Cálculo de dimensiones de regiones de interés del cerebro en resonancias magnéticas.

Miguel Segura Anaya
Máster Ingeniería Informática
Inteligencia Artificial

Samir Kanaan Izquierdo
Carles Ventura Royo

05/06/2018

A mi adorada mujer. Mi faro en los días de niebla.

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Cálculo de dimensiones de regiones de interés del cerebro en resonancias magnéticas.*
Nombre del autor:	*Miguel Segura Anaya*
Nombre del consultor/a:	*Samir Kanaan Izquierdo*
Nombre del PRA:	*Carles Ventura Royo*
Fecha de entrega (mm/yyyy):	*06/2018*
Titulación o programa:	Máster Ingeniería Informática
Área del Trabajo Final:	*Inteligencia Artificial*
Idioma del trabajo:	*Castellano*
Palabras clave	*Diagnóstico por imagen, Machine learning, Red neuronal convolucional*

Resumen del Trabajo (máximo 250 palabras):

El uso de la informática como herramienta de ayuda a la medicina es una realidad. El manejo de la información es algo integrado en la práctica clínica. Algunas de las aplicaciones más conocidas se encuentran en el diagnóstico por imagen. Debido a la calidad de la imagen y la sensibilidad, la MRI está considerada actualmente como la mejor prueba para evaluar las anormalidades o trastornos en el ser humano.

Una de las últimas tendencias en el campo de la bioinformática, es la aplicación del "machine learning" como herramienta de evaluación de estas pruebas médicas. Este trabajo se centra en explorar este campo y definir un escenario de trabajo donde poder unir estas dos tecnologías.

Tomando como referencia inicial FreeSurfer, una de las herramientas más populares en el estudio de las MRI, intentaremos obtener los mismos resultados utilizando un sistema de inteligencia artificial.

Comenzaremos nuestro trabajo analizando las capacidades de FreeSurfer y seleccionando de entre todas las medidas que calcula nuestro objetivo inicial.

A continuación, analizaremos las opciones de fuentes de datos y el formato de estos datos de cara a disponer de la suficiente información para poder entrenar y verificar nuestros sistemas.

Por último, seleccionaremos un entorno de trabajo para nuestros modelos de inteligencia artificial, los configuraremos y los entrenaremos con la finalidad de descubrir el nivel de precisión que nuestros resultados pueden llegar a tener.

El trabajo termina con una evaluación de los resultados obtenidos, una lista de conclusiones y propuestas de evolución.

Abstract (in English, 250 words or less):

The use of information technology as a tool to help medicine is a reality. The management of information is something integrated in clinical practice. Some of the most well-known applications are found in diagnostic imaging. Due to the quality of the image and sensitivity, MRI is currently considered the best test to evaluate abnormalities or disorders in humans.

One of the latest trends in the field of bioinformatics is the application of "machine learning" as a tool for evaluating these medical tests. This work focuses on exploring this field and defining a framework where we can merge these two technologies.

Taking as initial reference FreeSurfer, one of the most popular tools in the study of MRI, we will try to obtain the same results using an artificial intelligence system.

We will start our work analyzing the capabilities of FreeSurfer and selecting among all the measures that calculate our initial objective.

Next, we will analyze the options of data sources and the format of these data in order to have enough information to be able to train and verify our systems.

Finally, we will select a work environment for our artificial intelligence models, we will configure them and train them in order to discover the level of precision that our results may have.

The work ends with an evaluation of the obtained results, a list of conclusions and proposals to evolve the system.

Índice

Contents

Lista de figuras

1. Introducción.

El desarrollo de las nuevas tecnologías informática ha traído consigo cambios en la forma de actuar y de pensar en la sociedad, creando nuevas pautas en el comportamiento humano. Resulta incuestionable el auge cada vez mayor del uso y aplicación de las Tecnologías de la Información y las Comunicaciones (TIC) en los centros especializados de la salud y hospitales alrededor del mundo, por medios de los especialistas de las diferentes disciplinas de la medicina.

La informática médica comienza en los primitivos métodos manuales para registrar la información de interés médico e historias clínicas. Estas se desarrollaron conforme a la necesidad de contar con una mayor cantidad de información, hasta llegar a los complejos archivos de los hospitales que se tornaron inmanejables. A partir de este momento, es cuando comienza la introducción de las computadoras para responder a la necesidad del manejo de toda esa información.

En la actualidad, los avances de la informática en la medicina han sido muy grandes, generando increíbles resultados tanto en las áreas administrativa, académica, de investigación y clínica.

El uso de la informática como herramienta de ayuda a la medicina es una realidad, el manejo de la información es algo integrado en la práctica clínica. Médicos y pacientes interactúan en una compleja estructura de información, que ayuda a mejorar la calidad asistencial, garantizando la recuperación de la salud a toda persona enferma, la prevención de enfermedades y la mejora de la salud de la población en general.

Algunas de las aplicaciones más conocidas de este campo de la informática se encuentran en el diagnóstico por imagen, la telemedicina, los sistemas de gestión hospitalaria y el registro clínico electrónico.

I. Contexto y justificación del Trabajo.

En el diagnóstico por imagen se puede poner como ejemplo la realización y posterior análisis de las resonancias magnéticas o MRI, como se las conoce comúnmente.

Debido a la calidad de la imagen y la sensibilidad, la MRI está considerada actualmente como la mejor prueba para evaluar las anormalidades o trastornos en el ser humano. Aplicadas al cerebro, por ejemplo, permite detectar entre

otros tumores cerebrales, accidentes cardiovasculares, esclerosis múltiple, convulsiones, demencia, aneurismas, etc.

Una de las últimas tendencias en el campo de la bioinformática, es la aplicación del "Machine Learning" como herramienta de evaluación de estas pruebas médicas.

"La inteligencia artificial tiene un potencial enorme para revolucionar el diagnóstico de enfermedades y su tratamiento, al ser capaz de hacer al instante análisis y clasificaciones que involucran una gran cantidad de datos muy difíciles o imposibles de manejar para los humanos", considera, en una entrevista a Big Vang, Kang Zhang, director del Instituto de medicina genómica de la Escuela de Medicina de la Universidad de California San Diego (EE.UU.).

Esta aplicación del "Machine Learning" en la evaluación del MRI es la temática escogida para este trabajo.

II. Objetivos del Trabajo.

El objetivo general de este trabajo es el de disponer de un sistema capaz de obtener los mismos resultados que la herramienta FreeSurfer, a partir de la misma fuente de datos, pero en menor tiempo.

Este sistema podrá ser evolucionado posteriormente para obtener resultados más complejos que ayuden al diagnóstico y a la prevención y tratamiento de todo tipo de enfermedades. Esta evolución posterior no entra en el alcance de este trabajo.

Los objetivos específicos de este trabajo se dividen en tres partes:

- En primer lugar, plantearemos unos objetivos de resultados iniciales que vendrán determinados por las capacidades actuales de FreeSurfer. Para ello realizaremos una instalación del programa FreeSurfer y estudiaremos cuales son las medidas que este software permite obtener.

  A partir de las medidas disponibles, realizaremos una selección, eligiendo aquellas que puedan ser más relevantes. El nivel de complejidad y la evolución de este trabajo determinarán la cantidad de medidas que podremos obtener finalmente.

- En segundo lugar, realizaremos un análisis sobre diferentes fuentes de libre acceso que ofrezcan resultados de resonancias magnéticas del cerebro disponibles actualmente.

  Estas fuentes nos tienen que permitir adquirir las muestras necesarias para comenzar el entrenamiento de nuestro sistema. Intentaremos que estas fuentes sean lo más diversas y grandes posibles, de manera que

permitan dotar a nuestro sistema de unos datos iniciales con los que obtener unos resultados de suficiente calidad.

El formato proporcionado por estas fuentes condicionará posiblemente la herramienta a utilizar para el análisis de estas.

- Por último, analizaremos y seleccionaremos la red neuronal convolucional, u otro sistema similar que sea capaz de procesar las imágenes tridimensionales resultantes de las resonancias magnéticas.

  Este sistema será entrenado con nuestras fuentes de datos seleccionadas para conseguir los resultados deseados.

III. Enfoque y método utilizado.

Para este proyecto vamos a utilizar un enfoque de establecimiento de bases de trabajo, es decir vamos a intentar construir las herramientas y estándares necesarios para poder posteriormente realizar todos los experimentos y pruebas necesarias y permitir una posterior evolución y perfeccionamiento de los resultados.

Inicialmente necesitamos saber cuáles son los tipos de resultados con los que se trabajan en el campo del diagnóstico por imagen dentro de la exploración del cerebro. Para ello utilizaremos como referencia una herramienta muy popular en este ámbito llamada FreeSurfer.

A partir del análisis de capacidades de FreeSurfer, seremos capaces de determinar que se esperaría de nuestro sistema, es decir, que tipo de resultados y variables son aplicables en este tipo de estudios.

Una vez sepamos cuales son nuestros objetivos pasaremos a analizar las opciones que tenemos para obtener datos de ejemplo para realizar nuestros experimentos. Para ello analizaremos las fuentes de resonancias magnéticas que actualmente se encuentran disponibles en internet, en paralelo realizaremos un estudio de los formatos más comunes de MRI para que no utilicemos un formato que no sea común entre las fuentes de datos o que no sea compatible con nuestro sistema de IA. El objetivo final es evitar las conversiones lo máximo posible.

El punto final en la construcción de nuestras bases de trabajo será la selección de un sistema de IA que nos permita cumplir con nuestros objetivos. Una vez seleccionado el sistema comenzará una segunda fase donde aplicaremos la base construida para intentar obtener los resultados definidos con la mayor precisión posible.

El proceso de experimentación comenzará con la definición de un sistema sencillo que nos sirva para familiarizarnos con el sistema y unir todas nuestras

piezas hasta obtener una primera versión funcional que arroje los primeros resultados.

Este primer modelo se evolucionará, posteriormente, manteniendo la base, pero aumentando su complejidad. Estos modelos posteriores serán analizados para comprobar si este aumento de complejidad tiene un efecto directo positivo sobre los resultados.

Cuando ya tengamos un modelo suficientemente complejo, aplicaremos cambios en otros factores, como la calidad de las fuentes de datos o el tamaño de los grupos de entrenamiento. Veremos también como estos factores influyen en los resultados finales.

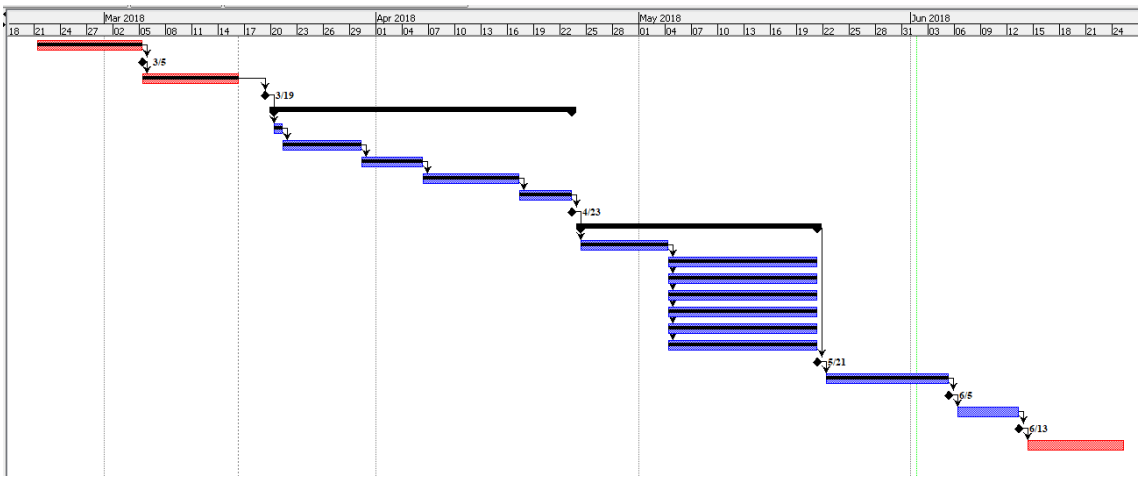
IV. Planificación del Trabajo.

Este trabajo está limitado temporalmente por la duración de un cuatrimestre lectivo. Considerando esta limitación, así como los hitos académicos que la asignatura presenta hemos establecido una planificación que utilizaremos para seguir la evolución del trabajo.

La planificación de Trabajo con sus correspondientes hitos es la siguiente:

	Name	Duration	Start	Finish	Predecessors
1	PAC0 - Definición de los contenidos del trabajo	8 days	2/21/18 9:00 AM	3/5/18 9:00 AM	
2	Entrega de Propuesta de TFM	0 days	3/5/18 9:00 AM	3/5/18 9:00 AM	1
3	PAC1 - Plan de trabajo	9 days	3/5/18 9:00 AM	3/16/18 9:00 AM	2
4	Entrega de Plan de Trabajo	0 days	3/19/18 9:00 AM	3/19/18 9:00 AM	3
5	**PAC2 - Desarrollo del trabajo - Fase 1**	24 days	**3/20/18 9:00 AM**	**4/23/18 9:00 AM**	
6	Descarga e instalacion de FreeSurfer	1 day	3/20/18 9:00 AM	3/21/18 9:00 AM	4
7	Analisis de capacidades	7 days	3/21/18 9:00 AM	3/30/18 9:00 AM	6
8	Seleccion de medidas objetivo	5 days	3/30/18 9:00 AM	4/6/18 9:00 AM	7
9	Análisis sobre diferentes fuentes de libre acceso de MRI	7 days	4/6/18 9:00 AM	4/17/18 9:00 AM	8
10	Determinacion de formato de MRI a utilizar	4 days	4/17/18 9:00 AM	4/23/18 9:00 AM	9
11	Entrega de Informe de Seguimiento	0 days	4/23/18 9:00 AM	4/23/18 9:00 AM	10
12	**PAC3 - Desarrollo del trabajo - Fase 2**	19 days	**4/24/18 9:00 AM**	**5/21/18 9:00 AM**	
13	Selección de sistema de IA	8 days	4/24/18 9:00 AM	5/4/18 9:00 AM	11
14	Seleccion de medida objetivo	11 days	5/4/18 9:00 AM	5/21/18 9:00 AM	13
15	Obtencion de resultados de en FreeSurfer de MRI de entrenan	11 days	5/4/18 9:00 AM	5/21/18 9:00 AM	13
16	Entrenamiento	11 days	5/4/18 9:00 AM	5/21/18 9:00 AM	13
17	Obtencion de resultados en FreeSurfer de MRI de testing	11 days	5/4/18 9:00 AM	5/21/18 9:00 AM	13
18	Verificacion del sistema entrenado	11 days	5/4/18 9:00 AM	5/21/18 9:00 AM	13
19	Ajustes del sistemas	11 days	5/4/18 9:00 AM	5/21/18 9:00 AM	13
20	Entrega de Informe de Seguimiento	0 days	5/21/18 9:00 AM	5/21/18 9:00 AM	12
21	PAC4 - Redacción de la memoria	10 days	5/22/18 9:00 AM	6/5/18 9:00 AM	20
22	Entrega de Memoria del proyecto	0 days	6/5/18 9:00 AM	6/5/18 9:00 AM	21
23	PAC5a - Elaboración de la presentación	5 days	6/6/18 9:00 AM	6/13/18 9:00 AM	22
24	Entrega de la Presentacion del TFM	0 days	6/13/18 9:00 AM	6/13/18 9:00 AM	23
25	PAC5b - Defensa publica	7 days	6/14/18 9:00 AM	6/25/18 9:00 AM	24

Il·lustració 1 - Planificación de tareas



Il·lustració 2 - Planificación de tareas Gant

V. Breve sumario de productos obtenidos.

Durante la realización de este trabajo hemos obtenido los siguientes productos:

- Lista de resultados de análisis de FreeSurfer
- Lista de fuentes de datos de MRI para experimentación y análisis
- Script Python para el formateado de imágenes.
- Script Python para automatizar la generación de modelos Keras, el entrenamiento y su verificación.
- Script Python para automatizar la aplicación de modelos entrenados de Keras en formato h5 sobre un grupo de imágenes.
- Modelo Keras de 4 capas para cálculo del volumen de la materia gris cortical del hemisferio izquierdo para imágenes con formato (112,112,88) entrenado con 20 imágenes. Formato h5.
- Modelo Keras de 6 capas para cálculo del volumen de la materia gris cortical del hemisferio izquierdo para imágenes con formato (112,112,88) entrenado con 20 imágenes. Formato h5.
- Modelo Keras de 8 capas para cálculo del volumen de la materia gris cortical del hemisferio izquierdo para imágenes con formato (112,112,88) entrenado con 20 imágenes. Formato h5.
- Modelo Keras de 8 capas para cálculo del volumen de la materia gris cortical del hemisferio izquierdo para imágenes con formato (256,256,128) entrenado con 50 imágenes. Formato h5.

VI. Breve descripción de los otros capítulos de la memoria.

La estructura de este TFG se divide en las siguientes secciones:

- Descripción de los datos: En este capítulo analizaremos el proceso de definición de nuestros resultados objetivos. Como obtendremos los datos para realizar nuestros experimentos y en que formato deberán estar.

- Selección de métodos y diseño de experimentos: En este capítulo describiremos el proceso de selección y preparación del entorno de “Machine Learning”. Como se han generado los modelos y como ha sido el proceso de entrenamiento.

- Resultados: En este capítulo expondremos y analizaremos los resultados obtenidos en nuestros experimentos.

- Conclusiones: En este capítulo expondremos las conclusiones de nuestro trabajo y cuáles son las vías y posibilidades de evolución y expansión del sistema que hemos construido.

- Glosario: En este capítulo encontraremos una referencia de los términos técnicos que se han utilizado en este documento y una definición de estos.

- Bibliografía: En este capítulo podremos encontrar las referencias a todos los recursos y fuentes de información que se han utilizado para documentar y realizar este trabajo.

- Anexos: En este capítulo encontraremos los scripts y otros recursos referenciados a lo largo de este documento.

2. Descripción de los datos.

I. Análisis de capacidades de FreeSurfer.

Descripción del producto

FreeSurfer es un paquete de software de imágenes cerebrales desarrollado por el Centro de Imágenes Biomédicas Athinoula A. Martinos en el Hospital General de Massachusetts. Se utiliza principalmente para el análisis de imágenes de resonancia magnética (MRI). Es una herramienta muy importante en el mapeo cerebral funcional y facilita la visualización de las regiones funcionales de la corteza cerebral.

FreeSurfer proporciona una serie de herramientas que permiten el procesamiento completo de los datos proporcionados por una resonancia magnética. Estas herramientas incluyen:

- Extracción del cráneo, corrección del campo de polarización B1 y segmentación de la materia gris-blanca.
- Reconstrucción de modelos de superficie cortical (superficie límite gris-blanca y superficie pial).
- Etiquetado de regiones en la superficie cortical, así como estructuras cerebrales subcorticales.
- Registro no lineal de la superficie cortical de un individuo con un atlas estereotáxico.
- Análisis estadístico de las diferencias de morfometría grupal.

Durante el proceso de la superficie cortical, las herramientas construyen modelos del límite entre la materia blanca y la materia gris cortical, así como la superficie pial. Una vez que se conocen estas superficies, se genera una serie de medidas anatómicas, que incluyen: grosor cortical, área superficial, curvatura y superficie normal en cada punto relevante de la corteza.

Tras el procesado inicial de los datos originales de la resonancia, los resultados obtenidos pueden explotarse a través de unas herramientas visuales la más importante se denomina FreeView. Las superficies pueden inflarse y / o aplanarse para una mejor visualización. Las superficies también se pueden usar para restringir las soluciones a los problemas ópticos inversos, EEG y MEG. Además, se ha definido un atlas basado en la superficie cortical basado en patrones de plegado promedio mapeados a una esfera.

Las superficies de los individuos se pueden alinear con este atlas con un algoritmo de registro no lineal de alta dimensión. El registro se basa en la alineación de los patrones de plegado cortical y alinea de forma directa la anatomía en lugar de las intensidades de la imagen.

El atlas esférico forma naturalmente un sistema de coordenadas en el que se puede lograr la correspondencia punto a punto entre sujetos. Este sistema de coordenadas se puede usar para crear mapas de grupo (similar a cómo se usa el espacio MNI para mediciones volumétricas). La mayor parte de la canalización de FreeSurfer está automatizada, lo que la hace ideal para usar en grandes conjuntos de datos.

Proceso de evaluación

Para evaluar las capacidades del producto, hemos considerado oportuno realizar una instalación en nuestro entorno de pruebas. El producto está disponible para la descarga en esta url:

https://surfer.nmr.mgh.harvard.edu/fswiki/DownloadAndInstall.

Los requerimientos de la solución son los siguientes:

- Sistema Operativo: Linux, macOS. No disponemos de ninguna de las dos opciones así que hemos optado por un entorno virtualizado. La plataforma de virtualización elegida es VMWare 12 y el sistema operativo OpenSuse Leap 42.3.
- Velocidad de Procesador: Mínimo 2GHz. Inicialmente hemos asignado un procesador a nuestra máquina virtual. Nuestro sistema dispone de una CPU Intel i7. El primer procesamiento de un MRI ha tardado aproximadamente 12 horas.
- RAM: 8GB recomendados. No disponemos de tanta memoria disponible así que inicialmente asignamos 4GB.
- Tarjeta Gráfica: Tarjeta gráfica 3D con memoria propia y drivers OpenGL acelerados. No disponemos de ellos.
- Tamaño de instalación: 10GB. Este tamaño es realmente el del fichero tar / zip con el que se distribuye la aplicación. La máquina necesita casi el triple para poder descomprimir y alojar el software descomprimido.
- Tamaño de un MRI procesado: 300MB. El tamaño requerido para almacenar una MRI procesada es bastante grande.

