UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Máster en Inteligencia Artificial, Reconocimiento de Formas e Imagen Digital

Master en Inteligencia artificial, reconocimiento de formas e imagen digital

Universitat Politècnica de València

Análisis poblacional de patología de Alzheimer a partir de imágenes de resonancia magnética

TRABAJO FIN DE MÁSTER

Curso 2018/2019

Autor:

Jose Manuel Saborit Torres

Tutor: Jon Ander Gómez Adrián

Tutor Externo: Maria de los Desamparados de la Iglesia Vayá

Agradecimientos

En primer lugar, me gustaría agradecer a Jon Ander Gomez Adrián y María de la Iglesia Vayá por su apoyo incondicional y su implicación durante el desarrollo del trabajo.

A mis compañeros de trabajo Jhon Jairo Saenz Gamboa y Joaquim Ángel Montell Serrano por la paciencia que han tenido conmigo durante estos dos últimos años de trabajo, pero también por los buenos momentos.

También a Alvaro López Chillet, Rafael Vicente Sánchez Romero y Silvia Nadal Almela por su ayuda en esta etapa y por los almuerzos de "Bocadillos Sorpresa"mientras hablábamos de los resultados obtenidos.

A mi familia, por apoyarme desde Burriana en los buenos y malos momentos de la vida, por los mensajes diciendo “¿Qué es de tu vida?” y también por poner altares con santos y velas para mi buenaventura.

A mis amigos, que me ayudaron de una manera desinteresada, gracias infinitas por toda su ayuda y buena voluntad.

Finalmente, dar las gracias a NVIDIA por la tarjeta gráfica QUADRO P5000 donada al CEIB . Y al CIPF por el uso del cluster HPC con el que también hemos podido trabajar.

Resumen

El problema que se aborda en este trabajo es el de usar técnicas de aprendizaje profundo para etiquetar imágenes de resonancia magnética cerebral para la Enfermedad del Alzheimer. Es importante detectar las primeras fases del Alzheimer ya que es una enfermedad sin cura por el momento y lo único que se puede hacer es diagnosticarla lo antes posible para poder disminuir al máximo su impacto a largo plazo mediante algunos tratamientos.

Para ello, primero se estudiarán las herramientas y técnicas de preprocesado utilizadas actualmente para sacar el mayor partido a las imágenes. Seguidamente se revisarán los modelos y estrategias seguidos por el estado del arte. Para finalmente proponer algunas topologías de redes neuronales nuevas que aborden el problema desde un nuevo punto de vista.

Palabras clave: Alzheimer, aprendizaje profundo, redes neuronales, resonancia magnética cerebral

---

Abstract

The problem addressed in this work is to use deep learning techniques to label brain magnetic resonance images in Alzheimer disease. It is important to detect the first phases of Alzheimer since it is a disease without cure at the moment and the only thing that can be done is to diagnose it as soon as possible in order to minimize its long-term impact through some treatments.

For this purpose, first, the preprocessing tools and techniques currently used will be studied to get the most out of the images. Then we will review the models and strategies followed by the state of the art. To finally propose some new neural network topologies that approach the problem from a new point of view.

Keywords: Alzheimer, deep learning, neural network, brain magnetic resonance

Índice general

Índice de figuras

Índice de tablas

Abreviaturas

ADNI: Alzheimer Disease Neuroimaging Initiative.
AROC: Area under the ROC curve .

BOLD: blood oxygen level dependent.

CN: Cognitivamente Normal (Cognitive Normal).
CNN: Convolutional Neural Network.

DNN: Red Neurona Profunda (Deep Neural Network).
DTI: Diffusion Tensor Imaging.

EA (AD): Enfermedad del Alzheimer (Alzheimer Disease).
EMCI: Deterioro cognitivo leve temprano (Earlier Mild Cognitive Impairment).

FLAIR: Fluid-attenuated inversion recovery.
FN: Falsos negativos.

GPU: Unidad de procesamiento gráfico (Graphics Processing Unit).

IA: Inteligencia Artificial.
IRM: Imagen por Resonancia Magnética.

LCE (CSF): Líquido Cerebroespinal (cerebrospinal fluid).
LMCI: Latest Mild Cognitive Impairment (Deterioro cognitivo leve Tardío).

MB (WM): Sustancia Blanca (White Matter).
MCI: Deterioro cognitivo leve (Mild Cognitive Impairment).

MG (GM): Sustancia Gris (Grey Matter).

NB: Naive Bayes.

NP: Falsos Positivos.

OASIS: Alzheimer Disease Neuroimaging Initiative.

OMS: Organización Mundial de la Salud.

PET: Tomografía por Emisión de Positrones.

RAM: Random Access Memory.

RM: Resonancia Magnética.

ROC: Receiver Operating Characteristic.

SVC: Support Vector Machine.

SWI: Susceptibility Weighted Imaging.

TAC: Tomografía Axial Computada.

VN: Verdaderos Negativos.

VP: Verdaderos Positivos.

CAPÍTULO 1

Introducción

1.1 Presentación de la enfermedad de Alzheimer.

La enfermedad de Alzheimer (EA o AD Alzheimer Disease en inglés) es una afección neurodegenerativa que se manifiesta como deterioro cognitivo (es decir, la capacidad para procesar el pensamiento) y trastornos conductuales más allá de lo que sería un envejecimiento cognitivo normal. Una de sus características más frecuentes es una pérdida de memoria inmediata debido a la muerte de las células nerviosas y, en consecuencia, la atrofia en ciertas zonas/áreas del cerebro. El primer síntoma que suele presentarse es la incapacidad de almacenar nuevos recuerdos pero suele confundirse con actitudes propias de la vejez y, por tanto, es difícil de diagnosticar sus posibles manifestaciones cognitivo-conductuales en etapas tempranas de la enfermedad.

La enfermedad de Alzheimer, es la causa más común de demencia entre los adultos mayores, convirtiéndose en un trastorno cerebral progresivo e irreversible que mata las células cerebrales, destruye gradualmente la memoria y finalmente afecta

al pensamiento, el comportamiento y la capacidad de llevar a cabo las tareas diarias de la vida.

Según las investigaciones, como el Estudio Alfa de la Fundación Pasqual Maragall [1, 2], antes de dar un diagnóstico de Alzheimer al paciente, se deben descartar otras posibles causas o enfermedades que puedan afectar al sistema cognitivo tales como procesos infecciosos, problemas vasculares cerebrales, trastornos del estado de ánimo, o incluso, los efectos secundarios de algún medicamento. Para descartar y diagnosticar correctamente se deben realizar las siguientes pruebas:

- **Análisis de Sangre:** Se usa para descarte de otras enfermedades.
- **Exploración neuropsicológica:** Se realizará una *anamnesis*, que es construir el relato de los síntomas iniciales, cuándo empezaron a aparecer y cuál ha sido su evolución para precisar las características y el alcance de la alteración cognitiva - conductual y de su posible impacto en la vida cotidiana. Debe ir acompañado de un familiar o persona próxima que la conozca bien.
- **Pruebas de neuroimagen:** Usados también para descartar enfermedades que puedan afectar a las alteraciones cognitivas. También para encontrar indicios de neurodegeneración, atrofia o lesiones neuronales. Estas pruebas son:
  - Resonancia Magnética Nuclear (RMN).
  - Tomografia Axial Computada (TAC).
  - Tomografía por Emisión de Positrones (PET, menos común debido a su alto coste).
- **Pruebas genéticas:** Se recomiendan en casos de inicio muy precoz y con antecedentes familiares, ya que la probabilidad de que sea hereditario es $> 1\,\%$ de los casos detectados.

- **Análisis de líquido cefalorraquídeo:** se extrae mediante una punción lumbar, se trata de una prueba cada vez más habitual en el ámbito de la investigación y los ensayos clínicos. Los resultados en niveles de proteínas como la proteína tau o la β - amiloide, orientan al facultativo en el diagnóstico de Alzheimer.

Aunque tradicionalmente, para diagnosticar la EA se parte del examen exhaustivo de la historia clínica del paciente, los criterios de definición de la enfermedad incluyen el estudio del déficit cognitivo que se realiza a través de la historia clínica detallada y de la aplicación de diversos test neuropsicológicos. Por otra parte, también resulta imprescindible descartar otras afecciones neurológicas que conllevan un deterioro cognitivo progresivo, como el producido por la enfermedad cerebrovascular, la enfermedad de Parkinson, tumores cerebrales, etc. **Es en este punto donde el diagnóstico por imagen tiene un valor insustituible.**

En general la imagen PET, se utilizan como biomarcador e indicación clínica en el estudio de las demencias, especialmente en la Enfermedad de Alzheimer. El PET aportan información acerca del volumen y flujo sanguíneo cerebral y del metabolismo glicolítico del córtex así como de las estructuras subcorticales. Además la técnica de PET permite obtener un patrón bastante específico de la enfermedad de Alzheimer. Actualmente se conoce que el deterioro cognitivo en la enfermedad de Alzheimer está producido por la acumulación de unos péptidos tóxicos en el cerebro que producen la degeneración neurofibrilar y la pérdida de las sinapsis. Mediante la PET es posible visualizar estos péptidos y obtener un mapa de su distribución cerebral, lo que sin duda contribuirá al diagnóstico precoz de la enfermedad.

La PET es una técnica más costosa que la Imagen de Resonancia Magnética, aun así y aunque las ventajas que nos ofrece el PET son evidentes, los costes de esta técnica son elevados con respecto a los de IMR. En este TFM queremos contribuir en la búsqueda de biomarcadores que detecten la enfermedad de Alzheimer con suficien-

te antelación y con costes asumibles para la Sanidad Pública, aplicando técnicas de Inteligencia Artificial.

1.2 Epidemiología

La EA es, además, una de las enfermedades de mayor impacto social y económico para los sistemas de salud, grupos familiares y los mismos pacientes, pues afecta a su funcionalidad y disminuye su calidad de vida. Es conocido que la principal barrera para implementar terapias y por lo tanto reducir los costos de la enfermedad, es la dificultad que tiene el personal médico en los servicios de atención primaria y especializada en detectar prematuramente la EA.

La prevalencia en demencia, de acuerdo con un metaanálisis de 157 estudios epidemiológicos realizados entre 1980 y 2009 a nivel mundial se sitúa, dependiendo de la zona geográfica, entre el 5-8 % para personas mayores de 60 años, y muestra un patrón de crecimiento exponencial con la edad, duplicándose cada cinco años aproximadamente, con un predominio en el sexo femenino [3] (Fig. 1.1).

En España, estudios poblacionales recientes indican prevalencias que oscilan entre el $[8,5 - 9,4]$ % [4, 5] en los mayores de 70 años, y entre el $[5,5 - 5,8]$ % [6, 7] en los mayores de 65 años.

Todos los estudios sobre la incidencia de demencia y de EA coinciden en particular en mostrar que se trata de una patología que tiene una incidencia mayor con el aumento de la edad y que presenta un patrón de crecimiento exponencial a partir de los 65 años (Fig. 2). La incidencia acumulada del parámetro de la demencia estimada para mayores de 60 años, de acuerdo con un metaanálisis es de 52,8 por 1.000, y la tasa de incidencia, del 17,1 por 1.000 personas/año [8].

Figura 1.1: Estimación de la prevalencia de demencia en mayores de 60 años [3].

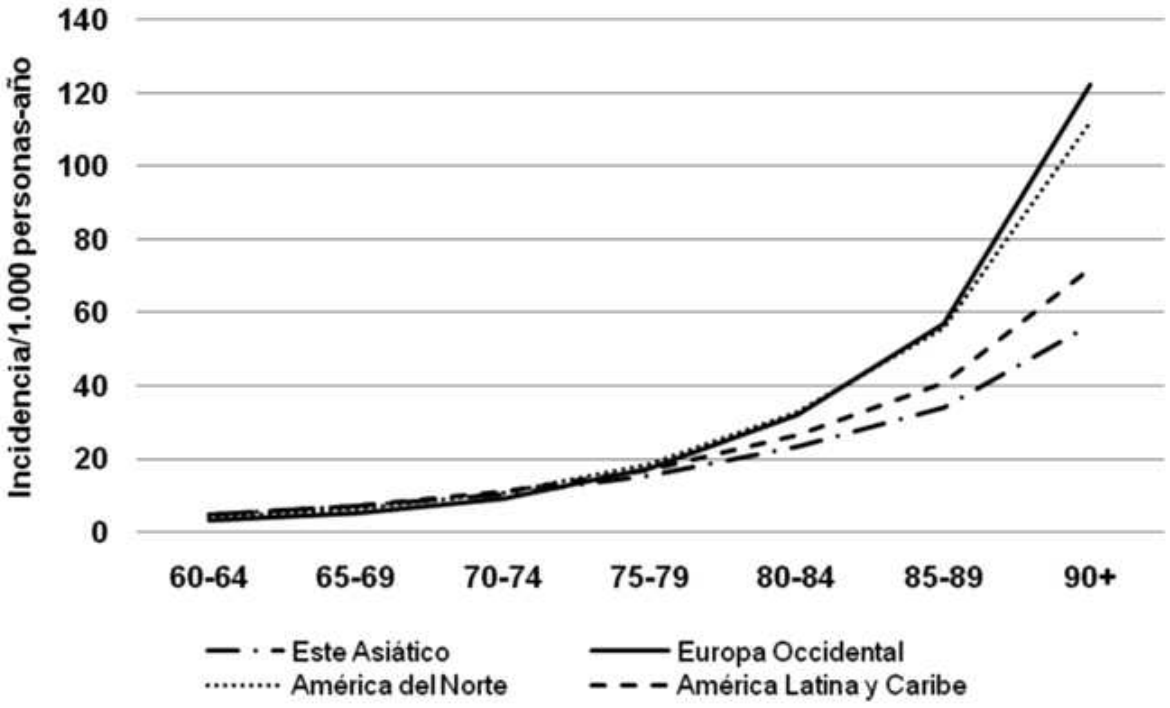


Figura 1.2: Estimación de la incidencia de demencia en mayores de 60 años (adaptado de Alzheimer's Disease International 2015 [9]

Las tasas de mortalidad y supervivencia tanto globales como específicas son indicadores epidemiológicos muy relevantes desde una perspectiva de salud pública. Para monitorizar la salud de la población, describiendo las consecuencias negativas de la enfermedad a lo largo del tiempo, se debe contribuir en la planificación de los recursos sanitarios. De acuerdo con los resultados del *Global Burden Disase Study*, la EA y otro tipo de demencias, conforman un grupo de enfermedades que durante el período 1990-2013, escaló un mayor número de posiciones dentro del ranking entre las 50 principales causas de mortalidad [10]

Según los datos de la Organización Mundial de la Salud (OMS) [11], actualmente, el Alzheimer es la causa más común en los casos de demencia, acaparando entre un [60-70] % de los casos, actualmente sin cura y con un gran impacto en la sociedad. Uno de cada diez personas mayores de 65 años cursan EA y en los casos graves necesitan de un cuidador a su cargo, donde el 80 % son familiares que reciben poca retribución monetaria o ninguna y una gran carga emocional negativa ante la impotencia de la evolución de la enfermedad, y con el agravante de una dependencia total del cuidador asignado. Los datos relativos al crecimiento de la incidencia de EA en la población, no son nada alentadores, El gráfico representa los biomarcadores como indicadores de AD. Las curvas indican cambios en cinco biomarcadores de normal a anormal en el transcurso de la EA (cognición normal a la demencia)e estima que la demencia afecta hoy en día a nivel mundial a unos 50 millones de personas y se prevé que el número total de personas con demencia alcance los 82 millones en 2030 y 152 millones en 2050.

1.3 Estudio de la patológica de Alzheimer

continuación vamos a explicar cómo se estudia la EA a nivel de neuroimagen estructural que es la técnica que hemos escogido para realizar este trabajo final de máster. Para ello hemos utilizado las técnicas de Imagen de Resonancia Magnética (IRM)

para cuantificar volumetrías en estructuras cerebrales utilizando los cerebros de los estudios epidemiológicos de proyectos internacionales muy conocidos en el estado del arte (ADNI, OASIS). En cuanto a las etapas de la enfermedad Alzheimer, hay que destacar los tres estadíos que determinan su progreso:

En el Alzheimer hay 3 estadios de la enfermedad:

- **Etapa temprana:** A menudo pasa desapercibida, ya que el inicio es paulatino y de larga duración
  - Tendencia al olvido.
  - Pérdida de la noción del tiempo.
  - Desubicación espacial, incluso en lugares conocidos.
- **Etapa intermedia:** Los signos y síntomas se vuelven cada vez más evidentes y más limitadores. En esta etapa las personas afectadas se caracterizan porque:
  - Empiezan a olvidar acontecimientos recientes, así como los nombres de las personas.
  - Se encuentran desubicadas en su propio hogar.
  - Tienen cada vez más dificultades para comunicarse.
  - Empiezan a necesitar ayuda con el aseo y cuidado personal.
  - Sufren cambios de comportamiento, como por ejemplo, dan vueltas por la casa o repiten las mismas preguntas.

- **Etapa tardía:** La dependencia y la inactividad son casi totales. Las alteraciones de la memoria son graves y los síntomas y signos físicos se hacen cada vez más evidentes. Entre estos síntomas se incluyen:
  - Una creciente desubicación en el tiempo y en el espacio.
  - Dificultades para reconocer a familiares y amigos.
  - Una necesidad cada vez mayor de ayuda para el cuidado personal.
  - Dificultades para caminar.
  - Alteraciones del comportamiento que pueden exacerbar y desembocar en agresiones.

La EA no es un problema de la vejez en las personas, sino que es una enfermedad que en la actualidad no tiene tratamiento, y lo que es más grave se desconoce todavía su génesis. Por otro lado se puede intentar paliar los efectos que puedan producir en el futuro mediante optimización de la salud física, la cognición, la actividad social y el bienestar. También se puede dar el caso de coexistir con enfermedades concomitantes[1], que se debería detectar si las hubiese para mejorar el estado de la enfermedad.

Llevamos hoy en día más de tres décadas de investigación sobre esta patología tan socialmente estigmatizante y no se han producido avances que impacten en tratamientos eficientes de las personas con el trastorno, según un experto de la UCLA que ha estudiado la bioquímica del cerebro y la enfermedad de Alzheimer durante casi 30 años. *"Nada ha funcionado"*, Steven Clarke, un distinguido profesor de química y bioquímica. *"Estamos listos para incorporar nuevas ideas"*. Ahora, Clarke y sus colaboradores de la Universidad de UCLA, han informado sobre nuevas ideas

[1]Que acompaña a una cosa o actúa junto a ella.

que pueden conducir al progreso en la lucha contra la devastadora enfermedad [12].

Según el *Alzheimer Disease Neuroimaging Initiative* (ADNI) [13], se han clasificado 5 biomarcadores que nos pueden indicar si un paciente sufre la enfermedad o no. Estos son los siguientes:

1. β-amiloide medido por punción extrayendo el líquido cefalorraquídeo o mediante imágenes de PET amiloide.

2. Neurodegeneración indicada por la proteína tau medida en líquido cefalorraquídeo, o por disfunción sináptica, medida por PET.

3. Atrofia cerebral, principalmente en el lóbulo temporal medial, medido por resonancia magnética estructural.

4. Pérdida de memoria, medida por pruebas cognitivas.

5. Función clínica, indicada por el deterioro cognitivo general medido por pruebas cognitivas.

Los indicadores del uno al tres, son biomarcadores que pueden observarse antes de un diagnóstico de demencia, mientras que los cambios observados en los puntos cuatro y cinco son los indicadores clásicos del diagnóstico de demencia.

De estos biomarcadores, hoy en día se ha decantado por el uso de los relacionados con pruebas de IRM (MRI por sus siglas en inglés Magnetic Resonance Imaging) para clasificar el problema planteado (figura 1.4) y que presentamos con más detalle en el próxima sección.

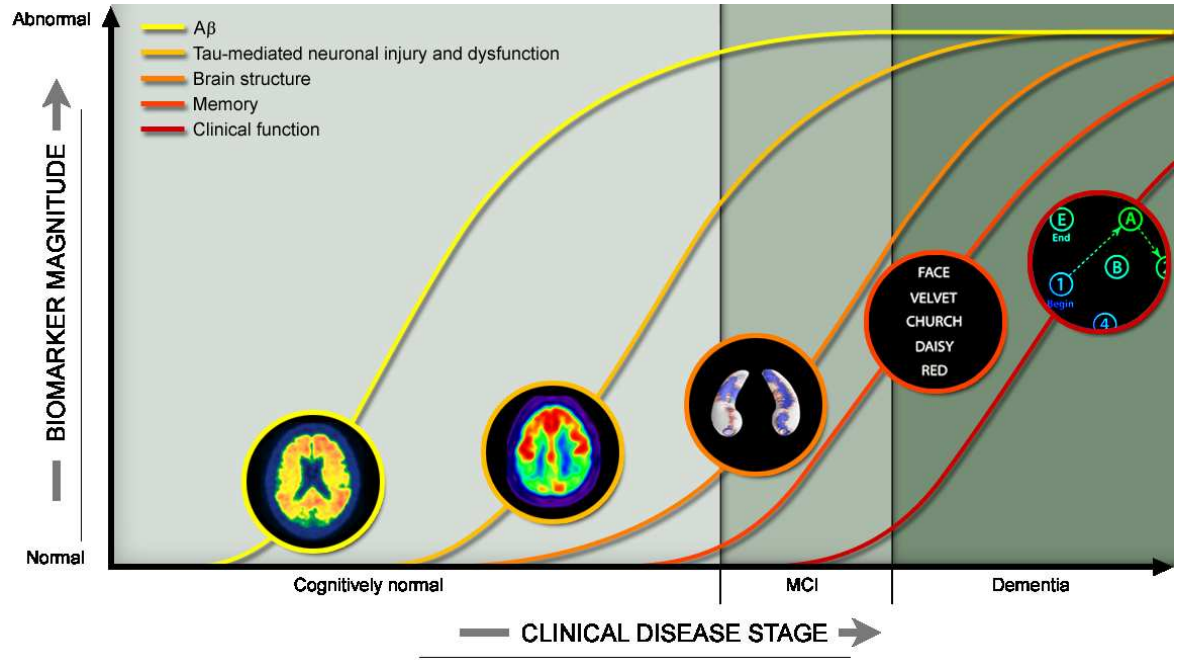


Figura 1.3: Un biomarcador, o marcador biológico, es una sustancia, medida o indicador de un estado biológico. Los biomarcadores pueden existir antes de que surjan los síntomas clínicos. ADNI utiliza y define varios biomarcadores para ayudar a predecir la aparición de la enfermedad de Alzheimer. En el gráfico se representa los biomarcadores como indicadores de la EA. Las curvas indican cambios en cinco biomarcadores de normal a patológico en el transcurso de la EA (cognición normal a la demencia) [13]

Figura 1.4: Maquina de resonancia magnética.

1.4 La Imagen de Resonancia Magnética Nuclear (IRM)

En este apartado se pretende introducir los fundamentos básicos de la imagen por resonancia magnética (MRI), una técnica fundamental para el estudio del EA y otros trastornos relacionados con la demencia. Estas técnicas nos permiten estudiar anatómicamente las distintas áreas y estructuras cerebrales de forma no invasiva.

La IRM es un tipo de imagen médica que permite ver el interior de una persona a partir de un fenómeno mediante el cual los átomos pueden absorber o emitir energía al ser excitados por señales de radiofrecuencia si están en el interior de un campo magnético intenso. De esta forma, por contraste, se pueden crear imágenes del interior del cuerpo denominados "cortes". Cada corte es una imagen 2D de una parte del cuerpo. Si se obtiene muchos cortes de secciones muy juntas entre ellas, se obtiene un volumen 3D de la parte del cuerpo que se está estudiando.

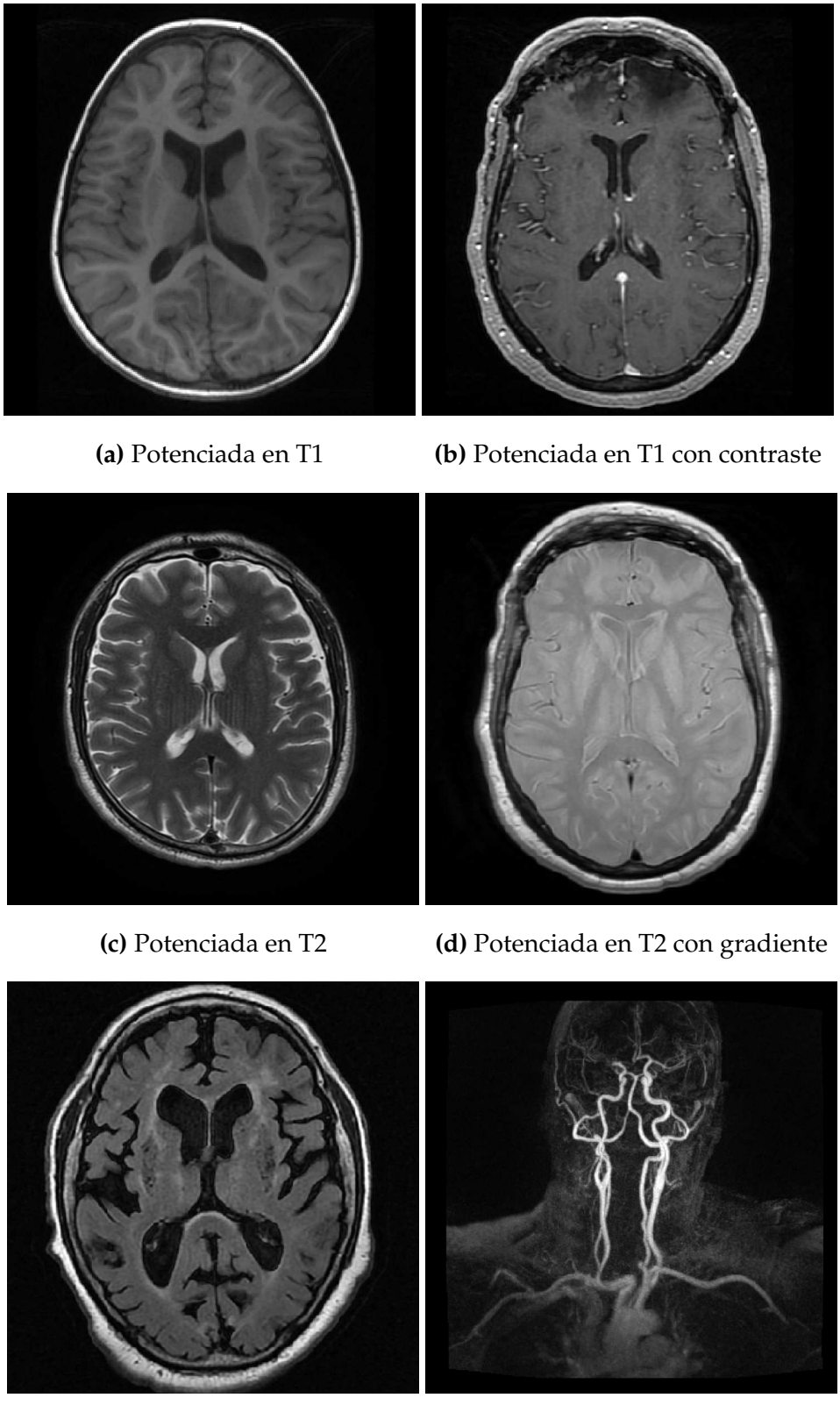
Esta herramienta es muy versátil debido a la existencia de diferentes tipos de configuraciones que se pueden usar para obtener distintos tipos de imágenes que identifican ciertas estructuras. Algunas de ellas se pueden observar en la figura 1.5 donde estas imágenes tiene diferentes objetivos a la hora de diagnosticar

Para el caso de estudio, las imágenes potenciadas en T1 (figura 1.5a) se trata de un tipo de imagen en donde la estructura se observa con gran precisión, en cuanto a la distinción de los tres principales tejidos que componen la estructura cerebral:

- **Líquido Cerebroespinal (LCE o CSF cerebrospinal fluid en ingles):** líquido incoloro que rodea el cerebro y ciertas partes en su interior y que tiene una función de regulación y protección y sustentación dentro de la cavidad craneal, entre otros.

- **Materia Gris (SG o GM Grey Matter en ingles):** zona grisácea del sistema nervioso central formada por somas neuronales y axones sin mielina. En el cerebro corresponde a la superficie de la misma formando la denominada corteza cerebral. Esta sustancia se asocia con la función del procesamiento de la información, razonamiento y la memoria.

- **Materia Blanca (SB o WM White Matter en ingles):** Zona del sistema nervioso central compuesta por fibras nerviosas mielinizadas (cada fibra contiene muchos axones) que se sitúa en el interior de la corteza cerebral. Esta parte modula la distribución de los potenciales de acción, actuando como un retransmisor y coordinando la comunicación entre las diferentes regiones del cerebro.



(a) Potenciada en T1

(b) Potenciada en T1 con contraste

(c) Potenciada en T2

(d) Potenciada en T2 con gradiente

(e) Fluid-attenuated inversion recovery (FLAIR)

(f) Angiografía

Figura 1.5: Algunos ejemplos de imágenes que se pueden obtener en las resonancias magnéticas.

El fundamento físico en el que se basa la técnica es la resonancia magnética nuclear, fue descrito en 1946 simultáneamente por Felix Block en Stanford y por Edward Purcell en Harvard, ambos ganaron el premio Nobel de Física en 1952. Durante las décadas posteriores se fueron desarrollando las aplicaciones para el análisis físico-químico de las moléculas y en 1971 Raymond Damadian mostraba cómo los tiempos de relajación nuclear magnéticos de tejidos y tumores, eran distintos, lo que resultaba útil a la hora del diagnóstico de enfermedades. En la década de los 70 se obtuvieron las primeras imágenes de MRI con una aplicación biosanitaria. Esta técnica tiene numerosas ventajas ya que puede atravesar el hueso sin distorsionar la señal, al utilizar radiación no ionizante haciéndola inocua al paciente (siempre que no tenga marcapasos u otras prótesis que contenga un material magnetizable), permitiendo diferenciar tejidos e imágenes en cualquier plano .[14]

En líneas generales el proceso para obtener imágenes con la resonancia magnética nuclear, viene dado por unos pasos generales:

1. Los núcleos atómicos son expuestos a un campo magnético elevado (B0) y se excitan.

2. Los núcleos al excitarse pasan a un estado de mínima energía o a uno de máxima.

3. Para volver al estado original deben liberar energía, originando el fenómeno de la resonancia nuclear.

4. La energía liberada es detectada por el escáner, generando así las imágenes.

Los átomos y las partículas elementales que los componen, ya sea el núcleo formado dos partículas, por protones (carga positiva), neutrones (sin carga neta), y electrones (con carga negativa). Estas partículas elementales tienen un espín, que es el giro de la partícula sobre su propio eje. El espín al tratarse del giro de una partícu-

la cargada, presenta propiedades magnéticas. Las partículas elementales tienen un momento magnético asociado con el vector de dirección al eje del giro y su sentido. La orientación de este vector es de vital importancia en todo proceso de la resonancia magnética. Cuando el número de protones o de electrones es par, los espines tienen sentidos opuestos y sus efectos magnéticos se anulan y esto es importante en la IMR, por lo que se utilizan átomos que tienen un número másico impar, como el hidrógeno, es el más usado debido a su abundancia en el cuerpo humano, aunque también se pueden utilizar el fósforo, flúor, carbono o el sodio. A partir de ahora cuando se hable de protones se hará referencia al protón del átomo de hidrógeno.

