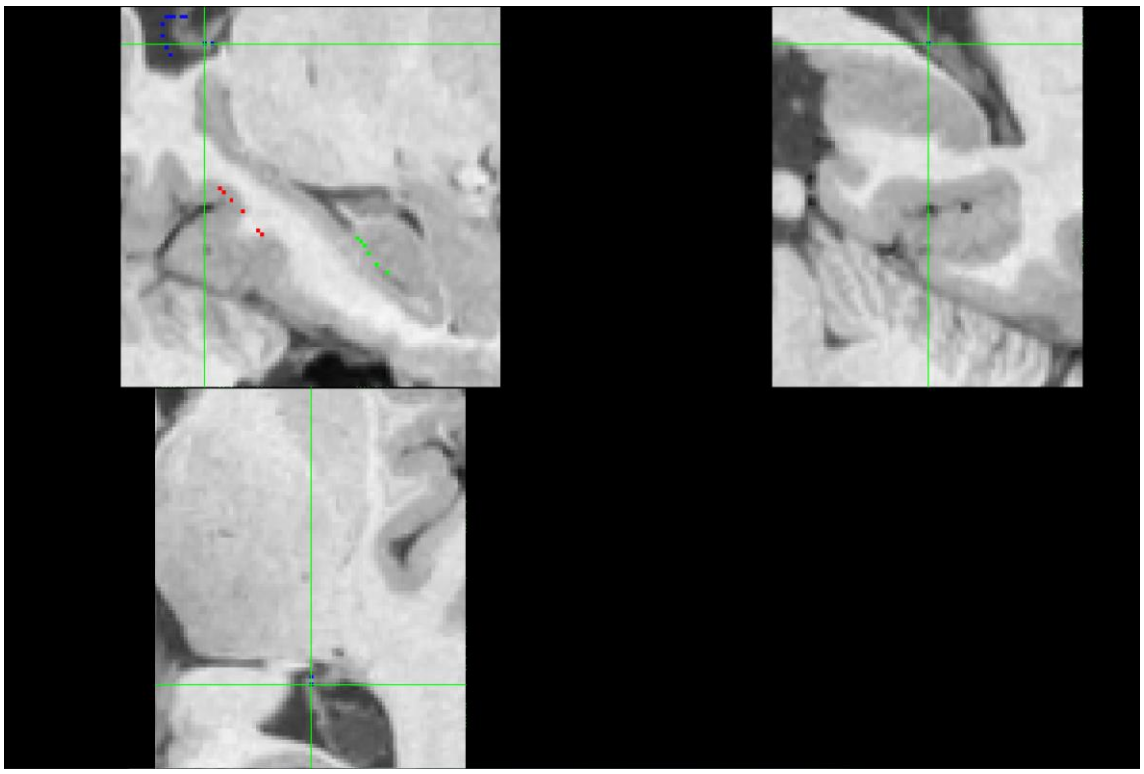


Ilustración 37. Ejemplo de introducción de puntos.

Entrenar la red

Una vez se tiene un número suficiente para un entrenamiento inicial, se selecciona el botón de "*Train*" para empezar el proceso de entrenamiento. En el terminal en el que se haya ejecutado la herramienta se podrá observar el curso del proceso, y la ventana emergente de "*Training*" desaparecerá una vez el proceso haya acabado.

Ver resultados del entrenamiento

Una vez el proceso de entrenamiento haya acabado, se puede seguir introduciendo puntos, como en el punto tres, o comprobar el resultado del entrenamiento. Para ver la segmentación generada, basta con seleccionar el botón de "*Test*", lo que lanzará el proceso que genera una segmentación en base a los parámetros de la red. Este proceso puede ser más o menos costoso en función de los parámetros de la red, o el tamaño de la imagen, ya que tiene que evaluar cada uno de los puntos de la red.

Una vez haya acabado el proceso, la ventana emergente de "*Testing*" se cerrará, se tendrá que hacer clic sobre alguna de las tres vistas para recargar el visualizador y ver la segmentación generada.

Corrección de puntos

En caso de que la segmentación no sea satisfactoria, basta con corregir los puntos mal clasificados, como se indica en el punto tres.

Reanudación de entrenamiento

En caso de que se tenga que cerrar la herramienta y se quiera continuar más tarde, basta con seleccionar el botón *"Save net*", con lo que se guardará toda la información necesaria en la carpeta seleccionada al crear la red, para poder continuar con el entrenamiento más adelante.

Una vez se quiera continuar con el proceso de entrenamiento, basta con abrir la imagen con lo que se quiera continuar, como se indica en el punto uno, crear una red idéntica a la creada, y sobrescribirla seleccionando el botón "*Load net*", lo que te abrirá un dialogo de ficheros, por lo que se tendrá que seleccionar el fichero donde se guardó toda la información relevante a la red.