La instalación consiste simplemente en descomprimir el fichero descargado en la ruta deseada.

```
linux-h5y4:~ # tar -C /UOC -xzvf freesurfer-Linux-centos6_x86_64-stable-pub-v6.0.0.tar.gz
```

El software requiere de una licencia para poderse utilizar. Esta licencia se puede solicitar directamente rellenado el formulario en esta url:

https://surfer.nmr.mgh.harvard.edu/registration.html.

Pasadas unas horas, la licencia es enviada al correo especificado en la solicitud junto a las instrucciones de instalación. La licencia es un fichero de texto llamado license.txt que tiene que copiarse en la carpeta $FREESURFER_HOME.

Para utilizar la aplicación necesitamos ejecutar unos scripts cada vez que abramos el terminal de Linux:

```
linux-h5y4:~ # export FREESURFER_HOME=/UOC/freesurfer
linux-h5y4:~ # export SUBJECTS_DIR=/opt
linux-h5y4:~ # source $FREESURFER_HOME/SetUpFreeSurfer.sh
-------- freesurfer-Linux-centos6_x86_64-stable-pub-v6.0.0-2beb96c --------
Setting up environment for FreeSurfer/FS-FAST (and FSL)
FREESURFER_HOME   /UOC/freesurfer
FSFAST_HOME       /UOC/freesurfer/fsfast
FSF_OUTPUT_FORMAT nii.gz
SUBJECTS_DIR      /opt
MNI_DIR           /UOC/freesurfer/mni
```

Una vez el Sistema está preparado podemos procesar la primera resonancia.

El software proporciona dos resonancias de prueba que pueden encontrarse en el directorio $FREESURFER_HOME/subjects/. Esto ejemplos vienen en formato mgz.

Un archivo ".mgz" (o .mgh.gz) es un archivo ".mgh" que se ha comprimido con ZLib. El formato de archivo ".mgh" se utiliza para almacenar datos estructurales de alta resolución y otros datos que se superpondrán en el volumen estructural de alta resolución.

Para procesar las imágenes de ejemplo primero han de convertirse a formato Nifti, Nifti es un formato estándar en el ámbito de las herramientas informáticas relacionadas con la neuroimagen. Se pueden consultar más detalles sobre este formato en la sección "El formato Nifti" de este documento.

Ejecutamos el siguiente comando:

```
linux-h5y4:~ #mri_convert sample-002.mgz sample-002.nii.gz
mri_convert.bin sample-002.mgz sample-002.nii.gz
$Id: mri_convert.c,v 1.226 2016/02/26 16:15:24 mreuter Exp $
reading from sample-002.mgz...
TR=7.25, TE=3.22, TI=600.00, flip angle=7.00
i_ras = (-0, -1, -0)
j_ras = (-0, 0, -1)
k_ras = (-1, 0, 0)
writing to sample-002.nii.gz...
```

Este proceso es bastante rápido, cuestión de un par de segundos. A continuación, podemos proceder al análisis de la imagen. Para ello utilizamos el comando:

```
linux-h5y4:~ #recon-all -i sample-002.nii.gz -s paciente2 -all
Subject Stamp: freesurfer-Linux-centos6_x86_64-stable-pub-v6.0.0-2beb96c
Current Stamp: freesurfer-Linux-centos6_x86_64-stable-pub-v6.0.0-2beb96c
INFO: SUBJECTS_DIR is /opt
Actual FREESURFER_HOME /UOC/freesurfer
Linux linux-h5y4 4.4.120-45-default #1 SMP Wed Mar 14 20:51:49 UTC 2018 (623211f) x86_64
x86_64 x86_64 GNU/Linux

....
```

El resultado de esta ejecución es una serie de ficheros que pueden ser explotados por herramientas que el propio FreeSurfer provee. Estas herramientas son principalmente un visor que permite visualizar las resonancias y tomar medidas, explorar las capas o reconocer automáticamente zonas del cerebro y un analizador estadístico que permite comparar los resultados de cada resonancia con unos datos estadísticos también proporcionados por la herramienta.

II. Selección de medidas objetivo.

FreeSurfer ofrece como resultado de su proceso de análisis de las resonancias las siguientes medidas principales:

- Medidas de volúmenes (expresados en mm^3)
    - Volumen de segmentación cerebral
    - Volumen de segmentación cerebral sin ventrículos
    - Volumen de segmentación cerebral sin ventrículos de Surf
    - Volumen total de materia blanca cerebral
    - Volumen total de materia gris cortical
    - Volumen intracraneal total estimado
    - Volumen de materia blanca cerebral del hemisferio izquierdo
    - Volumen de materia gris cortical del hemisferio izquierdo
    - Volumen de la máscara
    - Volumen de materia blanca cerebral del hemisferio derecho
    - Volumen de materia gris cortical del hemisferio derecho
    - Volumen de materia gris subcortical
    - Volumen supratentorial
    - Volumen supratentorial de Voxel
    - Volumen total de materia gris
    - Volumen de ventrículos y plexo coroideo

- Medidas de superficies (expresadas en mm^2)
    - Área total de la superficie Pial

    - Área total de la superficie Blanca

- Valores medios de grosor (expresados en mm)

- Recuento de vértices

- Diversas medidas de grados de curvatura

A parte de estas medidas principales, se ofrecen también otro gran número de medidas más detalladas pertenecientes a zonas más específicas que no consideraremos en este momento.

De las medidas principales, hay algunas que se calculan como sumas de otras así que para optimizar las medidas a calcular plantearemos los siguientes objetivos iniciales:

1. Volumen de materia gris cortical del hemisferio izquierdo
2. Volumen de materia gris cortical del hemisferio derecho
3. Volumen de materia blanca cerebral del hemisferio izquierdo
4. Volumen de materia blanca cerebral del hemisferio derecho
5. Volumen de ventrículos y plexo coroideo

III. Análisis de archivos públicos de MRI.

Existen múltiples fuentes de datos que proporcionan acceso a resonancias magnéticas reales del cerebro, efectuada a multitud de sujetos anónimos.

Estas fuentes de datos están normalmente relacionadas con determinados proyectos o estudios concretos centrados en enfermedades como el autismo o el Alzheimer.

A continuación, listamos una serie de fuentes de datos disponibles en internet con algunas de sus características y el tipo y cantidad de imagen que proporcionan.

Nombre: **Connectome Coordination Facility (CCF)**

Descripción: Connectome Coordination Facility (CCF) alberga y distribuye datos públicos de investigación para una serie de estudios que se centran en las conexiones dentro del cerebro humano. Estos se conocen como proyectos de conexión humana.

El CCF actualmente admite 19 estudios de conectomas humanos.

URL: https://www.humanconnectome.org

Necesita registro: Si
Formato Nifti: Si
Requerimientos especiales de software: Aspera connect

Estudios disponibles:
HCP Young Adult > 1200 muestras
Características adicionales: Dispone de resultados de procesados por FreeSurfer.

Nombre: **Alzheimer's Disease Neuroimaging Initiative (ADNI)**
Descripción: ADNI es un espacio de colaboración para los investigadores que trabajan para definir la progresión de la enfermedad de Alzheimer. Los investigadores de ADNI recopilan, validan y utilizan datos como imágenes de MRI y PET, genética, pruebas cognitivas, LCR y biomarcadores sanguíneos como predictores de la enfermedad.

Incluye los datos de los participantes del estudio ADNI de América del Norte, incluidos pacientes con enfermedad de Alzheimer, sujetos con deterioro cognitivo leve y controles de edad avanzada.

URL: http://adni.loni.usc.edu/

Necesita registro: Si
Formato Nifti: Si
Estudios disponibles:
ABIDE > 1000 muestras
PAD 3 muestras

Nombre: **Open Access Series of Imaging Studies (OASIS)**

Descripción: OASIS es un proyecto dirigido a hacer que los conjuntos de datos de neuroimagen del cerebro estén disponibles gratuitamente para la comunidad científica. Al compilar y distribuir libremente conjuntos de datos de neuroimágenes, esperan facilitar descubrimientos futuros en neurociencias básicas y clínicas.

URL: http://www.oasis-brains.org/#data

Necesita registro: No
Formato Nifti: Si
Estudios disponibles:
OASIS-2: 150 muestras
OASIS-1: 416 muestras

Nombre: **OpenfMRI**

Descripción: La base de datos OpenfMRI es un repositorio de datos de imágenes del cerebro humano recogidos mediante técnicas de MRI y EEG. Ha estado aceptando datos desde 2010; inicialmente solo aceptaba conjuntos de datos que incluían una IRMF basada en tareas (una técnica que mide cómo la realización de una tarea particular influye en la activación cerebral), pero posteriormente se ha abierto a todas las formas de datos de neuroimágenes que incluyen datos de IRM.

URL: https://openfmri.org

Necesita registro: No
Formato Nifti: Si
Estudios disponibles:
96 estudios diferentes
3372 muestras distribuidas entre los diferentes estudios.

Nombre: **BRAINS Imagebank**

Descripción: BRAINS Imagebank está diseñado para proporcionar datos detallados de imágenes estructurales del cerebro de individuos sanos a lo largo del curso de la vida humana. El banco de imágenes es una base de datos de búsqueda de conjuntos de datos integrados ya recopilados como parte de estudios de investigación que incluyen sujetos sanos.

URL: http://www.brainsimagebank.ac.uk

Necesita registro: No
Formato Nifti: Si

Tras estudiar todas las opciones presentadas anteriormente, hemos decidido utilizar dos fuentes.

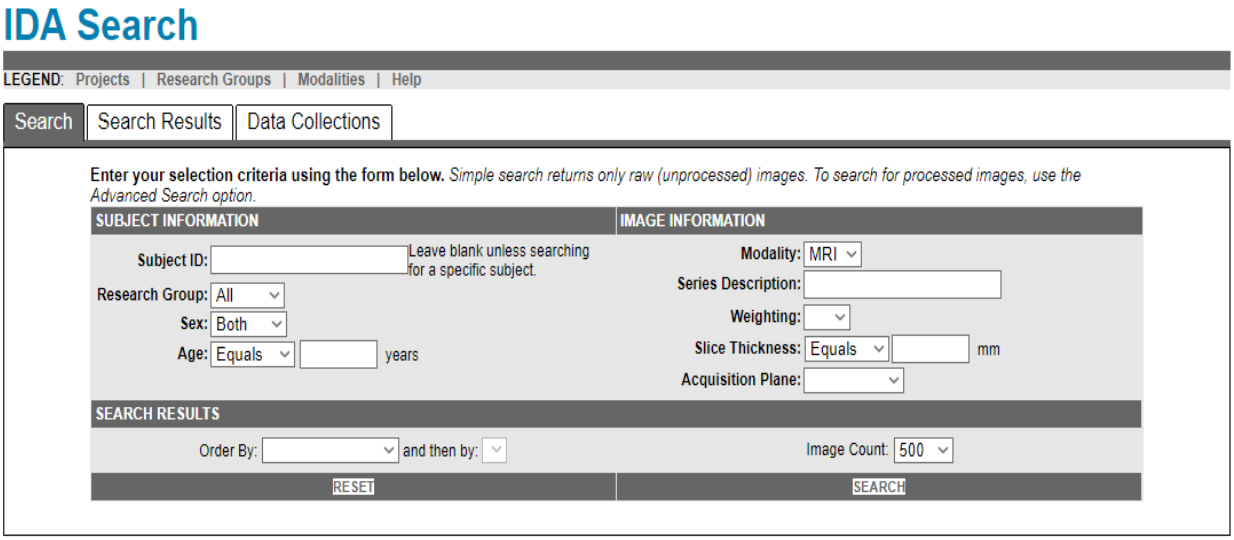
La primera es la base de datos del proyecto ABIDE, alojado dentro de la iniciativa ADNI.

Esta base de datos además de ofrecer una gran cantidad de muestras y el formato adecuado para nuestros intereses dispone de una interface de búsqueda y descarga de ficheros muy sencilla de utilizar. Las muestras combinan sujetos afectados por el autismo con otros usuarios sanos dentro del grupo de control.

Esta fuente es ideal para la descarga de muestras individuales que queremos utilizar para evaluar nuestros modelos.

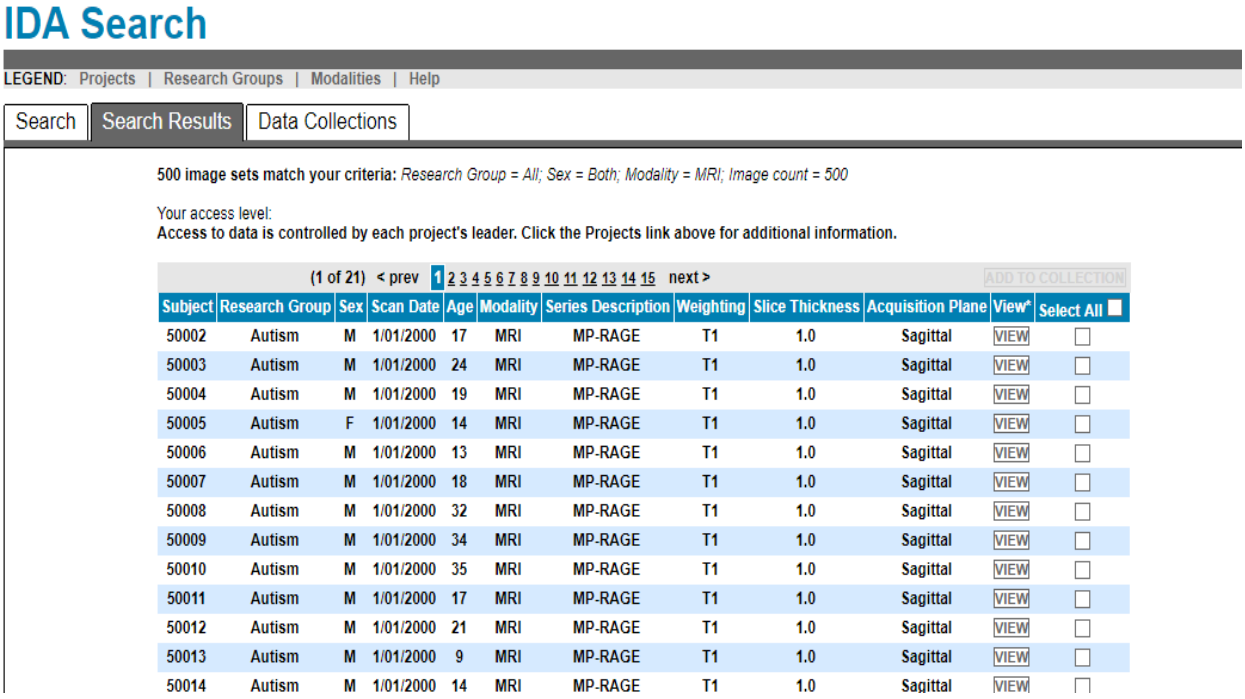
El funcionamiento de la fuente de datos es el siguiente:

La herramienta presenta inicialmente una pantalla de búsqueda donde podemos introducir ciertos criterios de búsqueda para seleccionar los sujetos de estudio deseados.



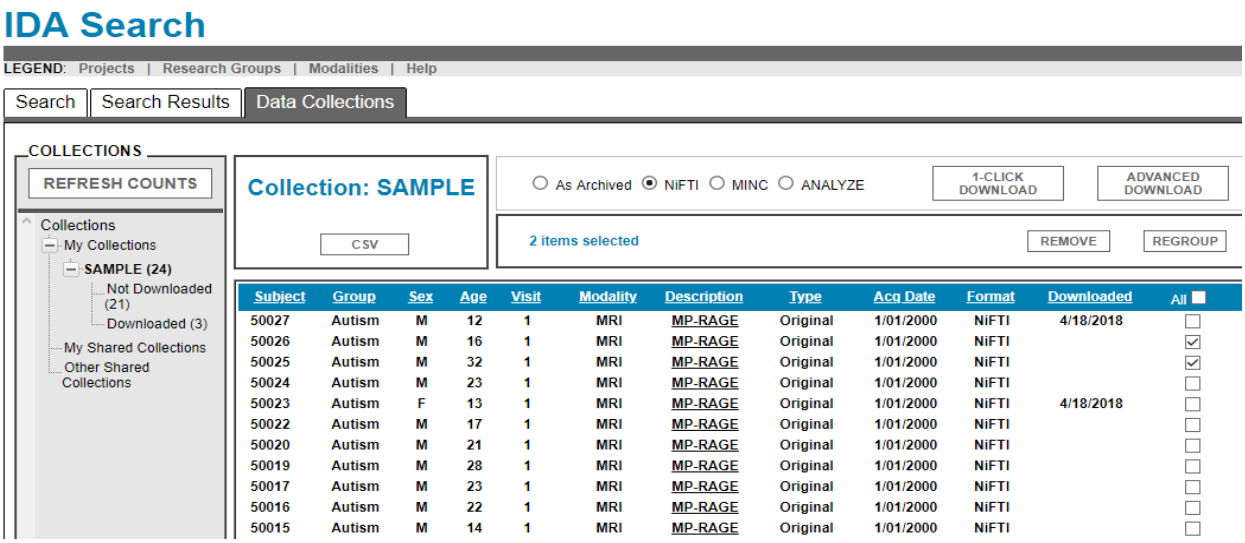
Il·lustració 3 – IDA. Búsqueda de sujetos

El resultado de la búsqueda es una lista de sujetos que podemos seleccionar y añadir a una colección para una posterior explotación.



Il·lustració 4 – IDA. Resultados de la búsqueda

La exploración de la colección nos permite seleccionar algunos sujetos, definir el formato de descarga que deseamos y realizar la descarga de los sujetos seleccionados.

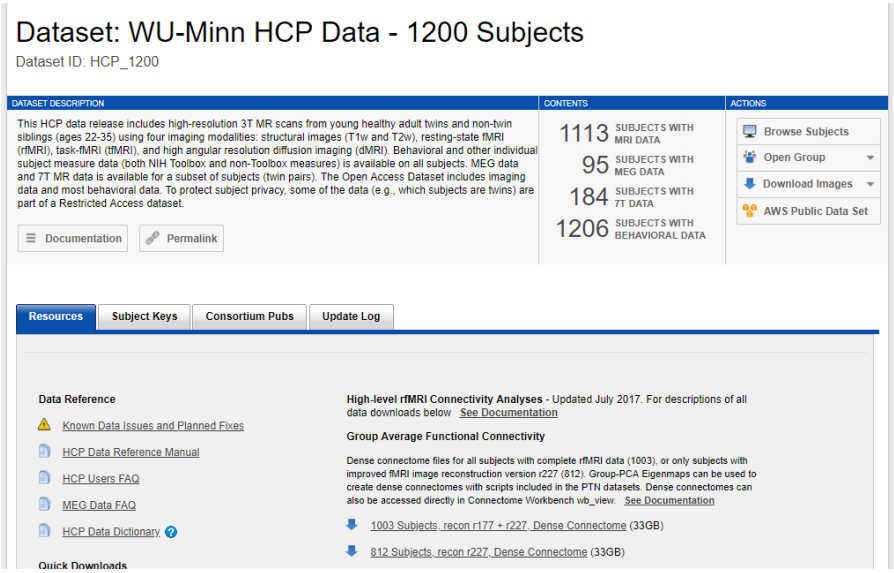


Il·lustració 5 – IDA. Colección para descarga

La otra fuente que hemos utilizado para nuestro trabajo es la del proyecto HCP Young Adult de Connectome Coordination Facility (CCF). Esta fuente es ideal para el proceso de entrenamiento ya que permite la descarga de los resultados del análisis de FreeSurfer lo que ahorra gran cantidad de tiempo.

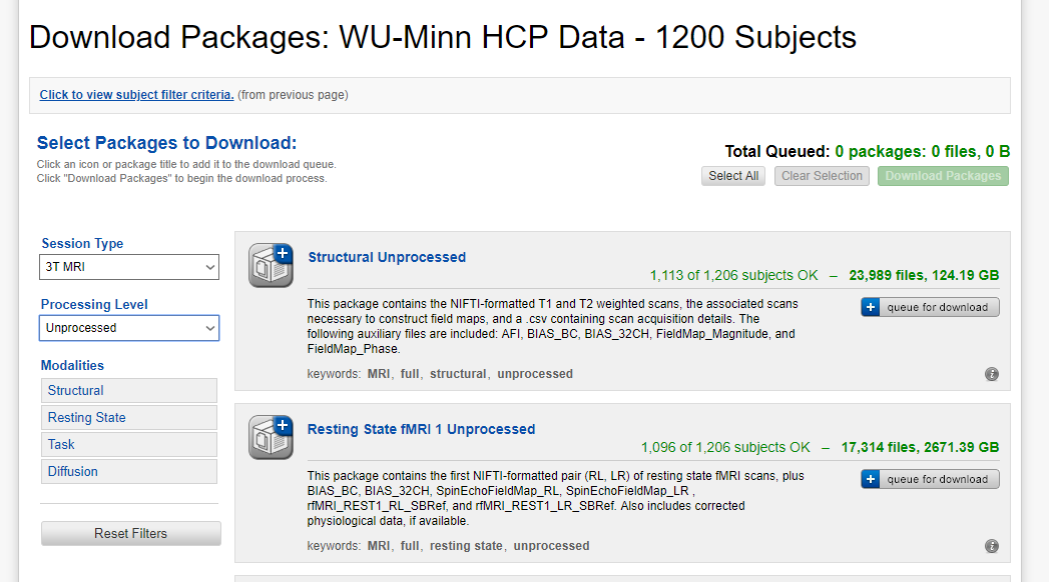
Estos resultados descargados en formato CSV pueden ser enlazados a las imágenes Nifti mediante el número de sujeto.

EL acceso a los datos es un poco más complejo, ya que estos están agrupados por grupos de estudio y están disponibles en multitud de formatos, con pre procesados y con otras transformaciones.



Il·lustració 6 - CCF. Selección de colecciones

Tras la selección de un grupo, la siguiente pantalla nos permite seleccionar el tipo de sesión de MRI y el nivel de procesado. Como respuesta a este filtrado obtenemos unos resultados en la parte derecha que nos permitirán descargar tanto las imágenes como los análisis en FreeSurfer.