En ausencia de campo magnético (B0 = 0) (Figura 1.6a), los espines de los protones están orientados al azar y el momento magnético resultante al sumar todos los espines es cero. Cuando un campo magnético intenso B0 (Figura 1.6b) actúa sobre los protones, estos tienden a orientarse de tal manera que el vector de dirección sea paralelo al campo magnético y el protón puede adoptar dos orientaciones posibles, orientación paralela o up que es un estado energético bajo o la orientación antiparalela o down y corresponde a un estado energético alto. Como en la muestra hay una mayor cantidad de protones en un estado paralelo, tendremos un momento magnético neto distinto de cero y con una dirección paralela al campo magnético B0. Las partículas pueden pasar de un espín en paralelo a uno antiparalelo si se le somete a una cantidad de energía determinada, en el caso de la resonancia magnética se utiliza un pulso de radiofrecuencia determinado a la partícula excitar, ya que la intensidad necesaria es única para cada partícula. Al emitir una cantidad determinada de pulsos, el número de protones en estado antiparalelo igualará a los paralelos, siendo por tanto el momento magnético igual a cero. Esto es equivalente a la inversión de los espines en 90 grados respecto a la dirección del campo magnético B0, de esta manera se encuentran en paralelo respecto al impulso magnético B1 (Figura 1.6c) y por tanto el momento magnético de la muestra nulo. Una vez que el impulso magnético B1 cesa, los espines vuelven al estado original (Figura 3b) y recuperando

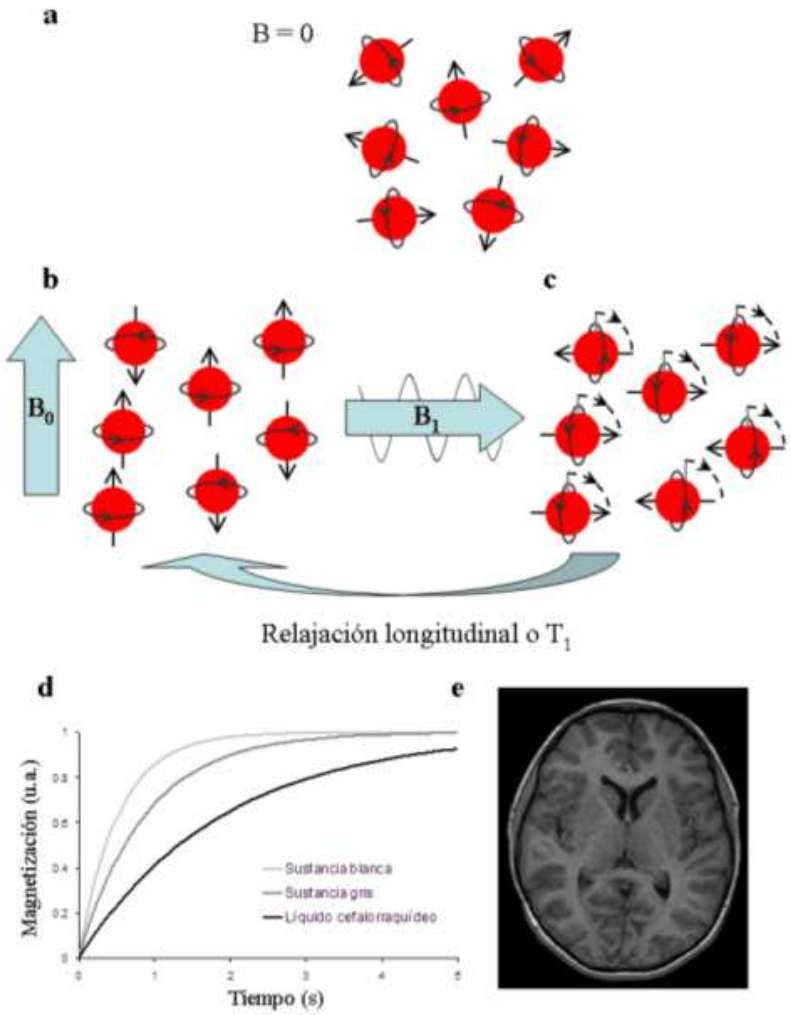


Figura 1.6: (a) Cuando no hay un campo magnético externo, los espines de los protones se orientan de forma aleatoria, siendo la suma vectorial cero por lo que el momento magnético de la muestra es cero. (b) En el momento que incide un campo magnético B0 se produce un reordenamiento de los espines y se orientan de manera paralela al impulso magnético, de manera antiparalela o paralela. (c) Un breve pulso electromagnético B1 perpendicular al B0 hace que los espines de los protones se inclinan al plano perpendicular obteniendo un momento magnético longitudinal total a cero. Cuando el impulso B1 cesa, los protones vuelven gradualmente a su posición original (paralelo al impulso B1), este proceso se denomina relajación longitudinal o T1. (d) Diferentes constantes de tiempo según el tejido. (e) Con los valores de la relajación longitudinal es posible obtener una imagen anatómica o de tipo T1 [15].

su orientación respecto a B0. El tiempo de este proceso se denomina relajación longitudinal, que está asociado con una constante de tiempo y que depende únicamente de las condiciones del medio en donde se encuentran los protones. Por tanto, con la medición (Figura 1.6d) de los T1 podemos diferenciar a los distintos tejidos que estén presentes en una imagen. En el caso del cerebro (Figura 1.6e) se puede diferenciar la sustancia gris, blanca y el líquido cefalorraquídeo, de esta manera es posible crear un imagen del cerebro anatómica o T1. Estas imágenes son tridimensionales y tiene un grosor determinado según el escáner utilizado, normalmente el grosor entre imagen y imagen es de 1-3 mm [15].

1.5 Planteamiento del problema

El problema que se pretende abordar es la creación de un sistema de ayuda a la decisión al diagnóstico que permitiría orientar al médico en la toma de decisiones. Para la implementación de estos procedimientos de entrenamiento de redes neuronales convolucionales y métodos clásicos, vamos a presentar los siguientes conceptos genéricos que nos ayudarán a entender el planteamiento genérico (figura 1.7):

- **Entrada:** En esta parte, la información necesaria (*curation data*) se introduce en el sistema para poder clasificar el problema. Para poder trabajar con IRM cerebral se hace necesario realizar un preprocesado que más adelante describiremos y que nos ayudará a mejorar la fase de entrenamiento de la experimentación en este trabajo.

- **Modelo:** En esta parte, se van a aplicar distintas técnicas de Aprendizaje Profundo (AP o DL *Deep Learning* por sus siglas en inglés) y métodos clásicos, para generar un modelo que sea capaz de preprocesar la información de entrada y pueda emitir un juicio de valor con una probabilidad de precisión estimada y comparable al estado del arte.

- **Salida:** En esta última parte, la información que se ha procesado en la fase anterior ofrecerá una probabilidad de precisión de la estimación del modelo para las diferentes etiquetas que se definieron en el conjunto de entrada.

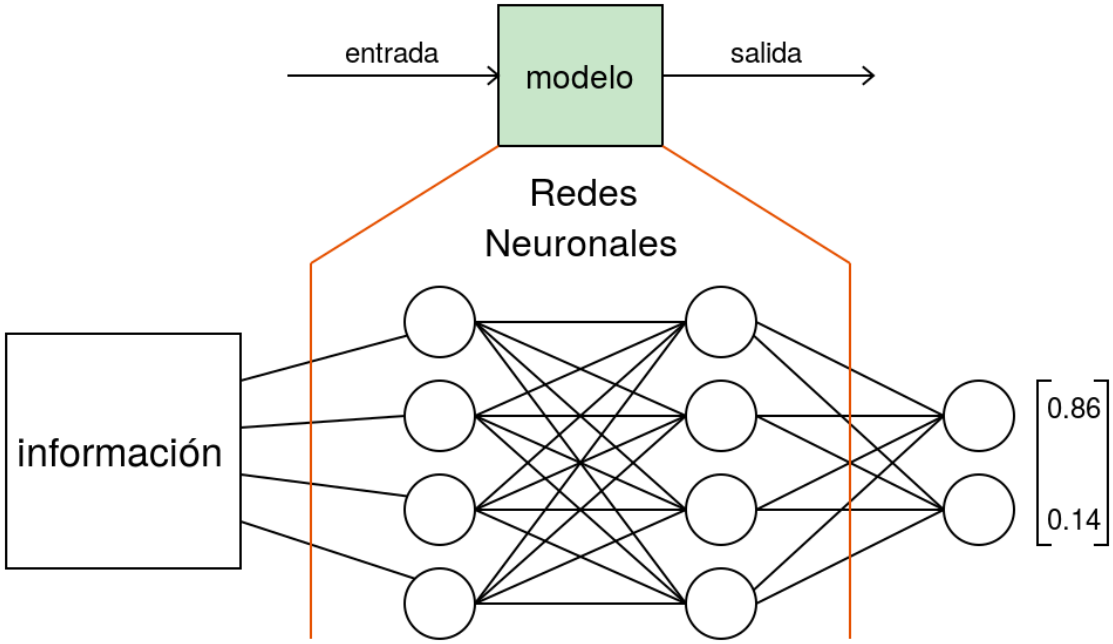


Figura 1.7: Esquema general de un modelo, En este caso, el ejemplo correspondería a un modelo basado en Deep Learning

Con esta idea, se pretende que el médico experto utilice esta información para que los profesionales sanitarios puedan centrar su atención en aquellos casos en el que la confianza obtenida por el modelo no sea óptima o tan precisa. Esto ahorraría tiempo al médico en los casos que el modelo es capaz de detectar con una precisión y un intervalo de confianza elevado.

1.6 Estado del arte en las técnicas de Aprendizaje Profundo o Deep Learning

En la literatura que hemos estudiado, se ha podido observar desde cuales son los conjuntos de datos que se utilizan para realizar este tipo de tarea para crear un modelo predictivo, cuales son las técnicas de AP o la aplicación de modelos clásicos que se suelen utilizar con este tipo de datos y cuales son las técnicas de preprocesado que suelen utilizarse para mejorar la calidad del modelo y obtener un mejor acierto en la predicción. Por ello, vamos a explicar todo esto con un poco más de detalle para ver qué técnicas, topologías de redes neuronales, caracterización de los datos y su clasificación según el estado del arte: *Convolutional Neural Network*, o CNN en adelante.

1.6.1 Descripción del conjunto de datos utilizado

Actualmente existen dos conjuntos de datos, procedentes de proyectos internacionales y que son los más utilizados en este tipo de problema:

- **Colección de datos del Alzheimer Disease Neuroimaging Initiative (ADNI).** Este conjunto de datos se componen de 1476 sujetos con imágenes tanto de RM como de PET, datos demográficos, evaluaciones clínicas y cognitivas, y en menor medida, muestras biológicas (orina, muestras de sangre y extracción de líquido cefalorraquídeo, obtenido mediante punción lumbar).

- **Colección de datos del Open Access Series of Imaging Studies (OASIS).** Este conjunto de datos se componen de 1098 en su tercera versión (OASIS-3) sujetos con neuroimagen longitudinal, clínico, cognitivo y de biomarcadores para el envejecimiento normal y la enfermedad de Alzheimer. Hay desde sesiones de

MR que incluyen secuencias T1w, T2w, FLAIR, SWI, time of flight, resting-state BOLD y DTI, sesiones de PET y datos extraídos por post-procesado.

Estos estudios abordan principalmente imágenes de tipo T1w, que son imágenes muy precisas a nivel anatómico y se pueden diferenciar con nitidez las tres sustancias de la zona cerebral. Por tanto es necesario para ver y cuantificar la posibilidad de atrofia cerebral en un paciente. También hay estudios que abordan los modelos con imágenes PET [16, 17]. el motivo del uso de estas imágenes puramente funcionales es que se pueden ver atrofias a partir de la observación del consumo de glucosa en el cerebro en determinados momento. Así, una persona con demencia pueden tener zonas donde no se consuma tanta glucosa como en una persona sin demencia.

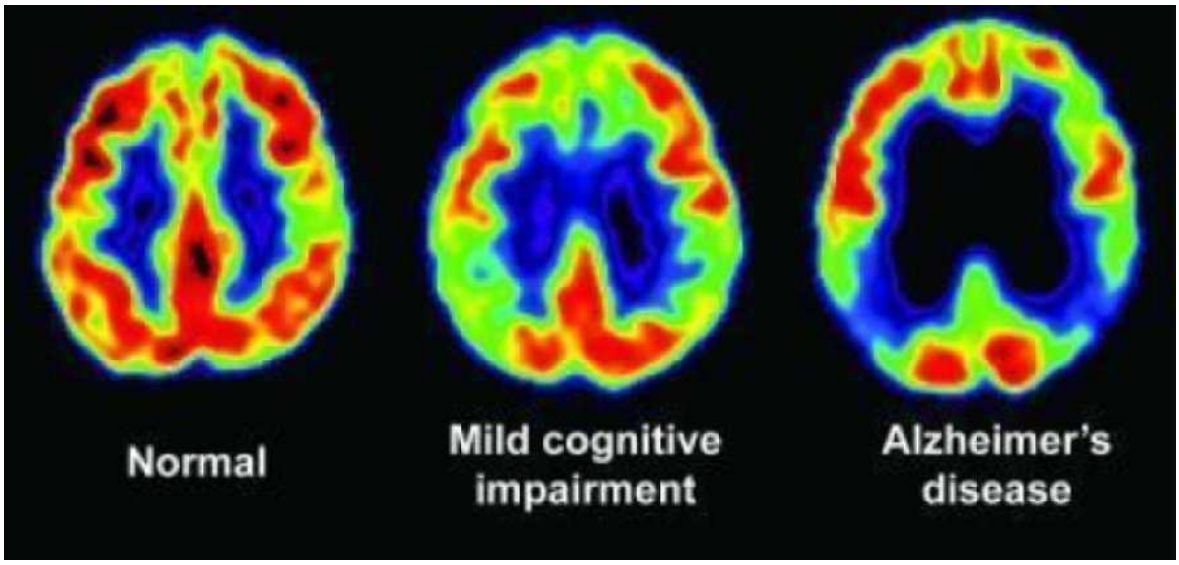


Figura 1.8: Diferenciación de imágenes PET en demencia, a la izquierda tenemos un caso control, en medio una demencia leve y en el derecho un caso de AD.

1.6.2 Técnicas de preprocesado más utilizadas.

A veces la propia imagen suele tener información que no es necesaria para realizar un entrenamiento de un modelo de CNN y que no nos aporta una información relevante para el clasificador que estamos construyendo. Se suele intentar extraer las

zonas de interés para reducir la cantidad de datos que introducen ruido a la CNN y para disminuir las dimensiones de las imágenes, técnica que permite acelerar la rapidez del entrenamiento de las redes o producen modelos más grandes. Las herramientas que se suelen utilizar son las que presentamos a continuación:

- **Brain Extraction Tool (BET):** Esta herramienta nos permite eliminar tejidos que no pertenecen al parénquima cerebral. Estos son el cráneo, cuello, globos oculares,... [18, 19]

- **FMRIB's Linear Image Registration Tool (FLIRT):** Esta herramienta pertenece a la librería software FSL[2], y que corregistra las imágenes de RM con transformaciones lineales a partir de una imagen de referencia. Este preprocesado se realiza para que todas las zonas del cerebro están alineadas y sean comparables, teniendo en cuenta la heterogeneidad y la variabilidad inter-sujeto. Además, este preprocesamiento nos ayuda a obtener imágenes con las mismas dimensiones, ya que originalmente el tamaño y proporción del conjunto de datos original son variantes y para introducirlas en la red han de ser iguales en las tres dimensiones con la que se adquiere la imagen.[18, 17].

- Freesurfer: Con esta herramienta se realizar la parcelación cerebral siguiendo diferentes modelos del cerebro con distintos atlas cerebrales propuestos en la literatura. Además obtiene los datos de volumetría de las parcelas, como el grosor y la superficie. En algunos estudios se han utilizado la segmentación para extraer distintas zonas de interés y procesarlas por separado [17] además de utilizar los datos numéricos de morfología cerebral para entrenar modelos clásicos como random forests [20].

[2]La biblioteca de software FMRIB, abreviada FSL, es una biblioteca que contiene herramientas de análisis de imágenes y estadísticas para datos de imágenes cerebrales de resonancia magnética funcional, estructural y de difusión. FSL está disponible como binarios precompilados y código fuente para computadoras Apple y PC (Linux) y disponible en abierto para uso no comercial.

En general, todas estas técnicas nombradas buscan limpiar los datos para reducir la información ruidosa que se introduce a las redes y que pueden alterar el aprendizaje de estas, además, supone una mejora en la velocidad y eficiencia de los entrenamientos ya que se reduce la dimensionalidad y en consecuencia la cantidad de operaciones a realizar.

Una estrategia poco utilizada en la literatura consultada es el data augmentation, o técnica de aumento de datos [21] que consiste en realizar transformaciones en las imágenes para generar otras nuevas, aumentando así de forma artificialmente el tamaño del conjunto de datos etiquetados. Normalmente estas transformaciones son: rotaciones, traslaciones, zooms, cambios en el brillo, deformaciones, etc. Con esta técnica se consigue generalizar la entrada al sistema y evitar el sobreentrenamiento.

1.6.3 Modelos de Aprendizaje Profundo.

Para la clasificación de estas imágenes los trabajos revisados usan principalmente redes neuronales convolucionales debido a su gran potencial en tareas de clasificación y reconocimiento de formas. A continuación vamos a comentar algunas de las técnicas y variantes de CNNs vistas en la literatura.

En la mayoría de estudios se utilizan redes convolucionales convencionales, con bloques de convolución + *pooling* y finalmente una red *fully connected* que acaba con una salida de probabilidades para cada clase [17, 18].

En el artículo de Donghuan Lu, [17], se utiliza una red multimodal que consiste en tener varias entradas a la red donde cada una pasará por sus propias capas convolucionales para finalmente concatenar las matrices resultantes antes de entrar en la etapa *fully connected* que realiza la clasificación a partir de las características extraídas. Esto hace que la red pueda dedicar toda una rama de convoluciones a cada una

de las diferentes entradas para así poder extraer características más concretas de cada una de ellas. En este estudio se dispone de diferentes regiones de interés que habían extraído mediante la segmentación realizada con FreeSurfer, de esta forma dedicaban cada una de las entradas para una zona de interés.

Por otro lado hemos detectado que en el artículo de Hongfei Wang [18], se utilizan bloques convolucionales densamente conectados. Estos están formados por capas convolucionales cuyas salidas están conectadas a cada una de las capas siguientes, como se puede ver en la figura 1.9.

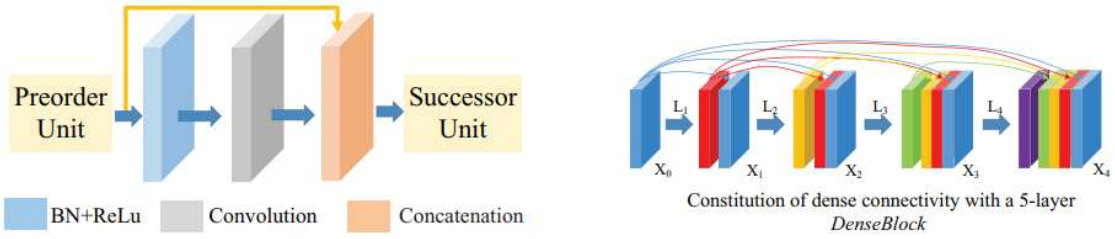


Figura 1.9: En la figura de la izquierda se pueden ver las conexiones de un bloque convolucional densamente conectado. Y a la derecha se muestra el flujo de los tensores para la combinación de 5 de estos bloques [18].

Con esto se consigue mejorar la conectividad de la red y por lo tanto una mejor retropropagación del error con la que se reduce el problema de *vanishing-gradient*[3]. Pero estos bloques aumentan mucho las dimensiones de los tensores debido a las múltiples concatenaciones que se realizan, por lo que para reducir la dimensionalidad, la imagen de entrada pasa por una capa convolucional con el *stride* de 2x2x2 que reduce el tamaño a un cuarto. Además, tras pasar por cada bloque densamente conectado se aplica una capa convolucional con *kernels* de 1x1x1 y otra capa de *pooling* para reducir todas las dimensiones del tensor resultante de cada bloque. Se puede ver el esquema de esta red en la figura 1.10.

[3]El gradiente durante el entrenamiento se va desvaneciendo a valores muy pequeños, impidiendo eficazmente el peso de cambiar su valor.

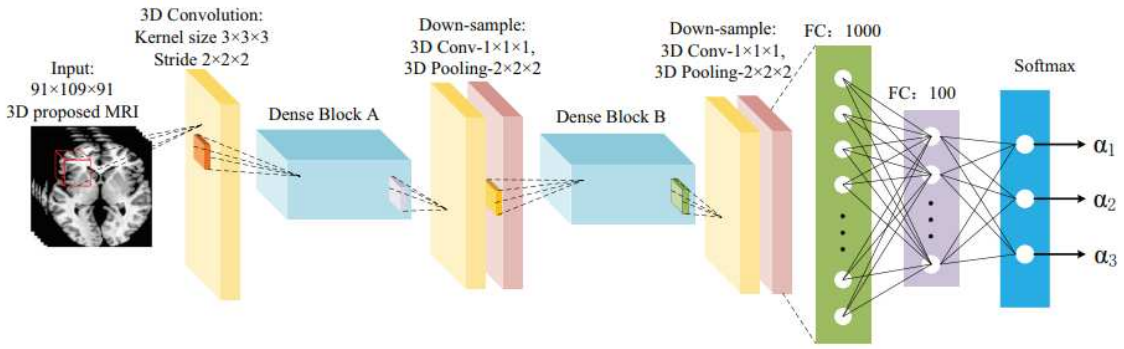


Figura 1.10: Concatenación de *dense blocks* con *pooling* para la extracción de características [18].

Tabla 1.1: Porcentajes de acierto del artículo de Hongfei Wang [18] para diferentes combinaciones de etiquetas.

Etiquetas	Precisión (%)
MCI/AD	93.61
AD/CN	98.83
MCI/CN	98.42
AD/MCI/CN	97.52

En este mismo estudio se entrenan 10 modelos con la misma topología mediante validación cruzada para finalmente hacer una clasificación por votación entre todos ellos para el conjunto de test, obteniendo así los mejores resultados vistos y que se muestran en la tabla 1.1.

Finalmente, una técnica muy utilizada es el pre-entrenamiento de los bloques convolucionales mediante autoencoders [22, 17]. Para ello, primero se entrena un autoencoder que aprenda a reconstruir las imágenes, esto hace que las capas convolucionales aprendan los kernels para reconocer características de ese tipo de imágenes ya que la parte del encoder ha de crear una representación compacta de la imagen en el cuello de botella de la red para que luego el decoder pueda reconstruir la imagen lo mejor posible. Una vez acabado el entrenamiento se extraen las capas del encoder y

se utilizan como bloque convolucional de la red que vamos a utilizar para la clasificación. El entrenamiento de esta red con pesos pre-entrenados se hace en dos pasos. Primero se entrena con los pesos congelados/fijos del encoder pre-entrenado, y que sólo aprenda primero los pesos restantes y tras este paso se descongelan los pesos del autoencoder para realizar un entrenamiento de grano fino que acabe de ajustar los pesos de toda la red en conjunto.

1.6.4 Crítica al estado del arte

Una parte muy importante a la hora de abordar estos problemas es la forma de hacer el particionado de los datos en los conjuntos de entrenamiento, validación y test. Ya que no siempre es válida una partición aleatoria de las muestras para separar los datos en estos tres conjuntos. En el caso del problema que se está abordando en este trabajo, este tipo de partición aleatoria no es válida ya que hay muestras que comparten características que pueden ser aprovechadas por la red para hacer "trampas"durante el entrenamiento y obtener resultados que aparentan ser correctos pero en realidad no está generalizando para la tarea que se les está queriendo entrenar.

En este caso se trata de que hay pacientes con varias imágenes/sesiones de resonancia magnética y una partición aleatoria podría hacer que sus imágenes queden separadas en estas tres particiones. Por lo que la red puede aprender características particulares de ese paciente en lugar de patrones del deterioro cognitivo con los que poder generalizar. Y aun así se obtendrían buenos resultados en las particiones de validación y test ya que al volver a reconstruir el cerebro del paciente que ha memorizado lo clasificará correctamente.

Para verificar esto hemos entrenado un mismo modelo con dos particiones distintas, una aleatoria y otra a nivel de sujeto para que todas las imágenes de un mismo suje-

to sólo puedan aparecer en una partición. Los resultados obtenidos son totalmente diferentes, para el modelo con la partición aleatoria se ha obtenido una precisión entorno al 99 % para todas las particiones como se puede ver en la figura y, mientras que el modelo correctamente particionado ha acabado haciendo un sobreentrenamiento en la partición de training (entrenamiento) mientras que en development (validación) y test se ha quedado estancado cerca del 70 %, figura 1.11 y 1.12

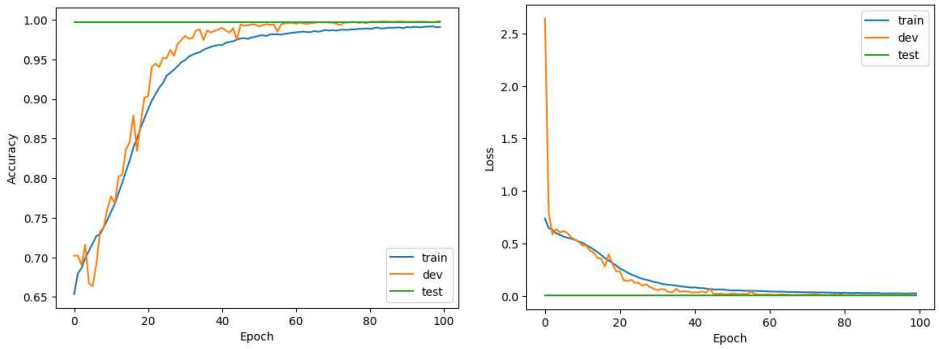


Figura 1.11: Resultados de clasificación para una red entrenada con un particionado incorrecto separando las imágenes de forma aleatoria.

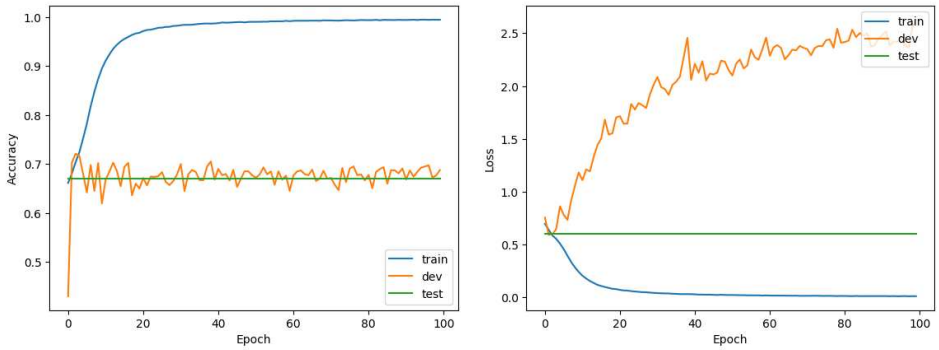


Figura 1.12: Resultados de clasificación para una red entrenada con un particionado del *dataset* a nivel de sujeto.

Esto es debido a que con la partición correctamente realizada la red no puede generalizar bien; aprendiendo las características del sujeto, en vez de aprender las ca-

racterísticas de la EA. Al ver los nuevos datos a partir de la partición acordada, en development y test, los resultados no son tan buenos. Por lo que si se llegara a implementar y poner en producción el modelo con una partición aleatoria, los resultados de clasificación con casos nuevos no serían fiables.

En todas las publicaciones que hemos estudiado, se han observado pocos artículos que especifiquen con claridad cómo se han realizado las particiones y que estas sean correctas, estos han sido [18, 16]. Otros artículos en los que no se ha especificado cómo se ha realizado la partición han sido [8, 17, 21], por lo que sus resultados podrían ser erróneos o poco fiables. Por último se encontró una publicación [19] en donde se especificaba que la partición era aleatoria, por lo tanto no se está haciendo uso de buenas prácticas pudiendo llevar a resultados no válidos. El objetivo de esta crítica es destacar esta mala praxis con el particionado y la falta de detalle en la explicación de algunas publicaciones que puede llevar a resultados aparentemente correctos y que pueden pasar desapercibidos.

1.7 Estructura de la memoria

La estructura de este trabajo se divide en siete capítulos. En el capítulo 2 se hablará de la misión y propuesta inicial del trabajo que se quiere realizar, el objetivo principal y los objetivos secundarios. A continuación en el capítulo 3, se describe los *datasets* y revisamos las herramientas software y hardware que se han utilizado a lo largo de todo el proyecto. En el capítulo 4 se comenta cómo se han procesado los *dataset* y su particionado para posteriormente aplicarles los modelos de predicción. Durante el capítulo 5 se van a presentar los resultados obtenidos durante el proceso de entrenamiento. En el capítulo 6 se realizará una discusión de los resultados y se explicaran las limitaciones que han acontecido durante los experimentos. Por último, en el capítulo 7 marcaremos las conclusiones obtenidas del trabajo.

Posteriormente en el apéndice A se plasmara el código utilizado en la experimentación para el preprocesado, particionado, creación de modelos y entrenamiento.

1.8 Colaboraciones

Este trabajo ha sido discutido además de con mis tutores Jon Ander Gómez y María de la Iglesia, con tres compañeros de grado con los que he compartido trabajo en equipo y amplias y elocuentes sesiones de intercambio de conocimiento y experiencias que han sido gran parte de la semilla de este TFM. Esta intensa colaboración concluyó con la presentación de un Póster en la segunda edición del congreso CONBIOPREVAL, en donde mis compañeros de equipo Álvaro López Chilet, Rafael Sánchez y Silvia Nadal Almela, participaron en la primera parte del proyecto; esta colaboración ha consistido principalmente en el análisis de los datos de los que disponíamos y concluyeron unos primeros experimentos de clasificación.

Además, este TFM se ha realizado en su totalidad bajo el paraguas de una beca concedida por la Conselleria de Sanidad Universal y Salud Pública, Subdirección General de Sistemas de Información para la salud, más concretamente dentro de la Unidad Mixta de Imagen Biomédica FISABIO-CIPF. Además se han utilizado tanto instalaciones asociadas al cluster de cómputo del CIPF como al cluster Artemisa ML, del IFIC, cuyo objetivo es permitir a los distintos grupos de investigación desarrollar y hacer correr sus programas de machine learning e inteligencia artificial en bioinformática.

Por último se ha colaborado con el centro de investigación de Pattern Recognition and Human Language Technology –PRHLT– en la validación de alguno de los modelos que han sido testados en este TFM.

CAPÍTULO 2

Misión, hipotesis y objetivos

En el día a día los profesionales de la Salud se ayudan de los sistemas de información para la toma de decisiones. En EA la IRM es una prueba en la que los neurólogos se apoyan para realizar el diagnóstico de la demencia aunque siempre necesitan de pruebas complementarias para asegurar sus diagnósticos. Esto es debido a que gracias a la IRM se puede apreciar con claridad las atrofias a nivel cerebral-estructural debidas a la EA. Por tanto, estas imágenes serán las que se utilicen para generar un modelo que pueda predecir cómo de probable es que un paciente al que se le ha realizado una prueba de imagen, pueda padecer Alzheimer o no.

Se pretende en este trabajo diseñar un modelo como prueba más que se solicitaría para realizarla al paciente y que verifique su diagnóstico a modo de un biomarcador de patología de Alzheimer. Esta prueba se realiza a partir de las imágenes RM potenciadas en T1 y como resultado se pretende obtener una probabilidad que cuantifique o prediga si padece la enfermedad o no. Por tanto, el médico experto en base a las pruebas solicitadas y según los resultados obtenidos ayudado por el modelo que se implemente, siempre tiene la última decisión. Por consiguiente, nuestro el modelo actúa como un software de ayuda al diagnóstico.

La **hipótesis principal** del presente TFM pretende demostrar que con el uso de técnicas de IRM potenciadas en T1, se pueden obtener modelos que podrán discernir y clasificar los pacientes que padecen EA respecto de los casos control.

El **objetivo principal** de este TFM es obtener un modelo que prediga si a través una red de aprendizaje profundo se clasifique con una probabilidad elevada de acierto la EA.

Por último se plantean en este TFM los siguientes **objetivos secundarios**:

- Estudiar aproximaciones y soluciones recientemente publicadas por varios investigadores y reproducir algunas de ellas para contrastar resultados.
- Analizar y probar distintas técnicas de preprocesado de datos para decidir, en base a los resultados obtenidos, cuál o cuáles son las más apropiadas para el problema que se aborda en este trabajo.
- Diseñar varias topologías de redes neuronales profundas y evaluar los resultados obtenidos utilizando el conjunto de datos ADNI [13] y OASIS [23] para poder compararlos con los resultados publicados por otros autores sobre el mismo conjunto de datos.
- Comprobar si la viabilidad de diagnosticar la enfermedad del Alzheimer es factible con IRM.

CAPÍTULO 3

Material y métodos

3.1 Conjunto de datos

Para este trabajo, se ha optado por trabajar los datos que se ofrecen en ADNI y OASIS por separado. Como se había hablado en el apartado 1.4.1, Este conjunto de datos está preparado para la investigación en EA y deterioro cognitivo leve para detectar la enfermedad del Alzheimer en fases tempranas de su desarrollo, ya que es el momento donde más efectivos van a ser los tratamientos. Este cuenta con muchas pruebas para el diagnóstico y están clasificadas en 5 etiquetas que las nombraran de mayor a menor grado de demencia:

- **Alzheimer Disease (AD):** los pacientes con esta etiqueta tienen un diagnóstico claro de la EA.
- **Latest Mild Cognitive Impairment (LMCI):** los pacientes con esta etiqueta tienen una situación alta de demencia.
- **Mild Cognitive Impairment (MCI):** representa una situación de cambios cognitivos que no se justifican por la edad, sino que son consecuencia de alguna

condición médica, ya sea la enfermedad de Alzheimer u otra. Estos pueden paliarse o, incluso, revertirse en algunos casos. Los pacientes con esta etiqueta tienen situación media de demencia.

- **Earlier Mild Cognitive Inpairment (EMCI):** los pacientes con esta etiqueta tienen una situación baja de demencia.

- **Cognitive Normal (CN):** los pacientes con esta etiqueta son casos control sin ningún tipo de demencia.

Para este trabajo se ha realizado la descarga de las imágenes T1w que tienen una resolución de imagen 3D, siento un total de 2576 imágenes. Elegimos este subconjunto ya que el problema va a ser abordado solamente con información estructural del cerebro y para ello necesitábamos imágenes que fuesen de tipo T1 y con una buena calidad, por lo que han de ser tridimensionales para aportar la mayor cantidad de información volumétrica posible. El recuento de imágenes por etiqueta, edad y sexo se puede ver en la tabla 3.1.

Por otro lado tenemos las imágenes del *dataset* OASIS. Este *dataset* un compendio de muchos tipos de imágenes de RM. Éste está compuesto por 2663 imágenes. Estas imágenes también están adquiridas con una resolución 3D y IRM potenciado en T1 . El *dataset* está etiquetado a partir de una escala denominada *Clinical Demential Rating* (CDR) donde adquiere valores continuos desde [0,3]. Cuando el valor es mayor, más grave es la enfermedad y estas serían las equivalencias con las etiquetas del ADNI:

- 0 equivale a CN

- (0,1) equivale MCI

- $[1, 3]$ equivale a AD

Tabla 3.1: En esta tabla se muestra el recuento de sujetos del *dataset* utilizado por edad, etiqueta y sexo.

Edad	LMCI		AD		CN		EMCI		MCI		Suma por edad
	M	H	M	H	M	H	M	H	M	H	
(50,55]	1	0	0	0	0	0	0	0	0	0	1
(55,60]	16	3	9	4	8	2	11	7	11	4	75
(60,65]	19	16	10	12	18	13	51	47	18	26	230
(65,70]	27	12	13	17	48	45	70	58	31	42	363
(70,75]	37	34	49	32	121	97	47	79	56	81	633
(75,80]	20	28	22	52	121	122	33	63	65	112	638
(80,85]	15	19	24	38	53	77	29	21	56	91	423
(85,90]	6	4	23	24	18	41	2	6	15	56	195
(90,95]	1	0	3	0	4	3	1	0	0	5	17
(95,100]	0	0	0	0	1	0	0	0	0	0	1
Suma por genero	142	116	153	179	392	400	244	281	252	417	2576

De esta forma se puede comparar este *dataset* con ADNI y ver como los modelos son capaces de aprender de estas imágenes. En la tabla 3.2, Se aprecia la distribución de imágenes en este caso sólo para las etiquetas de AD y CN y por sexo.

Tabla 3.2: En esta tabla se muestra el recuento de imágenes del *dataset* utilizado por etiqueta y sexo.

Clase	H	M	total por clase
CN	731	1174	1905
AD	236	205	441
Total por genero	967	1379	2346

3.2 Software utilizado

En este apartado se va describir y desarrollar las herramientas y softwares que hemos utilizado durante el desarrollo del TFM. Principalmente para la realización de los modelos de clasificación y preprocesado se ha utilizado Python 3. Esto se debe a que es un lenguaje de programación multiparadigma, que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Los motivos por los que se han utilizado este lenguaje son:

- Python es un lenguaje de alto nivel con muchas librerías adecuadas para el problema que se está estudiando.
- Hay un gran desarrollo en librerías dentro del campo del Aprendizaje Profundo, con gran cantidad de información, desarrollos y tutoriales.
- Dispone del software Jupyter Notebook [24] que se ha utilizado para probar código por bloques bien documentados. Además facilita métodos de preprocesado de imagen, desarrollo e implementación de estadísticas y gráficas que se van a presentar en el desarrollo de esta memoria.

3.2.1 National Library of Medicine Insight Segmentation and Registration Toolkit

El National Library of Medicine Insight Segmentation and Registration Toolkit (ITK) es un conjunto de herramientas dedicado fundamentalmente al análisis de imágenes. Se trata de una librería para el manejo de imagen médica diseñado con distintas interfaces que abordan las necesidades de los desarrolladores, programadores de aplicaciones y usuarios que utilizan esta toolkit de herramientas. ITK, dispone de una interfaz de uso sencilla denominada SimpleITK que tiene su adaptación en Pyt-

hon y se ha utilizado para cargar en los modelos que se han probado, las imágenes de RM que estan el el formato NIfTI [25].

3.2.2 FMRIB Software Library

El FMRIB Software Library (FSL) [26] es una herramienta software creada por el grupo de análisis de la Universidad de Oxford que dispone de multitud de herramientas de análisis y tratamiento de imagen para RM. Este software no esta integrado en Python pero existe una librería FSL que actúa de interfaz wrapper entre python y la librería de FSL en la máquina donde se encuentre instalada. De todas las herramientas existentes, se han utilizado principalmente dos para la realización de este trabajo: BET y FLIRT.

3.2.2.1. Brain Extraction Tool (BET)

La función Brain Extraction Tool (BET) es un software que se encarga de eliminar todo tipo de tejido de la imagen que no sea cerebro. Se le pasa como parámetro de entrada la imagen obtenida por el escáner y como salida de la función se obtiene la imagen con sólo el parénquima cerebral, y de forma opcional, si es de interés, se obtiene la segmentación en los tres tejidos fundamentales que componen el cerebro.

Para poder realizar una extracción de parénquima ajustada y sin perder tejido de interés hay que modificar alguno de los parámetros de entrada del BET. Unos valores de umbral demasiado altos pueden hacer desaparecer tejidos y áreas que son de interés para nuestro estudio, mientras que valores de umbral demasiado bajos podrían no eliminar tejidos que no son de interés. Después de revisar la literatura en este ámbito, hemos comprobado que los valores que mejor resultado dan para este conjunto de datos son: un umbral de 0.1 con la activación de la opción que elimina

el cuello. En la figura 3.1 se puede observar el resultado obtenido a partir de una imagen nativa.

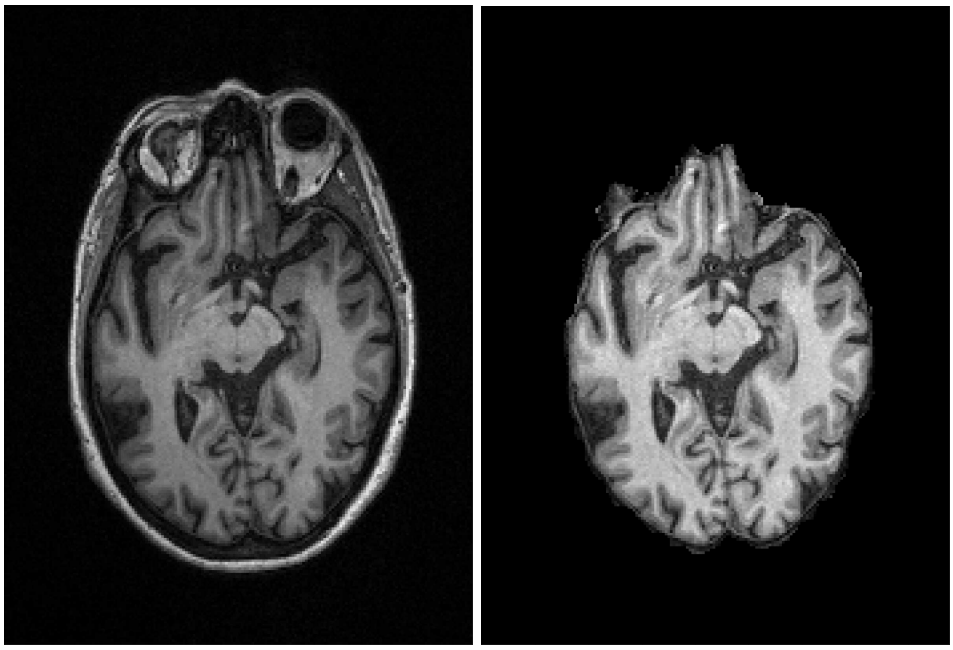


Figura 3.1: En la figura de la izquierda tenemos un corte 2D de la imagen original y en la de la derecha el mismo corte después utilizar BET. Se observa cómo elimina perfectamente globo ocular y cráneo.

3.2.2.2. FMRIB's Linear Image Registration Tool

La función Flirt de la librería FSL, FMRIB's Linear Image Registration Tool (FLIRT) realiza una transformación robusta, precisa y totalmente automática para el registro lineal (afín) de imágenes cerebrales a partir de una imagen de referencia. Para esta parte se ha utilizado como referencia la plantilla MNI152, que es un atlas cerebral generado a partir de 152 muestras de sujetos normales.

Así pues, dada una imagen de nuestro conjunto de datos, FLIRT realiza operaciones de traslación, rotación y zoom para que la imagen dada se ajuste a la posición y forma de la cabeza de referencia. Como se puede apreciar en la figura 3.2.

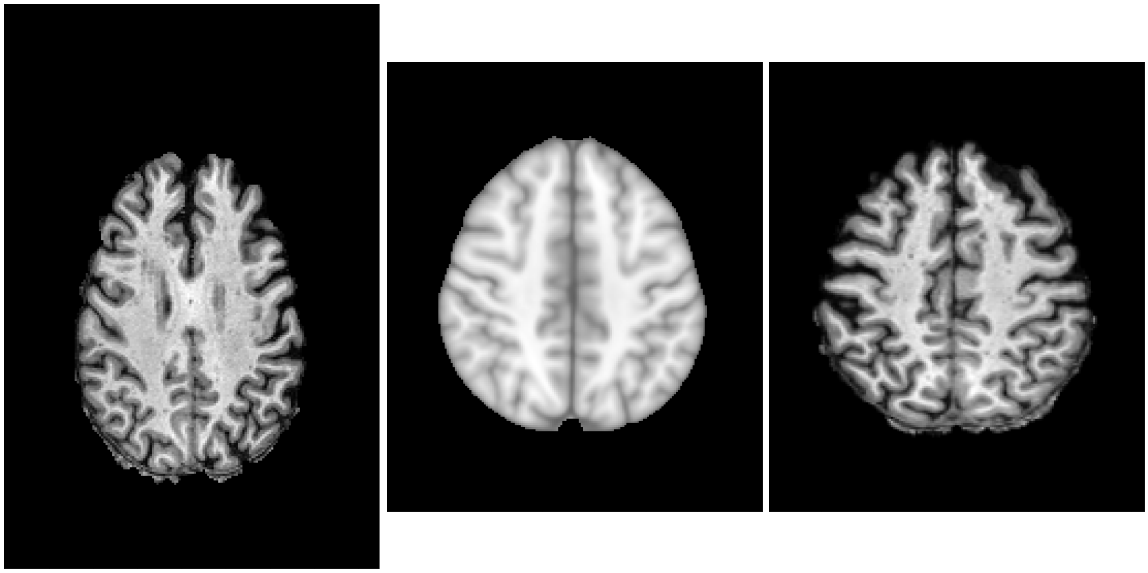


Figura 3.2: A la izquierda se puede ver un corte 2D de la imagen original, la figura central es un corte del MNI152 usado como referencia y en la derecha se observa el resultado aplicar la función FLIRT a la imagen de la izquierda

3.2.3 *FreeSurfer*

Como ya hemos comentado en puntos anteriores, para realizar este trabajo, del conjunto de imágenes que hemos obtenido, sólo hemos trabajado con aquellos vóxeles que forman parte del parénquima cerebral y para ello se ha hecho necesario realizar un preprocesado de todas las imágenes. Este preprocesado se ha realizado utilizando la librería *FreeSurfer* [27]. Este *software* que ha sido desarrollado por el Centro de Imagen Biomédico de Athinoula A. Martinos en el Hospital General de Massachusetts, y cuyo objetivo principal es el pretratamiento de imágenes de MRI del cerebro, se puede utilizar tanto para obtener información morfológica (área, volumen y grosor de la sustancia gris) con respecto a diferentes atlas anatómicos, como para estudiar información funcional.

A continuación, en la figura 3.3, se muestra un esquema conceptual del flujo de trabajo desde que se obtiene la imagen desde el escáner hasta los resultados que se obtienen con Freesurfer.

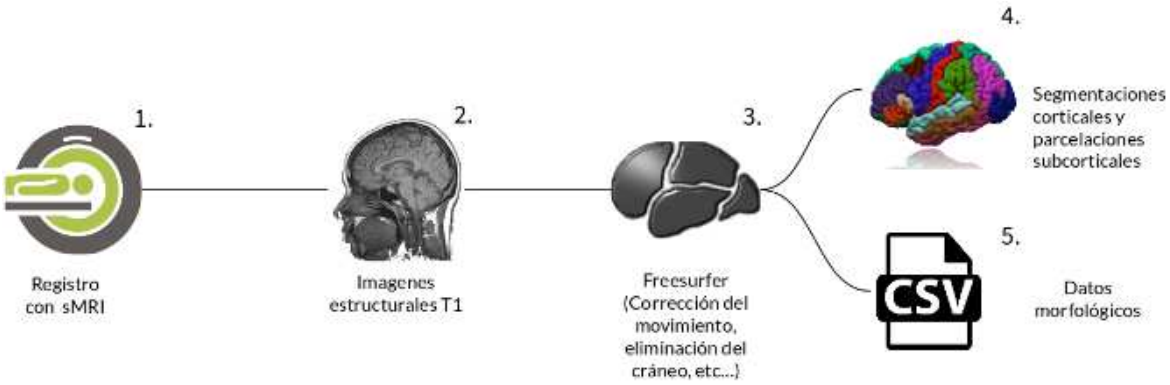


Figura 3.3: Flujo de trabajo general para el tratamiento de los datos las imágenes de MRI para obtener información estructural. (1) Se toman las imágenes anatómicas T1 de los pacientes. (2) Se obtiene una imagen T1, la cual es útil para obtener datos anatómicos de las distintas estructuras cerebrales, pero también obtendremos información que no es de interés, como por ejemplo todos los tejidos blandos que están en la cabeza, el cráneo, la barrera hematoencefálica, entre otras estructuras morfológicas/anatómicas. (3) Utilizando *FreeSurfer* es posible hacer un preprocesamiento de la imagen y hacer múltiples correcciones, además de eliminar aquellos vóxeles que no formen parte de las estructuras corticales y subcorticales en el cerebro. (4) El córtex es segmentado en áreas según diferentes atlas anatómicos, mientras que las estructuras subcorticales son parceladas en sus diferentes estructuras. (5) Obtención de datos morfológicos, como volumen en mm3, área en mm2 y grosor de la sustancia gris en mm.

En el caso el preprocesamiento de imágenes estructurales T1 de MRI, *FreeSurfer* tiene integrado un flujo de trabajo para realiza el preprocesamiento de las imágenes, la corrección de inhomogeneidad del campo magnético, y la extracción de información estadística que está incorporada dentro del flujo; para lanzar este flujo de trabajo se utiliza el comando **"recon-all"** donde se indica el directorio de trabajo, el nombre del archivo en formato nifti o ".nii" con las imágenes MRI y el nombre de la carpeta en donde se van a dejar los resultados o outputs. Todo este proceso ha sido ejecutado en el cluster del Centro de Investigación Príncipe Felipe (CIPF) utilizando un sistema de colas en High Performance Computing (HPC), que nos ha permitido

lanzar 100 ejecuciones en paralelo una por cada imagen de cerebro. Pese a tener muchos recursos, el coste de cómputo de esta herramienta suele ser de entre 6 a 8 horas por IRM en 3D cerebral, sin paralelizar.

A continuación describimos de forma detallada cada uno de los pasos requeridos para completar el flujo de trabajo (Figura zz):

1. Primero se elimina el cráneo y aquellos tejidos que no formen parte de las estructuras cerebrales; para ello se usa un histograma con los valores de intensidad de los diferentes voxeles y crear un mapa global, si hacemos un símil geográfico, los puntos más altos serán aquellos vóxeles más blancos (más intensos), mientras que los valles y depresiones serían los voxeles más oscuros. Si vemos una imagen potenciado en T1 podemos ver que existe una gran cantidad de vóxeles intensos que corresponden a aquellos que forman parte las estructuras cerebrales, rodeados por vóxeles de color oscuro, por tanto, el encéfalo sería toda una meseta rodeada por valles, de esta manera es posible extraer aquellos vóxeles que forman parte del encéfalo.

2. A continuación se realiza el etiquetado volumétrico para diferenciar la sustancia gris de la sustancia blanca, además de la parcelación de las estructuras subcorticales. Con la utilización de histogramas de intensidad, campos aleatorios de Markov y de atlas probabilísticos (un grupo de cerebros que han sido etiquetados con sus estructuras corticales y subcorticales a mano) es posible etiquetar las diferentes regiones subcorticales.

3. La intensidad de la imagen se normaliza con un objetivo dual, primero reducir el ruido de fondo y segundo realizar una corrección de la imagen. Ambos son necesarios para intensificar las diferencias entre la sustancia gris y blanca, además de separar ambos hemisferios.

4. Al segmentar la sustancia blanca de la gris es posible realizar un "inflado" del córtex.

5. El "inflado" del córtex tiene como objetivo representar los surcos (en rojo) y los giros (en verde) de la corteza cerebral, por un lado crear un visualización "sencilla" de un elemento tan complejo como son las estructuras corticales, y por otro, tener una plantilla para realizar la segmentación y etiquetado de las diferentes áreas corticales.

6. Se extrae la corteza y se determina el grosor de la sustancia gris delimitado los límites de la sustancia blanca y la superficie pial.

7. Por último se realiza el etiquetado del córtex con diferentes atlas probabilísticos anatómicos y se obtienen los datos estadísticos [27].

El resultado final son diferentes archivos en formato *"mgz"* que contienen la información de las imágenes procesadas (el cerebro sin el cráneo, solo la sustancia gris o blanca, etc...), además de numerosos archivos ***.label*** que contienen las coordenadas tridimensionales de las diferentes estructuras cerebrales, por último, también se obtiene una lista de archivos CSV que contienen los datos cuantitativos de diversas estadísticas de las áreas que se han parcelado, como el volumen en mm^3, el área en mm^2, el grosor de la sustancia gris en mm, el número de vóxeles y la curvatura. Se describen a continuación los ficheros generados:

- **aparc:** Parcelación anatómica del córtex cerebral, atlas de Desikan-Killiany (70 áreas).

- **DKTatlas40:** Parcelación anatómica del córtex cerebral, atlas de DKT (64 áreas).

- **aparc.a2009s:** Parcelación anatómica del córtex cerebral, atlas de Destrieux (152 áreas).

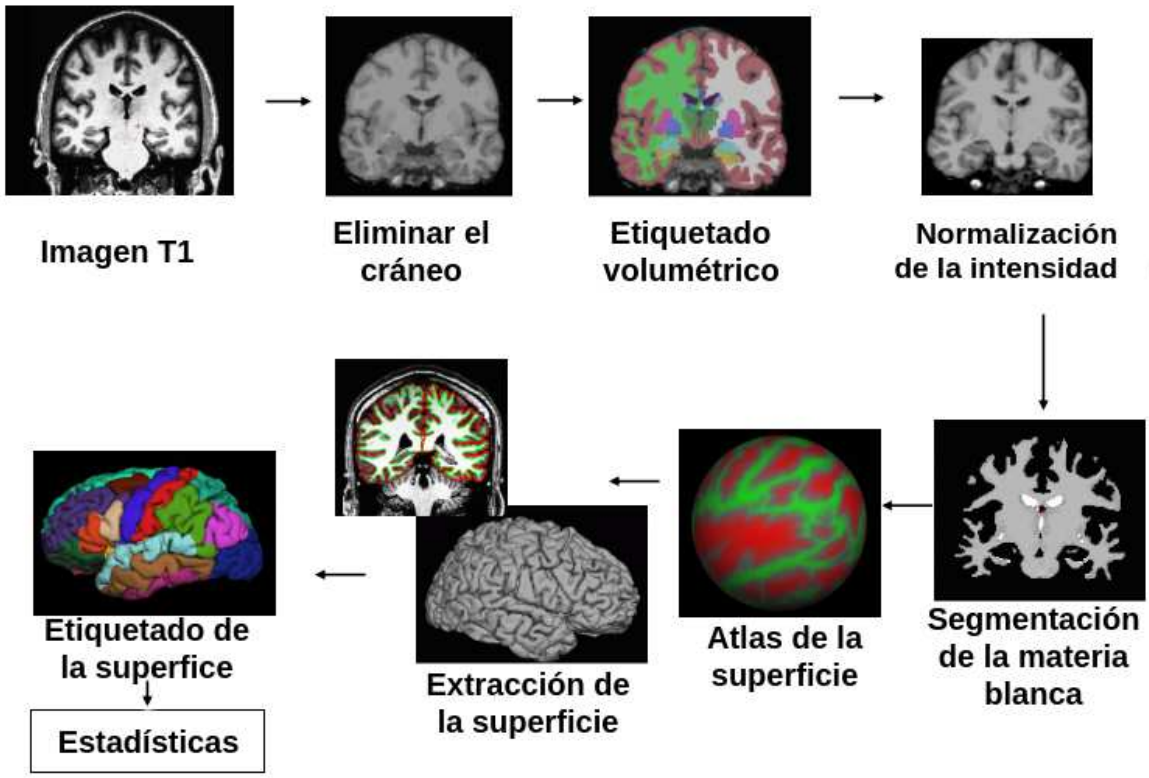


Figura 3.4: Flujo de trabajo general del comando recon-all. Primero se elimina el cráneo y aquellos tejidos que no pertenezcan al encéfalo; a continuación se produce el etiquetado volumétrico, la normalización de la intensidad, paso necesario para separar y segmentar la sustancia blanca de la gris. En la segmentación del córtex se realiza primero un "inflado" del cerebro para separar los surcos y giros del córtex con el objetivo de extraer las superficies del cerebro. Por último se produce el etiquetado del córtex con diferentes atlas anatómicos [27].

- **aseg:** Parcelación anatómica de estructuras subcorticales (46 estructuras).
- **BA:** Parcelación anatómica del córtex cerebral según las áreas de Brodmann y sin ser corregidas, no están todas las áreas sino aquellas afectadas por la enfermedad de Alzheimer (30 áreas).
- **BA.thresh:** Parcelación anatómica del córtex cerebral según las áreas de Brodmann y corregidas. (30 áreas).

- **Entorhinal ex vivo:** Se trata de la parcelación del córtex entorrinal, lo obtiene por separado debido a su importancia ya que tiene una citoarquitectura única en el cerebro. Actúa como un hub neuronal y está especialmente interconectado al hipocampo, en el Alzheimer esta área es de la primeras en ser afectadas por la enfermedad, una única área.

- **w-g-pct:** Muestra el contraste entre la sustancia gris y la blanca. La alteración de este ratio (pérdida de sustancia gris), que también se va modificando por la edad, puede ser importante como marcador para algunas enfermedades, como por ejemplo en el alzheimer o esquizofrenia con el atlas de Desikan-Killiany (70 áreas).

- **wmparc:** Muestra sólo datos sobre la sustancia blanca de regiones corticales usando el atlas de Desikan-Killiany (70 áreas).

3.3 Bibliotecas Keras y Tensorflow

Keras [28] es una interfaz de aplicación o API de redes neuronales de alto nivel capaz de ejecutarse sobre frameworks como TensorFlow, CNTK o Theano, además Keras es una biblioteca de Redes Neuronales de Código Abierto escrita en Python. Fue desarrollado con un enfoque que permite la experimentación rápida. Poder pasar de la idea al resultado con el menor tiempo posible es clave para hacer una investigación eficiente y rápida.

Para nuestro estudio, se ha utilizado Tensorflow [29]. Tensorflow es una biblioteca de código abierto utilizada para generar modelos de AP. Esta biblioteca fue desarrollada por el equipo Google Brain de Google y se lanzó por primera vez en 2015 como sucesora de Distbelief –biblioteca para el mismo uso, pero de código cerrado, que Google utilizaba internamente para crear sus propios modelos de redes neuronales.

Se ha utilizado por su facilidad de integración en sistemas mediante una instalación de Anaconda [30] en donde automáticamente se incorpora python con todas las librerías necesarias para trabajar con estas herramientas y con el software necesario para que funcionen correctamente las GPUs.

3.3.1 Hardware utilizado

Los modelos de AP necesitan realizar cálculos matriciales para entrenar los modelos. Ésto necesita de una tecnologia que nos permita acelerar estas operaciones. Una solución es el uso de tarjetas gráficas donde nos permite paralelizar estos cálculos y acelerar enormemente el proceso, sobretodo en el proceso de entrenamiento donde la carga es mayor. Hay que tener en cuenta en este aspecto la memoria RAM de la tarjeta gráfica, ya que en ella se va al albergar el modelo y, en nuestro caso, las imágenes que se usarán para entrenar. Por consiguiente, cuanto más RAM tenga la gráfica, más rápido es el entrenamiento.

En nuestro caso, al estar trabajando con imágenes 2D y 3D que requieren de mucha potencia de cálculo y en el caso de los modelos 3D no es suficiente con utilizar gráficas de 8GB si queremos utilizar modelos más pesados.

Las gráficas utilizadas son las siguientes:

- **NVIDIA QUADRO P5000** de 16 GB [31]. Los 16 GB de VRAM han permitido entrenar modelos de bastante más capacidad con imágenes 3D debido a su gran dimensionalidad.

- **NVIDIA GeForce GTX 1080** de 8GB [32]. Esta gráfica ha sido utilizada para entrenamientos con entrada de imágenes en 2D.

- **ARTEMISA** es una infraestructura informática de alto rendimiento basada en aceleradores de GPU. perteneciente al Instituto de Física Corpuscular (IFIC) de la Universitat de València. Se contó con el acceso a **4 GPU Tesla Volta V100 SMX2** [33]. Los 32 GB de VRAM han permitido entrenar modelos de bastante más capacidad con imágenes 3D debido a su gran dimensionalidad.

CAPÍTULO 4

Experimentación

En este capítulo, presentaremos toda la experimentación realizada durante el proyecto, desde el inicio, con la creación y preparación/curación del dato; hasta la fase de modelado computacional, haciendo hincapié en la descripción de los preprocesados que se han realizado para crear el conjunto óptimo y correctamente equilibrado. Se va a exponer el preprocesado que se ha llevado a cabo para alimentar la entrada al modelo y cuáles han sido las estructuras cerebrales que se han tenido en cuenta para la detección de la EA. Como paso previo para adentrarnos con más detenimiento en la descripción de todo el proceso de experimentación nos vamos a fijar en la figura fff, que expone y sintetiza el planteamiento y el esquema global de todo el proceso de experimentación, realizado en este TFM. Como se puede observar, en la figura fff se centra en dar una visión de conjunto dentro de la fase de experimentación, y se presenta dividida en tres grandes bloques: (a) Preprocesado del dato, (b) particionado equilibrado y por último (c) modelado computacional. Cada uno de ellos serán explicados con detalle en este capítulo.

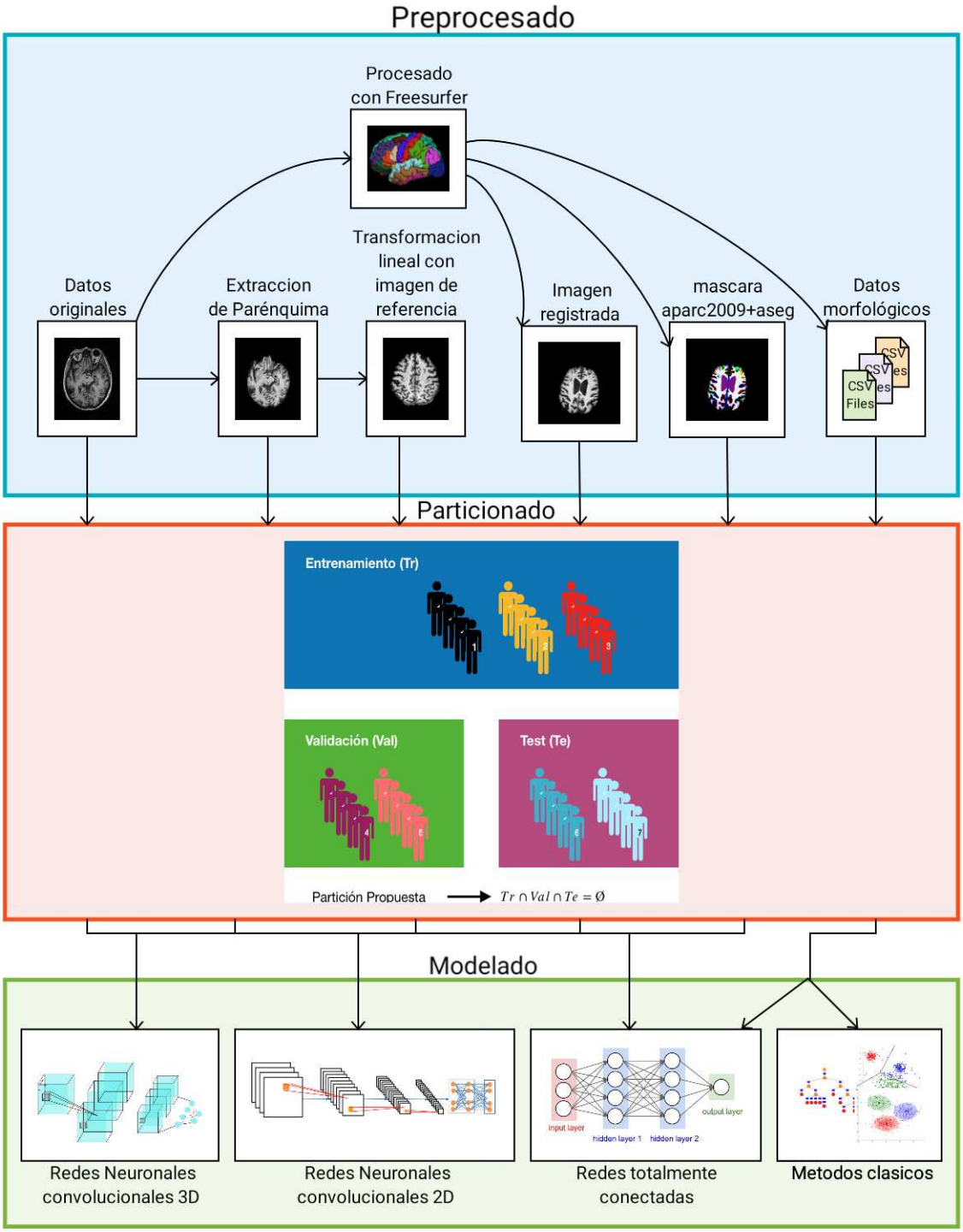


Figura 4.1: Esquema gráfico del proceso realizado en la fase de experimentación del TFM, Se puede observar todas las etapas realizadas que llevan desde la fase de preprocesado o tratamiento de la imagen, su división en las particiones de entrenamiento, desarrollo y test y por último la creación de modelos de clasificación que han sido planteados .