Il·lustració 7 - CCF. Descarga de datos

IV. Determinación de formatos de MRI a utilizar.

Tras analizar los formatos compatibles con FreeSurfer y la disponibilidad de este formato en los diferentes repositorios públicos de datos, tomamos la decisión de utilizar el formato de fichero Nifti como la base para nuestro proyecto.

El formato Nifti

Nifti es el acrónimo de “Neuroimaging Informatics Technology Initiative”. Su objetivo principal es el de proporcionar un estándar común para acelerar el desarrollo y mejorar la utilidad de las herramientas informáticas relacionadas con la neuroimagen. El Instituto Nacional de Salud Mental y el Instituto Nacional de Trastornos Neurológicos y Accidentes Cerebrovasculares son patrocinadores conjuntos de esta iniciativa.

Como hemos comentado al comienzo de este capítulo, FreeSurfer es capaz de analizar la MRI en formato Nifti sin que se requiera ningún tipo de conversión ni pre-proceso inicial. El formato Nifti es también un formato común de descarga para los repositorios de imágenes MRI existentes en internet que hemos analizado.

Además de diferentes formatos multi-fichero, Nifti dispone de una versión de un único fichero con extensión ".nii". Este fichero único puede encontrarse también en formato comprimido con la extensión ".nii.gz". Este formato comprimido es el que hemos utilizado inicialmente en este proyecto.

El formato Nifti se creó hace alrededor de una década como un reemplazo al entonces extenso pero problemático formato de archivo de analyze 7.5. Surgió como un intento de solucionar la falta de información adecuada sobre la orientación en el espacio que hacía que los datos almacenados no pudieran interpretarse de manera inequívoca.

El nuevo formato se definió en dos reuniones del llamado Data Format Working Group (dfwg) en los Institutos Nacionales de Salud (nih), uno el 31 de marzo y otro el 02 de septiembre de 2003. Representantes de algunos de los programas de neuroimagen más populares acordaron un formato que incluiría nueva información y el uso de este nuevo formato, ya fuera forma nativa o como opción para importar y exportar.

Otros de los problemas del formato analyze 7.5 era la utilización de múltiples ficheros para la representación de una única imagen. Un encabezado con extensión .hdr, para almacenar metainformación, y los datos reales, con extensión .img. El formato Nifti eliminó la necesidad de trabajar con múltiples ficheros y sintetizó toda la información en un solo contenedor con extensión ".nii".

Sin embargo, para mantener la compatibilidad con el formato de analyze, el tamaño del encabezado se mantuvo en 348 bytes como en el formato antiguo. Algunos campos se reutilizaron, otros se conservaron, pero se ignoraron, y algunos se sobrescribieron por completo.

Por tanto en el formato nifty se mantiene una separación entre la cabecera de metadatos y la información de imagen. Esta información de imagen es evidentemente multidimansional.

En el formato nifti, las primeras tres dimensiones se reservan para definir las tres dimensiones espaciales - x, y y z -, mientras que la cuarta dimensión se reserva para definir los puntos de tiempo - t. Las dimensiones restantes, del quinto al séptimo, son para otros usos. La quinta dimensión, sin embargo, puede tener algunos usos predefinidos, como almacenar parámetros de distribución específicos de voxel o contener datos basados en vectores.

Para obtener una completa descripción del formato Nifti puede consultarse la siguiente Web:

https://nifti.nimh.nih.gov

3. Selección de métodos y diseño de experimentos

I. Proceso de selección de biblioteca de “Machine Learning”.

Hoy en día, tenemos a nuestra a nuestra disposición una gran cantidad de biblioteca de “Machine Learning”. Estas plataformas nos permiten desarrollar herramientas que den solución a los problemas con un mayor nivel de abstracción simplificando tareas de programación y permitiendo al usuario centrarse más en tareas de explotación.

Cada herramienta está construida de una manera diferente y con el objetivo de dar solución a diferentes desafíos.

Estos son algunos de las propuestas actuales más importantes.

Tensorflow: Es una de las mejores plataformas de “Deep Learning”. Ha sido adoptado por varios gigantes como Airbus, Twitter, IBM y otros, principalmente debido a la gran flexibilidad de su arquitectura.

La aplicación más conocida de TensorFlow es el Traductor de Google que implementa capacidades tales como el procesamiento del lenguaje natural, clasificación / resumen de texto, reconocimiento de voz / imagen / escritura, previsión y etiquetado.

TensorFlow está disponible tanto en computadoras de escritorio como en dispositivos móviles y es compatible con lenguajes como Python y C ++.

Es muy recomendado para principiantes, está soportado por Google y dispone de una excelente documentación y comunidad que la respalda.

Caffe: Librería de “Deep Learning” compatible con lenguajes como C, C ++, Python y MATLAB. Es muy conocido por su velocidad, su portabilidad y su aplicabilidad en el modelado de redes neuronales de convolución (CNN).

El mayor beneficio de usar Caffe es la capacidad de acceder a las redes disponibles desde su repositorio Caffe Model Zoo. Estas redes están pre-entrenadas y pueden usarse de inmediato.

Es la plataforma más adecuada para resolver problemas con imágenes y redes de convolución.

Microsoft Cognitive Toolkit/CNTK: Conocida popularmente por su sencillo proceso de entrenamiento y la combinación de diferentes modelos entre varios servidores, CNTK es un framework de aprendizaje de código abierto muy

eficiente en el procesado de redes convolucionales y el tratamiento de imágenes.

Cuando se trata de crear nuevos tipos de capas complejas, es más sencillo que Caffe, ya que los usuarios no necesitan implementarlos en un lenguaje de bajo nivel debido a la fina granularidad de los componentes básicos disponibles.

Actualmente, debido a la falta de soporte en la arquitectura ARM, sus capacidades en los dispositivos móviles son bastante limitadas.

Torch/PyTorch: Es un framework de “Deep Learning” basado en Lua que se usa ampliamente entre los gigantes de la industria como Facebook, Twitter y Google. Utiliza CUDA junto con bibliotecas C / C ++ para el procesamiento y se construyó principalmente para poder escalar la construcción de modelos y proporcionar una flexibilidad general.

Recientemente, esta plataforma ha ganado en popularidad entre la comunidad de Deep Learning y se le considera actualmente una competidora de TensorFlow.

A diferencia de Torch, PyTorch se ejecuta en Python, lo que significa que cualquier persona con un conocimiento básico de Python puede comenzar a construir sus propios modelos de aprendizaje profundo. Dado el estilo arquitectónico, todo el proceso de modelado es mucho más simple y transparente en comparación con Torch.

MXNet: Soportado por Python, R, C++ y Julia, este framework fue diseñado específicamente con el propósito de lograr gran eficiencia, productividad y flexibilidad. Su gran virtud es la de permitir al usuario utilizar gran variedad de lenguajes de programación lo que permite al usuario utilizar sus conocimientos actuales sin necesidad de aprender un nuevo lenguaje.

Escrito en C++ y CUDA, es fácilmente escalable y puede trabajar con una gran cantidad de GPUs. Destaca por sus capacidades en imágenes, escritura a mano / reconocimiento de voz, previsión y PNL. Es muy popular a nivel empresarial siendo Amazon una de las empresas más populares que lo utilizan.

Chainer: Muy potente, dinámico e intuitivo, Chainer es un marco de Deep Learning basado en Python para redes neuronales diseñado por la estrategia de ejecución. En comparación con otros marcos que utilizan la misma estrategia, puede modificar las redes durante el tiempo de ejecución, lo que le permite ejecutar sentencias de flujo de control arbitrarias.

Chainer es compatible con CUDA y multi-GPU. Se utiliza principalmente en el análisis de sentimiento, traducción automática, reconocimiento de voz, etc. utilizando RNN y CNN

Keras: Conocida por su simplicidad, la biblioteca de red neuronal Keras (con una interfaz de soporte de Python) admite redes convolucionales y recurrentes que son capaces de ejecutarse en TensorFlow o Theano.

Dada la complejidad de la interfaz de TensorFlow y su bajo nivel, puede ser realmente complicada para los nuevos usuarios. Keras se creó para proporcionar una interfaz simplista con el objetivo de generar prototipos rápidamente mediante la construcción de redes neuronales que pueden funcionar con TensorFlow.

Keras es ligero, fácil de usar y muy sencillo cuando se trata de construir un modelo de aprendizaje profundo mediante la secuencia de múltiples capas. Estas son las razones por las que Keras es parte de la API central de TensorFlow.

Keras está principalmente indicado para la clasificación, generación y resumen de texto, etiquetado y traducción, junto con reconocimiento de voz y más

Deeplearning4j: Está desarrollado tanto en Java como en Scala y es también compatible con otros lenguajes JVM. Ampliamente adoptada como una plataforma de “Deep learning” distribuida. Enfocada en la industria, la mayor ventaja de este marco de aprendizaje profundo es que puede reunir todo el ecosistema de Java para ejecutar el “Deep learning”.

Administrable sobre Hadoop y Spark para orquestar múltiples hilos de host. DL4J usa MapReduce para capacitar a la red mientras depende de otras bibliotecas para ejecutar grandes operaciones matriciales.

Deeplearning4j soporta RBM, DBN, redes neuronales de convolución (CNN), redes neuronales recurrentes (RNN), redes de tensores neuronales recursivas (RNTN) y memoria larga a corto plazo (LTSM).

Dado que este marco de aprendizaje profundo se implementa en Java, es mucho más eficiente en comparación con Python. Cuando se trata de tareas de reconocimiento de imágenes usando múltiples GPU, es tan rápido como Caffe.

Este framework muestra su máximo potencial en el reconocimiento de imágenes, la detección de fraudes, la minería de textos, el etiquetado de partes de la voz y el procesamiento del lenguaje natural.

Entre todas estas opciones propuestas, nos hemos decantado por Keras sobre TensorFlow como herramienta para realizar nuestro proyecto. Las principales razones son las siguientes:

- Ha sido recomendado por nuestro tutor.
- Es ideal para usuarios sin grandes conocimientos de Machine Learning como es nuestro caso.

- Es ligero y no requiere excesivos recursos de hardware. Aunque luego veremos cómo esto no es completamente cierto.
- Permite obtener resultados rápidos. Esto es perfecto para nuestros objetivos debido al poco tiempo del que disponemos.
- Se centra más en el modelado y definición que en la programación. Este es un proyecto de análisis, no de programación.
- Es compatible con Python. Esto nos interesa debido a todas las herramientas de procesado de imágenes Nifti disponibles en Python.
- Su base es TensorFlow que sin duda es uno de los grandes.

II. La Librería Keras

Instalación

Supongamos que partimos de un sistema con el entorno de Python ya instalado. El primer paso es la instalación de la librería base TensorFlow. Para instalar TensorFlow primero actualizamos el pip:

```
C:\> python -m pip install --upgrade pip
```

A continuación, instalamos TensorFlow. Tensorflow puede instalarse con soporte para GPU o sin soporte para GPU. Una instalación con GPU en un sistema que no dispone de GPU no funciona. Una instalación en un sistema con GPU requiere instalar las librerías de CUDA.

```
C:\> pip3 install --upgrade tensorflow        //Instalación sin GPU
o
C:\> pip3 install --upgrade tensorflow-gpu        //Instalación sin GPU
```

Tras instalar y verificar TensorFlow, podemos pasar a instalar la librería h5py, esta librería es necesaria para poder guardar y recuperar los ficheros desde disco.

```
C:\> pip install h5py
```

A continuación, instalamos Keras. Como podemos ver, para keras no existe una versión GPU o no GPU, esto solo afecta a TensorFlow de manera que para pasar activar o desactivar la GPU no hace falta tocar la instalación de Keras.

```
C:\> pip install keras
```

Con esto se termina la instalación de Keras y el sistema puede empezar a funcionar. Para nuestro proyecto, hemos utilizado una librería adicional para la gestión de las imágenes Nifti llamada nibabel. http://nipy.org/nibabel/installation.html

Para instalarla basta con ejecutar.

```
c:\> pip install nibabel
```

Adicionalmente para nuestra documentación hemos utilizado el pintado de modelos. Para ello es necesario instalar Graphviz (https://graphviz.gitlab.io/download/) y el módulo de Python pydot.

```
c:\> pip install pydot
```

El listado final de los módulos y sus versiones instalados en nuestro sistema para este proyecto es el siguiente:

absl-py==0.2.0
astor==0.6.2
bleach==1.5.0
gast==0.2.0
grpcio==1.11.0
h5py==2.8.0rc1
html5lib==0.9999999
Keras==2.1.6
Markdown==2.6.11
nibabel==2.2.1
numpy==1.14.3
protobuf==3.5.2.post1
pydot==1.2.4
pyparsing==2.2.0
PyYAML==3.12
scipy==1.1.0
six==1.11.0
tensorboard==1.8.0
tensorflow==1.8.0
termcolor==1.1.0
Werkzeug==0.14.1

Selección del tipo de Red

Dada la naturaleza de nuestro proyecto y el tipo de información que tenemos que procesar vamos a utilizar una red neuronal convolucional para lograr nuestros objetivos.

Las redes neuronales convolucionales son muy similares a las redes neuronales ordinarias, como el perceptrón multicapa y se usan principalmente para el tratamiento de imágenes cuando estas son grandes complejas y como en nuestro caso multidimensionales.

En este tipo de redes, las neuronas tienen una seria de pesos, reciben unos datos de entrada con los que realiza unas operaciones matremáticas y sobre los que aplica una función de activación, tienen una función de perdida, etc.

Trabajan dividiendo y modelando la información en partes más pequeñas, y combinando esta información en las capas más profundas de la red.

Las primeras capas se encargan de detectar los bordes de las imagenes. Las siguientes capas buscan combinar patrones de detección de bordes para conseguir formas más simples, y aplicar patrones de posición de objetos, iluminación, etc.

Las últimas capas intentarán hacer coincidir la imagen con los patrones ya descubiertos, para conseguir una predicción final con la suma de todos ellos. De esta manera las redes neuronales convolucionales son capaces de manejar grandes cantidades de datos, simplificando el problema en partes mas pequeñas para conseguir predicciones más sencillas y precisas.

Estructura de una red neuronal en Keras

Generalmente las redes neuronales convolucionales en Keras estan compuesta por varias capas:

- Una o varias capas convolucionales: Estas capas son las que dan nombre de la red. Utilizan una operación llamada convolución. Esta operación recibe como entrada una imagen multidimensional, y aplica sobre ella un filtro que devuelve el mapa de características, reduciendo de esta manera el tamaño de sus parámetros.

- Una o varias capas de pooling: Estas capas se colocan generalmente después de una o varias capas convolucionales. Se encargadan de reducir el número e parámetros del problema reduciendo las dimensiones espaciales, manteniendo las características más comunes. La operación llevado a cabo en esta capa se conoce también como reducción de muestreo o submuestreo.

- Capa de reducción de dimensiones: Esta capa recibe una entrada de da parámetros multidimensional y lo convierte en una salida unidimensional con el propósito de poder aplicarla a una capa totalmente conectada.

- Una o varias capas totalmente conectada: Se aplican al final y pierden la información espacial. En estas capa cada pixel se considera como una neurona separada al igual que en una red neuronal regular. La última capa clasificadora tendrá tantas neuronas como el número de clases que se debe predecir. En nuestro caso, nuestro problema no es un problema de clasificación sino una regresion por lo que la salida final no será una clasificación sino un resultado continuo.

III. Definición del Modelo

Como hemos explicado en el punto anterior, Keras es una librería de “Deep learning” escrita en Python que permite expresar redes neuronales de manera modular. Considera un modelo como una secuencia o un solo grafo. La

estructura de datos principal en Keras es el modelo (*model*), que no es más que una secuencia de capas.

La definición del modelo consiste en definir primero cual es el formato de entrada de los datos, y encadenar una serie de capas que nos llevarán al resultado final.

Keras simplifica enormemente la interacción de estas capas ya que, tras la única definición del formato de datos de entrada, la adaptación de datos entre capas se hace de forma automática.

Para la definición del modelo, se crea un objeto de tipo keras.models.Sequential mediante la instrucción:

```
model = Sequential()
```

Los métodos básicos del modelo secuencial son:

- compile() : Prepara el modelo para una forma concreta de entrenamiento.
- fit() : Entrena el modelo para un número fijado de ciclos, con actualizaciones de gradiente cada ciertos ejemplos.
- evaluate() : Calcula la función de perdida dados unos datos de entrada, lote por lote.
- predict() : Genera predicciones para unos valores de entrada

A continuación, se define la secuencia de capas. Estas capas pueden ser de diferentes tipos y cada tipo tendrá una configuración diferente. Las capas se añaden al modelo mediante la función add.

Las principales capas disponibles son:

- Dense: Para capas regulares totalmente conectadas.
- Activation: Se aplica una función de activación a una salida. Las principales funciones que nos ofrece Keras son:
    - softmax: generalización de la función logística.
    - softplus
    - softsign
    - relu: función rectificadora
    - tanh: función tangente hiperbólica
    - sigmoid: función sigmoide
    - hard_sigmoid
    - linear: función lineal
- Dropout: Consiste en establecer al azar una fracción de unidades de entrada a 0 en cada actualización durante la fase de entrenamiento. Esto ayuda a evitar el sobreajuste.
- Reshape: Transforma la salida en una forma concreta.

- Capas convolucionales: Keras ofrece funciones para entradas de una, dos o tres dimensiones.
- Capas pooling: Al igual que las capas convolucionales, Keras ofrece funciones para entradas de una, dos o tres dimensiones.
- Capas recurrentes.

Como hemos explicado anteriormente, el modelo necesita saber cuál va a ser el formato de entra de los datos. Esto se define en la primera capa mediante el parámetro "input_shape". Este parámetro no aparecerá en el resto de las capas ya que como hemos comentado, es Keras quien se encarga de a partir de utilizar en formato de datos resultado de la capa anterior para establecer el formato de datos de entrada de la capa siguiente.

Un ejemplo de definición de la primera capa sería

```
model.add(Conv3D(32, (8,8,8), strides=4, activation='relu', padding='same', input_shape = (256, 256, 256,1)))
```

En este ejemplo, hemos añadido una capa del tipo Conv3D como primera capa, y hemos definido que el formato de entrada será de tridimensional de dimensiones 256 ,256 ,256.

Podemos observar cómo, aunque el formato de entrada es tridimensional, nuestra definición tiene cuatro parámetros. Este cuarto parámetro determina el número de canales. Este número de canales puede ser utilizado por ejemplo cuando disponemos de imágenes en color (RGB) y estos están descompuestos en tres canales, uno para rojo, otro para el azul y otro para el verde.

Una vez tenemos nuestro modelo construido, tenemos que configurar el proceso de aprendizaje. Esto se realiza mediante la instrucción compile.

Al compilar, debemos especificar algunas propiedades adicionales requeridas para entrenar la red, como la función de pérdida "loss", para evaluar un conjunto de pesos, el optimizador utilizado para buscar en diferentes pesos de la red y cualesquiera otras métricas opcionales que deseemos recopilar durante el entrenamiento. Por ejemplo:

```
model.compile(loss='mean_squared_error', optimizer='adam',  metrics=['accuracy'])
```

Tras compilar el modelo, podemos obtener una descripción de este mediante el comando summary. Un ejemplo de resultado sería el siguiente:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv3d_1 (Conv3D)            (None, 112, 112, 88, 64)  32832
```

```
_________________________________________________________________
max_pooling3d_1 (MaxPooling3 (None, 56, 56, 44, 64)    0
_________________________________________________________________
flatten_1 (Flatten)          (None, 8830976)           0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 8830977
=================================================================
Total params: 8,863,809
Trainable params: 8,863,809
Non-trainable params: 0
_________________________________________________________________
None
```

IV. Proceso de entrenamiento

Entrenamiento

Una vez tenemos definido y compilado nuestro modelo y ya está listo para ser entrenado. El siguiente paso consiste en aplicarle nuestro grupo de imágenes de entrenamiento. Esto se realiza mediante la instrucción fit.

A esta instrucción tenemos que pasarle varios parámetros, principalmente, los datos de entrenamiento y validación, el número de iteraciones que se realizaran sobre esos mismos datos, el número de instancias que se evaluarán antes de que se realice una actualización de pesos o la cantidad de información que se mostrara en pantalla durante el proceso de entrenamiento.

Es posible especificar la distribución de las imágenes de entrenamiento y las imágenes de validación. Esto puede realizarse especificando un grupo independiente de test o indicando un porcentaje de imágenes destinado a ese propósito. En nuestro caso hemos especificado un valor del 0.1, por lo que se asigna el 10% de las imágenes a validación.

Un ejemplo de ejecución del proceso de entrenamiento seria:

```
model.fit(X_train, Y_train,
        batch_size=1,
        epochs=epochs,
        verbose=1,
        validation_split=0.1)
```

Un ejemplo de resultado obtenido por pantalla sería:

```
18/18 [==============================] - 32558s 1809s/step - loss: 137017574286.2222 - acc: 0.0000e+00 - val_loss: 66809565184.0000 - val_acc: 0.0000e+00
Epoch 2/25
18/18 [==============================] - 4862s 270s/step - loss: 60619969422.2222 - acc: 0.0000e+00 - val_loss: 66567124992.0000 - val_acc: 0.0000e+00
```

```
Epoch 3/25
18/18 [==============================] - 4876s 271s/step - loss: 32338262926.2222 - acc: 0.0000e+00 - val_loss: 170974011392.0000 - val_acc: 0.0000e+00
Epoch 4/25
....
```

En este listado de resultados parciales podemos ver como obtenemos para cada iteración un valor de “loss” y otro “val_loss”. El valor “val_loss” representa el valor de la función de costo para los datos de validación y el valor “loss” es el valor de la función de costo para los datos de entrenamiento.

Como podemos ver los valores de “loss” y “val_loss” son muy grandes. Esto es debido a que estan expresados en milimetros cúbicos y hemos mantenido esta unidad para poder mantener el formato original de las fuentes de datos sin tener que modificar los datos ni en la entrada ni en el resultado final.

Los valores que estamos considerando se situan alrededor de los 250000 mm cúbicos. Este factor junto a la utilización de la función de error cuadrático como función de validación provocan estos valores de pérdida tan altos.

En los datos de validación, las neuronas que utilizan la técnica de “drop” no eliminan las neuronas aleatorias. La razón es que durante el entrenamiento usamos drop out para agregar algo de ruido para evitar el exceso de ajuste. Durante el cálculo de la validación, estamos en la fase de recuperación y no en la fase de entrenamiento. Usamos todas las capacidades de la red.

Los datos de entrenamiento y validación se han tomado de la fuente del proyecto The Human Connectome Project (HCP) de forma totalmente aleatoria, sin considerar sexo, edad o posibles patologias.

Script de definición y entrenamiento del modelo

A partir de lo descrito en el apartado anterior, hemos desarrollado nuestro propio script para automatizar el proceso de definición y entrenamiento lo máximo posible. Hay dos decisiones que hemos tomado y que han afectado directamente a la construcción del script:

1. El script no se encarga de adaptar las imágenes al formato requerido por el modelo. Se entiende que las imágenes utilizadas ya se encuentran debidamente formateadas y este formato es el que determina el formato del modelo.

2. Nuestra plataforma tiene como objetivo generar diferentes modelos que permitan obtener rápidamente diferentes medidas de volumen. Por tanto, nuestro script está diseñado para poder cambiar fácilmente la medida que deseamos obtener y generar un nuevo modelo con esa nueva medida.

La descripción del script generado es la siguiente:

1. Importación de librerías auxiliares

```
import os                    // Librería de gestión de archivos en disco
import nibabel as nib        // Librería de lectura de imágenes Nifti
import numpy as np           // Librería de procesado de arrays
import glob                  // Librería de lectura de ficheros en disco
import csv                   // Librería de lectura de archivos CSV.
```

2. Importación de librerías de Keras

```
from keras.models import Sequential          // Importación de modelo secuencial
from keras.layers import Conv3D,Activation   // Definición de las diferentes capas y utilidades
,Flatten,MaxPooling3D,Dense
```

3. Lectura del fichero de resultados disponibles en CSV. Cargamos en un diccionario todos los resultados disponibles por sujeto. Utilizamos el identificador de sujeto como clave del diccionario.

```
resultDictionary = {}
with open(<Path_To_CSV_File>) as csvfile:
  reader = csv.DictReader(csvfile)
  for row in reader:
    resultDictionary[row['Subject']] = row
```

4. Leemos todas las imágenes para realizar el entrenamiento. Estas se encuentran en un directorio concreto y tienen la extensión ".nii.gz".

```
trainingElements = glob.glob("<Path_To_Training Images>\\*.nii.gz")
```

5. Para cada imagen del directorio extraemos el identificador del sujeto del nombre del fichero. Cargamos la imagen Nifti y extraemos los datos de la imagen al array imageData. Obtenemos la fila de datos del CSV a partir del nombre del sujeto extraído y de la fila extraemos el valor deseado para el modelo y la añadimos al array resultData.

```
imageData = []
resultData = []
for trainingElement in trainingElements:
  subject = os.path.basename(trainingElement)[:6]
  n1_img = nib.load(trainingElement)
  imageData.append(n1_img.get_data())
  resultData.append(resultDictionary[subject]['<Medida_Seleccioanda>'])
```

6. Realizamos un reformateado de los datos de entrada convirtiendo nuestros arrays en arrays numpy y añadiendo una cuarta dimensión al array de los datos de las imágenes

```
X_train = np.array(imageData)
X_train = np.expand_dims(X_train, axis=4)

Y_train = np.array(resultData)
```

7. Definimos el modelo a utilizar. En los apartados siguientes describimos detalladamente la definición de cada uno de los modelos. El siguiente ejemplo corresponde al modelo 4.

```
model = Sequential()

model.add(Conv3D(32, (8,8,8), strides=4, activation='relu',
padding='same',input_shape=(n1_img.shape[0], n1_img.shape[1],
n1_img.shape[2],1)))
model.add(MaxPooling3D((2, 2, 2)))
model.add(Conv3D(64, (6,6,6), strides=2, activation='relu', padding='same'))
model.add(MaxPooling3D((2, 2, 2)))
model.add(Conv3D(128, (4,4,4), strides=1, activation='relu', padding='same'))
model.add(MaxPooling3D((2, 2, 2)))
model.add(Flatten())
model.add(Dense(1,kernel_initializer='normal'))
```

8. Definimos el número de iteraciones del entrenamiento, compilamos el modelo y comenzamos el entrenamiento

```
epochs = 25      // Número de iteraciones

model.compile(loss='mean_squared_error',     // Compilación del modelo
        optimizer='adam',
        metrics=['accuracy']) # reporting the accuracy

model.fit(X_train,                                  // Entrenamiento
      Y_train,
      batch_size=1,
      epochs=epochs,
      verbose=1,
      validation_split=0.1)
```

9. Una vez el modelo esta entrenado podemos realizar diferentes operaciones con él. En nuestro caso lo guardamos para poder utilizarlo posteriormente y lo probamos con una predicción.

```
model.save('<Nombre_Del_Modelo>.h5')
```

```
n1_img = nib.load("<Ruta_Imagen_DeEvaluación>.nii.gz")
predictionData = []
predictionData.append(n1_img.get_data())
X_predict = np.array(predictionData)
X_predict = np.expand_dims(X_predict, axis=4)

prediction = model.predict(X_predict)
print(prediction)
```

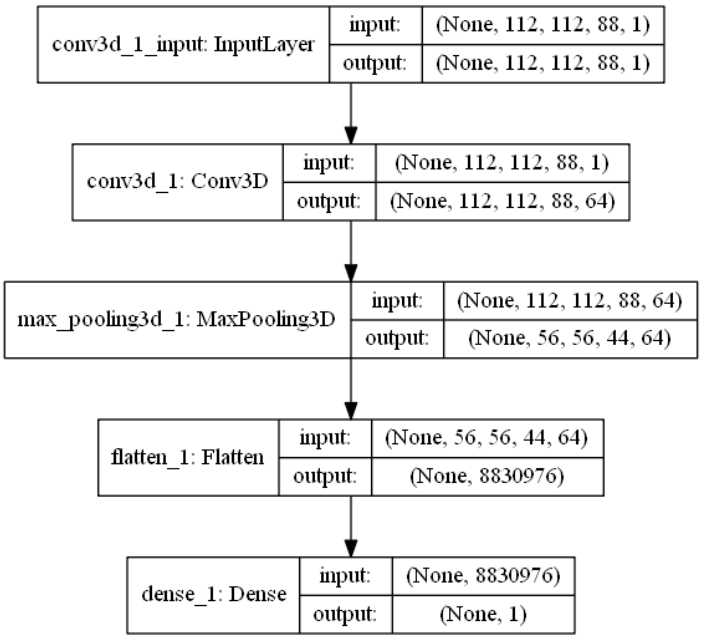
El script completo está disponible en el apartado de anexos.

Primer modelo

El primer modelo desarrollado es un modelo básico de cuatro capas. La primera capa es una capa de convolución 3D a la que sigue una capa de Max_pooling 3D que realizara una reducción de parámetros. La siguiente capa (Flatten) convertirá la matriz multidimensional en un vector, permitiendo de esta forma que la salida pueda ser procesada por una capa totalmente conectada (Dense). Esta capa totalmente conectada es la última de nuestras capas.

Podemos ver como esta ultima capa la reducción de parámetros es muy grande y puede traer consigo una gran pérdida de información. No hemos añadido más complejidad el modelo para no aumentar el tiempo de cálculo, pero en modelos más avanzados se podrían añadir más capas Dense con el propósito de hacer una reducción de parámetros más progresiva y reducir de esta manera la pérdida de información.

El grafo resultante de este primer modelo es el siguiente.



Il·lustració 8 - Grafo modelo1

Los primeros intentos de entrenamiento de este modelo han chocado con problemas de memoria que hacen que el ejecutor de Python se termine de forma incontrolada reportando errores de falta de memoria.

Para intentar solucionar el problema hemos optado por reducir la resolución de las imágenes que en formato original eran de (256, 256, 128). El tamaño al que las hemos convertido ha sido (112,112,88). Para ello hemos construido un sencillo script de conversión basado en la librería Nibabel que permite transformar todas las imágenes de un directorio a un formato definido por una imagen base.

La descripción del script generado es la siguiente:

1. Importación de librerías auxiliares

```
import os                          // Librería de gestión de archivos en disco
import nibabel as nib              // Librería de lectura de imágenes Nifti
import glob                        // Librería de lectura de ficheros en disco
from nibabel.processing import resample_from_to
                                   // Utilidad de resampleado.
```

2. Leemos todas las imágenes para realizar el entrenamiento. Estas se encuentran en un directorio concreto y tienen la extensión “.nii.gz”.

```
trainingElements = glob.glob("<Path_To_Training Images>\\*.nii.gz")
```

3. Leemos la imagen base.

```
base_img = nib.load((‘<Path_To_Base Image.nii.gz')
```

4. Para cada imagen la cargamos con Nibabel y realizamos el resampleado utilizando la imagen base como referencia de formato. Guardamos la nueva imagen con el nombre que deseamos.

```
for trainingElement in trainingElements:
    n1_img = nib.load(trainingElement)
    new_img = nib.processing.resample_from_to(n1_img, base_img)
    subject = os.path.basename(trainingElement)[:6]
    nib.save(new_img, ("<Path_To_Target_Images>/'+subject+'_3T.nii.gz')
```

El script completo está disponible en el apartado de anexos. ResampleTests.py.

Tras el cambio de formato, el proceso de entrenamiento se ejecuta correctamente, eso sí consumiendo hasta el último recurso de la máquina.

Al iniciar el proceso de entrenamiento, la consola arroja el siguiente error:

T:\src\github\tensorflow\tensorflow\core\platform\cpu_feature_guard.cc:140] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2

Este mensaje hace referencia a la posibilidad de disponer de una versión de TensorFlow que utiliza ciertas operaciones incluidas en los nuevos procesadores Intel. Tras intentar conseguir de forma insatisfactoria una versión de TensorFlow ya compilada que incluya estas extensiones, hemos optado por no intentar compilar la librería por nosotros mismos dada la complejidad del proceso. Este mensaje es solo informativo, las librerías funcionan, pero teóricamente el rendimiento es menor.

Para este primer modelo, la configuración final de cada una de las capas es la siguiente:

Name	***conv3d_1***
Batch_input_shape	(None, 112, 112, 88, 1)
Dtype	float32
Filters	64
Kernel_size	(8, 8, 8)
Strides	(1, 1, 1)
Otros parámetros	'padding': 'same', 'data_format': 'channels_last', 'dilation_rate': (1, 1, 1), 'activation': 'relu', 'use_bias': True, 'kernel_initializer': {'class_name': 'VarianceScaling', 'config': {'scale': 1.0, 'mode': 'fan_avg', 'distribution': 'uniform', 'seed': None}}, 'bias_initializer': {'class_name': 'Zeros', 'config': {}}, 'kernel_regularizer': None,

	'bias_regularizer': None, 'activity_regularizer': None, 'kernel_constraint': None, 'bias_constraint': None

Name	***max_pooling3d_1***
Pool_size	(2, 2, 2)
Padding	valid
Strides	(2, 2, 2)
Data_format	channels_last

Name	***flatten_1***
Data_format	channels_last

Name	***dense_1***
Otros parámetros	'units': 1, 'activation': 'linear', 'use_bias': True, 'kernel_initializer': {'class_name': 'VarianceScaling', 'config': {'scale': 1.0, 'mode': 'fan_avg', 'distribution': 'uniform', 'seed': None}}, 'bias_initializer': {'class_name': 'Zeros', 'config': {}}, 'kernel_regularizer': None, 'bias_regularizer': None, 'activity_regularizer': None, 'kernel_constraint': None, 'bias_constraint': None}

Esta es la salida del proceso de entrenamiento. En ella podemos ver como se realizan las 25 iteraciones configuradas sobre 18 imágenes. 2 imágenes son usadas para verificación. Cada iteración dura aproximadamente entre 5300 y 6800 segundos para un tiempo total de entrenamiento de 38 horas. Los valores de perdida se van reduciendo progresivamente lo que indica una convergencia en el modelo.

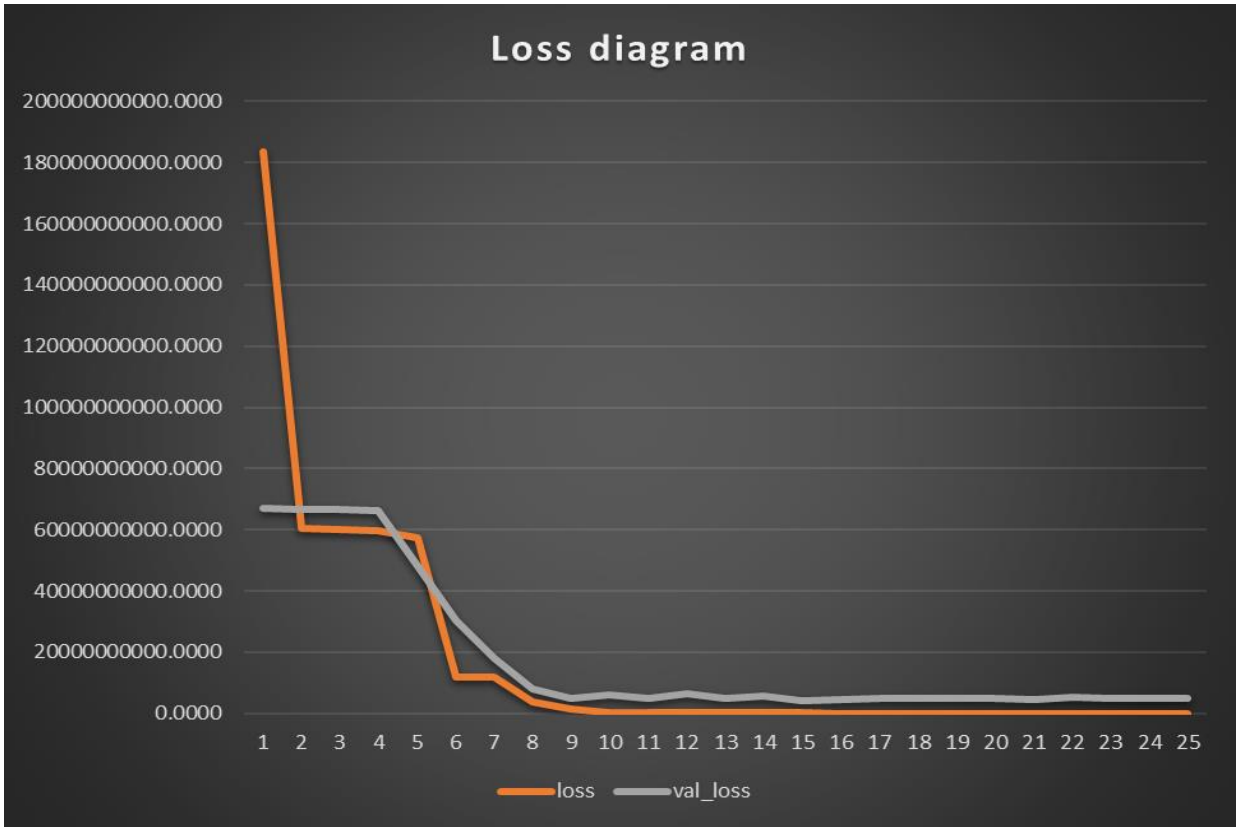
```
Epoch 1/25
 18/18 [==============================] - 5417s 301s/step - loss: 183455182890.6667 - acc: 0.0000e+00 - val_loss: 66863306752.0000 - val_acc: 0.0000e+00
Epoch 2/25
18/18 [==============================] - 5364s 298s/step - loss: 60474422613.3333 - acc: 0.0000e+00 - val_loss: 66762221568.0000 - val_acc: 0.0000e+00
Epoch 3/25
18/18 [==============================] - 5341s 297s/step - loss: 60243016590.2222 - acc: 0.0000e+00 - val_loss: 66554732544.0000 - val_acc: 0.0000e+00
Epoch 4/25
18/18 [==============================] - 5394s 300s/step - loss: 59831679886.2222 - acc: 0.0000e+00 - val_loss: 66148237312.0000 - val_acc: 0.0000e+00
Epoch 5/25
18/18 [==============================] - 5434s 302s/step - loss: 57557729735.1111 - acc: 0.0000e+00 - val_loss: 48162950144.0000 - val_acc: 0.0000e+00
Epoch 6/25
18/18 [==============================] - 5760s 320s/step - loss: 12017891987.3333 - acc: 0.0000e+00 - val_loss: 30540083200.0000 - val_acc: 0.0000e+00
Epoch 7/25
18/18 [==============================] - 5364s 298s/step - loss: 11731721634.6111 - acc: 0.0000e+00 - val_loss: 18082777088.0000 - val_acc: 0.0000e+00
Epoch 8/25
18/18 [==============================] - 6881s 382s/step - loss: 3707471305.8889 - acc: 0.0000e+00 - val_loss: 8128333824.0000 - val_acc: 0.0000e+00
Epoch 9/25
18/18 [==============================] - 6276s 349s/step - loss: 1342386091.7778 - acc: 0.0000e+00 - val_loss: 4847563904.0000 - val_acc: 0.0000e+00
Epoch 10/25
18/18 [==============================] - 5854s 325s/step - loss: 389785193.5556 - acc: 0.0000e+00 - val_loss: 6147196032.0000 - val_acc: 0.0000e+00
Epoch 11/25
18/18 [==============================] - 5610s 312s/step - loss: 415246899.7257 - acc: 0.0000e+00 - val_loss: 4936909696.0000 - val_acc: 0.0000e+00
Epoch 12/25
18/18 [==============================] - 5537s 308s/step - loss: 276438549.8333 - acc: 0.0000e+00 - val_loss: 6244257280.0000 - val_acc: 0.0000e+00
Epoch 13/25
18/18 [==============================] - 5890s 327s/step - loss: 266085881.1823 - acc: 0.0000e+00 - val_loss: 4856825088.0000 - val_acc: 0.0000e+00
Epoch 14/25
18/18 [==============================] - 5667s 315s/step - loss: 94138163.6042 - acc: 0.0000e+00 - val_loss: 5537867008.0000 - val_acc: 0.0000e+00
Epoch 15/25
18/18 [==============================] - 5663s 315s/step - loss: 77036383.0694 - acc: 0.0000e+00 - val_loss: 4313257088.0000 - val_acc: 0.0000e+00
Epoch 16/25
18/18 [==============================] - 5337s 296s/step - loss: 60805928.7526 - acc: 0.0000e+00 - val_loss: 4636764800.0000 - val_acc: 0.0000e+00
Epoch 17/25
18/18 [==============================] - 5332s 296s/step - loss: 42538640.8872 - acc: 0.0000e+00 - val_loss: 5014618368.0000 - val_acc: 0.0000e+00
Epoch 18/25
18/18 [==============================] - 5324s 296s/step - loss: 21374885.6137 - acc: 0.0000e+00 - val_loss: 4707144960.0000 - val_acc: 0.0000e+00
```