4.1 Preprocesado de los datasets

Como ya se ha comentado, vamos a utilizar los dos dataset explicados con detenimiento en el capítulo 3.1 para realizar el entrenamiento de los distintos modelos computacionales que han sido testados en este trabajo.

Entre otras acciones se ha desarrollado *ad hoc* un código que genera, a partir del dataset original y una tabla en formato CSV, unas etiquetas normalizadas para cada paciente y además se procede a realizar de forma semi-automática, distintos preprocesados del dataset.

Como resultado se almacena en un array del tipo **numpy** y con dimensiones $[X, Y]$ donde X es la entrada que utilizará el sistema e Y la etiqueta en formato *One Hot Vector* para cada imagen de RM. Esta práctica es muy común y siempre se ejecuta de esta forma tanto si trabajamos con imágenes 2D como 3D.

A continuación veremos los casos de uso con los que se ha trabajado partir del dataset que se han generado, ordenado desde el tipo de preprocesamiento más grosero al más detallado y preciso (figura 4.2).

Datos Originales.

El conjunto de datos de IMR no ha sufrido ningún preprocesado en la imagen, se incorporan en formato raw al modelo.

Datos preprocesados a partir de aplicar BET y FLIRT.

- **Extracción del Parénquima (BET):** el conjunto de datos de IRM se le ha extraído el parénquima cerebral y se ha eliminado las partes que no aportan información de valor.

- **Transformación lineal con imagen de referencia (FLIRT):** Al conjunto de datos de IRM (solo parénquima) se le ha llevado a un espacio común usando, como estándar de IRM, la referencia MNI 152 con FLIRT.

Datos preprocesados con *FreeSurfer*

- **Con imagen registrada:**En este caso se ha utilizado el preprocesado con *FreeSurfer* descrito anteriormente y utilizando como estándar de IRM, la referencia MNI 305.

- **Con máscara aparc.a2009s+aseg:** La salida de este proceso además de incorporar la imagen registrada, también se obtiene una máscara asociada a la segmentación. Esta segmentación tiene para cada píxel una etiqueta codificada a valores de enteros donde cada valor corresponde con una estructura del cerebro. A priori esto puede suponer un problema, ya que la codificación a enteros puede incorporar un sesgo a partir de la distancia de dos etiquetas según sean más o menos próximas, por tanto se ha decidido transformar la codificación numérica al formato RGB estandarizado del *FreeSurfer*, ya que este estándar está diseñado para que las áreas que se estén tocando entre ellas tengan un alto contraste y las diferencie.

- **Imagen registrada y la máscara aparc.a2009s+aseg:** Se unen las dos imágenes obtenidas como entrada usando los canales para albergar en cada pixel la intensidad y la codificación RGB.

- **Valores estadísticos**: La salida del *FreeSurfer* ofrece tablas de valores numéricos medidos para cada hemisferio como la volumetría, area, grosor del córtex, etc. Esto se puede utilizar tanto en modelos clásicos como redes *fully connected*.

Durante la parte de experimentación, se va a utilizar únicamente dos clases de etiquetado que aparecen en el conjunto de datos: AD y CN (AD: *Alzheimer's Disease*

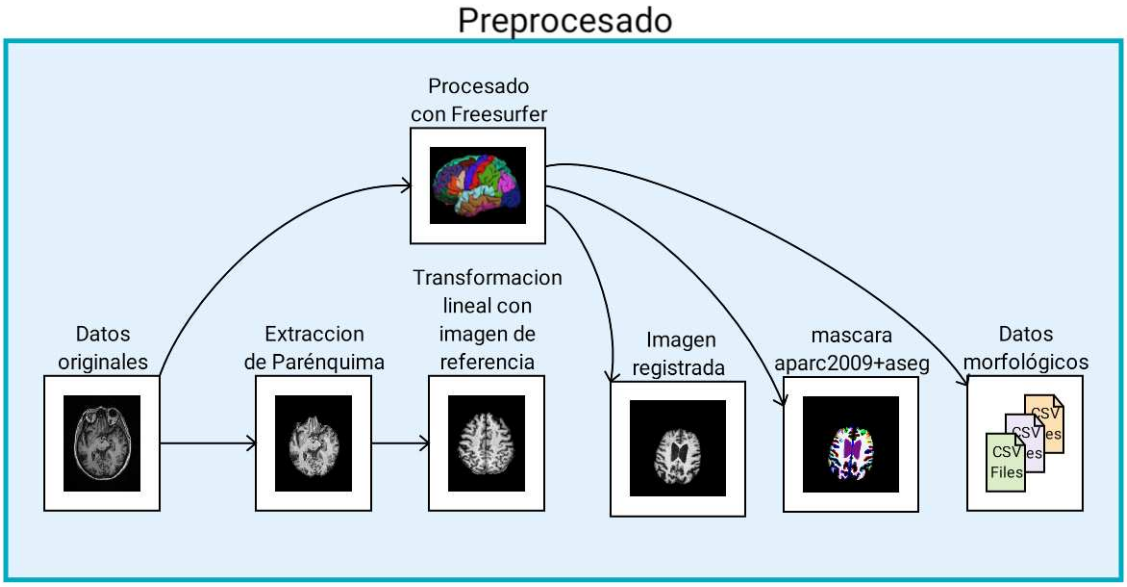


Figura 4.2: Preprocesado realizado a partir de los datos originales de IRM donde se generan las distintas entradas para los modelos.

y CN: *Cognitive Normal*). Esto se debe principalmente porque en el estado de arte, ninguno de los estudios observados, utilizan las sub-clases de *Mild Cognitive Impairment* (MCI) debido a que por el momento, no se encuentra en la literatura buenos resultados para este tipo de análisis con imágenes estructurales. Por otro lado a nivel de la enfermedad, es muy difícil categorizar el deterioro cognitivo leve a partir de Imagen de Resonancia Magnética.

4.2 Particionado equilibrado

Para el conjunto de datos de ADNI y OASIS con los que se ha trabajado en este TFM, se ha organizado en tres particiones:

- **Partición de entrenamiento (Tr):** Se utiliza para entrenar los modelos que se generen durante el trabajo y esta partición contiene el 60 % de los datos.

- **Partición de validación (Val):** Se utiliza para validar los modelos y realizar un tuning o afinamiento de los modelos. Esta partición contiene el 20 % de los datos.

- **Partición de test (Te):** Se utiliza para comprobar la eficiencia de los modelos y esta partición contiene el 20 % de los datos restantes.

Como se ha resaltado en el apartados anteriores, para este tipo de datos de RM es un punto clave y muy importante saber cómo separar los datos entre las tres particiones. Como se ha demostrado en el apartado del estado del arte, la pertenencia de imágenes de un paciente en más de una partición hace que el modelo no extraiga las características de la enfermedad, si no las del paciente, haciendo que memorice que etiqueta corresponde a qué paciente. Por lo tanto estaríamos entrenando modelos que memorizan en vez de entrenar modelos que clasifican. Por consiguiente, las particiones que se generen, deben cumplir que las imágenes de un paciente se incluyan únicamente en una de las particiones (figura 4.3).

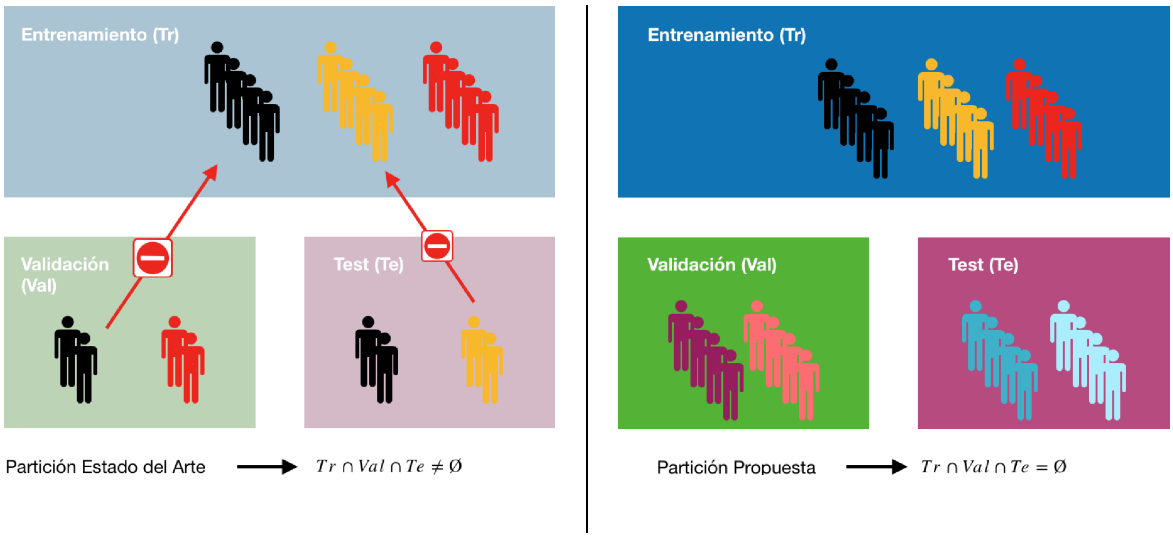


Figura 4.3: En esta figura se pueden ver esquemáticamente los dos modos de partir los datos que se presentan en este trabajo. A la izquierda, se observa la partición completamente aleatoria, donde puede caer un sujeto en más de un subconjunto de valores. Imágenes de un mismo paciente aparecen en más de un subconjunto. A la derecha, las imágenes de un mismo paciente pertenecen solo a un subconjunto.

4.3 Modelos computacionales propuestos en el proyecto

En este apartado vamos a presentar diferentes planteamientos de los modelos que se han utilizado para cada *datasets* generado en el apartado anterior. Dependiendo de la forma en la que organicemos los datos, podemos utilizar unas técnicas u otras.

Se han utilizado principalmente tres tipos de modelos para los *datasets* descritos (figura 4.4):

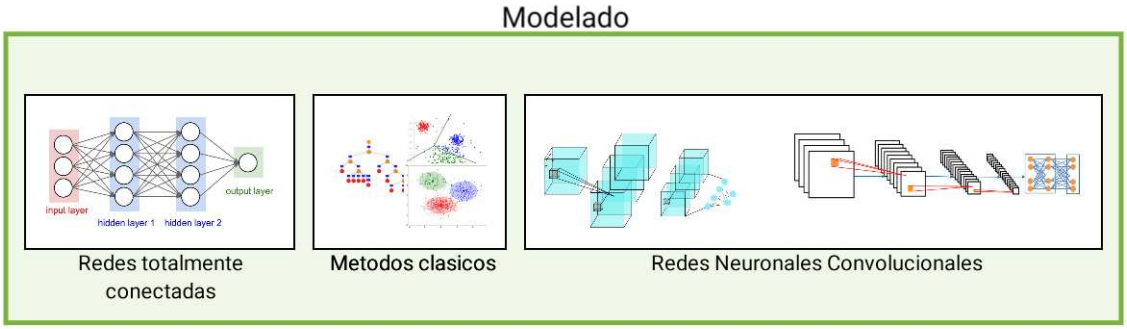


Figura 4.4: Distintos modelos genéricos utilizados en la fase de experimentación.

Tanto las imágenes 2D y 3D ocupan y consumen bastante memoria RAM para los por lo que resulta muy complicado entrenar modelos. Por tanto, se ha utilizado una clase definida dentro de la librería **Keras** llamada *Sequence* que nos permite cargar los batches al modelo desde disco a medida que se van necesitando las imágenes a procesar. Además esta clase nos ofrece más funcionalidades como por ejemplo aplicar la técnica de aumento de datos en el momento de la carga de la imagen y, además nos ayuda a completar el *dataset* si observamos una descompensación entre el número de imágenes etiquetadas como AD y CN (figura 4.5). Para ello se replican las imágenes del *dataset* con menor número de etiquetas hasta alcanzar y equiparar al de mayor número. Esto se realiza para que la red neuronal no le llegue únicamente imágenes de una etiqueta y no aprenda a diferenciar correctamente.

En cuanto al Aumento de datos, se ha propuesto para el entrenamiento las siguientes transformaciones:

- **Zoom**
- **Traslación**
- **Rotación**
- **Cambio de luminancia**

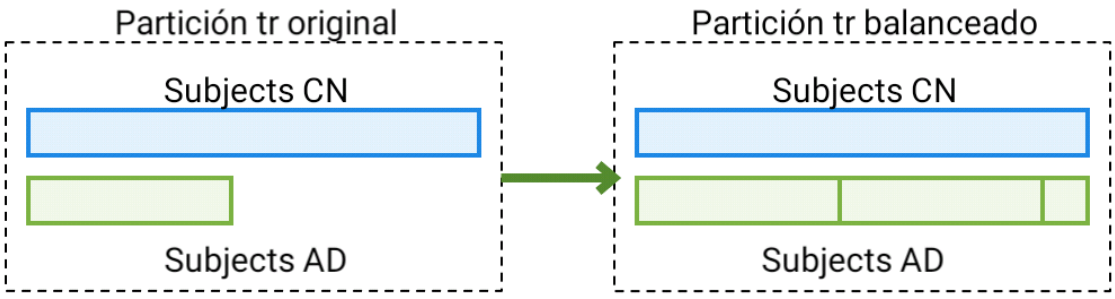


Figura 4.5: Sistema de balanceo para la partición del entrenamiento como solución al desbalance de etiquetas.

4.3.1 Redes Totalmente conectadas

Estas redes se caracterizan por conectar la entrada de los datos con el resultado esperado a partir de capas de neuronas donde la capa anterior esté totalmente conectada con la siguiente (véase el esquema de conexión en la figura 4.8). Cada neurona se encarga de recibir la información sináptica de las neuronas a las que está conectada y transmite la información que procesa a las siguientes neuronas (figura 4.7), a semejanza de cómo se comporta la transmisión sináptica de nuestras neuronas

En la experimentación planteada, estos modelos se han utilizado con los valores morfológicos obtenidos con el *FreeSurfer* donde el preprocesado realizado a estos

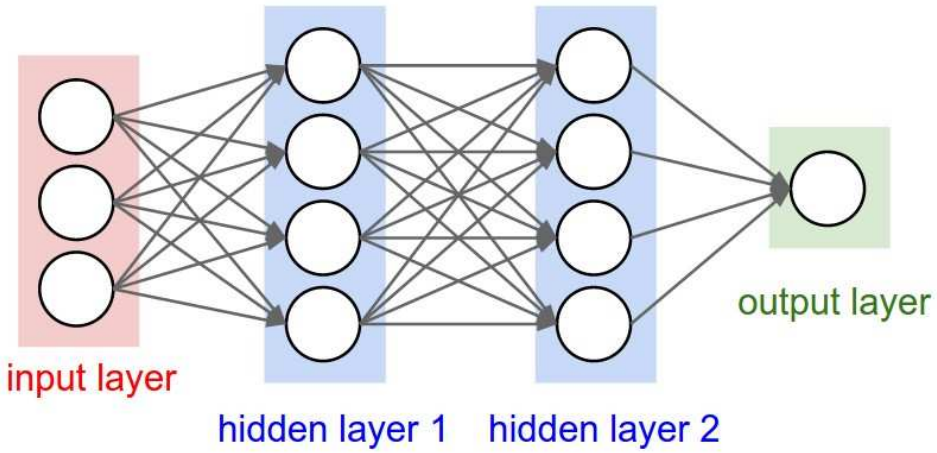


Figura 4.6: Ejemplo de una DNN totalmente conectada donde cada capa se conecta con relación todos a todos con la siguiente.

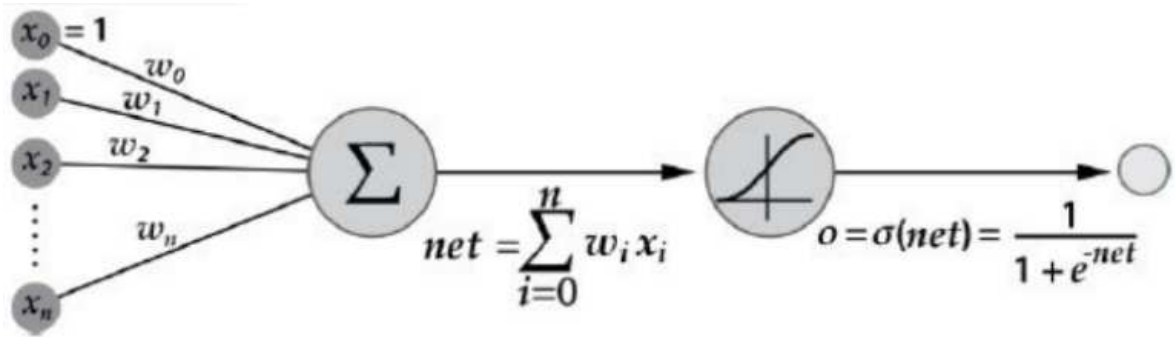


Figura 4.7: Comportamiento interno de una neurona en AP, donde recibe los valores de la neuronas a las que ésta conectada, realiza una función lineal a partir de unos pesos entrenables y realiza una función de no-linealidad para obtener un valor que es enviado a otras neuronas o como salida del sistema.

datos consiste en elegir qué variables de las generadas van a ser la entrada para la red neuronal. Además se realiza un normalizado de las variables propuestas y que se incluyen en el modelo. Para realizar la elección de estas variables se ha decidido a partir de qué estructuras están relacionadas con la enfermedad basándonos fundamentalmente en el estado del arte [20].

En este tipo de redes existen técnica que mejoran la capacidad de aprendizaje de estos modelos. Estas técnicas son las siguientes:

- Gaussian Noise: Función que aplica ruido gaussiano para alterar levemente los datos de entrada y que de esta forma se evite el sobre entrenamiento.

- Batch Normalization: Función que aplica a los datos una normalización $\mathcal{N}(\mu = 0, \sigma^2 = 1)$. Esta función aplicada después de cada neurona, permite que las salidas de las neuronas no tomen valores muy altos y mejora la capacidad de aprendizaje.

- Dropout: Función que se le aplica a la capa anterior, una desconexión aleatoria de neuronas a partir de una probabilidad entre 0 y 1. Ésto provoca que la red no sobreentrene y generalize.

Un ejemplo esquemático de este tipo de redes lo encontramos en la figura 4.8, en donde aplicamos una entrada con 105 variables y en medio de las estructuras que se han explicado anteriormente, se extrae la probabilidad para cada etiqueta.

4.3.2 Métodos Clásicos

Por otro lado y en paralelo, también se ha trabajado con los datos morfológicos extraídos de los csv que proporciona el *FreeSurfer* en su carpeta de *stats* y estos datos se han utilizado para aplicar los métodos clásicos.

Se ha utilizado la librería *Scikit-learn* para aplicar estas técnicas de *Machine Learning* con el conjunto de datos que hemos comentado y se ha realizado una comparación con los modelos basados en AP. El preprocesado aplicado en esta parte del modelaje es el mismo que en el apartado 4.3.1.

A continuación enumeramos los métodos que se han utilizado en este apartado:

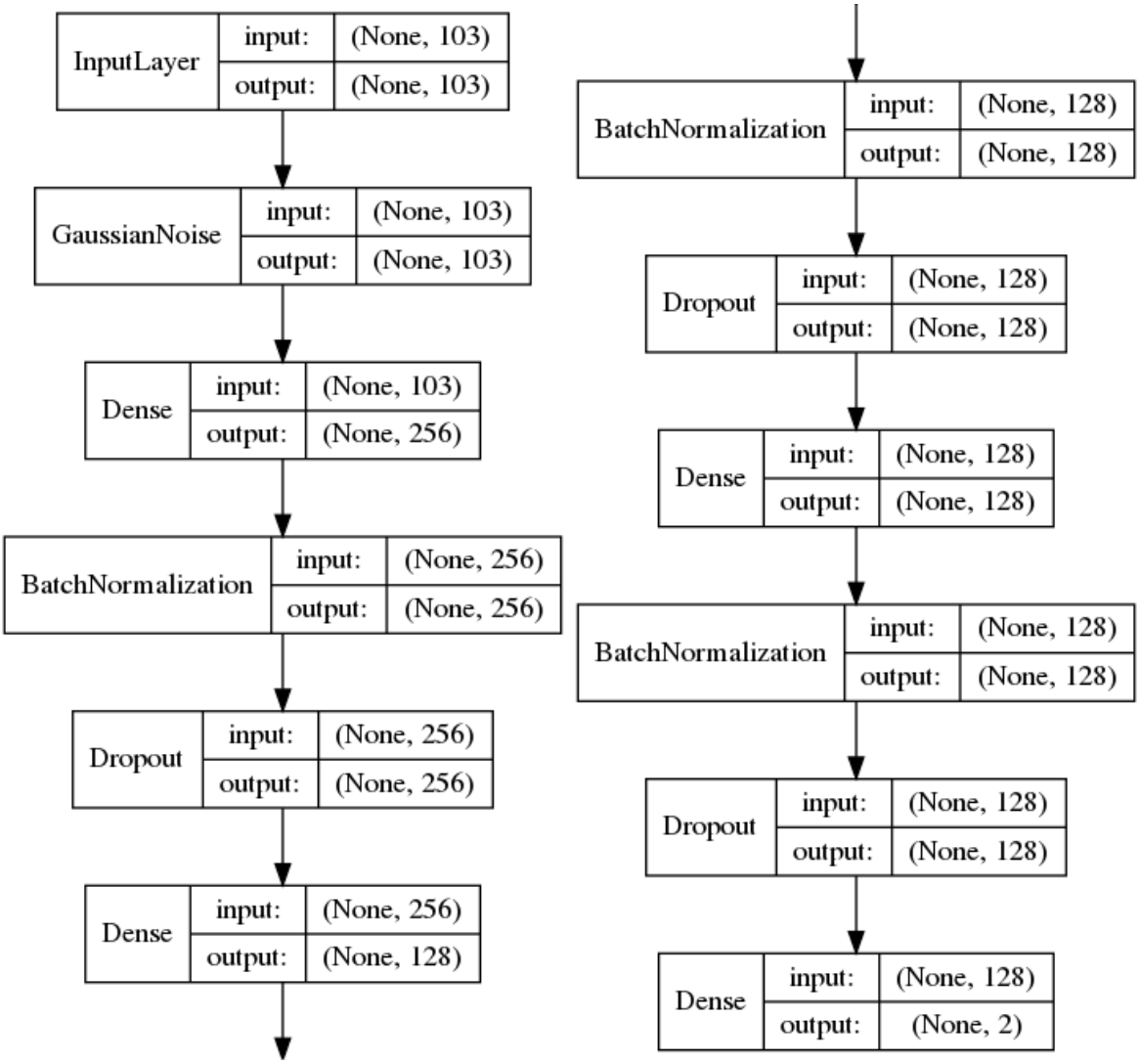


Figura 4.8: Esquema de un a red neuronal totalmente conectada donde se muestra las operaciones que se están realizando para obtener la probabilidad de la etiqueta.

- Extra Trees Classifier
- Gradient Boosting
- Bagging Classifier
- AdaBoost Classifier
- Random Forest Classifier
- K-Nearest Neighbors Classifier
- SVC
- NuSVC
- Gaussian NB
- Gaussian Process Classifier

- Linear Discriminant Analysis
- Gaussian Mixture Classifier
- Quadratic Discriminant Analysis

4.3.3 Redes Neuronales Convolucionales para imágenes 2D y 3D

Las redes neuronales convolucionales se basan en el uso de filtros de imagen para realizar transformaciones de una matriz de entrada a una matriz de salida. El filtro se aplica como una ventana deslizante que pasa por todos los píxeles de la imagen (véase figura 4.9) y la funcionalidad principal de ésta es la detección de elementos dentro de la imagen. Para cada proceso de convolución dentro de una CNN se puede utilizar un número determinado de filtros que permitirán encontrar distinto patrones en la imagen.

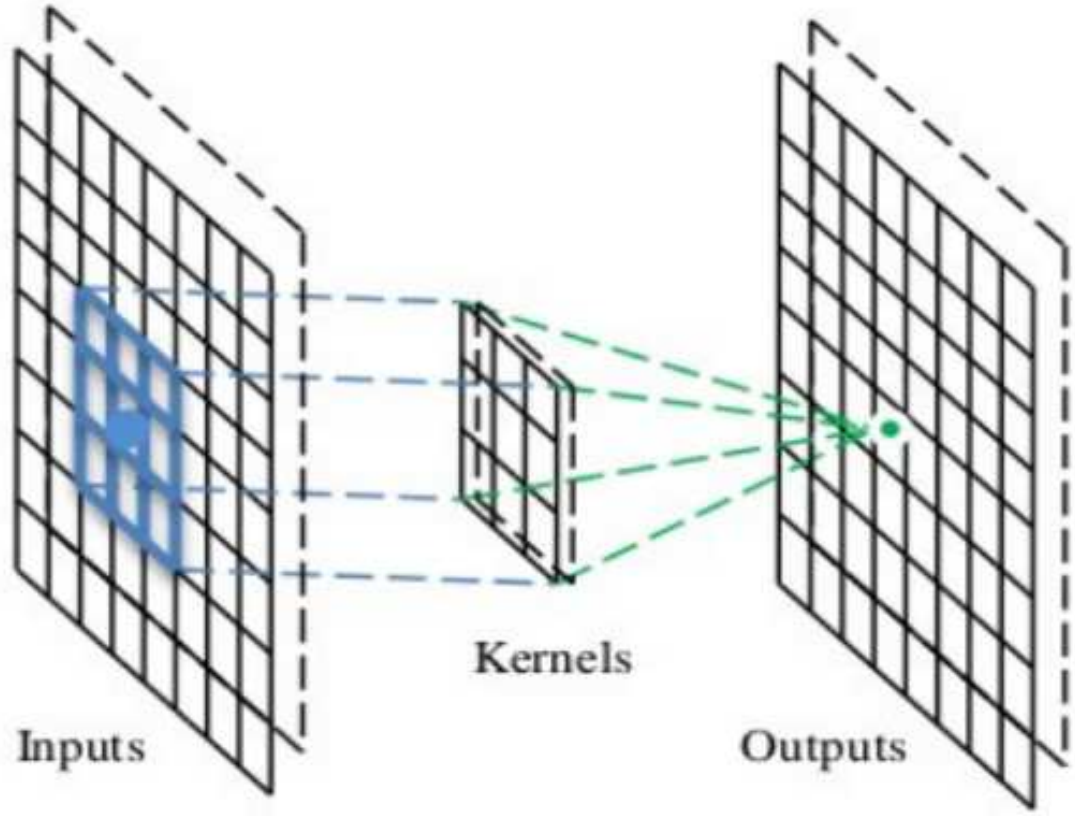


Figura 4.9: Proceso de filtrado de una imagen de entrada para generar una imagen de salida.

Otra operación que se suele utilizar en las CNN son las reducciones de muestreo o (pooling) donde se aplica una ventana deslizante de un tamaño predefinido y además se aplica una regla para reducir la información de entrada. En la figura 4.10, se puede observar una operación de pooling. En este caso, la regla que aplica es escoger el valor máximo en cada deslizamiento y con esto se consigue reducir el tamaño de la imagen.

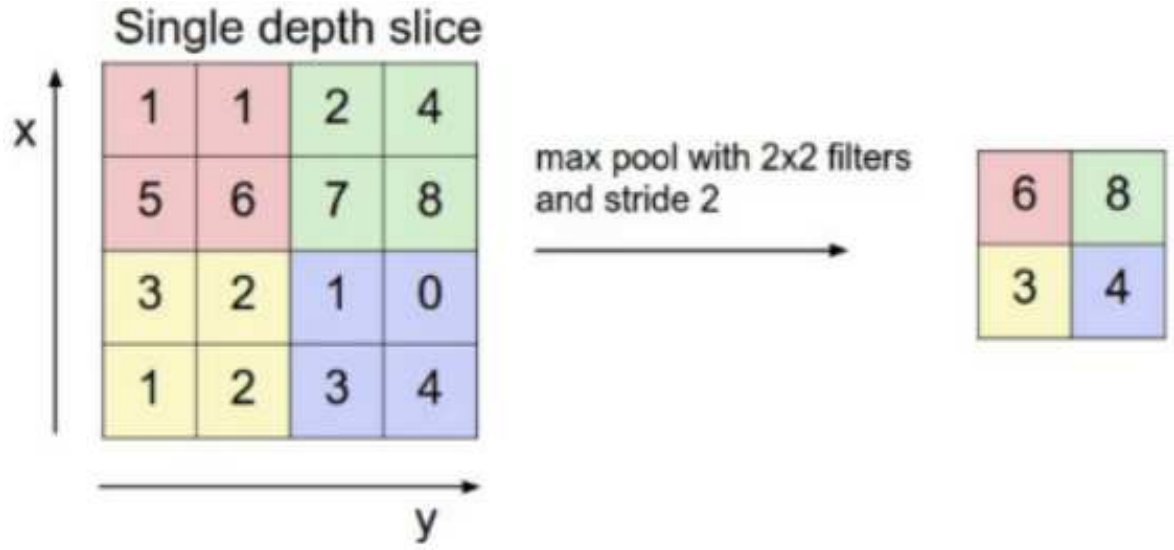


Figura 4.10: Reducción de una matriz por el método de reducción de muestreo en CNN. En este caso se utiliza el método de MaxPooling, donde realizando ventanas de deslizamiento con saltos de 2 pixeles y tamaño de ventana de 2x2, se escogen los valores máximos en cada iteración.

Las CNN se caracterizan tener dos partes diferenciadas:

- **Bloque convolucional:** Donde se aplica las técnicas de filtrado y reducción de muestreo para obtener un vector de características.

- **Bloque altamente conectado:** Donde, a partir de este vector de características, se aplica las operaciones descritas en el apartado 4.3.1 para clasificar la imagen.

Este tipo de estructura se ha utilizado con las IRM al escoger los cortes que componen las imágenes 3D y entrenar estos modelos con estas imágenes sustrayendo

el contexto posicional dentro de la imagen 3D. Se pretende objetivar si el contexto posicional dentro de la imagen 3D favorece, o no, la predicción en el modelo.