```
Epoch 19/25
18/18 [==============================] - 5330s 296s/step - loss: 9510185.6797 - acc: 0.0000e+00 - val_loss: 4723847424.0000 - val_acc: 0.0000e+00
Epoch 20/25
18/18 [==============================] - 5321s 296s/step - loss: 19007046.4010 - acc: 0.0000e+00 - val_loss: 4761959936.0000 - val_acc: 0.0000e+00
Epoch 21/25
18/18 [==============================] - 5343s 297s/step - loss: 17771438.0872 - acc: 0.0000e+00 - val_loss: 4641987840.0000 - val_acc: 0.0000e+00
Epoch 22/25
18/18 [==============================] - 5471s 304s/step - loss: 22286854.1658 - acc: 0.0000e+00 - val_loss: 5400693760.0000 - val_acc: 0.0000e+00
Epoch 23/25
18/18 [==============================] - 5394s 300s/step - loss: 9918069.7027 - acc: 0.0000e+00 - val_loss: 4940108288.0000 - val_acc: 0.0000e+00
Epoch 24/25
18/18 [==============================] - 5491s 305s/step - loss: 5786445.5191 - acc: 0.0000e+00 - val_loss: 5052015104.0000 - val_acc: 0.0000e+00
Epoch 25/25
18/18 [==============================] - 5457s 303s/step - loss: 5485310.8882 - acc: 0.0000e+00 - val_loss: 4822229760.0000 - val_acc: 0.0000e+00
```

Si expresamos la evolución de los valores de perdida obtenemos la siguiente gráfica.

En ella podemos observar cómo tras unas iteraciones iniciales donde parece estancarse la mejora en los valores de pérdida, esta mejora se produce finalmente para tras conseguir una mejora rápida estancarse un poco en las últimas iteraciones.

Esto sucede tanto para los valores de los como para los de val_loss.



Il·lustració 9 - Gráfica entrenamiento Modelo 1

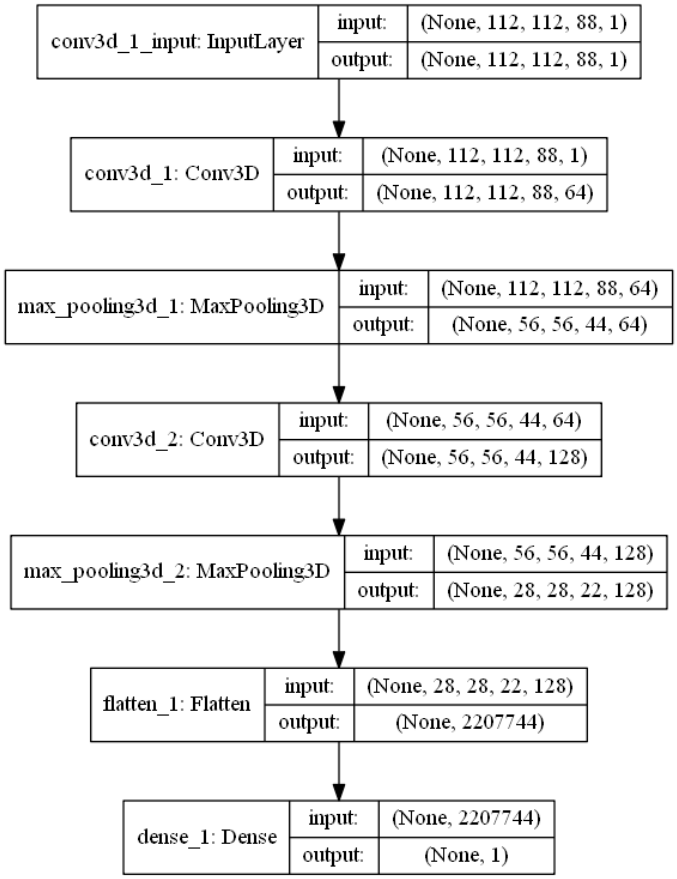
Segundo modelo

El segundo modelo desarrollado es un modelo de seis capas resultado de evolucionar el primer modelo.

Para no sufrir de nuevo los problemas de memoria hemos mantenido la misma resolución en las imágenes y solo hemos introducido cierta complejidad adicional en el modelo.

Las dos primeras capas son idénticas a las del primer modelo. La tercera capa vuelve a ser una capa de convolución 3D a la que también sigue una capa de Max pooling 3D. Utilizamos de nuevo una capa (Flatten) para convertir la matriz multidimensional en un vector que conectaremos a la última capa de nuevo una capa totalmente conectada (Dense).

El grafo resultante de este segundo modelo es el siguiente.



Il·lustració 10 - Grafo modelo 2

La configuración en detalle de cada una de las capas es la siguiente:

Name	***conv3d_1***
Batch_input_shape	(None, 112, 112, 88, 1)
Dtype	float32
Filters	64

Kernel_size	(8, 8, 8)
Strides	(1, 1, 1)
Otros parámetros	'padding': 'same', 'data_format': 'channels_last', 'dilation_rate': (1, 1, 1), 'activation': 'relu', 'use_bias': True, 'kernel_initializer': {'class_name': 'VarianceScaling', 'config': {'scale': 1.0, 'mode': 'fan_avg', 'distribution': 'uniform', 'seed': None}}, 'bias_initializer': {'class_name': 'Zeros', 'config': {}}, 'kernel_regularizer': None, 'bias_regularizer': None, 'activity_regularizer': None, 'kernel_constraint': None, 'bias_constraint': None

Name	***max_pooling3d_1***
Pool_size	(2, 2, 2)
Padding	valid
Strides	(2, 2, 2)
Data_format	channels_last

Name	***conv3d_2***
Dtype	float32
Filters	128
Kernel_size	(8, 8, 8)
Strides	(1, 1, 1)
Otros parámetros	'padding': 'same', 'data_format': 'channels_last', 'dilation_rate': (1, 1, 1), 'activation': 'relu', 'use_bias': True, 'kernel_initializer': {'class_name': 'VarianceScaling', 'config': {'scale': 1.0, 'mode': 'fan_avg', 'distribution': 'uniform', 'seed': None}}, 'bias_initializer': {'class_name': 'Zeros', 'config': {}}, 'kernel_regularizer': None, 'bias_regularizer': None, 'activity_regularizer': None, 'kernel_constraint': None, 'bias_constraint': None

Name	***max_pooling3d_2***
Pool_size	(2, 2, 2)
Padding	valid
Strides	(2, 2, 2)
Data_format	channels_last

Name	***flatten_1***
Data_format	channels_last

Name	***dense_1***
Otros parámetros	'units': 1, 'activation': 'linear', 'use_bias': True, 'kernel_initializer': {'class_name': 'VarianceScaling', 'config': {'scale': 1.0, 'mode': 'fan_avg', 'distribution': 'uniform', 'seed': None}}, 'bias_initializer': {'class_name': 'Zeros', 'config': {}}, 'kernel_regularizer': None, 'bias_regularizer': None, 'activity_regularizer': None, 'kernel_constraint': None, 'bias_constraint': None}

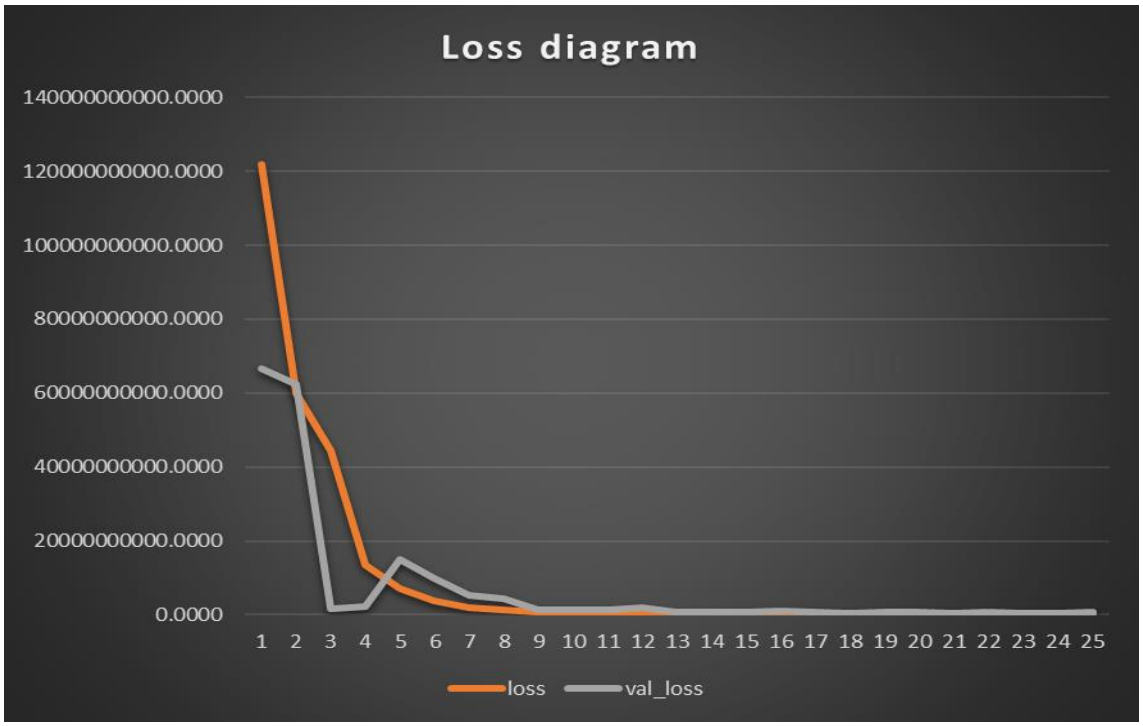
Esta es la salida del proceso de entrenamiento. En ella podemos ver como se realizan de nuevo las 25 iteraciones configuradas sobre 18 imágenes y las 2 de evaluación. Cada iteración dura esta vez aproximadamente entre 9500 y 13000 segundos, prácticamente el doble del modelo 1. El modelo tarda 71 horas en

entrenarse. Este importante incremento viene provocado por la adición de las dos nuevas capas y el aumento del número de cálculos que traen consigo.

Como sucedía en el primer modelo, los valores de perdida se van reduciendo progresivamente lo que indica que este segundo modelo también tiene una tendencia a la convergencia. Esta convergencia es más rápida que en el modelo 1.

```
Epoch 1/25
18/18 [==============================] - 9708s 539s/step - loss: 121992834389.3333 - acc: 0.0000e+00 - val_loss: 66744526848.0000 - val_acc: 0.0000e+00
Epoch 2/25
18/18 [==============================] - 9928s 552s/step - loss: 59898669283.5556 - acc: 0.0000e+00 - val_loss: 62300258304.0000 - val_acc: 0.0000e+00
Epoch 3/25
18/18 [==============================] - 9626s 535s/step - loss: 44340296135.1111 - acc: 0.0000e+00 - val_loss: 1729023240.0000 - val_acc: 0.0000e+00
Epoch 4/25
18/18 [==============================] - 9576s 532s/step - loss: 13509115332.4215 - acc: 0.0000e+00 - val_loss: 2321663779.0000 - val_acc: 0.0000e+00
Epoch 5/25
18/18 [==============================] - 9579s 532s/step - loss: 7084724048.0000 - acc: 0.0000e+00 - val_loss: 14968238656.0000 - val_acc: 0.0000e+00
Epoch 6/25
18/18 [==============================] - 9599s 533s/step - loss: 3693184882.8802 - acc: 0.0000e+00 - val_loss: 9896176512.0000 - val_acc: 0.0000e+00
Epoch 7/25
18/18 [==============================] - 9619s 534s/step - loss: 2034082202.3333 - acc: 0.0000e+00 - val_loss: 5199507776.0000 - val_acc: 0.0000e+00
Epoch 8/25
18/18 [==============================] - 10443s 580s/step - loss: 1306456291.8889 - acc: 0.0000e+00 - val_loss: 4253492224.0000 - val_acc: 0.0000e+00
Epoch 9/25
18/18 [==============================] - 9621s 534s/step - loss: 638698518.4201 - acc: 0.0000e+00 - val_loss: 1385063680.0000 - val_acc: 0.0000e+00
Epoch 10/25
18/18 [==============================] - 9648s 536s/step - loss: 840037023.9826 - acc: 0.0000e+00 - val_loss: 1297687616.0000 - val_acc: 0.0000e+00
Epoch 11/25
18/18 [==============================] - 9855s 548s/step - loss: 631876842.0347 - acc: 0.0000e+00 - val_loss: 1471295408.0000 - val_acc: 0.0000e+00
Epoch 12/25
18/18 [==============================] - 9677s 538s/step - loss: 338061522.5625 - acc: 0.0000e+00 - val_loss: 1789028887.5000 - val_acc: 0.0000e+00
Epoch 13/25
18/18 [==============================] - 9583s 532s/step - loss: 248659750.2674 - acc: 0.0000e+00 - val_loss: 808788688.0000 - val_acc: 0.0000e+00
Epoch 14/25
18/18 [==============================] - 9603s 534s/step - loss: 199615699.3672 - acc: 0.0000e+00 - val_loss: 658381296.0000 - val_acc: 0.0000e+00
Epoch 15/25
18/18 [==============================] - 9662s 537s/step - loss: 111816108.3902 - acc: 0.0000e+00 - val_loss: 826564816.0000 - val_acc: 0.0000e+00
Epoch 16/25
18/18 [==============================] - 10286s 571s/step - loss: 77580419.3194 - acc: 0.0000e+00 - val_loss: 886158184.0000 - val_acc: 0.0000e+00
Epoch 17/25
18/18 [==============================] - 12077s 671s/step - loss: 58393260.3113 - acc: 0.0000e+00 - val_loss: 695820918.0000 - val_acc: 0.0000e+00
Epoch 18/25
18/18 [==============================] - 11525s 640s/step - loss: 47026131.7934 - acc: 0.0000e+00 - val_loss: 533108396.0000 - val_acc: 0.0000e+00
Epoch 19/25
18/18 [==============================] - 11322s 629s/step - loss: 18110769.1721 - acc: 0.0000e+00 - val_loss: 651713336.0000 - val_acc: 0.0000e+00
Epoch 20/25
18/18 [==============================] - 13081s 727s/step - loss: 12794266.6381 - acc: 0.0000e+00 - val_loss: 701038872.0000 - val_acc: 0.0000e+00
Epoch 21/25
18/18 [==============================] - 10602s 589s/step - loss: 12741578.8932 - acc: 0.0000e+00 - val_loss: 405792848.0000 - val_acc: 0.0000e+00
Epoch 22/25
18/18 [==============================] - 10087s 560s/step - loss: 9964599.6263 - acc: 0.0000e+00 - val_loss: 622621664.0000 - val_acc: 0.0000e+00
Epoch 23/25
18/18 [==============================] - 10030s 557s/step - loss: 10575363.8125 - acc: 0.0000e+00 - val_loss: 469655944.0000 - val_acc: 0.0000e+00
Epoch 24/25
18/18 [==============================] - 10254s 570s/step - loss: 8308566.5491 - acc: 0.0000e+00 - val_loss: 527387288.0000 - val_acc: 0.0000e+00
Epoch 25/25
18/18 [==============================] - 11086s 616s/step - loss: 15913195.3090 - acc: 0.0000e+00 - val_loss: 618406244.0000 - val_acc: 0.0000e+00
```

Si expresamos la evolución de los valores de perdida obtenemos la siguiente gráfica. Podemos observar como esa indecisión inicial que se observaba en el primer modelo no existe en este segundo y que la convergencia inicial es casi lineal hasta alcanzar los mejores valores:



Il·lustració 11 - Gráfica entrenamiento Modelo 2

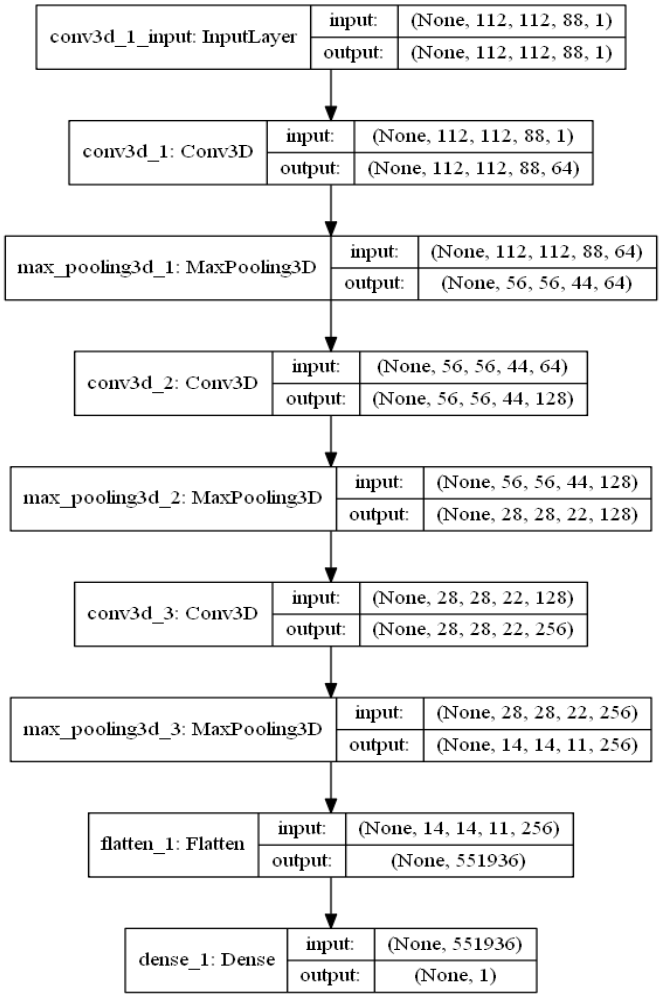
Tercer modelo

El tercer modelo desarrollado es un modelo de ocho capas resultado de evolucionar el segundo modelo de la misma manera que este evolucionaba del primero.

Para este tercer modelo disponemos con un hardware más potente, y más cantidad de memoria, pero para poder verificar el efecto del procesador sobre la velocidad de proceso hemos mantenido la misma resolución en las imágenes y hemos vuelto a introducir complejidad adicional al modelo. Los detalles sobre el hardware utilizado pueden consultarse en la sección de Anexos.

Las cuatro primeras capas son idénticas a las del segundo modelo. La quinta capa vuelve a ser una capa de convolución 3D a la que también sigue una capa de Max pooling 3D. Utilizamos de nuevo una capa (Flatten) para convertir la matriz multidimensional en un vector que conectaremos a la última capa de nuevo una capa totalmente conectada (Dense).

El grafo resultante de este tercer modelo es el siguiente.



Il·lustració 12 - Grafo Modelo 3

La configuración en detalle de cada una de las capas es la siguiente:

Name	***conv3d_1***
Batch_input_shape	(None, 112, 112, 88, 1)
Dtype	float32
Filters	64
Kernel_size	(8, 8, 8)
Strides	(1, 1, 1)
Otros parámetros	'padding': 'same', 'data_format': 'channels_last', 'dilation_rate': (1, 1, 1), 'activation': 'relu', 'use_bias': True, 'kernel_initializer': {'class_name': 'VarianceScaling', 'config': {'scale': 1.0, 'mode': 'fan_avg', 'distribution': 'uniform', 'seed': None}}, 'bias_initializer': {'class_name': 'Zeros', 'config': {}}, 'kernel_regularizer': None, 'bias_regularizer': None, 'activity_regularizer': None, 'kernel_constraint': None, 'bias_constraint': None

Name	***max_pooling3d_1***
Pool_size	(2, 2, 2)
Padding	valid

Strides	(2, 2, 2)
Data_format	channels_last

Name	***conv3d_2***
Dtype	float32
Filters	128
Kernel_size	(8, 8, 8)
Strides	(1, 1, 1)
Otros parámetros	'padding': 'same', 'data_format': 'channels_last', 'dilation_rate': (1, 1, 1), 'activation': 'relu', 'use_bias': True, 'kernel_initializer': {'class_name': 'VarianceScaling', 'config': {'scale': 1.0, 'mode': 'fan_avg', 'distribution': 'uniform', 'seed': None}}, 'bias_initializer': {'class_name': 'Zeros', 'config': {}}, 'kernel_regularizer': None, 'bias_regularizer': None, 'activity_regularizer': None, 'kernel_constraint': None, 'bias_constraint': None

Name	***max_pooling3d_2***
Pool_size	(2, 2, 2)
Padding	valid
Strides	(2, 2, 2)
Data_format	channels_last

Name	***conv3d_3***
Dtype	float32
Filters	256
Kernel_size	(8, 8, 8)
Strides	(1, 1, 1)
Otros parámetros	'padding': 'same', 'data_format': 'channels_last', 'dilation_rate': (1, 1, 1), 'activation': 'relu', 'use_bias': True, 'kernel_initializer': {'class_name': 'VarianceScaling', 'config': {'scale': 1.0, 'mode': 'fan_avg', 'distribution': 'uniform', 'seed': None}}, 'bias_initializer': {'class_name': 'Zeros', 'config': {}}, 'kernel_regularizer': None, 'bias_regularizer': None, 'activity_regularizer': None, 'kernel_constraint': None, 'bias_constraint': None

Name	***max_pooling3d_3***
Pool_size	(2, 2, 2)
Padding	valid
Strides	(2, 2, 2)
Data_format	channels_last

Name	***flatten_1***
Data_format	channels_last

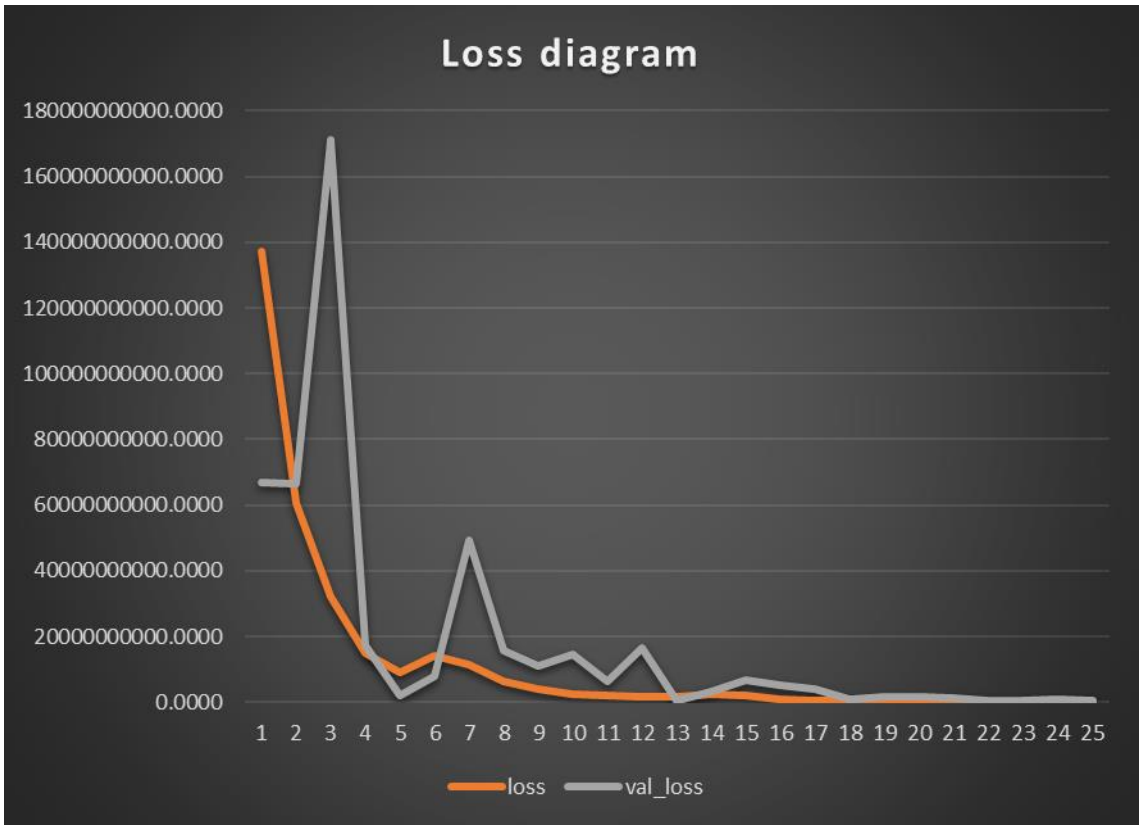
Name	***dense_1***
Otros parámetros	'units': 1, 'activation': 'linear', 'use_bias': True, 'kernel_initializer': {'class_name': 'VarianceScaling', 'config': {'scale': 1.0, 'mode': 'fan_avg', 'distribution': 'uniform', 'seed': None}}, 'bias_initializer': {'class_name': 'Zeros', 'config': {}}, 'kernel_regularizer': None, 'bias_regularizer': None, 'activity_regularizer': None, 'kernel_constraint': None, 'bias_constraint': None}

La salida del proceso de entrenamiento para este tercer modelo es la siguiente. En ella podemos ver como se realizan de nuevo las 25 iteraciones configuradas sobre 18 imágenes. Se mantienen las dos imágenes de evaluación. Para este tercer modelo disponemos con un hardware más potente, y más cantidad de memoria. Cada iteración dura esta vez aproximadamente entre 4800 y 5000 segundos para un total de 33 horas. Vemos como a pesar de incorporar complejidad al modelo 2 hemos vuelto a los tiempos del modelo 1.

El comportamiento de este modelo durante el entrenamiento es muy similar al del modelo 2.

```
Epoch 1/25
18/18 [==============================] - 32558s 1809s/step - loss: 137017574286.2222 - acc: 0.0000e+00 - val_loss: 66809565184.0000 - val_acc: 0.0000e+00
Epoch 2/25
18/18 [==============================] - 4862s 270s/step - loss: 60619969422.2222 - acc: 0.0000e+00 - val_loss: 66567124992.0000 - val_acc: 0.0000e+00
Epoch 3/25
18/18 [==============================] - 4876s 271s/step - loss: 32338262926.2222 - acc: 0.0000e+00 - val_loss: 170974011392.0000 - val_acc: 0.0000e+00
Epoch 4/25
18/18 [==============================] - 4901s 272s/step - loss: 14803764061.7778 - acc: 0.0000e+00 - val_loss: 17419767808.0000 - val_acc: 0.0000e+00
Epoch 5/25
18/18 [==============================] - 4886s 271s/step - loss: 9206113748.4444 - acc: 0.0000e+00 - val_loss: 2259950924.0000 - val_acc: 0.0000e+00
Epoch 6/25
18/18 [==============================] - 4968s 276s/step - loss: 14380059094.1806 - acc: 0.0000e+00 - val_loss: 7781719680.0000 - val_acc: 0.0000e+00
Epoch 7/25
18/18 [==============================] - 4912s 273s/step - loss: 11572899445.5000 - acc: 0.0000e+00 - val_loss: 49190656000.0000 - val_acc: 0.0000e+00
Epoch 8/25
18/18 [==============================] - 4884s 271s/step - loss: 6582656216.0000 - acc: 0.0000e+00 - val_loss: 15890708736.0000 - val_acc: 0.0000e+00
Epoch 9/25
18/18 [==============================] - 5058s 281s/step - loss: 3937490823.4062 - acc: 0.0000e+00 - val_loss: 11062178176.0000 - val_acc: 0.0000e+00
Epoch 10/25
18/18 [==============================] - 5050s 281s/step - loss: 2630638912.4444 - acc: 0.0000e+00 - val_loss: 14648478848.0000 - val_acc: 0.0000e+00
Epoch 11/25
18/18 [==============================] - 4849s 269s/step - loss: 2037162349.5556 - acc: 0.0000e+00 - val_loss: 6234406656.0000 - val_acc: 0.0000e+00
Epoch 12/25
18/18 [==============================] - 4856s 270s/step - loss: 1777616805.9861 - acc: 0.0000e+00 - val_loss: 16680660416.0000 - val_acc: 0.0000e+00
Epoch 13/25
18/18 [==============================] - 4847s 269s/step - loss: 1818732727.5556 - acc: 0.0000e+00 - val_loss: 547119304.0000 - val_acc: 0.0000e+00
Epoch 14/25
18/18 [==============================] - 4867s 270s/step - loss: 2434929381.6343 - acc: 0.0000e+00 - val_loss: 3107065472.0000 - val_acc: 0.0000e+00
Epoch 15/25
18/18 [==============================] - 4850s 269s/step - loss: 1925323487.8125 - acc: 0.0000e+00 - val_loss: 6920516936.0000 - val_acc: 0.0000e+00
Epoch 16/25
18/18 [==============================] - 4848s 269s/step - loss: 957330230.8819 - acc: 0.0000e+00 - val_loss: 5130506782.0000 - val_acc: 0.0000e+00
Epoch 17/25
18/18 [==============================] - 4851s 269s/step - loss: 517101236.0634 - acc: 0.0000e+00 - val_loss: 4202301707.0000 - val_acc: 0.0000e+00
Epoch 18/25
18/18 [==============================] - 4839s 269s/step - loss: 533372685.4627 - acc: 0.0000e+00 - val_loss: 986567840.0000 - val_acc: 0.0000e+00
Epoch 19/25
18/18 [==============================] - 4849s 269s/step - loss: 664985211.7222 - acc: 0.0000e+00 - val_loss: 1709308418.0000 - val_acc: 0.0000e+00
Epoch 20/25
18/18 [==============================] - 4849s 269s/step - loss: 353069886.0417 - acc: 0.0000e+00 - val_loss: 1757374857.7500 - val_acc: 0.0000e+00
Epoch 21/25
18/18 [==============================] - 5146s 286s/step - loss: 241707972.4562 - acc: 0.0000e+00 - val_loss: 1144344968.0000 - val_acc: 0.0000e+00
Epoch 22/25
18/18 [==============================] - 4925s 274s/step - loss: 269585502.7674 - acc: 0.0000e+00 - val_loss: 523220580.0000 - val_acc: 0.0000e+00
Epoch 23/25
18/18 [==============================] - 4861s 270s/step - loss: 232220737.1632 - acc: 0.0000e+00 - val_loss: 675767078.5000 - val_acc: 0.0000e+00
Epoch 24/25
18/18 [==============================] - 4858s 270s/step - loss: 116800566.2118 - acc: 0.0000e+00 - val_loss: 765717477.7109 - val_acc: 0.0000e+00
Epoch 25/25
18/18 [==============================] - 4847s 269s/step - loss: 119226946.9306 - acc: 0.0000e+00 - val_loss: 698961662.6953 - val_acc: 0.0000e+00
```

Si expresamos la evolución de los valores de perdida obtenemos la siguiente gráfica. Como podemos observar, la función de loss es prácticamente idéntica a la del modelo 2 mientras que la de val_loss tiene una figura mucho más abrupta.



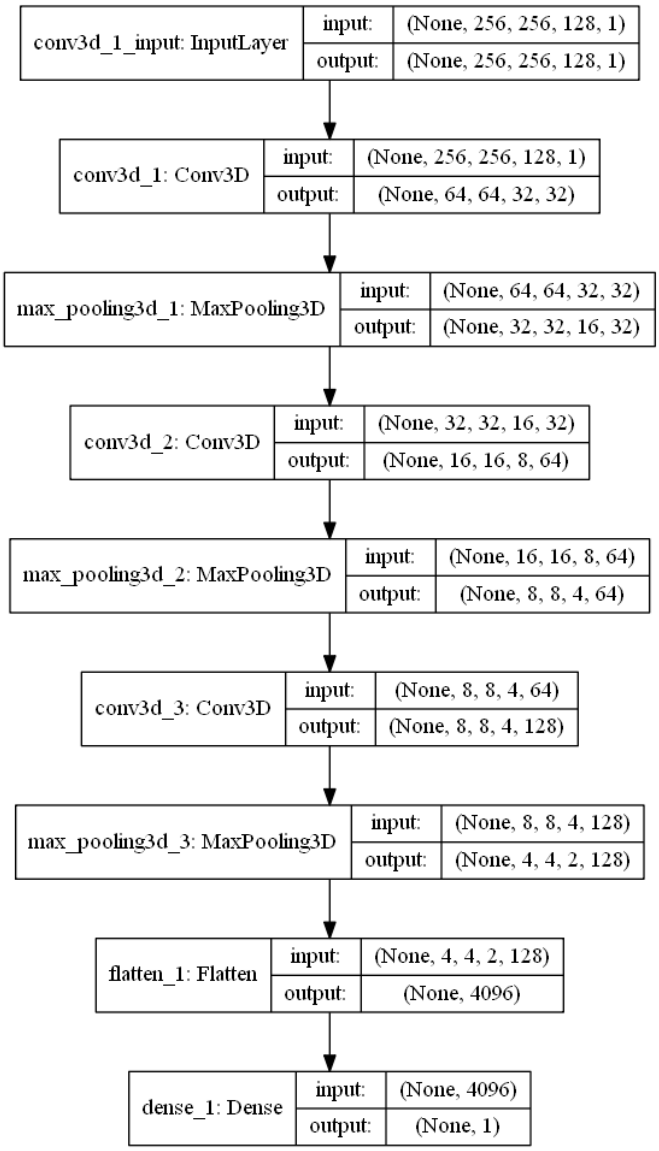
Il·lustració 13 - Gráfica entrenamiento Modelo 3

Cuarto modelo

Para el cuarto modelo hemos cambiado la estrategia ya que por un lado disponemos ahora de un ordenador con más capacidad de memoria lo que nos va a permitir poder procesar imágenes con mayor resolución.

Por otro lado, entendemos que el nivel de complejidad del modelo de ocho capas es suficiente por lo que vamos a optar por modificar alguna de las configuraciones, concretamente reduciendo la cantidad de filtros en las convoluciones y reduciendo el tamaño de kernel en las convoluciones 2 y 3.

La resolución ahora será la inicial de (256, 256, 128)



Il·lustració 14 - Grafo Modelo 4

Name	***conv3d_1***
Batch_input_shape	(None, 256, 256, 128, 1)
Dtype	float32
Filters	32
Kernel_size	(8, 8, 8)
Strides	(4, 4, 4)
Otros parámetros	'padding': 'same', 'data_format': 'channels_last', 'dilation_rate': (1, 1, 1), 'activation': 'relu', 'use_bias': True, 'kernel_initializer': {'class_name': 'VarianceScaling', 'config': {'scale': 1.0, 'mode': 'fan_avg', 'distribution': 'uniform', 'seed': None}}, 'bias_initializer': {'class_name': 'Zeros', 'config': {}}, 'kernel_regularizer': None, 'bias_regularizer': None, 'activity_regularizer': None, 'kernel_constraint': None, 'bias_constraint': None

Name	***max_pooling3d_1***

Pool_size	(2, 2, 2)
Padding	valid
Strides	(2, 2, 2)
Data_format	channels_last

Name	***conv3d_2***
Dtype	float32
Filters	64
Kernel_size	(6,6,6)
Strides	(2, 2, 2)
Otros parámetros	'padding': 'same', 'data_format': 'channels_last', 'dilation_rate': (1, 1, 1), 'activation': 'relu', 'use_bias': True, 'kernel_initializer': {'class_name': 'VarianceScaling', 'config': {'scale': 1.0, 'mode': 'fan_avg', 'distribution': 'uniform', 'seed': None}}, 'bias_initializer': {'class_name': 'Zeros', 'config': {}}, 'kernel_regularizer': None, 'bias_regularizer': None, 'activity_regularizer': None, 'kernel_constraint': None, 'bias_constraint': None

Name	***max_pooling3d_2***
Pool_size	(2, 2, 2)
Padding	valid
Strides	(2, 2, 2)
Data_format	channels_last

Name	***conv3d_3***
Dtype	float32
Filters	128
Kernel_size	(4, 4, 4)
Strides	(1, 1, 1)
Otros parámetros	'padding': 'same', 'data_format': 'channels_last', 'dilation_rate': (1, 1, 1), 'activation': 'relu', 'use_bias': True, 'kernel_initializer': {'class_name': 'VarianceScaling', 'config': {'scale': 1.0, 'mode': 'fan_avg', 'distribution': 'uniform', 'seed': None}}, 'bias_initializer': {'class_name': 'Zeros', 'config': {}}, 'kernel_regularizer': None, 'bias_regularizer': None, 'activity_regularizer': None, 'kernel_constraint': None, 'bias_constraint': None

Name	***max_pooling3d_3***
Pool_size	(2, 2, 2)
Padding	valid
Strides	(2, 2, 2)
Data_format	channels_last

Name	***flatten_1***
Data_format	channels_last

Name	***dense_1***
Otros parámetros	'units': 1, 'activation': 'linear', 'use_bias': True, 'kernel_initializer': { 'class_name': 'RandomNormal', 'config': {'mean': 0.0, 'stddev': 0.05, 'seed': None}}, 'bias_initializer': {'class_name': 'Zeros', 'config': {}}, 'kernel_regularizer': None, 'bias_regularizer': None, 'activity_regularizer': None, 'kernel_constraint': None,

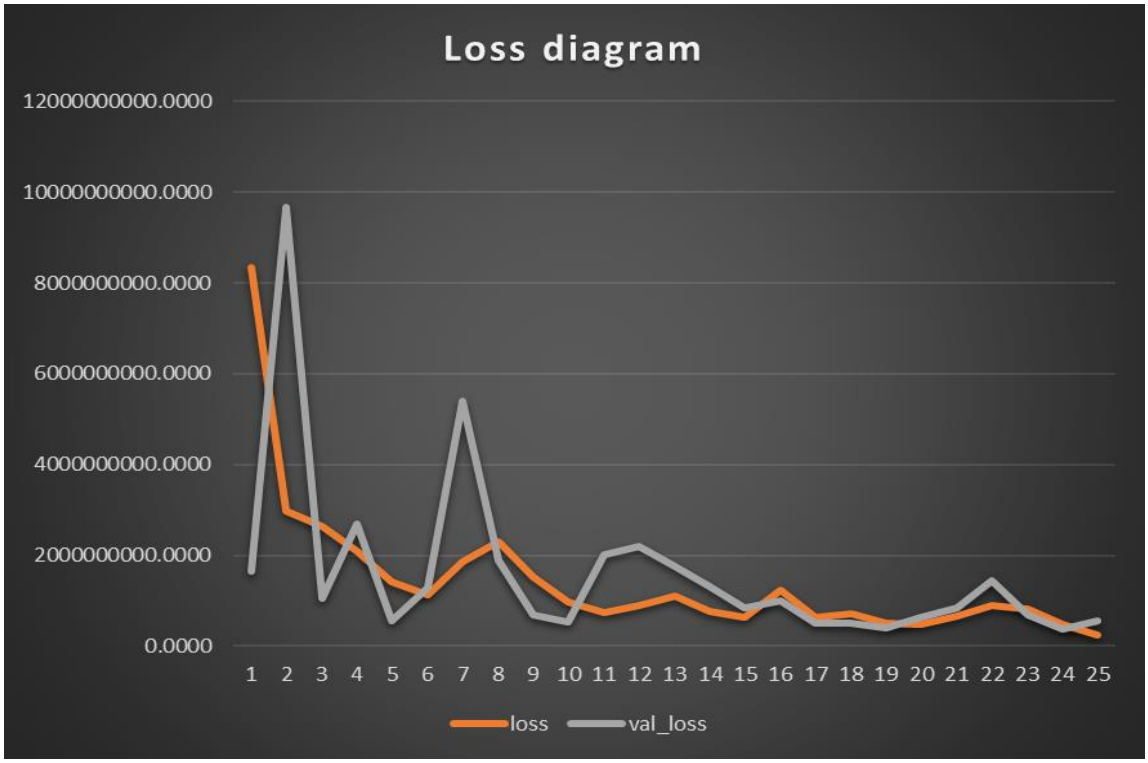
	'bias_constraint': None}

La salida del proceso de entrenamiento para este tercer modelo es la siguiente. En ella podemos ver como se realizan de nuevo las 25 iteraciones configuradas sobre 46 imágenes. Se asignan 6 imágenes de evaluación. Cada iteración dura esta vez aproximadamente 20000 segundos unas 4 veces en tiempo utilizado en el modelo 3. El modelo necesita aproximadamente 140 horas para completar el entrenamiento. Este aumento de tiempo es debido ahora por el aumento de las muestras de entrenamiento y por el cambio de resolución de las imágenes.

El comportamiento de este modelo durante el entrenamiento es diferente a los anteriores porque en la primera iteración ya se consiguen mejores resultados por lo que el progreso del proceso de aprendizaje es más lento y un poco mas irregular.

```
Epoch 1/25
46/46 [==============================] - 20173s 439s/step - loss: 8329890980.5082 - acc: 0.0000e+00 - val_loss: 1642668576.1250 - val_acc: 0.0000e+00
Epoch 2/25
46/46 [==============================] - 20101s 437s/step - loss: 2982502502.5000 - acc: 0.0000e+00 - val_loss: 9657304234.6667 - val_acc: 0.0000e+00
Epoch 3/25
46/46 [==============================] - 20038s 436s/step - loss: 2638361445.6705 - acc: 0.0000e+00 - val_loss: 1038780268.0000 - val_acc: 0.0000e+00
Epoch 4/25
46/46 [==============================] - 20076s 436s/step - loss: 2099940127.7201 - acc: 0.0000e+00 - val_loss: 2702513322.6667 - val_acc: 0.0000e+00
Epoch 5/25
46/46 [==============================] - 20135s 438s/step - loss: 1428130481.6675 - acc: 0.0000e+00 - val_loss: 548533103.1667 - val_acc: 0.0000e+00
Epoch 6/25
46/46 [==============================] - 20111s 437s/step - loss: 1119114421.1338 - acc: 0.0000e+00 - val_loss: 1298313913.3333 - val_acc: 0.0000e+00
Epoch 7/25
46/46 [==============================] - 20081s 437s/step - loss: 1850276563.1522 - acc: 0.0000e+00 - val_loss: 5395355328.0000 - val_acc: 0.0000e+00
Epoch 8/25
46/46 [==============================] - 20068s 436s/step - loss: 2306845049.7391 - acc: 0.0000e+00 - val_loss: 1891941808.0000 - val_acc: 0.0000e+00
Epoch 9/25
46/46 [==============================] - 20050s 436s/step - loss: 1532190636.9130 - acc: 0.0000e+00 - val_loss: 685988228.5000 - val_acc: 0.0000e+00
Epoch 10/25
46/46 [==============================] - 20079s 437s/step - loss: 960981445.5679 - acc: 0.0000e+00 - val_loss: 535318933.5000 - val_acc: 0.0000e+00
Epoch 11/25
46/46 [==============================] - 20058s 436s/step - loss: 731589779.8784 - acc: 0.0000e+00 - val_loss: 2017294570.2500 - val_acc: 0.0000e+00
Epoch 12/25
46/46 [==============================] - 20042s 436s/step - loss: 903374239.5326 - acc: 0.0000e+00 - val_loss: 2186761381.3333 - val_acc: 0.0000e+00
Epoch 13/25
46/46 [==============================] - 20049s 436s/step - loss: 1105947705.5000 - acc: 0.0000e+00 - val_loss: 1741033730.8333 - val_acc: 0.0000e+00
Epoch 14/25
46/46 [==============================] - 20043s 436s/step - loss: 754006731.4239 - acc: 0.0000e+00 - val_loss: 1315669709.3333 - val_acc: 0.0000e+00
Epoch 15/25
46/46 [==============================] - 20061s 436s/step - loss: 635265042.4674 - acc: 0.0000e+00 - val_loss: 853834878.6667 - val_acc: 0.0000e+00
Epoch 16/25
46/46 [==============================] - 20042s 436s/step - loss: 1241282189.8043 - acc: 0.0000e+00 - val_loss: 1009268302.0365 - val_acc: 0.0000e+00
Epoch 17/25
46/46 [==============================] - 20095s 437s/step - loss: 635408827.9761 - acc: 0.0000e+00 - val_loss: 504739309.9583 - val_acc: 0.0000e+00
Epoch 18/25
46/46 [==============================] - 20090s 437s/step - loss: 714262841.1591 - acc: 0.0000e+00 - val_loss: 513270115.0000 - val_acc: 0.0000e+00
Epoch 19/25
46/46 [==============================] - 20110s 437s/step - loss: 510313550.7174 - acc: 0.0000e+00 - val_loss: 412531704.0104 - val_acc: 0.0000e+00
Epoch 20/25
46/46 [==============================] - 20061s 436s/step - loss: 466157632.9579 - acc: 0.0000e+00 - val_loss: 632366613.5000 - val_acc: 0.0000e+00
Epoch 21/25
46/46 [==============================] - 20057s 436s/step - loss: 655597478.7723 - acc: 0.0000e+00 - val_loss: 846551849.1276 - val_acc: 0.0000e+00
Epoch 22/25
46/46 [==============================] - 20114s 437s/step - loss: 899562166.1579 - acc: 0.0000e+00 - val_loss: 1440617445.6667 - val_acc: 0.0000e+00
Epoch 23/25
46/46 [==============================] - 20115s 437s/step - loss: 808833592.7418 - acc: 0.0000e+00 - val_loss: 676054249.2500 - val_acc: 0.0000e+00
Epoch 24/25
46/46 [==============================] - 20094s 437s/step - loss: 481114642.5380 - acc: 0.0000e+00 - val_loss: 373303701.2083 - val_acc: 0.0000e+00
Epoch 25/25
46/46 [==============================] - 20061s 436s/step - loss: 232104427.1464 - acc: 0.0000e+00 - val_loss: 564187728.5000 - val_acc: 0.0000e+00
```

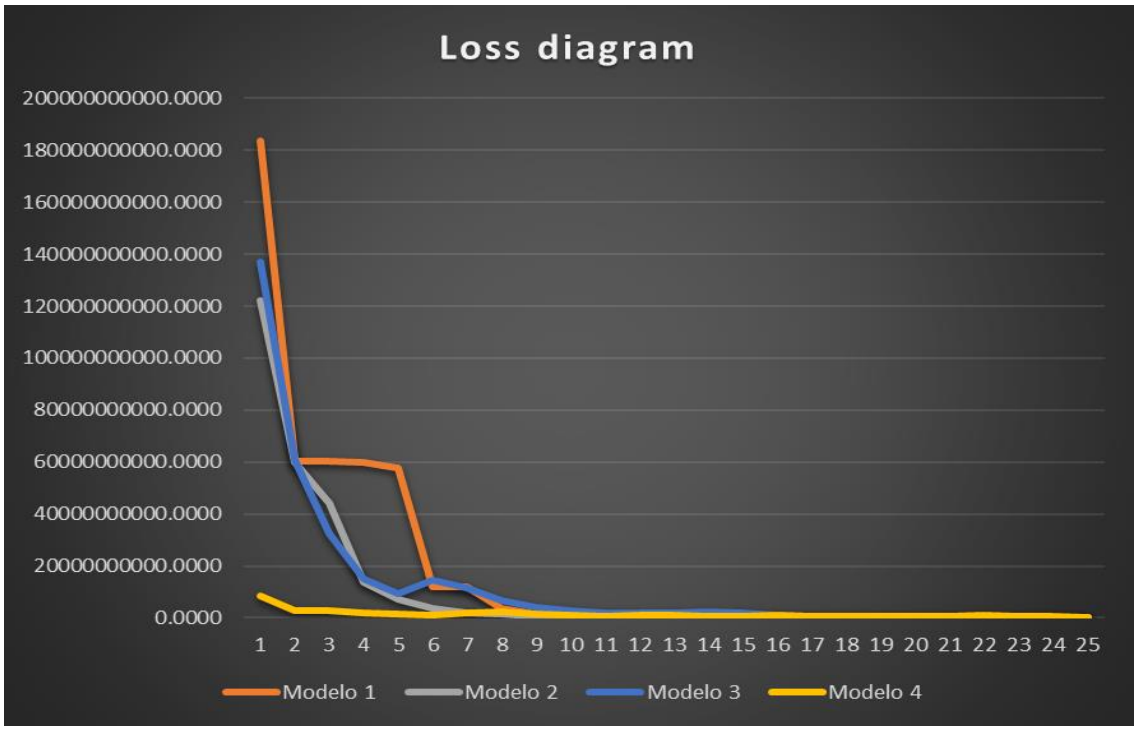
Si expresamos la evolución de los valores de perdida obtenemos la siguiente gráfica. En ella podemos ver como las curvas son mucho menos pronunciadas y un poco menos regulares que en los modelos anteriores.



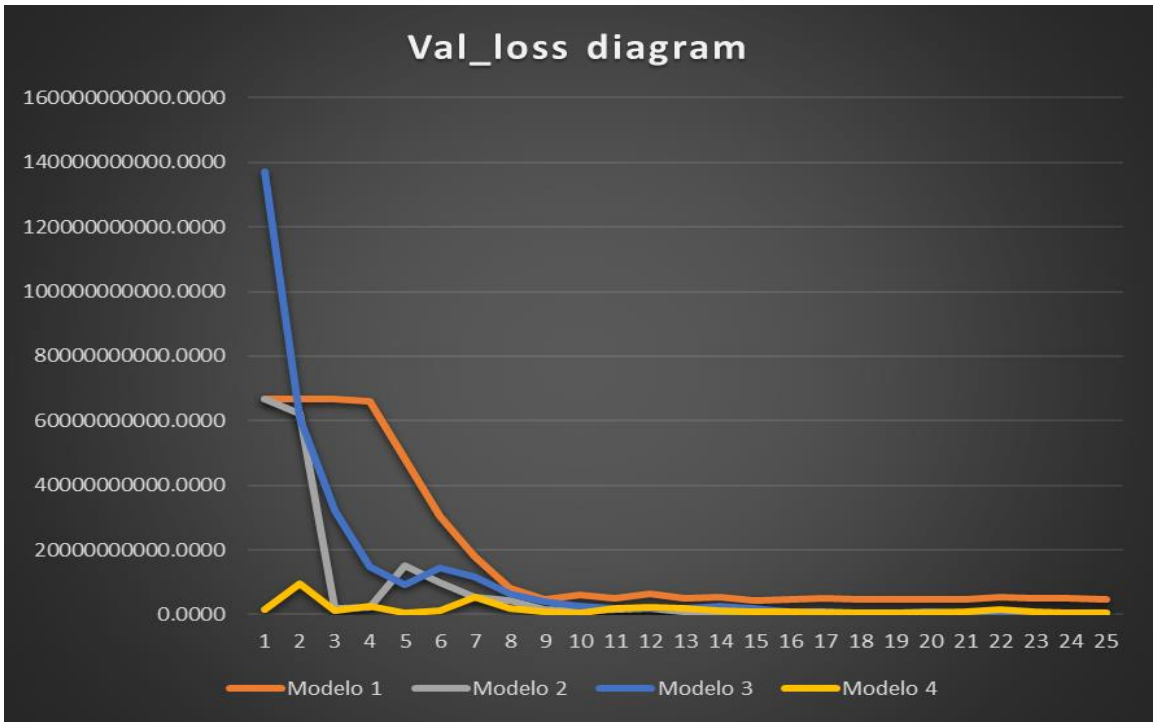
Il·lustració 15 - Gráfica entrenamiento Modelo 4

Si comparamos los diagramas de valores de pérdidas de los cuatro modelos obtenemos los siguientes resultados. En ellas podemos apreciar las diferencias que ya hemos comentado.

Podemos ver como el modelo 2 y 3 convergen más rapidamente que el primero siendo el cuarto modelo el que parte desde el mejor resultado inicial, esto es sin duda debido al mayor número de muestras que permite que desde un principio el modelo parta con mejores ajustes en su parametrización.



Il·lustració 16 - Grafica comparativa Loss



Il·lustració 17 - Grafica comparativa Val_Loss

4. Resultados

Para evaluar los resultados de los modelos generados, los hemos sometido a un test de 52 imágenes no usadas durante el entrenamiento. De cada una de ellas hemos obtenido el resultado de aplicarles el modelo y hemos calculado el error respecto al valor proporcionado por FreeSurfer.

Estos datos de test, al igual que los de entrenamiento y validación se han tomado de la fuente del proyecto The Human Connectome Project (HCP) de forma totalmente aleatoria, sin considerar sexo, edad o posibles patologias.

Para automatizar las pruebas hemos desarrollado un script en Python al que hemos llamado TestModelDefinition.py. El contenido de script es el siguiente:

1. Importación de librerías auxiliares