Las redes neuronales Convolucionales 3D (Figura 4.11) tienen los mismo principios y funciones que las utilizadas en las 2D que ya hemos comentado, pero a estos modelos se les aplica una dimensión extra como entrada al sistema. Esta estructura facilita el entrenamiento para cualquier tipo de imagen volumetrica. En nuestro caso, este modelo es adecuado para este tipo de imagen.

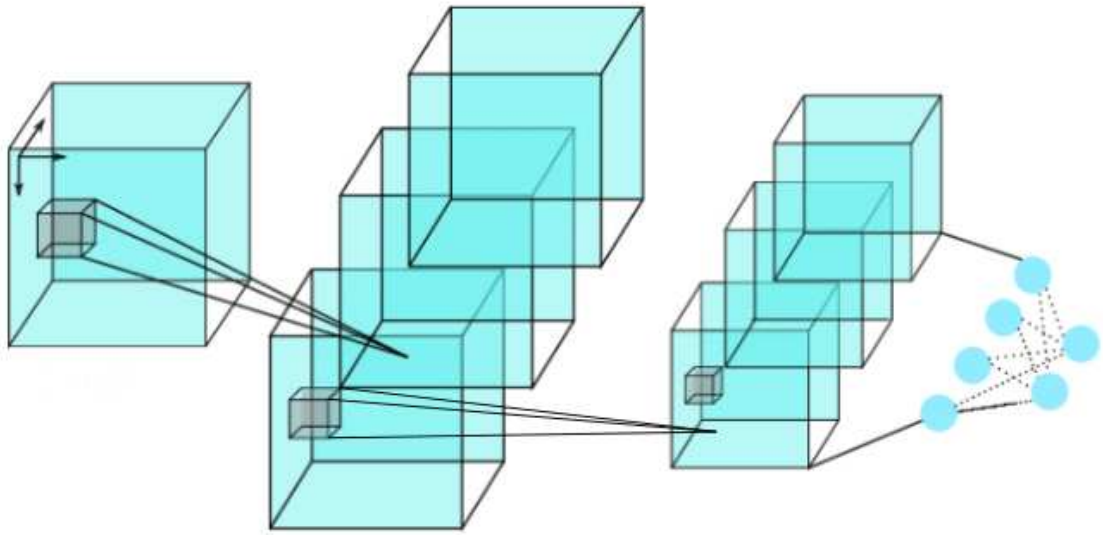


Figura 4.11: Modelo esquemático de una CNN para matrices en 3D.

A continuación se presenta un ejemplo esquemático de este tipo de redes a modo de abstracción conceptual que se muestra en la figura 4.12, donde se aplica una entrada tridimensional donde se van incorporando las funciones descritas con anterioridad y se extrae el vector de características para aplicar el bloque fully conected y así obtener la predicción del modelo.

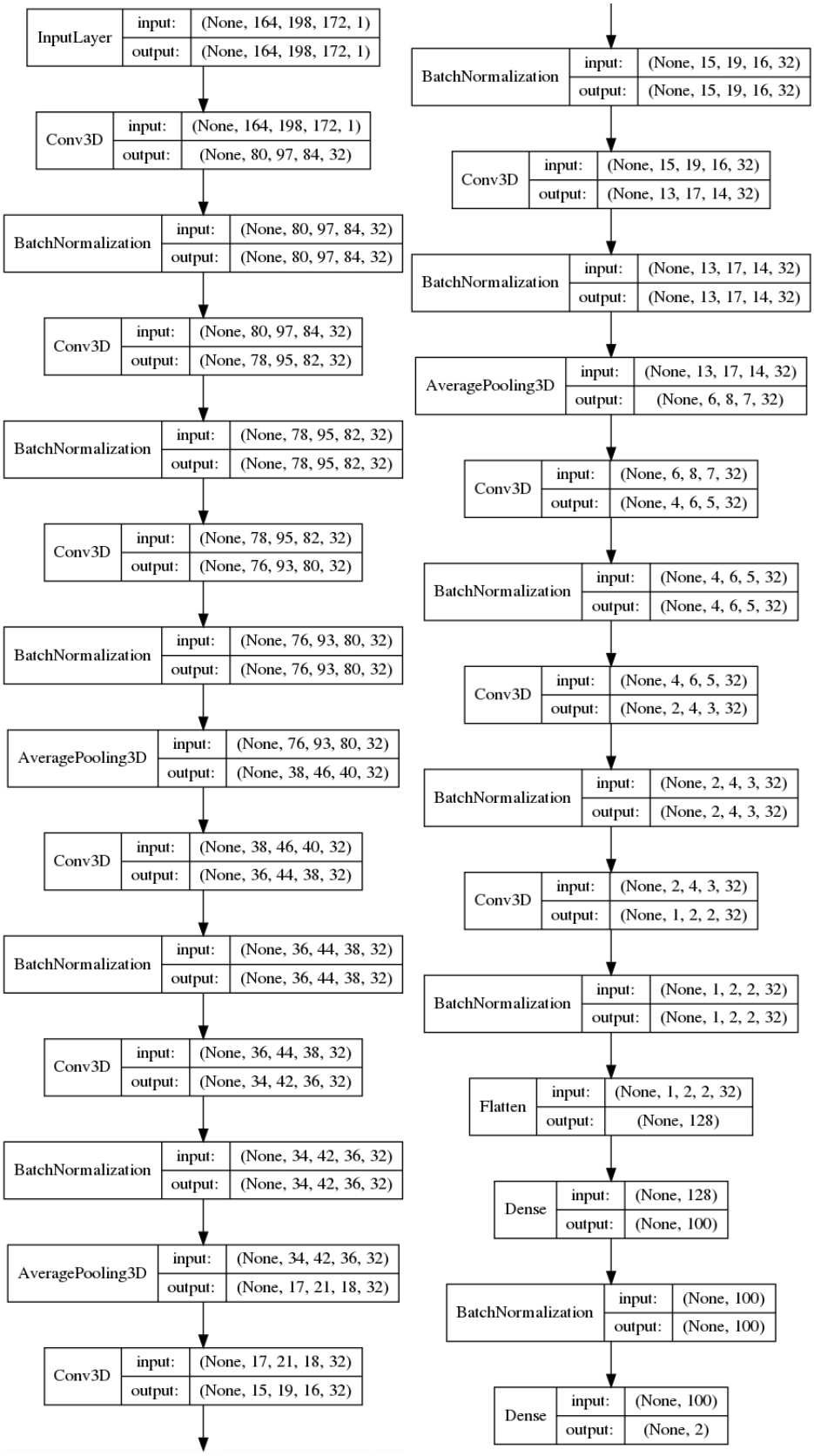


Figura 4.12: Esquema de un a red neuronal convolucional donde se muestran las capas utilizadas de uno de los modelos utilizados en la experimentación.

CAPÍTULO 5

Resultados

En este capítulo, se va a resaltar las métricas que se han utilizado y los resultados obtenidos a partir de los distintos preprocesados que hemos especificado en el sección 4.1. También vamos a hablar desde los primeros experimentos que se realizaron hasta los mejores modelos obtenido durante la fase de experimentación.

5.1 Métricas utilizadas para la evaluación de los resultados

Para la evaluación de los resultados obtenidos, se ha decidido utilizar un conjunto de métricas aparte de la precisión media (accuracy) para tener una visión más exacta de los modelos. Pero antes hay que definir los siguientes términos que se utilizaran para la formulación de las métricas cuando un modelo nos devuelva los resultados a analizar. Estos términos son:

- **Verdaderos positivos (VP):** Número de casos que la prueba declara positivos y que son verdaderamente positivos.

- **Falsos Positivos (FP):** Número de casos que la prueba declara positivos y que en realidad son negativos.

- **Falsos Negativos (FN):** Número de casos que la prueba declara positivos y que en realidad son negativos.

- **Verdaderos Negativos(VN):** Número de casos que la prueba declara negativos y que son realmente negativos.

Una vez aclarado estos términos, presentamos a continuación las métricas que se han utilizado para medir la calidad de los modelos.

- **Sensibilidad (precision):** Proporción de casos positivos que están bien detectadas por la prueba. La definición matemática es:

$$Sensibilidad = \frac{VP}{(VP + FN)}$$

- **Especificidad (recall):** proporción de casos negativos que son bien detectadas por la prueba. La definición matemática es:

$$Especificidad = \frac{VN}{(VN + FP)}$$

- **F1-score:** Media armónica entre la precisión y la sensibilidad. La definición matemática es:

$$\text{F1-score} = 2 * \frac{Sensibilidad * Especificidad}{Sensibilidad + Especificidad}$$

- **Macro average:** métrica que calcula el promedio entre precisión y sensibilidad independientemente para cada clase (por lo tanto, tratará a todas las clases por igual, independientemente de la aportación de cada etiqueta) esto se calcula a partir de la sensibilidad media y especificidad media:

  - Macro-average precision = $\frac{precision_1 + precision_2}{2}$

- Macro-average recall = $\frac{recall_1+recall_2}{2}$
- Macro-average F1-score = $2 * \frac{MAprecision*MArecall}{MAprecision+MArecall}$

- **Matriz de confusión:** matriz que nos indica el número de aciertos y fallos para cada etiqueta. Cómo en este TFM se evalúan dos etiquetas, AD y CN, se van a definir el significado de cada elemento de la matriz.

$$\begin{array}{cc} & \text{Predicción} \\ & \begin{array}{cc} \text{AD} & \text{CN} \end{array} \\ \begin{array}{c} \text{Valor} \\ \text{real} \end{array} & \begin{array}{c} \text{AD} \\ \text{CN} \end{array} \begin{bmatrix} VP & FN \\ FN & VN \end{bmatrix} \end{array}$$

- **Análisis ROC (Receiver Operating Characteristic):** una curva ROC es una representación gráfica del funcionamiento de un clasificador para todos los contextos posibles. Se puede utilizar el porcentaje de aciertos para elegir el mejor modelo, pero hemos visto que no es muy adecuado si por ejemplo la muestra está desequilibrada; Existe otra medida que se basa en las curvas ROC y que es inmune al desequilibrio en la muestra: el área bajo la curva de la curva ROC (Area Under the Curve AUC). Cuanto mayor sea esa área, más cerca está la curva ROC de la esquina superior izquierda, y por tanto mejor es la separación de los datos (independientemente del desequilibrio de la muestra)

5.2 Primeros experimentos

La experimentación inicial, tomando como base el *dataset* de ADNI y revisando el estado del arte de los estudios realizados a dicho *dataset*, pudimos constatar que usar más de dos etiquetas en este tipo de datos es una tarea de clasificación bastante compleja, ya que la clasificación de los diferentes estadios de la enfermedad se

originaron por criterios de los especialistas durante el tratamiento, con lo cual, morfológicamente no se diferencian a menos que se encuentre en un estado bastante avanzado.

Las primeras pruebas que se realizaron fueron usando el *dataset* de imágenes 2D sin ningún tipo de preprocesamiento, con una topología de red convolucional profunda para el reconocimiento de objetos desarrollada y capacitada por el Grupo de Geometría Visual (VGG) de Oxford, en las versiones VGG-16 y VGG-19 e implementada en las librerías de Keras. Los resultados se aproximaban al estado del arte que hemos descrito en la sección 1.3. Obteniendo una precisión del 98Las primeras pruebas que se realizaron fueron usando el *dataset* de imágenes 2D sin ningún tipo de pre-procesamiento, con una topología de red convolucional profunda para el reconocimiento de objetos desarrollada y capacitada por el Grupo de Geometría Visual (VGG) de Oxford, en las versiones VGG-16 y VGG-19 e implementada en las librerías de Keras. Los resultados se aproximaban al estado del arte que hemos descrito en la sección 1.6.4. Obteniendo una precisión del 98 %. Se puede constatar que estos resultados eran cuestionables, ya que en una parte del conjunto de imágenes tan solo aparecen zonas de la cabeza y cuello en las que no tiene sentido estudiar la enfermedad, con lo que después de revisar los resultados se concluyó, que los modelos no aprendían la enfermedad sino que aprendían a reconocer la morfología del paciente en la imagen. A partir de este punto, se planteó organizar los datos en las distintas particiones descritas en la sección 4.2, en donde un sujeto no puede pertenecer a más de una partición, garantizando de esta forma, independencia estadística de los datos; sin embargo los resultados obtenidos en el entrenamiento ya no eran favorables con los datos sin llevar a cabo el preprocesado.

Definidas las particiones del conjunto de datos, se realizó la experimentación siguiendo dos vertientes comunes en el estado del arte. Por un lado, se utilizó la herramienta *FreeSurfer* para procesar el *dataset* (Usando el cluster HPC del CIPF). y por el otro, se realizó el preprocesado de BET + FLIRT.

Se empezó a probar redes convolucionales con distintas estructuras más allá de las predefinidas en Keras. Estos modelos se empezaron a probar usando el preprocesado con BET. La hipótesis eran ver si lo que dificulta a la red para aprender era los tejidos que no pertenecían al cerebro. Después de entrenar los modelos propuestos en ese momento, las redes siguen sin aprender la enfermedad dejando la precisión donde no llegaba a superar el voto mayoritario de la clase. En ese momento se tomó la decisión de registrar todos los cerebros al mismo espacio, pero no se empezaron a ver resultados favorables.

La experimentación se realiza usando la IRM por sujeto como una imagen en 3 dimensiones (3D), como también extrayendo los cortes de la IRM en *n* imágenes de 2 dimensiones (2D). Para el caso de las imagenes 2D, las alternativas para evaluar la partición de test fueron:

- **Evaluación individual:** etiquetar cada corte
- **Evaluación por sujeto:** Se evalúan todos los cortes de un sujeto y se promedian los resultados para clasificar al paciente en una clase..

La evaluación por sujeto, comparado con la evaluación individual, dio mejores resultados pero no superan la votación mayoritaria. Una hipótesis que puede explicar estos resultados es que entrenar las CNN con imágenes en 3D dotan de más contexto de la estructura y su forma, que las imágenes en 2D, por consiguiente facilitan el aprendizaje.

5.3 Resultados con BET + FLIRT

En esta parte del estudio observamos que las clases no tienen la misma proporción (no están bien balanceadas) por lo tanto como la evaluación de los modelos es sensible a la distribución. Debido a eso y como es obvio, una distribución desbalanceada

a la hora de evaluar un clasificador, el porcentaje de acierto debería ser mayor, en la clase mayoritaria.Es decir, el clasificador tiene que ser mejor que el clasificador trivial.

En problemas de muestras desbalanceadas, sucede que aprende mejor la clase mayoritaria a costa de aprender mal la minoritaria. Para solucionar ésto, alguna: soluciones son:

- **Solución 1:** aprender varios modelos (p. ej. con distintos algoritmos) y seleccionar aquel que aprenda bien la clase minoritaria.
- **Solución 2:** remuestreo.
  - **Sub-muestreo:** eliminar datos de la clase mayoritaria para equilibrarse con la minoritaria.
  - **Sobre-muestreo:** replicar datos de la clase minoritaria (se trata del método utilizado en la partición de datos y explicado en el figura 4.5)

En esta ocasión utilizamos la matriz de confusión para quedarnos con el mejor clasificador, que en este caso es el que minimiza los FP, puesto que lo peor es presuponer que no tiene la enfermedad cuando realmente la tiene.

Esta técnica se ha realizado tanto en imágenes 3D como en imágenes 2D obteniendo resultados por encima del voto mayoritario en imagen 3D.

A continuación se muestra en la tabla 5.1 los mejores modelos que se han obtenido en los dos *datasets* por separado.

El modelo que mejores resultados ha obtenido ha sido el **modelo 17** el cual es el modelo descrito en la figura 5.2. En la figura 5.1, está representada la curva ROC del modelo 17 obtenido para el *dataset* ADNI.

Tabla 5.1: Resultados obtenidos a partir del preprocesado de *BET + FLIRT* para los *datasets* de ADNI y OASIS

model	Acuracy	AD				CN				Macro average				Matrix confusion
		precision	recall	f1-score	support	precision	recall	f1-score	support	precision	recall	f1-score	support	
ADNI Majority voting	0.7065	0.00	0.00	0.00	59	0.70	1.00	0.82	142	0.35	0.50	0.41	201	$\begin{bmatrix} 0 & 59 \\ 0 & 142 \end{bmatrix}$
adni-3D-17-a_adadelta_flirt	0.6965	0.49	0.92	0.64	59	0.95	0.61	0.74	142	0.72	0.76	0.69	201	$\begin{bmatrix} 54 & 5 \\ 56 & 86 \end{bmatrix}$
adni-3D-17-a_adadelta_DA_flirt-last	0.8458	0.74	0.73	0.74	59	0.89	0.89	0.89	142	0.81	0.81	0.81	201	$\begin{bmatrix} 43 & 16 \\ 15 & 127 \end{bmatrix}$
adni-cnn-classifier-3D-8-a_adadelta_DA_flirt-acc	0.7910	0.63	0.71	0.67	59	0.87	0.82	0.85	142	0.75	0.77	0.76	201	$\begin{bmatrix} 42 & 17 \\ 25 & 117 \end{bmatrix}$
OASIS Majority voting	0.7970	0.00	0.00	0.00	94	0.80	1.00	0.89	369	0.40	0.50	0.44	463	$\begin{bmatrix} 0 & 94 \\ 0 & 369 \end{bmatrix}$
oasis-3D-17-a_SGD_DA_flirt	0.8035	0.80	0.04	0.08	94	0.80	1.00	0.89	369	0.80	0.52	0.49	463	$\begin{bmatrix} 4 & 90 \\ 1 & 368 \end{bmatrix}$
oasis-3D-17-a_SGD_flirt	0.7235	0.40	0.77	0.53	94	0.92	0.71	0.80	369	0.66	0.74	0.67	463	$\begin{bmatrix} 72 & 22 \\ 106 & 263 \end{bmatrix}$
oasis-3D-8-a_adadelta_DA_flirt	0.7192	0.32	0.35	0.34	94	0.83	0.81	0.82	369	0.58	0.58	0.58	463	$\begin{bmatrix} 33 & 61 \\ 69 & 300 \end{bmatrix}$
oasis-3D-8-a_adadelta_flirt	0.8013	0.67	0.04	0.08	94	0.80	0.99	0.89	369	0.73	0.52	0.48	463	$\begin{bmatrix} 4 & 90 \\ 2 & 367 \end{bmatrix}$

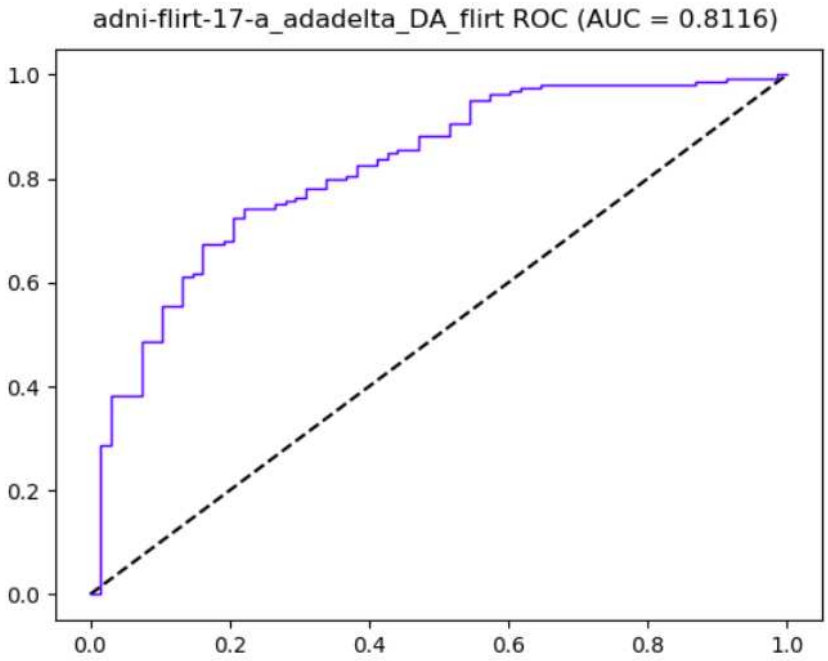


Figura 5.1: curva AROC del modelo 17 para los datos de ADNI 3D generados con BET+FLIRT.

El **modelo 17** se denomina CNN Bilineal [34] este tipo de red neuronal se caracteriza por tener dos bloques convolucionales que se inicializan aleatoriamente, diferentes procesos de aprendizaje generarán distintas redes incluso a partir del mismo conjunto de datos disponibles, lo que nos permite caracterizar de manera distinta una misma imagen.

Los estudios que se han realizado con este tipo de redes, han demostrado que funcionan bien cuando las imágenes que se pretenden clasificar, tienen características similares. Ejemplos de ello sería a la hora de clasificar especies de pájaros, ya que tienen características similares.

Llevando esta premisa a nuestro problema, ya que todas las imágenes son cerebrales en donde a nivel estructural tienen las mismas características, lo que los diferencia entre ellas son pequeñas diferencias como la asimetría entre los hemisferios del cerebro, la forma de las partes anatómicas, etc. Por ello, era imperativo probar este tipo

de redes CNN ya que realizan una clasificación más fina cuando las características en la entrada del sistema son similares.

En el caso del *dataset* OASIS, el mejor resultado obtenido, observando los valores Macro-average, también corresponde con el **Modelo 17**, pese a tener un accuracy peor que el de voto mayoritario, este logra aprender los sujetos de la clase minoritaria AD con respecto al resto de modelos. Pese a ser el mejor modelo, este sigue sin ser muy óptimo debido a que el número de FN es muy alto. Por tanto, no se ha encontrado un modelo que clasifique correctamente este dataset.

5.4 Resultados con *FreeSurfer*

En este apartado, se utilizó el dataset de ADNI para los experimentos realizados. De este proceso, se decidió utilizar las imágenes que contenían el parénquima cerebral. Esta imagen fue obtenida por el preprocesado de *FreeSurfer* durante el proceso de segmentación, ya que se necesita eliminar las áreas del parénquima cerebral que no están relacionadas con la EA, como habíamos realizado con el preprocesado con BET. Para ello utilizamos la máscara del **Aparc 2009** porque contiene todas las áreas que implica, en mayor o menor medida a la EA.

En la tabla5.2 se muestran los mejores modelos que se han obtenido con el dataset ADNI. Este ha sido el **modelo 8** el cual se encuentra descrito en la figura 4.12, donde presentamos una red convolucional con el número de filtros prefijados a 32 por convolución y la primera convolución es un filtro de $5x5$ y el resto de $3x3$. En la figura 5.3, se representa la curva ROC del **modelo 8** obtenido para el *dataset* ADNI.

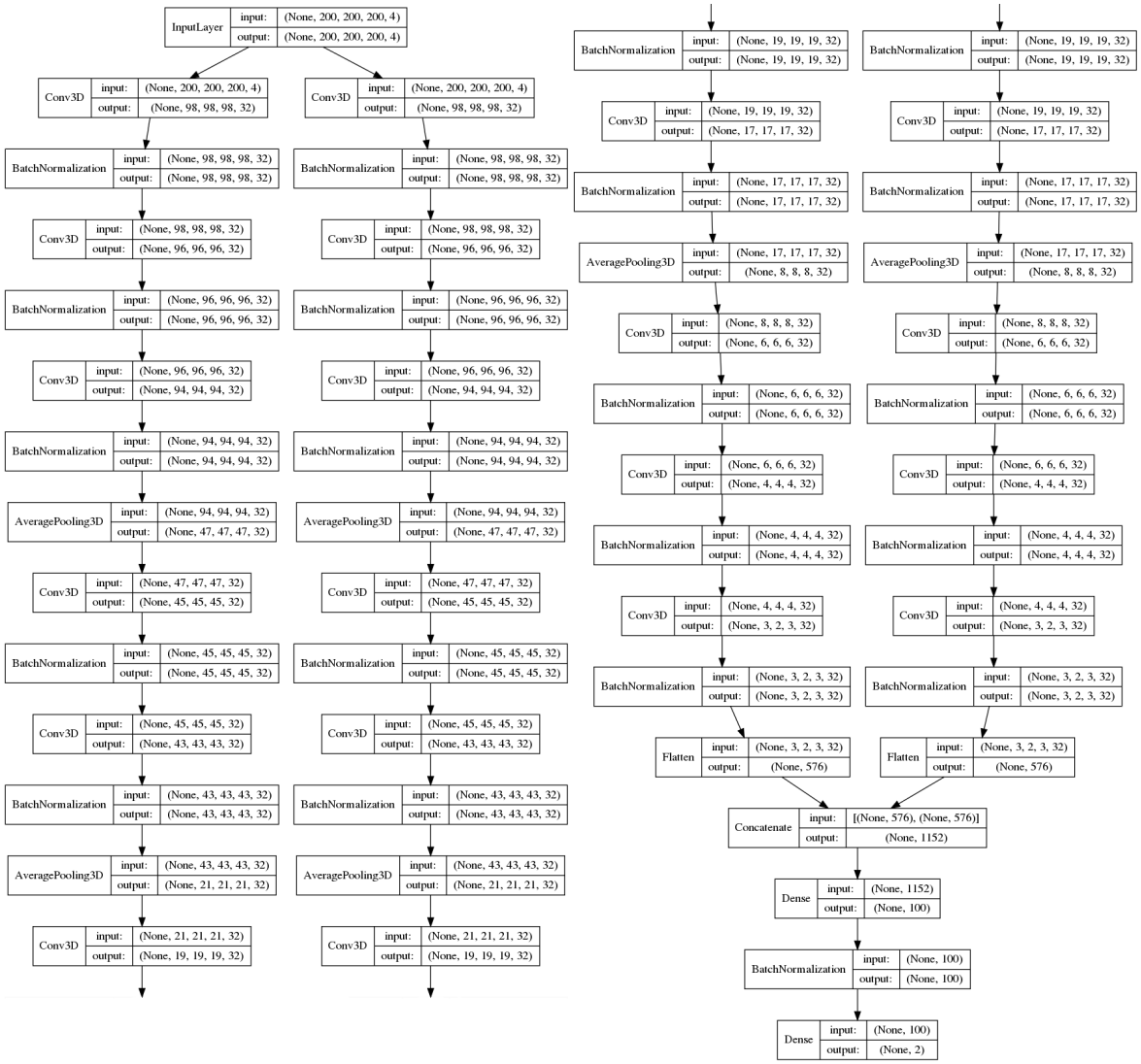


Figura 5.2: Esquema de un a red neuronal Bilineal donde se muestran las capas utilizadas de uno de los modelos pertenecientes a la experimentación.

Tabla 5.2: Resultados obtenidos a partir de la combinación de la imagen procesada con *FreeSurfer* con la segmentación

model	Acuracy	AD				CN				Macro average				Matrix confusion
		precision	recall	f1-score	support	precision	recall	f1-score	support	precision	recall	f1-score	support	
Adni Majority voting	0.7017	0.00	0.00	0.00	68	0.70	1.00	0.82	160	0.35	0.50	0.41	228	$\begin{bmatrix} 0 & 68 \\ 0 & 160 \end{bmatrix}$
adni-3D-8-d_adadelta (imagen y mascara RGB)	0.7895	0.64	0.66	0.65	68	0.85	0.84	0.82	160	0.75	0.75	0.75	228	$\begin{bmatrix} 45 & 23 \\ 25 & 135 \end{bmatrix}$
adni-3D-13-a_adadelta (imagen y mascara RGB)	0.7325	0.53	0.85	0.66	68	0.92	0.68	0.78	160	0.72	0.77	0.72	228	$\begin{bmatrix} 58 & 10 \\ 51 & 109 \end{bmatrix}$
adni-3D-8-d_adadelta_mask_rgb-lost(mascara RGB)	0.7763	0.60	0.76	0.67	68	0.89	0.78	0.83	160	0.74	0.86	0.75	228	$\begin{bmatrix} 52 & 16 \\ 35 & 125 \end{bmatrix}$

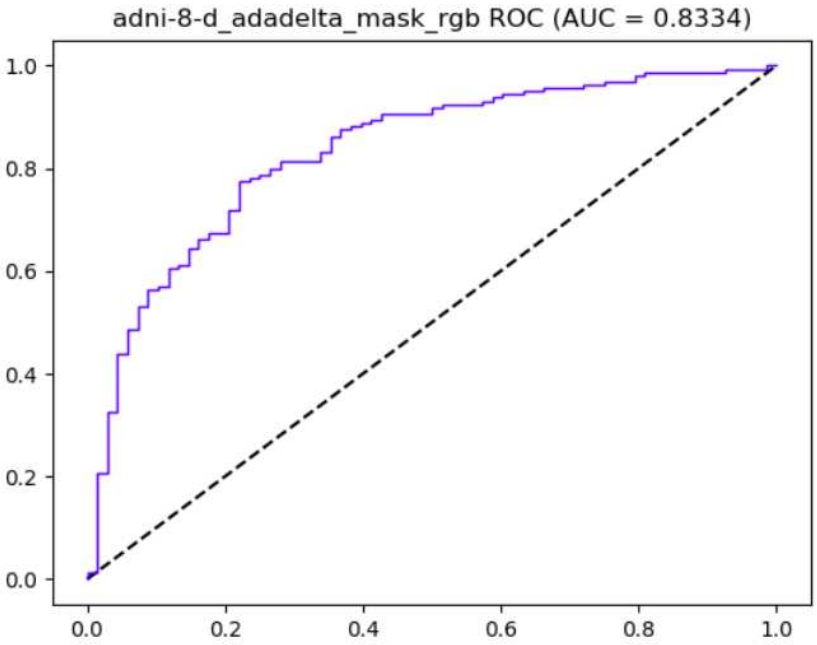


Figura 5.3: Curva AROC del modelo 8 para los datos de la segmentación obtenida en FreeSurfer con ADNI.

En la tabla 5.3, tenemos los resultados obtenidos a partir de los datos morfológicos extraídos en *FreeSurfer*. El mejor resultado se ha obtenido a partir del método clásico SCV y también se ha conseguido una red neuronal que se equipara al método tradicional. En las figuras 5.4 y 5.5 se puede observar las curvas ROC generadas a partir de estos dos modelos.

Tabla 5.3: Resultados obtenidos para los datos morfológicos extraídos con *FreeSurfer*

model	Acuracy	AD				CN				Macro average				Matrix confusion
		precision	recall	f1-score	support	precision	recall	f1-score	support	precision	recall	f1-score	support	
Adni Majority voting	0.7129	0.00	0.00	0.00	60	0.71	1.00	0.83	149	0.36	0.50	0.42	209	$\begin{bmatrix} 0 & 68 \\ 0 & 160 \end{bmatrix}$
Modelo Fully conected	0.8852	0.81	0.78	0.80	60	0.91	0.93	0.92	149	0.86	0.85	0.85	209	$\begin{bmatrix} 47 & 13 \\ 11 & 138 \end{bmatrix}$
LinearDiscriminantAnalysis	0.8708	0.84	0.68	0.75	60	0.88	0.95	0.91	149	0.86	0.81	0.83	209	$\begin{bmatrix} 48 & 15 \\ 15 & 138 \end{bmatrix}$
SVC (kernel=rbf)	0.8947	0.84	0.78	0.81	60	0.92	0.94	0.93	149	0.88	0.86	0.87	209	$\begin{bmatrix} 47 & 13 \\ 9 & 140 \end{bmatrix}$
AdaBoostClassifier	0.8851	0.83	0.75	0.79	60	0.90	0.94	0.92	149	0.87	0.84	0.86	209	$\begin{bmatrix} 45 & 15 \\ 9 & 140 \end{bmatrix}$

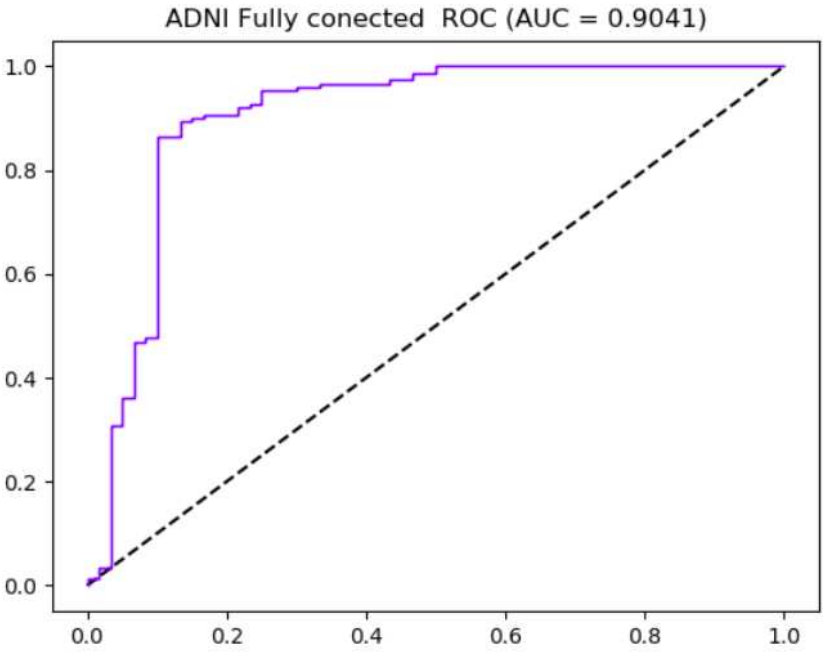


Figura 5.4: Curva AROC del modelo Fully Conected para los datos morfológicos obtenidos en *FreeSurfer* con ADNI.