```
import os                   // Librería de gestión de archivos en disco
import nibabel as nib       // Librería de lectura de imágenes Nifti
import numpy as np          // Librería de procesado de arrays
import glob                 // Librería de lectura de ficheros en disco
import csv                  // Librería de lectura de archivos CSV.
```

2. Importación de librerías de Keras

```
from keras.models import load_model     // Importación de la utilidad de lectura
                                        de modelos existentes
```

3. Lectura del fichero de resultados disponibles en CSV. Cargamos en un diccionario todos los resultados disponibles por sujeto. Utilizamos el identificador de sujeto como clave del diccionario.

```
resultDictionary = {}
with open(<Path_To_CSV_File>) as csvfile:
   reader = csv.DictReader(csvfile)
   for row in reader:
      resultDictionary[row['Subject']] = row
```

4. Leemos todas las imágenes para realizar el testing. Estas se encuentran en un directorio concreto y tienen la extensión “.nii.gz”.

```
trainingElements = glob.glob("<Path_To_Testing Images>\\*.nii.gz")
```

5. Cargamos el modelo que deseamos testear

```
model = load_model('<ruta_fichero_modelo_h5>')
```

6. Iteramos las diferentes imágenes de testeo obteniendo el identificador del sujeto, el valor resultado de aplicar el modelo y el valor proveniente del análisis de FreeSurfer.

```
for testingElement in testingElements:
    subject = os.path.basename(testingElement)[:6]   // Recuperados el id del sujeto
    n1_img = nib.load(testingElement)                // Cargamos la imagen
    predictionData = []
    predictionData.append(n1_img.get_data())
    X_predict = np.array(predictionData)             // Formateamos la magen
    X_predict = np.expand_dims(X_predict, axis=4)
    prediction = model.predict(X_predict)
    print(subject+' - '+str(prediction[0][0])+' -    // Mostramos los resultados.
          '+str(resultDictionary[subject]['FS_LCort_GM_Vol']))
```

Los resultados para el primer modelo son los siguientes:

ID de sujeto	*Resultado*	*Resultado esperado*	*Error*	*Error Relativo*
111312	213374.60	213370	-4.60	-0.002156%
125525	230083.36	229085	-998.36	-0.435803%
131217	240805.84	239830	-975.84	-0.406888%
131722	247175.03	247773	597.97	0.241338%
140925	241674.42	240843	-831.42	-0.345212%
144832	199110.78	254890	55779.22	21.883644%
146432	275072.75	292953	17880.25	6.103453%
147737	265272.78	215145	-50127.78	-23.299533%
148335	236406.86	268209	31802.14	11.857223%
149539	293506.38	293631	124.62	0.042441%
149741	295151.56	255448	-39703.56	-15.542717%
151526	225684.61	282473	56788.39	20.104006%
151627	201205.19	220038	18832.81	8.558890%
153025	225047.33	256430	31382.67	12.238299%
154734	267977.40	267286	-691.40	-0.258674%
156637	284131.00	283167	-964.00	-0.340435%
159340	254350.97	253141	-1209.97	-0.477983%
160123	238024.55	239593	1568.45	0.654631%
161731	228667.97	226695	-1972.97	-0.870319%
162733	267075.62	267932	856.38	0.319626%
163129	241109.36	239875	-1234.36	-0.514585%
176542	226927.00	226864	-63.00	-0.027770%
188347	242654.25	241430	-1224.25	-0.507083%
189450	290876.34	289038	-1838.34	-0.636020%
190031	227499.78	225928	-1571.78	-0.695700%

192540	245506.33	244392	-1114.33	-0.455960%
196750	242607.53	242957	349.47	0.143840%
198451	225253.50	297941	72687.50	24.396609%
199655	226183.69	287918	61734.31	21.441629%
201111	245155.64	259367	14211.36	5.479248%
208226	206807.84	234102	27294.16	11.659089%
211417	224661.10	237935	13273.90	5.578793%
211720	302222.75	270787	-31435.75	-11.609032%
212318	274412.80	227348	-47064.80	-20.701656%
214423	207831.03	310341	102509.97	33.031398%
221319	233299.67	230038	-3261.67	-1.417883%
239944	207104.23	249837	42732.77	17.104260%
245333	213277.75	228207	14929.25	6.541977%
280739	232311.00	213818	-18493.00	-8.648944%
298051	273912.40	264715	-9197.40	-3.474454%
366446	273080.40	272152	-928.40	-0.341133%
397760	207994.16	236113	28118.84	11.909060%
414229	251076.38	190371	-60705.38	-31.887935%
499566	189558.28	248280	58721.72	23.651410%
654754	145755.23	211793	66037.77	31.180336%
672756	296922.62	253963	-42959.62	-16.915700%
751348	228874.53	281548	52673.47	18.708522%
756055	207740.90	218348	10607.10	4.857887%
792564	217755.84	261609	43853.16	16.762864%
856766	173035.47	244272	71236.53	29.162790%
857263	293161.72	276254	-16907.72	-6.120353%
899885	251909.98	237975	-13934.98	-5.855649%
		Media	10522.49	3.688917%
		Maximo	102509.97	33.031398%
		Mínimo	4.60	0.002156%

Como podemos observar en los resultados, 20 de los 52 resultados tienen un error inferior al 1%, 11 inferior al 10%, 10 inferior al 20%, 8 inferior al 30% y 3 están ligeramente por encima del 30% de error relativo.

Evidentemente estos datos no son excesivamente buenos, pero sí que pueden considerarse aceptables si tenemos en cuenta la escasa cantidad de muestras utilizadas para el training y el pequeño número de iteraciones.

Los resultados del segundo modelo son:

ID de sujeto	*Resultado*	*Resultado esperado*	*Error*	*Error Relativo*
111312	215259.70	213370	-1889.70	-0.885645%
125525	229858.64	229085	-773.64	-0.337709%
131217	242686.64	239830	-2856.64	-1.191110%
131722	250047.77	247773	-2274.77	-0.918086%

140925	249237.52	240843	-8394.52	-3.485474%
144832	323384.66	254890	-68494.66	-26.872243%
146432	428179.22	292953	-135226.22	-46.159698%
147737	437669.30	215145	-222524.30	-103.429919%
148335	137261.94	268209	130947.06	48.822769%
149539	524017.78	293631	-230386.78	-78.461327%
149741	115316.69	255448	140131.31	54.857079%
151526	28581.01	282473	253891.99	89.881860%
151627	281629.16	220038	-61591.16	-27.991147%
153025	275815.22	256430	-19385.22	-7.559654%
154734	268554.22	267286	-1268.22	-0.474481%
156637	283566.90	283167	-399.90	-0.141224%
159340	256511.78	253141	-3370.78	-1.331582%
160123	241698.31	239593	-2105.31	-0.878703%
161731	227960.81	226695	-1265.81	-0.558376%
162733	268152.03	267932	-220.03	-0.082122%
163129	239890.53	239875	-15.53	-0.006474%
176542	229524.06	226864	-2660.06	-1.172535%
188347	246640.22	241430	-5210.22	-2.158067%
189450	290523.25	289038	-1485.25	-0.513860%
190031	229279.80	225928	-3351.80	-1.483570%
192540	245562.77	244392	-1170.77	-0.479054%
196750	244890.72	242957	-1933.72	-0.795910%
198451	332003.53	297941	-34062.53	-11.432643%
199655	233751.03	287918	54166.97	18.813332%
201111	371556.47	259367	-112189.47	-43.255106%
208226	260792.47	234102	-26690.47	-11.401214%
211417	15968.55	237935	221966.45	93.288691%
211720	432331.00	270787	-161544.00	-59.657221%
212318	266204.03	227348	-38856.03	-17.090993%
214423	79644.09	310341	230696.91	74.336589%
221319	230661.69	230038	-623.69	-0.271125%
239944	323073.80	249837	-73236.80	-29.313833%
245333	221689.90	228207	6517.10	2.855784%
280739	291652.53	213818	-77834.53	-36.402235%
298051	355401.90	264715	-90686.90	-34.258316%
366446	482341.88	272152	-210189.88	-77.232532%
397760	52263.69	236113	183849.31	77.864968%
414229	348007.34	190371	-157636.34	-82.804807%
499566	192820.11	248280	55459.89	22.337639%
654754	203043.34	211793	8749.66	4.131232%
672756	462058.06	253963	-208095.06	-81.939125%
751348	401543.10	281548	-119995.10	-42.619766%
756055	496140.47	218348	-277792.47	-127.224646%
792564	41637.25	261609	219971.75	84.084168%
856766	358144.90	244272	-113872.90	-46.617255%
857263	533230.10	276254	-256976.10	-93.021676%
899885	279458.66	237975	-41483.66	-17.431940%

	Media	-24493.70	-10.539775%
	Maximo	277792.47	127.224646%
	Mínimo	15.53	0.006474%

Si comparamos los resultados del primer modelo respecto al segundo, los resultados del segundo son considerablemente peores.

En este caso solo 13 resultados están por debajo del 1% de error relativo, 9 están por debajo del 10%, 5 por debajo del 20%, el resto presentan errores relativos mucho más altos del primer modelo superando incluso el 100% de error relativo.

Esto nos hace pensar que, para modelos más complicados, un entrenamiento más complejo es todavía más necesario. A la larga el modelo será mejor pero inicialmente su comportamiento es más impreciso.

Está afirmación confirma también los resultados de las gráficas obtenidas durante los procesos de entrenamiento donde podíamos ver cómo cuanto más complejo era el modelo más abruptas e inconstantes eran las gráficas de progreso de los valores de pérdida

Veamos si estas suposiciones se confirman con el tercer modelo.

Los resultados del tercer modelo son los siguientes:

ID de sujeto	*Resultado*	*Resultado esperado*	*Error*	*Error Relativo*
111312	203181.67	213370	10188.33	4.774959%
125525	228404.77	229085	680.23	0.296933%
131217	247590.14	239830	-7760.14	-3.235684%
131722	253634.39	247773	-5861.39	-2.365629%
140925	235273.78	240843	5569.22	2.312386%
144832	345274.72	254890	-90384.72	-35.460285%
146432	438972.78	292953	-146019.78	-49.844098%
147737	485241.62	215145	-270096.62	-125.541667%
148335	144916.27	268209	123292.73	45.968901%
149539	494542.12	293631	-200911.12	-68.422993%
149741	100170.11	255448	155277.89	60.786497%
151526	36499.45	282473	245973.55	87.078605%
151627	290146.72	220038	-70108.72	-31.862097%
153025	251268.70	256430	5161.30	2.012752%
154734	270430.30	267286	-3144.30	-1.176380%
156637	306946.60	283167	-23779.60	-8.397730%
159340	253234.10	253141	-93.10	-0.036778%
160123	241003.39	239593	-1410.39	-0.588661%
161731	197431.11	226695	29263.89	12.908926%
162733	289725.60	267932	-21793.60	-8.134004%

163129	239992.94	239875	-117.94	-0.049167%
176542	226098.61	226864	765.39	0.337378%
188347	246462.22	241430	-5032.22	-2.084339%
189450	290188.16	289038	-1150.16	-0.397927%
190031	228407.48	225928	-2479.48	-1.097465%
192540	248518.83	244392	-4126.83	-1.688611%
196750	242942.02	242957	14.98	0.006166%
198451	335326.47	297941	-37385.47	-12.547944%
199655	258146.40	287918	29771.60	10.340305%
201111	375041.20	259367	-115674.20	-44.598658%
208226	263799.44	234102	-29697.44	-12.685684%
211417	19346.68	237935	218588.32	91.868924%
211720	463754.97	270787	-192967.97	-71.261903%
212318	237440.36	227348	-10092.36	-4.439168%
214423	55557.48	310341	254783.52	82.097926%
221319	223241.81	230038	6796.19	2.954377%
239944	385004.84	249837	-135167.84	-54.102411%
245333	210517.12	228207	17689.88	7.751682%
280739	289308.10	213818	-75490.10	-35.305774%
298051	340638.97	264715	-75923.97	-28.681401%
366446	371503.40	272152	-99351.40	-36.505850%
397760	58199.38	236113	177913.62	75.351047%
414229	362370.66	190371	-171999.66	-90.349717%
499566	199456.75	248280	48823.25	19.664592%
654754	212293.11	211793	-500.11	-0.236132%
672756	423000.40	253963	-169037.40	-66.559853%
751348	405572.88	281548	-124024.88	-44.051061%
756055	527144.06	218348	-308796.06	-141.423810%
792564	44668.58	261609	216940.42	82.925442%
856766	373618.72	244272	-129346.72	-52.951922%
857263	490374.47	276254	-214120.47	-77.508550%
899885	276547.66	237975	-38572.66	-16.208703%
		Media	-23748.55	-10.391620%
		Maximo	308796.06	141.423810%
		Mínimo	14.98	0.006166%

La comparación de estos resultados con los del primer modelo reafirma la creencia de que un entrenamiento insuficiente es perjudicial para los modelos más complicados, aunque los resultados obtenidos en este tercer modelo son bastante similares al segundo.

En este tercer modelo solo 8 resultados están por debajo del 1% de error relativo, 14 están por debajo del 10%, 6 por debajo del 20%, el resto presentan errores relativos mucho más altos del primer modelo y similares a los del tercer modelo. También llegan a superar el 100% de error relativo.

Por último, analizamos los valores del cuarto modelo:

ID de sujeto	*Resultado*	*Resultado esperado*	*Error*	*Error Relativo*
111312	206892.89	213370	6477.11	3.035624%
125525	217285.92	229085	11799.08	5.150525%
131217	227778.47	239830	12051.53	5.025030%
131722	236076.88	247773	11696.12	4.720498%
140925	229749.10	240843	11093.90	4.606279%
144832	228430.33	254890	26459.67	10.380819%
146432	274370.03	292953	18582.97	6.343328%
147737	211128.50	215145	4016.50	1.866880%
148335	238349.84	268209	29859.16	11.132796%
149539	264344.75	293631	29286.25	9.973828%
149741	237848.33	255448	17599.67	6.889727%
151526	251612.33	282473	30860.67	10.925175%
151627	203116.78	220038	16921.22	7.690135%
153025	225746.66	256430	30683.34	11.965581%
154734	248638.19	267286	18647.81	6.976725%
156637	263423.22	283167	19743.78	6.972486%
159340	232941.70	253141	20199.30	7.979466%
160123	237048.05	239593	2544.95	1.062197%
161731	213234.23	226695	13460.77	5.937833%
162733	245603.69	267932	22328.31	8.333573%
163129	221035.56	239875	18839.44	7.853857%
176542	216073.10	226864	10790.90	4.756550%
188347	212560.98	241430	28869.02	11.957511%
189450	258444.53	289038	30593.47	10.584584%
190031	206024.45	225928	19903.55	8.809687%
192540	217920.16	244392	26471.84	10.831713%
196750	227109.50	242957	15847.50	6.522759%
198451	261787.00	297941	36154.00	12.134617%
199655	261271.94	287918	26646.06	9.254739%
201111	254083.78	259367	5283.22	2.036967%
208226	224903.90	234102	9198.10	3.929099%
211417	234451.38	237935	3483.62	1.464106%
211720	239943.88	270787	30843.12	11.390178%
212318	219404.38	227348	7943.62	3.494036%
214423	291179.97	310341	19161.03	6.174186%
221319	203672.10	230038	26365.90	11.461541%
239944	232056.60	249837	17780.40	7.116800%
245333	207168.48	228207	21038.52	9.219051%
280739	213708.39	213818	109.61	0.051263%
298051	249629.00	264715	15086.00	5.698959%
366446	251416.64	272152	20735.36	7.619036%
397760	219513.89	236113	16599.11	7.030155%
414229	191924.30	190371	-1553.30	-0.815933%
499566	227948.88	248280	20331.12	8.188787%
654754	189619.62	211793	22173.38	10.469364%

672756	228799.53	253963	25163.47	9.908321%
751348	251185.19	281548	30362.81	10.784239%
756055	234006.84	218348	-15658.84	-7.171506%
792564	221771.33	261609	39837.67	15.227943%
856766	222860.62	244272	21411.38	8.765384%
857263	263199.75	276254	13054.25	4.725452%
899885	236516.89	237975	1458.11	0.612716%
		Media	17666.07	6.866436%
		Maximo	39837.67	15.227943%
		Mínimo	109.61	0.051263%

Como podemos ver en estos resultados, el incremento de los elementos de entrenamiento ha conseguido estabilizar bastante los resultados. En este modelo ya no tenemos grandes errores como teníamos en los modelos dos y tres, pero mantenemos algunos buenos resultados y nos dé esperanza de que con un entrenamiento más complejo estos resultados serán cada vez mejor.

Si comparamos los cuatro modelos el resultado es el siguiente. En el se confirma como este último modelo empieza a dar resultados bastante interesantes.

Descripción Modelo	*Entrenamiento*	*Tiempo entrenamiento*	*Error medio*	*Desviación*
Modelo 1 4 capas Resolución (112,112,88)	25 iteraciones 18 imágenes training 2 imágenes validación	5300-6800 s/iter. 38 h total	3.688917%	13.466472%
Modelo 2 6 capas Resolución (112,112,88)	25 iteraciones 18 imágenes training 2 imágenes validación	9500-13000 s/iter. 71 h total	10.539775%	47.386478%
Modelo 3 4 capas Resolución (112,112,88)	25 iteraciones 18 imágenes training 2 imágenes validación	4800-5000 s/iter. 33 h total	10.391620%	48.299904%
Modelo 4 4 capas Resolución (256, 256, 128)	25 iteraciones 46 imágenes training 6 imágenes validación	20000 s/iter. 140 h total	6.866436%	4.098694%

5. Conclusiones

La primera conclusión es sin duda que el sistema es viable. Tanto los resultados de los procesos de entrenamiento como los de las pruebas indican que el sistema es capaz de encontrar los resultados en un periodo de tiempo muy rápido, que si recordamos era el primer de los objetivos de este trabajo.

Sin embargo, los resultados están lejos todavía de ser lo suficientemente precisos, pero sí que son suficientemente buenos como para pensar que un proceso de entrenamiento suficientemente complejo nos lleve a un sistema que cumpla con nuestras expectativas.

En resumen, podemos afirmar que se han cumplido todos los objetivos de este proyecto. Tenemos el conocimiento de las dimensiones utilizadas en el ámbito médico y un sistema para obtener resultados con los que comparar, tenemos las fuentes de datos adecuadas, tenemos las herramientas adecuadas y solo nos falta el tiempo para un entrenamiento correcto de la Inteligencia Artificial.

Si recordamos la primera motivación de arranque de este proyecto, encontrábamos que el tiempo de proceso de FreeSurfer para una imagen podía acercarse a las 12 h. Ahora hemos pasado a una respuesta del sistema de segundos, pero a un proceso de entrenamientos de muchos días. La principal ventaja está clara, el proceso de entrenamiento se realiza una vez, mientras que el procesado de la imagen tiene que hacerse para cada imagen por lo que está claro que no tardaríamos demasiado tiempo en compensar y hacer rentable el tiempo empleado en el entrenamiento.

Como conclusiones individuales de cada uno de los objetivos. Podemos afirmar que FreeSurfer es una herramienta excepcional, capaz de realmente diseccionar las resonancias magnéticas dándole al usuario la capacidad de explorar, medir, resaltar y comparar con bases estadísticas, pero tiene dos limitaciones importantes.

La primera es la gran capacidad de computo que necesita, requiriendo infraestructuras muy grandes para reducir el tiempo de procesamiento de las imágenes.

La segunda es que es una herramienta tonta, no es capaz de pensar y decidir, y nunca lo va a ser. Nunca va a ser capaz de realizar predicciones o clasificaciones y solo se podrá equiparar a los modelos que implementen regresiones.

En mi opinión es allí donde un sistema de Inteligencia Artificial tiene que aportar sus cualidades. Para calcular dimensiones el sistema de inteligencia Artificial podrá ser más rápido, pero nunca más preciso.

El siguiente objetivo era el de poder acceder a fuentes de datos que nos permitieran entrenar y probar nuestro sistema. Como ha quedado demostrado, la oferta disponible es completa y variada, y lo es más si consideramos que solamente hemos analizado fuentes de datos de acceso rápido y fácil. Las opciones se multiplican con un adecuado registro y solicitud de los datos.

A pesar de nuestro precario acceso, hemos tenido disponibles miles de imágenes de diferentes sujetos con diferentes tipologías y diferentes patologías. Algunas fuentes nos han dado incluso los datos estadísticos y métricas necesarios para nuestros procesos de entrenamiento y pruebas los que ha conseguido reducir significativamente el tiempo de este proyecto.

Imaginemos por un momento que hubiéramos tenido que procesar con FreeSurfer las cerca de 100 imágenes que hemos utilizado en nuestros experimentos, considerando que cada una requiere 12 horas de proceso hubiéramos necesitado 50 días solo para preparar los datos de prueba, y eso sin contar el tiempo de recopilación y formateado de los datos.

Resaltar también en este punto la gran idoneidad del formato Nifti. Es un formato ofrecido por la mayoría de las fuentes de datos. El formato no solo permite la compresión, lo que reduce en gran modo las necesidades de almacenaje, sino que da acceso de forma muy directa y sencilla a la información multidimensional. Hay que agradecer también en este punto la librería de Python Nibabel (http://nipy.org/nibabel/) que proporciona todas las herramientas para poder manipular de forma sencilla estos ficheros.

El último objetivo era la selección y uso de un sistema de Inteligencia Artificial que pudiera dar solución a nuestro problema. En este punto Keras se ha mostrado como una herramienta perfecta para principiantes. Ofuscando todo el complejo proceso matemático permite al usuario centrarse en los conceptos de Inteligencia Artificial, no solo dando todas las facilidades para modelar, entrenar y utilizar un modelo sino también proporcionando herramientas adicionales de documentación y monitorización.

TensorFlow ha resultado más complicado, sobre todo a nivel de instalación y adecuación del entorno. Cuando lo hemos utilizado en un sistema sin GPU, no hemos podido utilizar todas las posibilidades de la CPU ya que esto requiere una compilación de la librería que se escapa completamente de nuestros conocimientos y posibilidades. Cuando hemos tenido acceso a un sistema con GPU, los requerimientos de librerías con versiones antiguas y la necesidad de una GPU con mayores capacidades que la disponible nos ha obligado a descartar la opción de utilizar la GPU y volver a la versión sin GPU no optimizada.

La consecuencia de todo esto ha sido un entorno ineficiente con gran lentitud en los cálculos lo que ha dilatado considerablemente cualquier proceso de entrenamiento obligándonos a adecuar nuestras expectativas a esos largos periodos de proceso.

La disponibilidad de una versión ye compilada que utilice las ultimas mejoras de los procesadores modernos o la posibilidad de utilizar GPUs más pequeñas nos hubiera resultado de gran ayuda y nos hubiera permitido llegar más lejos en nuestros experimentos.

V. Posibilidades de evolución del sistema

Evidentemente el primer paso en la evolución del sistema es el entrenamiento más avanzado del último de los modelos, para poder saber hasta dónde puede llegar y realmente comprobar el nivel de precisión que pueden tener los resultados. Para ello vamos a necesitar más imágenes para añadir a nuestro juego de imágenes de entrenamiento e incrementar el número de iteraciones del entrenamiento.

Sin embargo, un incremento en las pruebas traerá consigo un gran incremento en el tiempo de entrenamiento. Esto hará necesario una mejora de rendimiento del sistema, incrementando las capacidades del sistema, añadiendo potentes GPUs y optimizando la compilación de TensorFlow para aprovechar todas las capacidades de los procesadores.

Una vez obtenido un modelo que nos dé la fiabilidad de resultados deseados, será muy sencillo generar otros modelos para obtener otras medidas utilizando el mismo modelo y las mismas imágenes de entrenamiento. Solo será necesario dirigir nuestros scripts a otra columna del CSV.

Otra opción para mejorar nuestros resultados sería limitar la zona del cerebro que estamos analizando. Actualmente estamos analizando el cerebro completo cuando por ejemplo solo queremos calcular un volumen del lado izquierdo. La reducción de la zona de análisis aumentaría la precisión de los resultados y reduciría el tiempo de entrenamiento y cálculo.

Una vez nuestros modelos sean capaces de calcular todas las medidas que necesitamos, será el momento de modificar los objetivos de este proyecto para pasar de modelos regresivos a modelos basados en clasificaciones. Será en este momento cuando dejaremos de buscar acelerar los mismos resultados que FreeSurfer y comenzaremos a obtener resultados que FreeSurfer nunca podrá obtener.

Estos resultados permitirán al sistema realizar predicciones medicas con el objetivo de detectar y prever posibles enfermedades o problemas futuros. Este sistema podrá ser alimentado constantemente con nuevas imágenes, nuevos diagnósticos aumentando su capacidad día tras día.

6. Glosario

Bioinformática: aplicación de tecnologías computacionales y la estadística a la gestión y análisis de datos biológicos.

MRI: técnica no invasiva que utiliza el fenómeno de la resonancia magnética nuclear para obtener información sobre la estructura y composición del cuerpo a analizar. Esta información es procesada por ordenadores y transformada en imágenes del interior de lo que se ha analizado.

Diagnóstico por imagen: constituye una prueba complementaria en el dictamen por médicos especialistas, que permite determinar patologías concretas y el alcance de estas.

Región cerebral: El cerebro humano es uno de los órganos más complejos de nuestro cuerpo. Se compone de diversas partes o estructuras que realizan funciones diferentes y trabajan de forma coordinada y unitaria a través de los miles de conexiones que establecen entre ellos y con el resto del cuerpo.

"Machine learning": subcampo de las ciencias de la computación y una rama de la inteligencia artificial cuyo objetivo es desarrollar técnicas que permitan a las computadoras aprender. De forma más concreta, se trata de crear programas capaces de generalizar comportamientos a partir de una información suministrada en forma de ejemplos.

Entrenamiento: proceso en el que se detectan los patrones de un conjunto de datos, es decir, es el corazón del "machine learning". Una vez identificados los patrones, se pueden hacer predicciones con nuevos datos que se incorporen al sistema.

Red neuronal convolucional: tipo de red neuronal artificial donde las neuronas corresponden a campos receptivos de una manera muy similar a las neuronas en la corteza visual primaria (V1) de un cerebro biológico. Este tipo de red es una variación de un perceptron multicapa, sin embargo, debido a que su aplicación es realizada en matrices bidimensionales, son muy efectivas para tareas de visión artificial, como en la clasificación y segmentación de imágenes, entre otras aplicaciones.

FreeSurfer: paquete de software de imágenes cerebrales desarrollado por el Centro de Imágenes Biomédicas Athinoula A. Martinos en el Hospital General de Massachusetts para el análisis de imágenes de resonancia magnética (MRI). Es una herramienta importante en el mapeo cerebral funcional y facilita la visualización de las regiones funcionales de la corteza cerebral altamente plegada

Nifti: Formato de imagen multidimensional utilizado para almacenar el resultado de resonancias magnéticas.

Nibabel: Librería de Python que da acceso a la lectura y escritura de algunos formatos de imagen comunes al ámbito de la neuroimagen.

CSV: Los archivos **CSV** (del inglés comma-separated values) son un tipo de documento en formato abierto para representar datos en forma de tabla, en las que las columnas se separan por comas.

GPU: Unidad de procesamiento gráfico o **GPU** (Graphics Processing Unit) es un coprocesador dedicado al procesamiento de gráficos u operaciones de coma flotante, para aligerar la carga de trabajo del procesador central en aplicaciones como los videojuegos o aplicaciones 3D interactivas.

7. Bibliografía

Imagen por resonancia magnética
https://es.wikipedia.org/wiki/Imagen_por_resonancia_magnética
https://www.nibib.nih.gov/espanol/temas-cientificos/imagen-por-resonancia-magnética-irm

Bioinformática
https://es.wikipedia.org/wiki/Bioinformática

Machine learning
http://cleverdata.io/conceptos-basicos-machine-learning/
https://dzone.com/articles/8-best-deep-learning-frameworks

La importancia del diagnóstico por imagen en la detección de enfermedades
http://www.zonahospitalaria.com/la-importancia-del-diagnostico-por-imagen-en-la-deteccion-de-enfermedades/

Partes del cerebro. Anatomía del cerebro
https://www.cognifit.com/es/partes-del-cerebro
https://www.onmeda.es/anatomia/anatomia_cerebro-estructura-del-cerebro-1478-2.html

Magnetic Resonance Imaging (MRI) – Head
https://www.radiologyinfo.org/en/info.cfm?pg=headmr

Segmentación de imágenes cerebrales de Resonancia Magnética basada en Redes Neuronales de Regresión Generalizada
http://scielo.sld.cu/scielo.php?script=sci_arttext&pid=S1684-18592013000100010

Redes neuronales convolucionales
https://es.wikipedia.org/wiki/Redes_neuronales_convolucionales
http://www.diegocalvo.es/red-neuronal-convolucional-cnn/
https://relopezbriega.github.io/blog/2016/08/02/redes-neuronales-convolucionales-con-tensorflow/

FreeSurfer
https://en.wikipedia.org/wiki/FreeSurfer
https://surfer.nmr.mgh.harvard.edu
https://surfer.nmr.mgh.harvard.edu/fswiki

Importancia de la aplicación de la informática en la medicina
http://www.monografias.com/trabajos98/importancia-aplicacion-informatica-medicina/importancia-aplicacion-informatica-medicina.shtml

Informática en la medicina

https://www.slideshare.net/jennifergalindo7186/informatica-en-la-medicina-41815124

Machine Learning Methods for Magnetic Resonance Imaging Analysis
https://deepblue.lib.umich.edu/bitstream/handle/2027.42/96103/gcen_1.pdf%3Bjsessionid%3D34101270E539D634DFA07B11B89651B0?sequence%3D1

La inteligencia artificial ya diagnostica enfermedades tan bien como los médicos
http://www.lavanguardia.com/ciencia/cuerpo-humano/20180223/44950677766/inteligencia-artificial-machine-learning-diagnosticar-enfermedades-medicos-eficiencia.html

Neuroimaging Informatics Technology Initiative
https://nifti.nimh.nih.gov

Primeros pasos en Keras
http://jorditorres.org/primeros-pasos-en-keras/

El Formato Nifti
https://brainder.org/2012/09/23/the-nifti-file-format/
https://nifti.nimh.nih.gov

GPU
https://es.wikipedia.org/wiki/Unidad_de_procesamiento_gráfico

8. Anexos

Descripción del hardware utilizado

Entrenamiento Modelos 1 y 2	Lenovo Yoga 2 PRO Procesador Intel Core i7 – 4510U CPU @ 2.00 GHz 2.60 GHz Memoria Ram 8 GB Sistema operativo Windows 10 64 bits
Entrenamiento Modelos 3 y 4 Pruebas	Lenovo Ideapad 720S-14IKB Procesador Intel Core i7 – 8550U CPU @ 1.80 GHz 2.00 GHz Memoria Ram 16 GB Sistema operativo Windows 10 64 bits

Muestra del CSV de resultados

Disponemos de un fichero CSV con todos los resultados del análisis de FreeSurfer para cada una de las resonancias magnéticas utilizadas para el entrenamiento y la evaluación.

La primera columna nos sirve para relacionar la fila del CSV con la prueba. El título de la columna nos permite decidir cuál es la medida de todas las posibles que queremos aplicar a nuestro modelo.

Como se puede ver en el script ModelDefinition.py, basta con cambiar la línea:

resultData.append(resultDictionary[subject]['FS_LCort_GM_Vol'])

para preparar generar un nuevo modelo con otra medida objetivo.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Subject	Gender	Release	FS_InterCr	FS_BrainS	FS_BrainS	FS_BrainS	FS_LCort_	FS_RCort_	FS_TotCor	FS_SubCo	FS_Total_(	FS_SupraT	FS_SupraT	FS_Su
2	100206	M	S900	1864518	1406924	1389007	1386836	304488	305634	610122	70240	807245	1239692	1225550	1225
3	100307	F	Q1	1512540	1109710	1098854	1097122	240030	243207	483238	56623	664124	957615	948872	948
4	100408	M	Q3	1684117	1280171	1260943	1259250	260695	265243	525938	63869	726206	1106881	1092248	1091
5	100610	M	S900	1816044	1321362	1306929	1304569	276506	278002	554508	74869	762308	1155067	1144617	1144
6	101006	F	S500	1437868	1053766	1038408	1037793	203718	210539	414257	58813	579632	916973	905210	903
7	101107	M	S500	1512727	1126655	1111763	1109747	242489	248807	491296	62092	665024	984952	972407	972
8	101309	M	S500	1540433	1239152	1230309	1228191	264768	270079	534847	59937	707674	1094546	1088148	1087
9	101410	M	S500	1824028	1328259	1307792	1306186	287013	299163	586176	62680	773472	1170042	1152443	1152
10	101915	F	Q3	1485035	1122818	1111376	1109464	247478	250042	497520	57557	676282	971287	962390	962
11	102008	M	S500	1675155	1222352	1205315	1203460	263712	274157	537869	61301	717215	1073196	1058855	1058
12	102109	M	S1200	1737018	1377371	1364617	1362497	293878	301370	595247	67152	784458	1216301	1206362	1205
13	102311	F	S500	1548451	1131734	1117529	1116008	258346	266220	524566	57682	690232	994481	983504	983
14	102513	M	S900	1737905	1350812	1339326	1337255	296722	310075	606797	66785	780767	1213131	1204817	1204
15	102614	M	S1200	1797372	1299066	1285069	1282984	284278	291130	575408	64397	772912	1130442	1120257	1120
16	102715	M	S1200	1878161	1422467	1395675	1393684	297676	306506	604182	68064	795282	1265335	1243708	1243
17	102816	F	Q3	1412821	1082089	1067703	1065492	226369	233817	460186	57502	609326	960625	948696	949
18	103010	M	S1200	1706654	1243204	1226639	1224952	266798	274607	541406	60361	726045	1084328	1072416	1071
19	103111	M	S500	1809464	1356292	1318005	1316523	279462	284325	563787	71466	771666	1181633	1146991	1145
20	103212	M	S1200	1578367	1240228	1213521	1211486	275111	280828	555939	62244	734371	1092274	1069856	1069
21	103414	F	Q2	1440750	1060512	1048224	1046786	233310	234625	467936	56970	628040	927334	917491	917

ResampleTests.py

```
import glob
import os
import nibabel as nib
from nibabel.processing import resample_from_to

trainingElements = glob.glob("<<Path de ficheros de entrenamiento>>\\*.nii.gz")
base_img = nib.load('<<Path de imagen referencia>>')


for trainingElement in trainingElements:
    n1_img = nib.load(trainingElement)
    new_img = nib.processing.resample_from_to(n1_img, base_img)
    subject = os.path.basename(trainingElement)[:6]
    nib.save(new_img, '<<Path de ficheros de
entrenamiento>>/'+subject+'_3T.nii.gz')
```

ModelDefinition.py

```
import os
import nibabel as nib
import numpy as np
import glob
import csv

from keras.models import Sequential
from keras.layers import Conv3D,Activation,Flatten,MaxPooling3D,Dense

resultDictionary = {}
```

```
with open('<<Fichero de resultados de imágenes de entrenamiento>>.csv') as csvfile:
    reader = csv.DictReader(csvfile)
    for row in reader:
        resultDictionary[row['Subject']] = row

print(os.path.join(os.path.dirname(__file__), "NIFTI", "Training", "*", "*"))

imageData = []
resultData = []

trainingElements = glob.glob("<<Path de ficheros de entrenamiento>>\\*.nii.gz")


print('Loading training subject images.')
print('')

for trainingElement in trainingElements:
    subject = os.path.basename(trainingElement)[:6]
    print('|', end='')
    n1_img = nib.load(trainingElement)
    imageData.append(n1_img.get_data())
    resultData.append(resultDictionary[subject]['FS_LCort_GM_Vol'])

X_train = np.array(imageData)
X_train = np.expand_dims(X_train, axis=4)

Y_train = np.array(resultData)

model = Sequential()

model.add(Conv3D(32, (8,8,8), strides=4, activation='relu', padding='same',input_shape=(n1_img.shape[0], n1_img.shape[1], n1_img.shape[2],1)))
model.add(MaxPooling3D((2, 2, 2)))
model.add(Conv3D(64, (6,6,6), strides=2, activation='relu', padding='same'))
model.add(MaxPooling3D((2, 2, 2)))
model.add(Conv3D(128, (4,4,4), strides=1, activation='relu', padding='same'))
model.add(MaxPooling3D((2, 2, 2)))
model.add(Flatten())
model.add(Dense(1,kernel_initializer='normal'))

epochs = 25

model.compile(loss='mean_squared_error',
              optimizer='adam',
              metrics=['accuracy']) # reporting the accuracy
print('')
print(model.summary())
```

```
print('')
print(Y_train.shape)
print('')

model.fit(X_train,
          Y_train,
          batch_size=1,
          epochs=epochs,
          verbose=1,
          validation_split=0.1)

model.save('nifti_modelX.h5')

n1_img = nib.load("<<imagen de test>>")
predictionData = []
predictionData.append(n1_img.get_data())
X_predict = np.array(predictionData)
X_predict = np.expand_dims(X_predict, axis=4)

prediction = model.predict(X_predict)
print(prediction)
```

TestModelDefinition.py

```
import os
import nibabel as nib
import numpy as np
import glob
import csv

from keras.models import load_model

resultDictionary = {}

with open(' \\unrestricted_hcp_freesurfer.csv') as csvfile:
    reader = csv.DictReader(csvfile)
    for row in reader:
        resultDictionary[row['Subject']] = row

testingElements = glob.glob("<<Path de ficheros de test>>\\*.nii.gz")

model = load_model('<<model_a_testear>>')

for testingElement in testingElements:
    subject = os.path.basename(testingElement)[:6]
    n1_img = nib.load(testingElement)
    predictionData = []
```

```
    predictionData.append(n1_img.get_data())
    X_predict = np.array(predictionData)
    X_predict = np.expand_dims(X_predict, axis=4)
    prediction = model.predict(X_predict)
    print(subject+' - '+str(prediction[0][0])+' -
'+str(resultDictionary[subject]['FS_LCort_GM_Vol']))
```