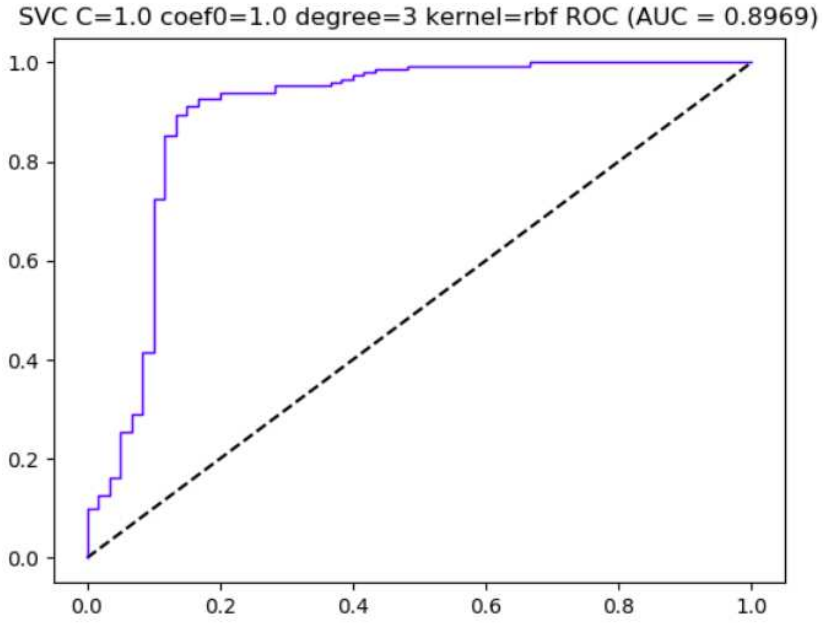


Figura 5.5: Curva AROC del modelo SCV para los datos morfológicos obtenidos en *FreeSurfer* con ADNI.

CAPÍTULO 6

Discusión

Durante el último año se han realizado importantes avances en el área de IA. La metodología presentada en este TFM pretende realizar un recorrido por las técnicas más importantes que se han planteado en esta materia y realizaremos alguna crítica con respecto al estado del arte. Se pretende discutir los resultados obtenidos a partir del marco metodológico utilizado y con los conjuntos de datos seleccionados obtenidos de bases de datos internacionales y en abierto, estas son los *datasets* ADNI y OASIS. También comentaremos las limitaciones acaecidas durante la elaboración de este trabajo.

6.1 Discusión de los resultados obtenidos con los distintos modelos

Con respecto a los resultados obtenidos en los inicios del trabajo, se observó que trabajar con más de un estadío de la enfermedad, en el caso de ADNI, con todas las etiquetas que nos ofrece este *dataset* (MCI, EMCI y LMCI), dificulta enormemente el

aprendizaje del modelo porque el nivel de atrofia cerebral en MCI es muy leve con respecto a las distintas fases y en estadios tempranos al AD.

Por otro lado, cuando se realizó un primer abordaje del estudio, utilizando los datos en crudo o sólo aplicando un preprocesado mínimo (extracción del parénquima cerebral) no ofrecieron resultados concluyentes para diferenciar la EA. Se empezaron a ver modelos con resultados más alentadores, a partir de que se realizó un registro lineal con FLIRT, que transforma todos los cerebros a un espacio común y que los hace más comparables.

En el caso del *dataset* de OASIS, los resultados obtenidos con FLIRT difieren de los obtenidos en ADNI comparándolos a partir del mismo preprocesado (BET+FLIRT) y utilizando los mismos modelos. Esto puede ser debido a que el *dataset* de OASIS no refleja las características morfológicas de la enfermedad.

Cuando comparamos los modelos basados en 2D y 3D, se observa que los 2D no consiguen resultados mejores que los de 3D, esto es debido a que la pérdida del contexto de localización del corte puede ser necesaria para la realizar una correcta clasificación de la enfermedad.

En cuanto a los resultados obtenidos aplicando el preprocesado con *FreeSurfer*, se ha podido observar que obtienen resultados que superan el voto mayoritario en donde el mejor resultado se encuentra con la utilización de la segmentación 3D. También se han constatado que usando sólo la segmentación en RGB, el modelo entrenado tiende a clasificar mejor a los casos de AD.

Por último, y después de realizar entrenamientos con múltiples combinaciones de topologías de redes CNN, la utilización de los datos morfológicos, extraídos con *FreeSurfer*, han obtenido los mejores resultados de todo el trabajo. Los resultados son comparables tanto con DNN como con métodos clásicos. Para concluir, se demuestra que gracias a los datos en abierto de ADNI, se puede llegar a caracterizar en

cierta medida la patología estudiada gracias a la morfología cerebral. Aún así queda mucho por hacer y todavía queda por probar diferentes topologías y configuraciones.

6.2 Limitaciones

Al trabajar con IRM, se ha podido observar que realizar una partición de los datos correcta para el entrenamiento de la enfermedad es fundamental para la obtención de buenos resultados. Este TFM ha sido posible realizarlo gracias a la comunidad open data, que pone a disposición de los investigadores estos datos para avanzar en nuevos hallazgos, inclusive algunas serendipias que nos ayudan a dar saltos cualitativos en I+D. Aun así, los conjuntos de datos públicos, no son de gran calidad aunque tienen etiquetas asociados que son revisadas por expertos, nos hemos encontrado que se dispone de casos bien balanceados y esto es un sesgo muy importante 1.6.4. Por esta razón los estudios realizados en el estado del arte que hemos estudiado, no definen con exactitud la forma de particionar los datos y los modelos utilizados para clasificar correctamente el estadio de la enfermedad, además de que hace que sea irreproducible.

Como ya hemos comentado en múltiples ocasiones, en este estudio tuvimos la oportunidad de trabajar con dos conjuntos de datos con clases desbalanceadas (no tienen la misma proporción). Esto supuso un problema muy importante en la fase de entrenamiento con clasificadores de machine learning como Random Forests o DNN que no lidian muy bien con *datasets* de entrenamiento desbalanceados ya que son sensibles a las proporciones de las diferentes clases. Se pretende enfatizar este punto ya que puede ser particularmente problemático cuando estamos interesados en la clasificación correcta de una clase "rara" o minoritaria, por esto es altamente recomendable llevar a cabo un análisis de sensibilidad del clasificador con respecto a la distribución de los datos de entrenamiento.

Un factor muy limitante a la hora de trabajar con los *datasets*, ha sido el tamaño muestral para el uso de las técnicas de *Deep Learning*, que en este caso claramente **es insuficiente**. En el estado del arte, se están utilizando estas técnicas de *Deep Learning* para entrenar estos conjuntos de datos pero, en la práctica, creemos que se necesita una cantidad mucho mayor de datos para utilizar estas técnicas después de aplicar el particionado que se propone.

CAPÍTULO 7

Conclusiones, recomendaciones y líneas de futuro

1. En este trabajo se han analizado y probado distintas técnicas de preprocesado de datos que principalmente se utiliza para el tratamiento de IRM y en base a los resultados obtenidos, se ha conseguido delimitar cuál o cuáles han sido más apropiadas para el problema que se aborda en este trabajo.

2. Estas herramientas han sido principalmente FSL y *FreeSurfer*. Con FSL se ha podido realizar una extracción del parénquima cerebral, es decir, la zona de interés y se ha podido registrar a un espacio común, vital para comparar morfologías. Por otro lado, se dispone de *FreeSurfer* que tiene un software de los más utilizados por su fiabilidad en IRM en donde preprocesa y extrae información que ha sido utilizada en múltiples experimentos.

3. En este trabajo se han analizado y probado distintas técnicas de preprocesado de datos que principalmente se utiliza para el tratamiento de IRM y en base a los resultados obtenidos, se ha conseguido delimitar cuál o cuáles han sido más apropiadas para el problema que se aborda en este trabajo.

4. Estas herramientas han sido principalmente FSL y *FreeSurfer*. Con FSL se ha podido realizar una extracción del parénquima cerebral, es decir, la zona de interés y se ha podido registrar a un espacio común, vital para comparar morfologías. Por otro lado, se dispone de *FreeSurfer* que tiene un software de los más utilizados por su fiabilidad en IRM en donde preprocesa y extrae información que ha sido utilizada en múltiples experimentos.

5. Se ha podido observar que con *FreeSurfer* se consiguen los mejores resultados a partir de los datos morfológicos que se extraen de cada sujeto y seleccionando aquellas variables que afectan en mayor medida.

6. Los resultados obtenidos en el trabajo no llegan a alcanzar a los del estado del arte. Estos resultados se han intentado replicar sin éxito debido a que, en muchos casos, no especifican cómo se han realizado las particiones de los sujetos a la hora de entrenar el modelo. En algunos casos, tampoco se podía reproducir con exactitud los modelos o los preprocesados utilizados al detalle ya que no se da la información requerida para replicar con exactitud los experimentos. Todo esto pone en tela de juicio la reproducibilidad en la investigación.

7. En cuanto a los *datasets* empleados, se puede decir que en ADNI se han encontrado modelos que puede aprender a generalizar la enfermedad, sin embargo en OASIS, no se a encontrado todavía un modelo que nos permita clasificar correctamente la EA. También hay que recalcar que estos *dataset* todavía no están preparados para su uso de técnicas en *Deep Learning* ya que hay pocos datos por etiqueta y existe un desbalance considerable de las mismas.

7.1 Recomendación y líneas de futuro

La obtención de imágenes de RM no es fácil, requiere de mucho tiempo y dinero extraer las imágenes para que un experto las clasifique. Aparte, estás RM deben estar anonimizadas para cumplir la legislación vigente, ya que son datos sensibles que identifican al paciente. Esta casuística, conlleva a que la creación de *dataset* voluminosos sean muy difíciles de adquirir.

Una de las posibles mejoras que se pueden realizar, es intentar unificar estos *datasets* y buscar nuevos casos que añadir, para intentar aumentar el número de casos necesarios para poder trabajar en técnicas de *Deep Learning*.

Otra vía de estudio es que, aunque sabemos que la EA se puede diferenciar de los casos control, debido a que las diferencias a nivel estructural son evidentes; un avance importante sería detectar la EA o otro tipo de demencia en sus fases iniciales, ya que podría darle al médico un biomarcador de la progresión de la EA temprana y al paciente le un tratamiento más adecuado y efectivo que mejorará tanto la esperanza como su calidad de vida que cuando se detecta en fases intermedias o finales.

La propuesta inicial de este TFM ha sido proponer un biomarcador de imagen médica con una prueba diagnóstica económicamente sostenible como la IRM, Aunque ante la duda, siempre sería necesario agregar otros tipos de pruebas diagnósticas que se utilizan habitualmente en el diagnóstico de la demencia.

Bibliografía

[1] Fundación Pascual Maragall. Hablemos de alzheimer. `https://blog.fpmaragall.org/diagnostico-alzheimer` (2019). Online; acceso: 17 de septiembre de 2019.

[2] Fundación Pascual Maragall. Introducción a la enfermedad. `https://fpmaragall.org/alzheimer-enfermedad/enfermedad-alzheimer` (2019). Online; acceso: 17 de septiembre de 2019.

[3] Prince, M. *et al.* The global prevalence of dementia: a systematic review and metaanalysis. *Alzheimer's & dementia* **9**, 63–75 (2013).

[4] Tola-Arribas, M. A. *et al.* Prevalence of dementia and subtypes in valladolid, northwestern spain: the deminvall study. *PloS one* **8**, e77688 (2013).

[5] Gascón-Bayarri, J. *et al.* Prevalence of dementia subtypes in el prat de llobregat, catalonia, spain: the praticon study. *Neuroepidemiology* **28**, 224–234 (2007).

[6] Gavrila, D. *et al.* Prevalence of dementia and cognitive impairment in southeastern spain: the ariadna study. *Acta Neurologica Scandinavica* **120**, 300–307 (2009).

[7] de Pedro-Cuesta, J. *et al.* Prevalence of dementia and major dementia subtypes in spanish populations: a reanalysis of dementia prevalence surveys, 1990-2008. *BMC neurology* **9**, 55 (2009).

[8] Fiest, K. M. *et al.* The prevalence and incidence of dementia due to alzheimer's disease: a systematic review and meta-analysis. *Canadian Journal of Neurological Sciences* **43**, S51–S82 (2016).

[9] Prince, M. *et al.* The global impact of dementia: an analysis of prevalence, incidence, cost and trends. *World Alzheimer Report* **2015** (2015).

[10] Abubakar, I., Tillmann, T. & Banerjee, A. Global, regional, and national age-sex specific all-cause and cause-specific mortality for 240 causes of death, 1990-2013: a systematic analysis for the global burden of disease study 2013. *Lancet* **385**, 117–171 (2015).

[11] Organización Mundial de la Salud. Demencia. `https://www.who.int/es/news-room/fact-sheets/detail/dementia` (2019). Online; acceso: 17 de septiembre de 2019.

[12] Neuroscience News. New insights into what may go awry in brains of alzheimer's patients. `https://neurosciencenews.com/alzheimers-brain-insights-14749/` (2019). Online; acceso: 17 de septiembre de 2019.

[13] Alzheimer Disease Neuroimage Initiative (ADNI). Diseño del estudio. `http://adni.loni.usc.edu/study-design/#background-container` (2019). Online; acceso: 17 de septiembre de 2019.

[14] de la Iglesia Vayá, M. *Técnicas de conectividad cerebral y transferencia de información aplicado al estudio de la esquizofrenia*. Ph.D. thesis (2011).

[15] Atenas, T. L., Díaz, E. C., Quiroga, J. C., Arancibia, S. U. & Rodríguez, C. C. Resonancia magnética funcional: principios básicos y aplicaciones en neurociencias. *Radiología* **60**, 368–377 (2018).

[16] Huang, Y. *et al.* Diagnosis of alzheimer's disease via multi-modality 3d convolutional neural network. *Frontiers in Neuroscience* **13** (2019).

[17] Lu, D., Popuri, K., Ding, G. W., Balachandar, R. & Beg, M. F. Multimodal and multiscale deep neural networks for the early diagnosis of alzheimer's disease using structural mr and fdg-pet images. *Scientific reports* **8**, 5697 (2018).

[18] Wang, H. *et al.* Ensemble of 3d densely connected convolutional network for diagnosis of mild cognitive impairment and alzheimer's disease. *Neurocomputing* **333**, 145–156 (2019).

[19] Sarraf, S., Tofighi, G. *et al.* Deepad: Alzheimer' s disease classification via deep convolutional neural networks using mri and fmri. *BioRxiv* 070441 (2016).

[20] Dimitriadis, S. I., Liparas, D., Tsolaki, M. N., Initiative, A. D. N. *et al.* Random forest feature selection, fusion and ensemble strategy: Combining multiple morphological mri measures to discriminate among healhy elderly, mci, cmci and alzheimer's disease patients: From the alzheimer's disease neuroimaging initiative (adni) database. *Journal of neuroscience methods* **302**, 14–23 (2018).

[21] Aderghal, K., Boissenin, M., Benois-Pineau, J., Catheline, G. & Afdel, K. Classification of smri for ad diagnosis with convolutional neuronal networks: A pilot 2-d+ *epsilon* study on i. In *International Conference on Multimedia Modeling*, 690–701 (Springer, 2017).

[22] Payan, A. & Montana, G. Predicting alzheimer's disease: a neuroimaging study with 3d convolutional neural networks. *arXiv preprint arXiv:1502.02506* (2015).

[23] Open Access Series of Imaging Studies (OASIS). Oasis brains datasets. `https://www.oasis-brains.org` (2019). Online; acceso: 17 de septiembre de 2019.

[24] Pérez, F. & Granger, B. E. Ipython: a system for interactive scientific computing. *Computing in Science & Engineering* **9**, 21–29 (2007).

[25] NIfTI. Neuroimaging informatics technology initiative, nifti. `https://nifti.nimh.nih.gov/background` (2005). Online; acceso: 17 de septiembre de 2019.

[26] Jenkinson, M., Beckmann, C. F., Behrens, T. E., Woolrich, M. W. & Smith, S. M. Fsl. *Neuroimage* **62**, 782–790 (2012).

[27] Fischl, B. Freesurfer. *Neuroimage* **62**, 774–781 (2012).

[28] Chollet, F. *et al.* Keras. `https://keras.io` (2015).

[29] Abadi, M. *et al.* TensorFlow: Large-scale machine learning on heterogeneous systems (2015). URL `http://tensorflow.org/`. Software available from tensorflow.org.

[30] Analytics, C. Anaconda software distribution, computer software. vers. 2-2.4. 0 (2015).

[31] Nvidia. Nvidia gtx 1080 whitepaper. `https://images.nvidia.com/content/pdf/quadro/data-sh` (2016). Online; acceso: 17 de septiembre de 2019.

[32] Nvidia. Nvidia gtx 1080 whitepaper. `https://www.gigabyte.com/es/Graphics-Card/GV-N1080WF` (2017). Online; acceso: 17 de septiembre de 2019.

[33] Nvidia. Nvidia tesla v100 gpu whitepaper. `https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepa` (2018). Online; acceso: 17 de septiembre de 2019.

[34] Lin, T.-Y., RoyChowdhury, A. & Maji, S. Bilinear cnn models for fine-grained visual recognition. In *Proceedings of the IEEE international conference on computer vision*, 1449–1457 (2015).

[35] Fundación Pascual Maragall. Hablemos de alzheimer. `https://fsl.fmrib.ox.ac.uk/fsl/fslwiki/` (2019). Online; acceso: 17 de septiembre de 2019.

[36] Woolrich, M. W. *et al.* Bayesian analysis of neuroimaging data in fsl. *Neuroimage* **45**, S173–S186 (2009).

[37] Smith, S. M. *et al.* Advances in functional and structural mr image analysis and implementation as fsl. *Neuroimage* **23**, S208–S219 (2004).

[38] Huang, G., Liu, Z., Van Der Maaten, L. & Weinberger, K. Q. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 4700–4708 (2017).

[39] Jenkinson, M., Bannister, P., Brady, M. & Smith, S. Improved optimization for the robust and accurate linear registration and motion correction of brain images. *Neuroimage* **17**, 825–841 (2002).

[40] Jenkinson, M. & Smith, S. A global optimisation method for robust affine registration of brain images. *Medical image analysis* **5**, 143–156 (2001).

[41] Marcus, C., Mena, E. & Subramaniam, R. M. Brain pet in the diagnosis of alzheimer's disease. *Clinical nuclear medicine* **39**, e413 (2014).

[42] Popescu, V. *et al.* Optimizing parameter choice for fsl-brain extraction tool (bet) on 3d t1 images in multiple sclerosis. *Neuroimage* **61**, 1484–1494 (2012).

[43] Smith, S. M. Fast robust automated brain extraction. *Human brain mapping* **17**, 143–155 (2002).

Anexo I: Código empleado

En este anexo vamos a visualizar el código utilizado para el trabajo realizado.

7.2 Preprocesado con paralelización en cpu

```
import os
import numpy as np
import nibabel as nib
import pandas as pd
from nipype.interfaces import fsl
from multiprocessing import Lock, Queue
from collections import deque
import threading
from bs4 import BeautifulSoup
import SimpleITK as sitk
import re
from lib.createdict import create_dict_freesurfer_codes

freesurfer_codes = create_dict_freesurfer_codes()

categorical_dict={"AD":np.array([1, 0, 0, 0, 0]),
      "CN":np.array([0, 1, 0, 0, 0]),
```

```
18        "EMCI":np.array([0, 0, 1, 0, 0]),
19        "LMCI":np.array([0, 0, 0, 1, 0]),
20        "MCI":np.array([0, 0, 0, 0, 1])
21       }
22
23  work_dir = '.'
24  origin_data_dir = "adni_opt"
25  origin_data_dir_oasis = "oasis3_opt"
26  freesurfer_dir = "adni_freesurfer_origin"
27  freesurfer_brainmask_file= os.path.join(work_dir,
      ↪ freesurfer_dir, "brainmask.txt")
28  freesurfer_aparc_aseg_file = os.path.join(work_dir,
      ↪ freesurfer_dir, "aparc_plus_aseg_files.txt")
29
30  data_dir_3d = "adni_opt_3d"
31  data_dir_3d_mask = "adni_opt_3d_parenquima"
32  #data_dir_3d_flirt = "adni_opt_3d_flirt"
33  data_dir_3d_flirt = "oasis_opt_3d_flirt2"
34  data_dir_3d_free = "adni_opt_3d_freemask_rgb"
35
36  data_dir_2d_mask = "adni_opt_2d_parenquima"
37  data_dir_2d_3c = "adni_opt_2d_3c"
38  #data_dir_2d_flirt = "adni_opt_2d_flirt"
39  data_dir_2d_flirt = "oasis_opt_2d_flirt2"
40  data_dir_2d_free = "adni_opt_2d_freemask_rgb_ax"
41
42  df_adni = pd.read_csv( origin_data_dir + "/
      ↪ ADNI_T1_MPRAGE_2_19_2019.csv" )
```

```
43 df_oasis= pd.read_csv( origin_data_dir_oasis + "/
     ↪ oasis_subjects.csv" )
44 df_oasis_subjects= pd.read_csv( origin_data_dir_oasis + "/
     ↪ oasis_real_subjects.csv" )
45 lock_queue_filenames = Lock()
46 lock_queue_2d = Lock()
47 lock_queue_2d_masks = Lock()
48 lock_queue_2d_3c = Lock()
49 lock_queue_2d_flirt = Lock()
50 lock_queue_3d = Lock()
51 lock_queue_3d_masks = Lock()
52 lock_queue_3d_flirt = Lock()
53
54 def label2rgb(mask):
55  mask_rgb = np.zeros( mask.shape + (3,) )
56  for i in range(mask.shape[0]):
57   for j in range(mask.shape[1]):
58    for k in range(mask.shape[2]):
59     if int(mask[i,j,k]) not in freesurfer_codes:
60      #print( mask[i,j,k], flush=True )
61      #print( freesurfer_codes, flush=True )
62      raise Exception ("invalid free surfer code %d!" % int(
          ↪ mask[i,j,k]))
63     codes = freesurfer_codes[int(mask[i,j,k])]
64     mask_rgb[i,j,k,:]=codes
65  return mask_rgb
66
67
```

```
68 # Crea la mascara de la imagen del path pasada como
     ↪ parametro y la devuelve
69 def get_mask( path, flirt = False ):
70  mask_bet = np.array([])
71  try:
72   mask_bet_path = os.path.join( 'masks', path.split("/")
       ↪ [-1] )
73   if not os.path.exists( mask_bet_path ):
74    print( 'generating the masks for %s' % mask_bet_path,
       ↪ flush=True )
75    btr = fsl.BET()
76    btr.inputs.in_file = path
77    btr.inputs.frac = 0.1 # HAY QUE JUGAR CON ESTE PARAMETRO
       ↪  PARA AJUSTAR EL NIVEL DE RECORTE
78    btr.inputs.out_file = mask_bet_path
79    btr.inputs.mask = False
80    #btr.inputs.remove_eyes = True
81    btr.inputs.reduce_bias = True
82    res = btr.run()
83   else:
84    print( 'already generated masks %s' % mask_bet_path,
       ↪ flush=True )
85   if flirt:
86    print( 'generating the FLIRT image %s_FLIRT.nii.gz' %
       ↪ mask_bet_path, flush=True )
87    flirt_out_path = mask_bet_path[:-7] + "_FLIRT.nii.gz"
88    if not os.path.exists( flirt_out_path ):
89     flt = fsl.FLIRT( bins = 640, cost_func = "mutualinfo")
90     flt.inputs.in_file = mask_bet_path
```

```
91      flt.inputs.reference = "/usr/share/fsl/5.0/data/
          ↪ standard/MNI152_T1_2mm_brain.nii.gz"
92      flt.inputs.output_type = "NIFTI_GZ"
93      flt.inputs.out_file = flirt_out_path
94      flt.inputs.out_matrix_file = "flirt-tmp/" +
          ↪ mask_bet_path[:-7].split("/")[-1] + ".mat"
95      res = flt.run()
96
97     mask_bet = nib.load(flirt_out_path)
98     mask_bet = mask_bet.get_data()
99    else:
100    mask_bet = nib.load(mask_bet_path[:-3])
101    mask_bet = mask_bet.get_data()
102
103  except RuntimeError:
104   print("Error en la imagen ", path)
105   return None
106
107  return mask_bet
108 #
      ↪ ----------------------------------------------------------------
      ↪
109
110
111 def preprocess_one_image( filename_list, csv_queues,
112   make_adni=True, make_oasis=False,
113   do_it_3D=False, do_it_3D_masks=False, do_it_3D_flirt=
        ↪ False, do_it_3D_freesurfer=False,
```

```
114   do_it_2D=False, do_it_2D_masks=False, do_it_2D_3c= False,
        ↪  do_it_2D_flirt=False, do_it_2D_freesurfer=False):
115   #
116   #adni_opt/002_S_0295/MPRAGE/2012-05-10_15_44_50.0/S150055
        ↪ /sub-0020295_ses-1_T1w
117   #    ^
118   #    \---- we want this
119   filename=filename_list[0].strip()
120   if filename_list[1]: filename_brain=filename_list[1].
        ↪ strip()
121   if filename_list[2]: filename_mask = filename_list[2].
        ↪ strip()
122   if filename[0] == '#' : return
123   if make_adni:
124    parts = filename.split('/')
125    session = parts[-2]
126    subject_id = parts[-5]
127    #
128    nifti_path = filename + ".nii.gz"
129    xml_path = filename + ".xml"
130    #
131    # abrimos el xml y buscamos el UID
132    with open(xml_path,"r") as xml_file:
133     content="".join([i for i in xml_file])
134    parse_xml = BeautifulSoup(content, "xml")
135    id_image=int(parse_xml.metadata.image["uid"][1:])
136    #df_element_aux=df_adni[df_adni["Subject"] == subject_id
         ↪ ]
137    df_element=df_adni[df_adni["Image Data ID"] == id_image]
```

```
138     try:
139      image_label = df_element.iloc[0]["Group"]
140      subject_sex = df_element.iloc[0]["Sex"]
141      subject_age = df_element.iloc[0]["Age"]
142     except:
143      print( "Error searching ", id_image )
144      return
145 
146     # si no tiene la etiqueta deseada, miramos la sigueinte
          ↪ imagen
147          if image_label in ["SMC","Patient"]: return
148   if make_oasis:
149 
150     parts = filename.split('/')[1].split('_')
151     session = parts[1]
152     subject_id = parts[0]
153     #
154     nifti_path = filename
155     df_element=df_oasis[df_oasis["Subject"] == subject_id.
          ↪ split("-")[1]]
156     df_element_subject=df_oasis_subjects[df_oasis_subjects["
          ↪ Subject"] == subject_id.split("-")[1]]
157     #
158     try:
159      label_str=str(df_element.iloc[0]["dx1"])
160      cdr=float(df_element.iloc[0]["cdr"])
161      if label_str=="None":return
162      if label_str == "Cognitively normal" and cdr==0.0:
163       image_label = "CN"
```

```
164     elif label_str.startswith("AD") or cdr>=1.0:
165      image_label = "AD"
166     else:
167      return
168     subject_sex = df_element_subject.iloc[0]["M/F"]
169     subject_age = df_element.iloc[0]["Age"]
170    except:
171     print( "Error searching ", id_image )
172     return
173   try:
174    image_3D = sitk.ReadImage(nifti_path)
175   except:
176    print("Error al abrir: ", nifti_path)
177    return
178   #abrimos la imagen y leemos el erray
179   if image_3D.GetSize()[0]<130: return # avoid sesions with
        ↪  less than 130 frames
180   # le damos la vuelta a la cabeza para tenerla de frente
        ↪ con el -1
181   array_3d = sitk.GetArrayFromImage(image_3D[:,:,::-1])
182   # Obtenemos loa valores max y minimo de la imagen 3D
183   max_value_aux = np.max(array_3d)
184   min_value_aux = np.min(array_3d)
185   # Guardamos una linea de texto con la informacion
        ↪ requerida del tsv
186   if do_it_3D:
187    target_filename = filename.replace( origin_data_dir,
        ↪ data_dir_3d )
```

```
188     os.makedirs( os.path.dirname( target_filename ),
            ↪ exist_ok=True )
189     image_3d_path = target_filename + ".npy"
190     if not os.path.exists(image_3d_path):
191      print( "saving " + image_3d_path + " ... " )
192      np.save( image_3d_path, [ array_3d, categorical_dict[
            ↪ image_label] ]  )
193     #lock_queue_3d.acquire()
194     csv_queues['3d'].append( ( subject_id + "\t"
195         + image_3d_path + "\t"
196         + subject_sex + "\t"
197         + str(subject_age) + "\t"
198         + image_label + "\t"
199         + "None" + "\t"
200         + str(max_value_aux) + "\t"
201         + str(min_value_aux) + "\t"
202         + str(image_3D.GetDepth()) + "\n") )
203     #lock_queue_3d.release()
204
205   if do_it_2D:
206     # FUTURIBLE: obtener la mascara del parenquima
207     # mask = get_mask(image)
208     target_filename = filename.replace( origin_data_dir,
            ↪ data_dir_2d )
209     os.makedirs( os.path.dirname( target_filename ),
            ↪ exist_ok=True )
210     for slide_num in range(image_3D.GetDepth()): # equivalen
            ↪ t to range(array_3d.shape[0])
211      #
```

```
212     # COGIDO CON PINZAS: miramos si la media de las
            ↪ imagenes para saber si la imagen es negra
213     if np.mean(array_3d[slide_num,:,:]) < 40: continue
214     # Se crea el array con la imagen y la etiqueta y se
            ↪ comprueba si da error.
215     array_2d = array_3d[slide_num,:,:]
216     # OJO, en este caso, nos guardamos el max y el minimo
            ↪ de cada imagen 2d
217     max_value_aux = np.max(array_2d)
218     min_value_aux = np.min(array_2d)
219     # guardamos el array
220     image_2d_path = target_filename + "-slide-%03d" + ".npy
            ↪ "
221     image_2d_path = image_2d_path % slide_num
222     if not os.path.exists(image_2d_path):
223      print( "saving " + image_2d_path + " ... " )
224      np.save( image_2d_path, [ array_2d, categorical_dict[
            ↪ image_label] ]  )
225
226     # Critical region
227     #lock_queue_2d.acquire()
228     csv_queues['2d'].append( ( subject_id + "\t"
229             + image_2d_path + "\t"
230             + subject_sex + "\t"
231             + str(subject_age) + "\t"
232             + image_label + "\t"
233             + "None" + "\t"
234             + str(max_value_aux) + "\t"
235             + str(min_value_aux) + "\n" ) )
```

```
236     #lock_queue_2d.release()
237     # Critical region
238   if do_it_3D_masks or do_it_2D_masks or do_it_2D_3c:
239    mask = get_mask( nifti_path )
240    max_value_aux = np.max(mask)
241    min_value_aux = np.min(mask)
242    if mask is None:
243     return
244
245    if do_it_3D_masks:
246     target_filename = filename.replace( origin_data_dir,
          ↪ data_dir_3d_mask )
247     os.makedirs( os.path.dirname( target_filename ),
          ↪ exist_ok=True )
248     image_3d_path = target_filename + ".npy"
249     if not os.path.exists(image_3d_path):
250      print( "saving " + image_3d_path + " ... " )
251      np.save( image_3d_path, [ mask, categorical_dict[
          ↪ image_label] ]  )
252     #lock_queue_3d.acquire()
253     csv_queues['3d_masks'].append( ( subject_id + "\t"
254         + image_3d_path + "\t"
255         + subject_sex + "\t"
256         + str(subject_age) + "\t"
257         + image_label + "\t"
258         + "None" + "\t"
259         + str(max_value_aux) + "\t"
260         + str(min_value_aux) + "\t"
261         + str(mask.shape[-1]) + "\n") )
```

```
262
263     if do_it_2D_masks:
264      target_filename = filename.replace( origin_data_dir,
           ↪ data_dir_2d_mask )
265      os.makedirs( os.path.dirname( target_filename ),
           ↪ exist_ok=True )
266      n_slices = mask.shape[-1]
267      first_slice = int(n_slices*3/10)
268      for slide_num in range( first_slice, n_slices ):
269       # COGIDO CON PINZAS: miramos si la media de las
            ↪ imagenes para saber si la imagen es negra
270       #if np.mean(mask[slide_num,:,:])<40:continue
271       # Se crea el array con la imagen y la etiqueta y se
            ↪ comprueba si da error.
272       array_2d = mask[:,:,slide_num]
273       max_value_aux = np.max(array_2d)
274       min_value_aux = np.min(array_2d)
275       if max_value_aux == min_value_aux: continue # Remove
            ↪ non-informative slices
276       #
277       mask_2d_path = target_filename + "-slide-%03d" + ".npy
            ↪ "
278       mask_2d_path = mask_2d_path % slide_num
279       if not os.path.exists( mask_2d_path ):
280        print( "saving " + mask_2d_path + " ... " )
281        np.save( mask_2d_path, [ array_2d, categorical_dict[
             ↪ image_label ] ] )
282
283       # escribimos para cada imagen, su linea csv
```

```
284         # Critical region
285         #lock_queue_2d_masks.acquire()
286         csv_queues['2d_masks'].append( ( subject_id + "\t"
287                 + mask_2d_path + "\t"
288                 + subject_sex + "\t"
289                 + str(subject_age) + "\t"
290                 + image_label + "\t"
291                 + "None" + "\t"
292                 + str(max_value_aux) + "\t"
293                 + str(min_value_aux) + "\n") )
294     if do_it_2D_3c:
295         target_filename = filename.replace( origin_data_dir,
              ↪ data_dir_2d_3c )
296         os.makedirs( os.path.dirname( target_filename ),
              ↪ exist_ok=True )
297         n_slices = mask.shape[-1]
298         first_slice = int(n_slices*0.40)
299         last_slice = int(n_slices*0.70)
300         for slide_num in range( first_slice, last_slice-3):
301             # COGIDO CON PINZAS: miramos si la media de las
                  ↪ imagenes para saber si la imagen es negra
302             #if np.mean(mask[slide_num,:,:])<40:continue
303             # Se crea el array con la imagen y la etiqueta y se
                  ↪ comprueba si da error.
304             array_2d = mask[:,:,slide_num:slide_num+3]
305             max_value_aux = np.max(array_2d)
306             min_value_aux = np.min(array_2d)
307             if max_value_aux == min_value_aux: continue # Remove
                  ↪ non-informative slices
```

```
308         #
309         mask_2d_path = target_filename + "-slide-%03d" + ".npy
                ↪ "
310         mask_2d_path = mask_2d_path % slide_num
311         if not os.path.exists( mask_2d_path ):
312          print( "saving " + mask_2d_path + " ... " )
313          np.save( mask_2d_path, [ array_2d, categorical_dict[
                ↪ image_label ] ] )
314
315         # escribimos para cada imagen, su linea csv
316         # Critical region
317         #lock_queue_2d_masks.acquire()
318         csv_queues['2d_3c'].append( ( subject_id + "\t"
319                  + mask_2d_path + "\t"
320                  + subject_sex + "\t"
321                  + str(subject_age) + "\t"
322                  + image_label + "\t"
323                  + "None" + "\t"
324                  + str(max_value_aux) + "\t"
325                  + str(min_value_aux) + "\n") )
326         #lock_queue_2d_masks.release()
327         # Critical region
328     if do_it_3D_flirt or do_it_2D_flirt:
329      mask = get_mask( nifti_path , flirt=True)
330      max_value_aux = np.max(mask)
331      min_value_aux = np.min(mask)
332      print(type(mask))
333      if mask is None:
334       return
```

```
335
336     if do_it_3D_flirt:
337      target_filename = filename.replace(
           ↪ origin_data_dir_oasis, data_dir_3d_flirt )
338      os.makedirs( os.path.dirname( target_filename ),
           ↪ exist_ok=True )
339      image_3d_path = target_filename + ".npy"
340      if not os.path.exists(image_3d_path):
341       print( "saving " + image_3d_path + " ... " )
342       np.save( image_3d_path, [ mask, categorical_dict[
           ↪ image_label] ]  )
343      #lock_queue_3d.acquire()
344      csv_queues['3d_flirt'].append( ( subject_id + "\t"
345          + image_3d_path + "\t"
346          + str(subject_sex) + "\t"
347          + str(subject_age) + "\t"
348          + image_label + "\t"
349          + "None" + "\t"
350          + str(max_value_aux) + "\t"
351          + str(min_value_aux) + "\t"
352          + str(mask.shape[-1]) + "\n") )
353
354     if do_it_2D_flirt:
355      target_filename = filename.replace(
           ↪ origin_data_dir_oasis, data_dir_2d_flirt )
356      os.makedirs( os.path.dirname( target_filename ),
           ↪ exist_ok=True )
357      n_slices = mask.shape[-1]
358      #first_slice = int(n_slices*3/10)
```

```
359     first_slice = 0
360     for slide_num in range( first_slice, n_slices ):
361      # COGIDO CON PINZAS: miramos si la media de las
           ↪ imagenes para saber si la imagen es negra
362      #if np.mean(mask[slide_num,:,:])<40:continue
363      # Se crea el array con la imagen y la etiqueta y se
           ↪ comprueba si da error.
364      array_2d = mask[:,:,slide_num]
365      max_value_aux = np.max(array_2d)
366      min_value_aux = np.min(array_2d)
367      if max_value_aux == min_value_aux: continue # Remove
           ↪ non-informative slices
368      #
369      mask_2d_path = target_filename + "-slide-%03d" + ".npy
           ↪ "
370      mask_2d_path = mask_2d_path % slide_num
371      if not os.path.exists( mask_2d_path ):
372       print( "saving " + mask_2d_path + " ... " )
373       np.save( mask_2d_path, [ array_2d, categorical_dict[
            ↪ image_label ] ] )
374
375      # escribimos para cada imagen, su linea csv
376      # Critical region
377      #lock_queue_2d_masks.acquire()
378      csv_queues['2d_flirt'].append( ( subject_id + "\t"
379               + mask_2d_path + "\t"
380               + str(subject_sex) + "\t"
381               + str(subject_age) + "\t"
382               + image_label + "\t"
```

```
383                 + "None" + "\t"
384                 + str(max_value_aux) + "\t"
385                 + str(min_value_aux) + "\n") )
386   if do_it_3D_freesurfer or do_it_2D_freesurfer:
387    if not filename_brain or not filename_mask: return
388    brain_free = nib.load(filename_brain)
389    brain_free = brain_free.get_data()
390    mask_free = nib.load(filename_mask)
391    mask_free = mask_free.get_data()
392    mask_free = label2rgb(mask_free)
393    mask_free = mask_free.astype("uint16")
394    max_value_aux = np.max(brain_free)
395    min_value_aux = np.min(brain_free)
396
397    if do_it_3D_freesurfer:
398     target_filename = filename.replace( origin_data_dir,
         ↪ data_dir_3d_free )
399     os.makedirs( os.path.dirname( target_filename ),
         ↪ exist_ok=True )
400     image_3d_path = target_filename + ".npy"
401     if not os.path.exists(image_3d_path):
402      print( "saving " + image_3d_path + " ... " )
403      x = np.transpose(np.array(
404       [brain_free,
405       mask_free[:,...,0],
406       mask_free[:,...,1],
407       mask_free[:,...,2]]
408      ), (1,2,3,0))
409      print("_____________________",x.shape)
```

```
410         np.save(image_3d_path,[x, categorical_dict[image_label
                ↪ ]])
411
412         #lock_queue_3d.acquire()
413         csv_queues['3d_freesurfer'].append( ( subject_id + "\t"
414             + image_3d_path + "\t"
415             + subject_sex + "\t"
416             + str(subject_age) + "\t"
417             + image_label + "\t"
418             + "None" + "\t"
419             + str(max_value_aux) + "\t"
420             + str(min_value_aux) + "\t"
421             + str(brain_free.shape[-1]) + "\n") )
422
423     if do_it_2D_freesurfer:
424         target_filename = filename.replace( origin_data_dir,
                ↪ data_dir_2d_free )
425         os.makedirs( os.path.dirname( target_filename ),
                ↪ exist_ok=True )
426         n_slices = brain_free.shape[-1]
427         #first_slice = int(n_slices*3/10)
428         first_slice = 0
429         for slide_num in range( first_slice, n_slices ):
430             # COGIDO CON PINZAS: miramos si la media de las
                    ↪ imagenes para saber si la imagen es negra
431             #if np.mean(mask[slide_num,:,:])<40:continue
432             # Se crea el array con la imagen y la etiqueta y se
                    ↪ comprueba si da error.
433             array_2d = brain_free[:,slide_num,:]
```

```
434     array_2d_mask_r = mask_free[:,slide_num,:,0]
435     array_2d_mask_g = mask_free[:,slide_num,:,1]
436     array_2d_mask_b = mask_free[:,slide_num,:,2]
437     max_value_aux = np.max(array_2d)
438     min_value_aux = np.min(array_2d)
439     if max_value_aux == min_value_aux: continue # Remove
          ↪ non-informative slices
440     #
441     mask_2d_path = target_filename + "-slide-%03d" + ".npy
          ↪ "
442     mask_2d_path = mask_2d_path % slide_num
443     if not os.path.exists( mask_2d_path ):
444      print( "saving " + mask_2d_path + " ... " )
445      x = np.transpose(np.array([array_2d,array_2d_mask_r,
          ↪ array_2d_mask_g,array_2d_mask_b]), (1,2,0))
446      np.save(mask_2d_path,[x, categorical_dict[image_label
          ↪ ]])
447
448
449     # escribimos para cada imagen, su linea csv
450     # Critical region
451     #lock_queue_2d_masks.acquire()
452     csv_queues['2d_freesurfer'].append( ( subject_id + "\t
          ↪ "
453             + mask_2d_path + "\t"
454             + subject_sex + "\t"
455             + str(subject_age) + "\t"
456             + image_label + "\t"
457             + "None" + "\t"
```

```
458                 + str(max_value_aux) + "\t"
459                 + str(min_value_aux) + "\n") )
460 #
    ↪ ------------------------------------------------------------------------
    ↪
461
462 def loop_for_processing_images(
463   queue, csv_queues, th_num,
464   make_adni=True, make_oasis=False,
465   do_it_3D=False, do_it_3D_masks=False, do_it_3D_flirt=
      ↪ False, do_it_3D_freesurfer=False,
466   do_it_2D=False, do_it_2D_masks=False, do_it_2D_3c=False,
      ↪ do_it_2D_flirt=False, do_it_2D_freesurfer=False
467  ):
468
469  print( 'starting thread number ', th_num )
470  while not queue.empty():
471   try:
472    lock_queue_filenames.acquire()
473    filename=queue.get()
474    lock_queue_filenames.release()
475   except:
476    filename=None
477   if filename is None: break
478   print( 'thread %d processing ' % th_num, filename )
479   preprocess_one_image(
480    filename, csv_queues,
481    make_adni, make_oasis,
```

```
482     do_it_3D, do_it_3D_masks, do_it_3D_flirt,
          ↪ do_it_3D_freesurfer,
483     do_it_2D, do_it_2D_masks, do_it_2D_3c, do_it_2D_flirt,
          ↪ do_it_2D_freesurfer
484   )
485
486  print( 'finishing thread number ', th_num )
487 #
     ↪ ------------------------------------------------------------
     ↪
488
489
490
491 def nifti_2_npy( index_filename, freesurfer_brainmask_file,
     ↪  freesurfer_aparc_aseg_file,
492   make_adni=True, make_oasis=False,
493   do_it_3D=False, do_it_3D_masks=False, do_it_3D_flirt=
        ↪ False,  do_it_3D_freesurfer=False,
494   do_it_2D=False, do_it_2D_masks=False, do_it_2D_3c=False,
        ↪ do_it_2D_flirt=False, do_it_2D_freesurfer=False
495  ):
496  #
497  # Identifadores de columna para los tsv tanto en 2D
498  columns=["Subject","T1_path","Sex","Age","Label","
       ↪ Partition","Max_value_pixel","Min_value_pixel","
       ↪ slides"]
499  # creamos un string para ir anyadiendo las lineas del tsv
       ↪ en 2D
```

```
500  # Buscamos los Sujetos (nombre de carpetas) en el
       ↪ directorio de las imagenes
501  if make_adni:
502   print("Loading adni file names...")
503   f=open( index_filename, 'rt' )
504   f_brain=open( freesurfer_brainmask_file, 'rt' )
505   f_mask=open( freesurfer_aparc_aseg_file, 'rt' )
506   f_brain_text=f_brain.read()
507   f_mask_text=f_mask.read()
508   filenames_queue = Queue()
509   for filename in f:
510    file_brain = ""
511    file_mask = ""
512    re_comp = re.compile(r".*"+filename.split(os.sep)[-1].
         ↪ strip()+".*", re.U)
513    valid_brain = re_comp.search(f_brain_text)
514    valid_mask = re_comp.search(f_mask_text)
515    if valid_brain and valid_mask:
516     file_brain = valid_brain.group(0)
517     file_mask = valid_mask.group(0)
518    filenames_queue.put([filename, file_brain, file_mask])
519   f.close()
520   f_brain.close()
521   f_mask.close()
522  if make_oasis:
523   print("Loading oasis file names...")
524   f=open( index_filename, 'rt' )
525   filenames_queue = Queue()
526   for filename in f:
```

```
527     filenames_queue.put([origin_data_dir_oasis + os.sep +fi
          ↪ lename[:-1], None, None])
528   f.close()
529   print("File names loaded.")
530   #
531   csv_queues=list()
532   threads=list()
533   num_threads=11
534   for th in range(num_threads):
535    csv_queues.append( {
536     '3d': deque(), '3d_masks': deque(), '3d_flirt': deque(),
          ↪  '3d_freesurfer': deque(),
537     '2d': deque(), '2d_masks': deque(), '2d_3c': deque(), '2
          ↪ d_flirt': deque(), '2d_freesurfer': deque()
538    } )
539    #
540    threads.append( threading.Thread(
541     target=loop_for_processing_images,
542     args=(
543      filenames_queue, csv_queues[-1],th,
544      make_adni, make_oasis,
545      do_it_3D, do_it_3D_masks, do_it_3D_flirt,
          ↪ do_it_3D_freesurfer,
546      do_it_2D, do_it_2D_masks, do_it_2D_3c, do_it_2D_flirt,
          ↪ do_it_2D_freesurfer
547     )
548    ) )
549   print("Starting threads...")
550   for th in threads: th.start()
```

```
551 print("Threads started.")
552 print("Waiting for all the threads...")
553 for th in threads: th.join()
554
555 print( 'All threads finished!', flush=True )
556
557 #filenames_queue.close()
558
559 if do_it_3D:
560  # Guardamos el tsv 3D
561  #with open(os.path.join(data_dir_3d,"adni_table_3D.tsv"),
       ↪  "w") as adni_tsv: adni_tsv.write(tsv_adni_3d)
562  with open(os.path.join(data_dir_3d,"adni_table_3D.tsv"),
       ↪ "w") as adni_tsv:
563   adni_tsv.write( "\t".join(columns) + "\n" )
564   for qs in csv_queues:
565    q = qs['3d']
566    while len(q) > 0:
567     adni_tsv.write( q.popleft() )
568   adni_tsv.close()
569
570 if do_it_3D_masks:
571  # Guardamos el tsv 3D
572  #with open(os.path.join(data_dir_3d,"adni_table_3D.tsv"),
       ↪  "w") as adni_tsv: adni_tsv.write(tsv_adni_3d)
573  with open(os.path.join(data_dir_3d_mask,"
       ↪ adni_table_3D_parenquima.tsv"), "w") as adni_tsv:
574   adni_tsv.write( "\t".join(columns) + "\n" )
575   for qs in csv_queues:
```

```
576     q = qs['3d_masks']
577     while len(q) > 0:
578      adni_tsv.write( q.popleft() )
579    adni_tsv.close()
580
581  if do_it_3D_flirt:
582   # Guardamos el tsv 3D
583   #with open(os.path.join(data_dir_3d,"adni_table_3D.tsv"),
        ↪  "w") as adni_tsv: adni_tsv.write(tsv_adni_3d)
584   with open(os.path.join(data_dir_3d_flirt,"
        ↪ adni_table_3D_flirt.tsv"), "w") as adni_tsv:
585    adni_tsv.write( "\t".join(columns) + "\n" )
586    for qs in csv_queues:
587     q = qs['3d_flirt']
588     while len(q) > 0:
589      adni_tsv.write( q.popleft() )
590    adni_tsv.close()
591
592  if do_it_3D_freesurfer:
593   # Guardamos el tsv 3D
594   #with open(os.path.join(data_dir_3d,"adni_table_3D.tsv"),
        ↪  "w") as adni_tsv: adni_tsv.write(tsv_adni_3d)
595   with open(os.path.join(data_dir_3d_free,"
        ↪ adni_table_3D_freesurfer.tsv"), "w") as adni_tsv:
596    adni_tsv.write( "\t".join(columns) + "\n" )
597    for qs in csv_queues:
598     q = qs['3d_freesurfer']
599     while len(q) > 0:
600      adni_tsv.write( q.popleft() )
```

```
601     adni_tsv.close()
602
603   if do_it_2D:
604    # Guardamos el tsv 2D
605    #with open(os.path.join(data_dir_2d,"adni_table_2D.tsv"),
         ↪  "w") as adni_tsv: adni_tsv.write(tsv_adni_2d)
606    with open(os.path.join(data_dir_2d,"adni_table_2D.tsv"),
         ↪ "w") as adni_tsv:
607     adni_tsv.write( "\t".join(columns) + "\n" )
608     for qs in csv_queues:
609      q = qs['2d']
610      while len(q) > 0:
611       adni_tsv.write( q.popleft() )
612     adni_tsv.close()
613
614   if do_it_2D_masks:
615    # Guardamos el tsv 2D
616    #with open(os.path.join(data_dir_2d_mask,"
         ↪ adni_table_2D_parenquima.tsv"), "w") as adni_tsv:
         ↪ adni_tsv.write(tsv_adni_2d_mask)
617    with open(os.path.join(data_dir_2d_mask,"
         ↪ adni_table_2D_parenquima.tsv"), "w") as adni_tsv:
618     adni_tsv.write( "\t".join(columns) + "\n" )
619     for qs in csv_queues:
620      q = qs['2d_masks']
621      while len(q) > 0:
622       adni_tsv.write( q.popleft() )
623     adni_tsv.close()
624
```

```
625 if do_it_2D_flirt:
626  # Guardamos el tsv 2D
627  #with open(os.path.join(data_dir_2d_mask,"
       ↪ adni_table_2D_parenquima.tsv"), "w") as adni_tsv:
       ↪ adni_tsv.write(tsv_adni_2d_mask)
628  with open(os.path.join(data_dir_2d_flirt,"
       ↪ adni_table_2D_flirt.tsv"), "w") as adni_tsv:
629   adni_tsv.write( "\t".join(columns) + "\n" )
630   for qs in csv_queues:
631    q = qs['2d_flirt']
632    while len(q) > 0:
633     adni_tsv.write( q.popleft() )
634   adni_tsv.close()
635
636 if do_it_2D_freesurfer:
637  # Guardamos el tsv 2D
638  #with open(os.path.join(data_dir_2d_mask,"
       ↪ adni_table_2D_parenquima.tsv"), "w") as adni_tsv:
       ↪ adni_tsv.write(tsv_adni_2d_mask)
639  with open(os.path.join(data_dir_2d_free,"
       ↪ adni_table_2D_freesurfer.tsv"), "w") as adni_tsv:
640   adni_tsv.write( "\t".join(columns) + "\n" )
641   for qs in csv_queues:
642    q = qs['2d_freesurfer']
643    while len(q) > 0:
644     adni_tsv.write( q.popleft() )
645   adni_tsv.close()
646
647
```

```
648  if do_it_2D_3c:
649   # Guardamos el tsv 2D
650   #with open(os.path.join(data_dir_2d_mask,"
        ↪ adni_table_2D_parenquima.tsv"), "w") as adni_tsv:
        ↪ adni_tsv.write(tsv_adni_2d_mask)
651   with open(os.path.join(data_dir_2d_3c,"adni_table_2D_3c.
        ↪ tsv"), "w") as adni_tsv:
652    adni_tsv.write( "\t".join(columns) + "\n" )
653    for qs in csv_queues:
654     q = qs['2d_3c']
655     while len(q) > 0:
656      adni_tsv.write( q.popleft() )
657    adni_tsv.close()
658
659  #for q in csv_queues.values(): q.close()
660 #
     ↪ ----------------------------------------------------------------------
     ↪
661
662
663 if __name__ == '__main__':
664  nifti_2_npy(
665    origin_data_dir_oasis + '/T1w_files.txt',
       ↪ freesurfer_brainmask_file,
       ↪ freesurfer_aparc_aseg_file,
666    make_adni=False, make_oasis=True,
667    do_it_3D=False, do_it_3D_masks=False, do_it_3D_flirt=
       ↪ True, do_it_3D_freesurfer=False,
```

```
668     do_it_2D=False, do_it_2D_masks=False, do_it_2D_3c=False,
            ↪ do_it_2D_flirt=True, do_it_2D_freesurfer=False
669 )
```

Código 7.1: prerpocesado para generar todos los dataset descritos

7.3 Data generator 3D freesurfer

```
import numpy as np
import os
import keras
from skimage.transform import resize
import pickle
import matplotlib.pyplot as plt

from utils import to_3D_cube_with_channels
from lib.createdict import create_dict_freesurfer_codes
from data_augmentation_3D import translateit_fast
from data_augmentation_3D import scaleit
from data_augmentation_3D import rotateit
from data_augmentation_3D import intensifyit

class DataGenerator3Dmask(keras.utils.Sequence):

 ''' Initialization of the generator '''
 def __init__( self, data_frame, target_shape=(256,256,256)
    ↪ , target_channels=1, indexes_output=None, batch_size
    ↪ =128, shuffle=True, rotation=False, do_balance =
    ↪ False, with_age = False, DA=False, autoencoder =
    ↪ False ):
  'Initialization'
  self.df = data_frame.reset_index(drop=True)
  self.target_shape = target_shape
  self.target_channels = target_channels
  self.batch_size = batch_size
  self.shuffle = shuffle
```

```
25    self.rotation = rotation
26    self.with_age = with_age
27    self.autoencoder = autoencoder
28    self.data_augmentation=DA
29    self.freesurfer_codes = create_dict_freesurfer_codes()
30    # HARD CODE KK
31    labels = np.unique( self.df['Label'] )
32    if do_balance and batch_size >= len(labels):
33     self.indexes = []
34     self.num_batches = 0
35     for label in labels:
36      indexes_label = np.array( self.df[ self.df["Label"]==
          ↪ label ].index.values.astype(int).tolist() )
37      self.indexes.append( indexes_label )
38      nb = int( len(labels) * np.ceil(len(indexes_label) /
          ↪ self.batch_size))
39      self.num_batches = max( self.num_batches, nb )
40     self.pointers = np.zeros( len(self.indexes), dtype=int )
41     self.samples_per_batch = int(self.batch_size // len(self
         ↪ .indexes))
42    else:
43     self.indexes = np.arange(len(data_frame.index))
44     self.num_batches = int(np.ceil(len(self.indexes) / self.
         ↪ batch_size))
45    #
46    if indexes_output is None:
47     self.indexes_output = 5*[True]
48    else:
49     self.indexes_output = indexes_output
```

```
50    #
51    self.on_epoch_end()
52
53   ''' Returns the number of batches per epoch '''
54   def __len__(self):
55    return self.num_batches
56    #return int(np.ceil(len(self.indexes) / self.batch_size))
57
58   ''' Returns a batch of data (the batches are indexed) '''
59   def __getitem__(self, index):
60    if type(self.indexes) is list:
61     X, Y, ages = [], [], [] # Batch initialization
62     for i in range(len(self.indexes)):
63      for j in range(self.samples_per_batch):
64       x,y = self.get_sample( self.indexes[i][ self.pointers[
           ↪ i] ] )
65       age=1
66       self.pointers[i] = (self.pointers[i]+1) % len(self.
           ↪ indexes[i])
67       if len(x.shape) == 3:
68        X.append( np.reshape( x, x.shape + (self.
             ↪ target_channels,)))
69       else:
70        X.append(x)
71       Y.append( y[self.indexes_output] )
72       ages.append( (age,) )
73     if self.autoencoder:
74      return np.array(X), np.array(X)
75     else:
```

```
76      return np.array(X), np.array(Y)
77    else:
78      indexes = self.indexes[index*self.batch_size:(index+1)*
          ↪ self.batch_size] # Take the id's of the batch
          ↪ number "index"
79
80      X, Y, ages = [], [], [] # Batch initialization
81
82      ''' For each index we take the sample an the label and
          ↪ we append them to the batch'''
83      for idx in indexes:
84        #x, y, age = self.get_sample(idx)
85        x, y = self.get_sample(idx)
86        age=1
87        X.append(x)
88        Y.append(y[self.indexes_output])
89        ages.append( (age,) )
90      if self.with_age:
91        return [np.array(X), np.array(ages)], np.array(Y)
92      else:
93        if self.autoencoder:
94          return np.array(X), np.array(X)
95        else:
96          return np.array(X), np.array(Y) #X:(batch_size, y, x),
              ↪  y:(batch_size, n_labels_types)
97      #return np.array(X), [np.array(X),np.array(Y)] #X:(
          ↪ batch_size, y, x), y:(batch_size, n_labels_types)
98
99    ''' Triggered at the end of each epoch '''
```

```
100 def on_epoch_end(self):
101  if self.shuffle == True:
102   if type(self.indexes) is list:
103    for i in range(len(self.indexes)):
104     np.random.shuffle(self.indexes[i])
105   else:
106    np.random.shuffle(self.indexes) # Shuffles the data
107
108 '''Returns the sample and the label with the id passed as
      ↪ a parameter'''
109 def get_sample(self, idx):
110  do_rotation = True if np.random.rand() > 0.7 else False
111  do_shift = True if np.random.rand() > 0.7 else False
112  do_zoom = True if np.random.rand() > 0.7 else False
113  do_intense= True if np.random.rand() > 0.7 else False
114  theta1 = 0
115  theta2 = 0
116  theta3 = 0
117  offset = [0, 0, 0]
118  zoom  = 1.0
119  factor = 1.0
120  if self.data_augmentation and np.random.rand() > 0.3: #
       ↪ Perform data augmentation
121   theta1 = float(np.around(np.random.uniform(-20.0,20.0,
        ↪ size=1), 3))
122   theta2 = float(np.around(np.random.uniform(-20.0,20.0,
        ↪ size=1), 3))
123   theta3 = float(np.around(np.random.uniform(-20.0,20.0,
        ↪ size=1), 3))
```

```
124     offset = list(np.random.randint(-15,15, size=3))
125     zoom   = float(np.around(np.random.uniform(0.9, 1.05,
            ↪ size=1), 2))
126     factor = float(np.around(np.random.uniform(0.8, 1.2,
            ↪ size=1), 2))
127
128
129
130   df_row = self.df.iloc[idx] # Get the row from the
        ↪ dataframe corresponding to the index "idx"
131   image_and_label = np.load(df_row["T1_path"], allow_pickle
        ↪ =True) # Load the numpy with the image and the
        ↪ label. With the format [image, label(one hot
        ↪ encoding)]
132   image = to_3D_cube_with_channels( image_and_label[0],
        ↪ target_shape=tuple((*self.target_shape, 4)) )
133
134   if do_rotation:
135     rotateit(image, theta1,theta2,theta3)
136   if do_shift:
137     translateit_fast(image, offset)
138   if do_zoom:
139     for channel in range(self.target_channels):
140       image[:,...,channel] = scaleit(image[:,...,channel],
            ↪ zoom)
141   if do_intense:
142     image[:,...,0]=intensifyit(image[:,...,0], factor)
143
144   image = self.norm(image)
```

```
145     return (image[:,...,0],image_and_label[1])#, float(df_row
          ↪ ["Age"])/100.0 # Return the resized image and the
          ↪ label
146
147   def norm(self, image):
148     #
149     #return image
150     #image_nonzero = image[np.nonzero(image)]
151     #return ((image - image_nonzero.mean()) / image_nonzero.
          ↪ std()) #Intentamos obtener la normalizacion con
          ↪ media 0 y desviacion tipica 1
152     """
153     """
154     image = np.minimum( 255, image )
155     image = np.maximum( 0, image )
156     image /= (255)
157     return image.astype( np.float32 )
158
159
160 class DataGenerator3Dmask_test(keras.utils.Sequence):
161
162   ''' Initialization of the generator '''
163   def __init__( self, data_frame, target_shape=(256,256,256)
          ↪ , target_channels=1, indexes_output=None, batch_size
          ↪ =128, shuffle=True, rotation=False, do_balance =
          ↪ False, with_age = False, autoencoder = False ):
164     'Initialization'
165     self.df = data_frame.reset_index(drop=True)
166     self.target_shape = target_shape
```

```
167     self.target_channels = target_channels
168     self.batch_size = batch_size
169     self.shuffle = shuffle
170     self.rotation = rotation
171     self.with_age = with_age
172     self.autoencoder = autoencoder
173     self.labels_array=[]
174     self.freesurfer_codes = create_dict_freesurfer_codes()
175     # HARD CODE KK
176     labels = np.unique( self.df['Label'] )
177     #print( labels )
178     if do_balance and batch_size >= len(labels):
179      self.indexes = []
180      self.num_batches = 0
181      for label in labels:
182       indexes_label = np.array( self.df[ self.df["Label"]==
          ↪ label ].index.values.astype(int).tolist() )
183       self.indexes.append( indexes_label )
184       nb = int( len(labels) * np.ceil(len(indexes_label) /
          ↪ self.batch_size))
185       self.num_batches = max( self.num_batches, nb )
186      self.pointers = np.zeros( len(self.indexes), dtype=int )
187      self.samples_per_batch = int(self.batch_size // len(self
          ↪ .indexes))
188     else:
189      self.indexes = np.arange(len(data_frame.index))
190      self.num_batches = int(np.ceil(len(self.indexes) / self.
          ↪ batch_size))
191     #
```

```
192   if indexes_output is None:
193    self.indexes_output = 5*[True]
194   else:
195    self.indexes_output = indexes_output
196   #
197   self.on_epoch_end()
198 
199  ''' Returns the number of batches per epoch '''
200 def __len__(self):
201  return self.num_batches
202  #return int(np.ceil(len(self.indexes) / self.batch_size))
203 
204  ''' Returns a batch of data (the batches are indexed) '''
205 def __getitem__(self, index):
206  if type(self.indexes) is list:
207   X, Y, ages = [], [], [] # Batch initialization
208   for i in range(len(self.indexes)):
209    for j in range(self.samples_per_batch):
210     x,y = self.get_sample( self.indexes[i][ self.pointers[
        ↪ i] ] )
211     age=1
212     self.pointers[i] = (self.pointers[i]+1) % len(self.
        ↪ indexes[i])
213     if len(x.shape) == 3:
214      X.append( np.reshape( x, x.shape + (self.
         ↪ target_channels,)))
215     else:
216      X.append(x)
217     Y.append( y[self.indexes_output] )
```

```
218         ages.append( (age,) )
219       if self.autoencoder:
220        return np.array(X), np.array(X)
221       else:
222        return np.array(X), np.array(Y)
223     else:
224       indexes = self.indexes[index*self.batch_size:(index+1)*
            ↪ self.batch_size] # Take the id's of the batch
            ↪ number "index"
225
226       X, Y, ages = [], [], [] # Batch initialization
227
228       ''' For each index we take the sample an the label and
            ↪ we append them to the batch'''
229       for idx in indexes:
230        #x, y, age = self.get_sample(idx)
231        x, y = self.get_sample(idx)
232        age=1
233        X.append(x)
234        Y.append(y[self.indexes_output])
235        ages.append( (age,) )
236       if self.with_age:
237        return [np.array(X), np.array(ages)], np.array(Y)
238       else:
239        if self.autoencoder:
240         return np.array(X), np.array(X)
241        else:
242         return np.array(X), np.array(Y) #X:(batch_size, y, x),
              ↪  y:(batch_size, n_labels_types)
```

```
243     #return np.array(X), [np.array(X),np.array(Y)] #X:(
            ↪ batch_size, y, x), y:(batch_size, n_labels_types)
244
245   ''' Triggered at the end of each epoch '''
246   def on_epoch_start():
247    self.labels_array=[]
248   # def on_epoch_end(self):
249   #  if self.shuffle == True:
250   #   if type(self.indexes) is list:
251   #    for i in range(len(self.indexes)):
252   #     np.random.shuffle(self.indexes[i])
253   #   else:
254   #    np.random.shuffle(self.indexes) # Shuffles the data
255
256   '''Returns the sample and the label with the id passed as
        ↪ a parameter'''
257   def get_sample(self, idx):
258    df_row = self.df.iloc[idx] # Get the row from the
        ↪ dataframe corresponding to the index "idx"
259    image_and_label = np.load(df_row["T1_path"], allow_pickle
        ↪ =True) # Load the numpy with the image and the
        ↪ label. With the format [image, label(one hot
        ↪ encoding)]
260    image = to_3D_cube_with_channels( image_and_label[0],
        ↪ target_shape=tuple((*self.target_shape, 4)) )
261
262
263    image = self.norm(image)
264    #mask=image[:, ..., 1:]
```

```
265     #mask_rgb = self.norm(mask)
266
267     #image[:, ..., 1] = self.normMask(image[:, ..., 1])
268     #if self.rotation:
269     # i=np.random.randint(4)
270     # if i > 0: norm_image = np.rot90(norm_image,i)
271     #mask=image[:,:,:,1]
272     #mask=np.reshape( mask, mask.shape + (self.
            ↪ target_channels,))
273     #image=image[:,:,:,0]
274     #image=np.reshape( image, image.shape + (self.
            ↪ target_channels,))
275     self.labels_array += [image_and_label[1]]
276     return (image[:,...,0].reshape(self.target_shape + (1,)),
            ↪ image_and_label[1])#, float(df_row["Age"])/100.0 #
            ↪ Return the resized image and the label
277
278   def norm(self, image):
279     #
280     #return image
281     #image_nonzero = image[np.nonzero(image)]
282     #return ((image - image_nonzero.mean()) / image_nonzero.
            ↪ std()) #Intentamos obtener la normalizacion con
            ↪ media 0 y desviacion tipica 1
283     """
284     """
285     #image = np.minimum( 1.0, image/255.0 )
286     image /= (255)
287     return image.astype( np.float32 )
```

Código 7.2: generador de datos para el modelo, en este caso para las imagenes de FreeSurfer

7.4 modelos CNN 2D

```
1  import keras
2  import os
3  import sys
4  import fnmatch
5  import random
6  import numpy as np
7  from keras.layers import Input, Conv2D, Conv2DTranspose,
     ↪ MaxPooling2D, UpSampling2D, AveragePooling2D
8  from keras.layers import Dropout, Activation, Flatten,
     ↪ Concatenate, Dense, Reshape
9  from keras.layers.normalization import BatchNormalization
     ↪ as BN
10 from keras.layers import GaussianNoise
11 from keras.models import Model, Sequential, load_model
12 from keras.optimizers import RMSprop, Adam, Adagrad, SGD,
     ↪ Adadelta
13 from keras.applications import vgg16, vgg19
14 from tensorflow.python.client import device_lib
15 import re
16 import math
17 import pickle
18 from skimage.transform import rescale, resize,
     ↪ downscale_local_mean
19 import sklearn
20 import pandas as pd
21 from sklearn import metrics
22
23 def block_convolv_a( _tensor_ ):
```

```
24      _tensor_=Conv2D( filters=  64, kernel_size=(11,11),
          ↪ strides=(1,1), padding='valid', activation='relu'
          ↪  )(_tensor_)
25      _tensor_=Conv2D( filters=  64, kernel_size=(5,5),
          ↪ strides=(1,1), padding='valid', activation='relu'
          ↪  )(_tensor_)
26      _tensor_=Conv2D( filters=  64, kernel_size=(3,3),
          ↪ strides=(1,1), padding='valid', activation='relu'
          ↪  )(_tensor_)
27      _tensor_=Conv2D( filters=  64, kernel_size=(3,3),
          ↪ strides=(1,1), padding='valid', activation='relu'
          ↪  )(_tensor_)
28      _tensor_=AveragePooling2D( pool_size=(2,2), strides
          ↪ =(2,2), padding='same' )(_tensor_)
29      _tensor_=Conv2D( filters=  64, kernel_size=(3,3),
          ↪ strides=(1,1), padding='valid', activation='relu'
          ↪  )(_tensor_)
30      _tensor_=Conv2D( filters=  64, kernel_size=(3,3),
          ↪ strides=(1,1), padding='valid', activation='relu'
          ↪  )(_tensor_)
31      _tensor_=AveragePooling2D( pool_size=(2,2), strides
          ↪ =(2,2), padding='same' )(_tensor_)
32      _tensor_=Conv2D( filters=  64, kernel_size=(3,3),
          ↪ strides=(1,1), padding='valid', activation='relu'
          ↪  )(_tensor_)
33      _tensor_=Conv2D( filters=  64, kernel_size=(3,3),
          ↪ strides=(1,1), padding='valid', activation='relu'
          ↪  )(_tensor_)
```

```
34      _tensor_=AveragePooling2D( pool_size=(2,2), strides
          ↪ =(2,2), padding='same' )(_tensor_)
35      _tensor_=Conv2D( filters=  64, kernel_size=(3,3),
          ↪ strides=(1,1), padding='valid', activation='relu'
          ↪  )(_tensor_)
36      _tensor_=Conv2D( filters=  64, kernel_size=(3,3),
          ↪ strides=(1,1), padding='valid', activation='relu'
          ↪  )(_tensor_)
37      _tensor_=AveragePooling2D( pool_size=(2,2), strides
          ↪ =(2,2), padding='same' )(_tensor_)
38      # _tensor_=Conv2D( filters=  64, kernel_size=(3,3),
          ↪ strides=(1,1), padding='valid', activation='relu'
          ↪  )(_tensor_)
39      # _tensor_=Conv2D( filters=  64, kernel_size=(3,3),
          ↪ strides=(1,1), padding='valid', activation='relu'
          ↪  )(_tensor_)
40      return _tensor_
41
42  def block_convolv_b( _tensor_ ):
43      _tensor_=Conv2D( filters=  32, kernel_size=(9,9),
          ↪ strides=(1,1), padding='valid', activation='relu'
          ↪  )(_tensor_)
44      _tensor_=Conv2D( filters=  32, kernel_size=(7,7),
          ↪ strides=(1,1), padding='valid', activation='relu'
          ↪  )(_tensor_)
45      _tensor_=Conv2D( filters=  64, kernel_size=(5,5),
          ↪ strides=(1,1), padding='valid', activation='relu'
          ↪  )(_tensor_)
```

```
46      _tensor_=Conv2D( filters=  64, kernel_size=(3,3),
          ↪ strides=(1,1), padding='valid', activation='relu'
          ↪  )(_tensor_)
47      _tensor_=Conv2D( filters=  64, kernel_size=(3,3),
          ↪ strides=(1,1), padding='valid', activation='relu'
          ↪  )(_tensor_)
48      _tensor_=AveragePooling2D( pool_size=(2,2), strides
          ↪ =(2,2), padding='same' )(_tensor_)
49      _tensor_=Conv2D( filters=  64, kernel_size=(3,3),
          ↪ strides=(1,1), padding='valid', activation='relu'
          ↪  )(_tensor_)
50      _tensor_=Conv2D( filters=  64, kernel_size=(3,3),
          ↪ strides=(1,1), padding='valid', activation='relu'
          ↪  )(_tensor_)
51      _tensor_=AveragePooling2D( pool_size=(2,2), strides
          ↪ =(2,2), padding='same' )(_tensor_)
52      _tensor_=Conv2D( filters=  64, kernel_size=(3,3),
          ↪ strides=(1,1), padding='valid', activation='relu'
          ↪  )(_tensor_)
53      _tensor_=Conv2D( filters=  64, kernel_size=(3,3),
          ↪ strides=(1,1), padding='valid', activation='relu'
          ↪  )(_tensor_)
54      _tensor_=AveragePooling2D( pool_size=(2,2), strides
          ↪ =(2,2), padding='same' )(_tensor_)
55      _tensor_=Conv2D( filters=  64, kernel_size=(3,3),
          ↪ strides=(1,1), padding='valid', activation='relu'
          ↪  )(_tensor_)
```

```
56      _tensor_=Conv2D( filters=  64, kernel_size=(3,3),
          ↪ strides=(1,1), padding='valid', activation='relu'
          ↪  )(_tensor_)
57      _tensor_=AveragePooling2D( pool_size=(2,2), strides
          ↪ =(2,2), padding='same' )(_tensor_)
58      _tensor_=Conv2D( filters=  64, kernel_size=(3,3),
          ↪ strides=(1,1), padding='valid', activation='relu'
          ↪  )(_tensor_)
59      _tensor_=Conv2D( filters=  64, kernel_size=(3,3),
          ↪ strides=(1,1), padding='valid', activation='relu'
          ↪  )(_tensor_)
60      return _tensor_
61
62  def block_convolv_c( _tensor_ ):
63      #_tensor_=Conv2D( filters=  64, kernel_size=(11,11),
          ↪ strides=(3,3), padding='valid', activation='relu'
          ↪  )(_tensor_)
64      _tensor_=Conv2D( filters=  64, kernel_size=(7,7),
          ↪ strides=(2,2), padding='valid', activation='relu'
          ↪  )(_tensor_)
65      _tensor_=Conv2D( filters=  64, kernel_size=(5,5),
          ↪ strides=(2,2), padding='valid', activation='relu'
          ↪  )(_tensor_)
66      #_tensor_=Flatten()(_tensor_)
67      #_tensor_=Conv2D( filters=  64, kernel_size=(5,5),
          ↪ strides=(1,1), padding='valid', activation='relu'
          ↪  )(_tensor_)
```

```
68      #_tensor_=Conv2D( filters=  64, kernel_size=(5,5),
            ↪ strides=(1,1), padding='valid', activation='relu'
            ↪  )(_tensor_)
69      #_tensor_=Conv2D( filters=  64, kernel_size=(5,5),
            ↪ strides=(1,1), padding='valid', activation='relu'
            ↪  )(_tensor_)
70      #_tensor_=Conv2D( filters=  64, kernel_size=(5,5),
            ↪ strides=(2,2), padding='valid', activation='relu'
            ↪  )(_tensor_)
71      #_tensor_=Conv2D( filters=  64, kernel_size=(3,3),
            ↪ strides=(1,1), padding='valid', activation='relu'
            ↪  )(_tensor_)
72      #_tensor_=Conv2D( filters=  64, kernel_size=(3,3),
            ↪ strides=(1,1), padding='valid', activation='relu'
            ↪  )(_tensor_)
73      return _tensor_
74
75  def block_convolv_d( _tensor_ ):
76      #_tensor_=Conv2D( filters=  64, kernel_size=(13,13),
            ↪ strides=(3,3), padding='valid', activation='relu'
            ↪  )(_tensor_)
77      _tensor_=Conv2D( filters=  64, kernel_size=(7,7),
            ↪ strides=(2,2), padding='valid', activation='relu'
            ↪  )(_tensor_)
78      _tensor_=Conv2D( filters=  64, kernel_size=(5,5),
            ↪ strides=(2,2), padding='valid', activation='relu'
            ↪  )(_tensor_)
```

```
79      _tensor_=Conv2D( filters=  64, kernel_size=(5,5),
          ↪ strides=(1,1), padding='valid', activation='relu'
          ↪  )(_tensor_)
80      _tensor_=Conv2D( filters=  64, kernel_size=(5,5),
          ↪ strides=(1,1), padding='valid', activation='relu'
          ↪  )(_tensor_)
81      _tensor_=Conv2D( filters=  64, kernel_size=(5,5),
          ↪ strides=(1,1), padding='valid', activation='relu'
          ↪  )(_tensor_)
82      _tensor_=Conv2D( filters=  64, kernel_size=(3,3),
          ↪ strides=(1,1), padding='valid', activation='relu'
          ↪  )(_tensor_)
83      _tensor_=Conv2D( filters=  64, kernel_size=(3,3),
          ↪ strides=(1,1), padding='valid', activation='relu'
          ↪  )(_tensor_)
84      _tensor_=Conv2D( filters=  64, kernel_size=(3,3),
          ↪ strides=(1,1), padding='valid', activation='relu'
          ↪  )(_tensor_)
85
86      return _tensor_
87
88  def block_convolv_5x5_b( _tensor_ ):
89      # 256 x 256 x 1
90      _tensor_=Conv2D( filters=  32, kernel_size=(5,5),
          ↪ strides=(1,1), padding='same', activation='relu'
          ↪ )(_tensor_)
91      _tensor_=Conv2D( filters=  64, kernel_size=(3,3),
          ↪ strides=(1,1), padding='same', activation='relu'
          ↪ )(_tensor_)
```

```
92      #_tensor_=BN()(_tensor_)
93      #_tensor_=Conv2D( filters=  64, kernel_size=(3,3),
           ↪ strides=(2,2), padding='valid', activation='relu'
           ↪  )(_tensor_)
94      _tensor_=MaxPooling2D( pool_size=(2,2), strides=(2,2),
           ↪ padding='same' )(_tensor_)
95      #_tensor_=BN()(_tensor_)
96      _tensor_=Conv2D( filters=  64, kernel_size=(3,3),
           ↪ strides=(1,1), padding='same', activation='relu'
           ↪ )(_tensor_)
97      _tensor_=Conv2D( filters=  64, kernel_size=(3,3),
           ↪ strides=(1,1), padding='same', activation='relu'
           ↪ )(_tensor_)
98      #_tensor_=BN()(_tensor_)
99      #_tensor_=Conv2D( filters= 128, kernel_size=(3,3),
           ↪ strides=(2,2), padding='valid', activation='relu'
           ↪  )(_tensor_)
100     _tensor_=MaxPooling2D( pool_size=(2,2), strides=(2,2),
           ↪ padding='same' )(_tensor_)
101     #_tensor_=BN()(_tensor_)
102     _tensor_=Conv2D( filters= 128, kernel_size=(3,3),
           ↪ strides=(1,1), padding='same', activation='relu'
           ↪ )(_tensor_)
103     _tensor_=Conv2D( filters= 128, kernel_size=(3,3),
           ↪ strides=(1,1), padding='same', activation='relu'
           ↪ )(_tensor_)
104     #_tensor_=BN()(_tensor_)
```

```
105     #_tensor_=Conv2D( filters= 256, kernel_size=(3,3),
            ↪ strides=(2,2), padding='valid', activation='relu'
            ↪  )(_tensor_)
106     _tensor_=MaxPooling2D( pool_size=(2,2), strides=(2,2),
            ↪ padding='same' )(_tensor_)
107     #_tensor_=BN()(_tensor_)
108     _tensor_=Conv2D( filters= 128, kernel_size=(3,3),
            ↪ strides=(1,1), padding='same', activation='relu'
            ↪ )(_tensor_)
109     _tensor_=Conv2D( filters= 128, kernel_size=(3,3),
            ↪ strides=(1,1), padding='same', activation='relu'
            ↪ )(_tensor_)
110     _tensor_=MaxPooling2D( pool_size=(2,2), strides=(2,2),
            ↪ padding='same' )(_tensor_)
111     #_tensor_=BN()(_tensor_)
112     _tensor_=Conv2D( filters= 256, kernel_size=(3,3),
            ↪ strides=(1,1), padding='same', activation='relu'
            ↪ )(_tensor_)
113     _tensor_=Conv2D( filters= 256, kernel_size=(3,3),
            ↪ strides=(1,1), padding='same', activation='relu'
            ↪ )(_tensor_)
114     #_tensor_=BN()(_tensor_)
115     return _tensor_
116
117 def block_reconstruction_5x5_b( _tensor_ ):
118     _tensor_=Conv2D( filters= 256, kernel_size=(3,3),
            ↪ strides=(1,1), padding='same', activation='relu'
            ↪ )(_tensor_)
```

```
119     _tensor_=Conv2D( filters= 256, kernel_size=(3,3),
            ↪ strides=(1,1), padding='same', activation='relu'
            ↪ )(_tensor_)
120     _tensor_=Conv2DTranspose( filters=256, kernel_size
            ↪ =(3,3), strides=(2,2), dilation_rate=1, padding='
            ↪ same', activation='relu' )(_tensor_)
121     #
122     _tensor_=Conv2D( filters= 256, kernel_size=(3,3),
            ↪ strides=(1,1), padding='same', activation='relu'
            ↪ )(_tensor_)
123     _tensor_=Conv2D( filters= 256, kernel_size=(3,3),
            ↪ strides=(1,1), padding='same', activation='relu'
            ↪ )(_tensor_)
124     _tensor_=Conv2DTranspose( filters=256, kernel_size
            ↪ =(3,3), strides=(2,2), dilation_rate=1, padding='
            ↪ same', activation='relu' )(_tensor_)
125     #
126     _tensor_=Conv2D( filters= 128, kernel_size=(3,3),
            ↪ strides=(1,1), padding='same', activation='relu'
            ↪ )(_tensor_)
127     _tensor_=Conv2D( filters= 128, kernel_size=(3,3),
            ↪ strides=(1,1), padding='same', activation='relu'
            ↪ )(_tensor_)
128     _tensor_=Conv2DTranspose( filters=128, kernel_size
            ↪ =(3,3), strides=(2,2), dilation_rate=1, padding='
            ↪ same', activation='relu' )(_tensor_)
129     #
```

```
130     _tensor_=Conv2D( filters= 128, kernel_size=(3,3),
          ↪ strides=(1,1), padding='same', activation='relu'
          ↪ )(_tensor_)
131     _tensor_=Conv2D( filters= 128, kernel_size=(3,3),
          ↪ strides=(1,1), padding='same', activation='relu'
          ↪ )(_tensor_)
132     _tensor_=Conv2DTranspose( filters=128, kernel_size
          ↪ =(3,3), strides=(2,2), dilation_rate=1, padding='
          ↪ same', activation='relu' )(_tensor_)
133     #
134     _tensor_=Conv2D( filters=  64, kernel_size=(3,3),
          ↪ strides=(1,1), padding='same', activation='relu'
          ↪ )(_tensor_)
135     _tensor_=Conv2D( filters=  64, kernel_size=(3,3),
          ↪ strides=(1,1), padding='same', activation='relu'
          ↪ )(_tensor_)
136     _tensor_=Conv2D( filters=   1, kernel_size=(3,3),
          ↪ strides=(1,1), padding='same', activation='
          ↪ sigmoid', name='reconstruction' )(_tensor_)
137     return _tensor_
138
139
140 def autoencoder_1( y, x, in_channel ):
141
142     input_tensor = Input((y, x, in_channel))
143
144     bottleneck = block_convolv_5x5_b( input_tensor )
145
```

```
146     reconstruction = block_reconstruction_5x5_b( bottleneck
            ↪ )
147
148     model = Model( inputs=input_tensor, outputs=
            ↪ reconstruction )
149
150     model.compile( loss='mse', optimizer = Adadelta() )
151
152     model.summary()
153
154     return model
155
156
157
158 def classifier_1( y, x, in_channel ):
159
160     input_tensor = Input((y, x, in_channel))
161
162     bottleneck = block_convolv_5x5_b( input_tensor )
163
164     reconstruction = block_reconstruction_5x5_b(bottleneck)
165
166     bottleneck = Flatten()(bottleneck)
167
168     tensor_dense = Dense(1024,activation='relu')(bottleneck
            ↪ )
169     tensor_dense = BN()(tensor_dense)
170     #tensor_dense = Dropout(0.3)(tensor_dense)
```

```
171     tensor_dense = Dense(1024,activation='relu')(
            ↪ tensor_dense)
172     tensor_dense = BN()(tensor_dense)
173     #tensor_dense = Dropout(0.3)(tensor_dense)
174     tensor_dense = Dense(1024,activation='relu')(
            ↪ tensor_dense)
175     tensor_dense = BN()(tensor_dense)
176     #tensor_dense = Dropout(0.3)(tensor_dense)
177     output_softmax = Dense(5,activation='softmax',name='
            ↪ output_softmax')(tensor_dense)
178
179     model = Model(inputs=input_tensor, outputs=[
            ↪ reconstruction,output_softmax])
180
181     model.compile( loss={'reconstruction':'mse', '
            ↪ output_softmax':'categorical_crossentropy'},
            ↪ optimizer = Adadelta(), metrics=['accuracy'] )
182
183     model.summary()
184
185     return model
186
187 def classifier_2( y, x, in_channel ):
188
189     _tensor_ = input_tensor = Input((y, x, in_channel))
190     _tensor_ = GaussianNoise(0.3)(_tensor_)
191     _tensor_ = bottleneck = block_convolv_a( _tensor_ )
192     _tensor_ = Flatten()(_tensor_)
193     _tensor_ = BN()(_tensor_)
```

```
194     _tensor_ = Dense(1024,activation='relu')(_tensor_)
195     _tensor_ = BN()(_tensor_)
196     output_softmax = Dense(2,activation='softmax',name='
          ↪ output_softmax')(_tensor_)
197
198     model = Model(inputs=input_tensor, outputs=
          ↪ output_softmax)
199     model.compile( loss='categorical_crossentropy',
          ↪ optimizer = Adadelta(), metrics=['accuracy'] )
200     model.summary()
201
202     return model
203
204 def classifier_3( y, x, in_channel ):
205
206     _tensor_ = input_tensor = Input((y, x, in_channel))
207     _tensor_ = GaussianNoise(0.3)(_tensor_)
208     _tensor_ = bottleneck = block_convolv_b( _tensor_ )
209     _tensor_ = Flatten()(_tensor_)
210     _tensor_ = BN()(_tensor_)
211     _tensor_ = Dense(1024,activation='relu')(_tensor_)
212     _tensor_ = BN()(_tensor_)
213     output_softmax = Dense(2,activation='softmax',name='
          ↪ output_softmax')(_tensor_)
214
215     model = Model(inputs=input_tensor, outputs=
          ↪ output_softmax)
216     model.compile( loss='categorical_crossentropy',
          ↪ optimizer = Adadelta(), metrics=['accuracy'] )
```

```
217     model.summary()
218
219     return model
220
221 def classifier_4( y, x, in_channel ):
222
223     _tensor_ = input_tensor = Input((y, x, in_channel))
224     _tensor_ = GaussianNoise(0.3)(_tensor_)
225     _tensor_ = bottleneck = block_convolv_c( _tensor_ )
226     _tensor_ = Flatten()(_tensor_)
227     for _ in range(3):
228         _tensor_ = BN()(_tensor_)
229         _tensor_ = GaussianNoise(0.3)(_tensor_)
230         _tensor_ = Dense(1024,activation='relu')(_tensor_)
231     _tensor_ = BN()(_tensor_)
232     output_softmax = Dense(2,activation='softmax',name='
        ↪ output_softmax')(_tensor_)
233
234     model = Model(inputs=input_tensor, outputs=
        ↪ output_softmax)
235     model.compile( loss='categorical_crossentropy',
        ↪ optimizer = Adadelta(), metrics=['accuracy'] )
236     model.summary()
237
238     return model
239
240 def classifier_5( y, x, in_channel ):
241
242     _tensor_ = input_tensor = Input((y, x, in_channel))
```

```
243     _tensor_ = GaussianNoise(0.3)(_tensor_)
244     _tensor_ = bottleneck = block_convolv_d( _tensor_ )
245     _tensor_ = Flatten()(_tensor_)
246     for _ in range(2):
247         _tensor_ = BN()(_tensor_)
248         _tensor_ = GaussianNoise(0.3)(_tensor_)
249         _tensor_ = Dense(1024,activation='relu')(_tensor_)
250     _tensor_ = BN()(_tensor_)
251     output_softmax = Dense(2,activation='softmax',name='
          ↪ output_softmax')(_tensor_)
252
253     model = Model(inputs=input_tensor, outputs=
          ↪ output_softmax)
254     model.compile( loss='categorical_crossentropy',
          ↪ optimizer = Adadelta(), metrics=['accuracy'] )
255     model.summary()
256
257     return model
```

Código 7.3: Libreria de modelos 2D

7.5 modelos CNN 3D

```
1  import keras
2  import os
3  import sys
4  import fnmatch
5  import random
6  import numpy as np
7  from keras.layers import Input, Conv3D, Conv3DTranspose,
     ↪ MaxPooling3D, UpSampling3D, AveragePooling3D,
     ↪ Cropping3D
8  from keras.layers import Dropout, Activation, Flatten,
     ↪ Concatenate, Dense, Reshape, Add, PReLU, LeakyReLU
9  from keras.activations import relu
10 from keras import regularizers
11 from keras.layers.normalization import BatchNormalization
     ↪ as BN
12 from keras.layers import GaussianNoise
13 from keras.models import Model, Sequential, load_model
14 from keras.optimizers import RMSprop, Adam, Adagrad, SGD,
     ↪ Adadelta
15 from keras.applications import vgg16, vgg19
16 from tensorflow.python.client import device_lib
17 import re
18 import math
19 import pickle
20 from skimage.transform import rescale, resize,
     ↪ downscale_local_mean
21 import sklearn
22 import pandas as pd
```

```
23 from sklearn import metrics
24
25 def block_convolv_a( _tensor_ ):
26     _tensor_=Conv3D( filters=  32, kernel_size=(7,7,7),
       ↪ strides=(2,2,2), padding='valid', activation='
       ↪ relu' )(_tensor_)
27     _tensor_=Conv3D( filters=  32, kernel_size=(5,5,5),
       ↪ strides=(2,2,2), padding='valid', activation='
       ↪ relu' )(_tensor_)
28     _tensor_=Conv3D( filters=  32, kernel_size=(5,5,5),
       ↪ strides=(2,2,2), padding='valid', activation='
       ↪ relu' )(_tensor_)
29     _tensor_=Conv3D( filters=  32, kernel_size=(5,5,5),
       ↪ strides=(2,2,2), padding='valid', activation='
       ↪ relu' )(_tensor_)
30     _tensor_=Conv3D( filters=  32, kernel_size=(5,5,5),
       ↪ strides=(2,2,2), padding='valid', activation='
       ↪ relu' )(_tensor_)
31     return _tensor_
32
33 def block_convolv_b( _tensor_ ):
34     _tensor_=Conv3D( filters=  32, kernel_size=(9,9,9),
       ↪ strides=(3,3,3), padding='valid', activation='
       ↪ relu' )(_tensor_)
35     _tensor_=Conv3D( filters=  32, kernel_size=(5,5,5),
       ↪ strides=(2,2,2), padding='valid', activation='
       ↪ relu' )(_tensor_)
```

```
36      _tensor_=Conv3D( filters=  32, kernel_size=(5,5,5),
          ↪ strides=(2,2,2), padding='valid', activation='
          ↪ relu' )(_tensor_)
37      _tensor_=Conv3D( filters=  32, kernel_size=(5,5,5),
          ↪ strides=(2,2,2), padding='valid', activation='
          ↪ relu' )(_tensor_)
38      return _tensor_
39
40  def block_convolv_c( _tensor_ ):
41      _tensor_=Conv3D( filters=  32, kernel_size=(11,11,11),
          ↪ strides=(3,3,3), padding='valid', activation='
          ↪ relu' )(_tensor_)
42      _tensor_=Conv3D( filters=  32, kernel_size=(5,5,5),
          ↪ strides=(2,2,2), padding='valid', activation='
          ↪ relu' )(_tensor_)
43      _tensor_=Conv3D( filters=  32, kernel_size=(5,5,5),
          ↪ strides=(2,2,2), padding='valid', activation='
          ↪ relu' )(_tensor_)
44      _tensor_=Conv3D( filters=  32, kernel_size=(5,5,5),
          ↪ strides=(2,2,2), padding='valid', activation='
          ↪ relu' )(_tensor_)
45      return _tensor_
46
47  def block_dense_conv( _tensor_, filters=16, blocks=5 ):
48      my_tensors = [_tensor_]
49      for i in range(blocks):
50          _tensor_ = BN()(_tensor_)
51          _tensor_ = Activation('relu')(_tensor_)
```

```
52          _tensor_conv_ = Conv3D( filters = filters,
                ↪ kernel_size = (3,3,3), strides = (1,1,1),
                ↪ padding = 'same', activation = 'linear' )(
                ↪ _tensor_)
53          my_tensors.append( _tensor_conv_ )
54          _tensor_ = Concatenate()( my_tensors )
55      return _tensor_
56
57
58  def classifier_1( z, y, x, in_channel ):
59
60      _tensor_ = input_tensor = Input( (z, y, x, in_channel)
            ↪ )
61      _tensor_ = GaussianNoise( 0.3 )(_tensor_)
62      _tensor_ = bottleneck = block_convolv_a( _tensor_ )
63      _tensor_ = Flatten()( _tensor_ )
64      _tensor_ = BN()(_tensor_)
65      _tensor_ = Dense(1024,activation='relu')(_tensor_)
66      _tensor_ = BN()(_tensor_)
67      output_softmax = Dense(2,activation='softmax',name='
            ↪ output_softmax')(_tensor_)
68
69      model = Model( inputs=input_tensor, outputs=
            ↪ output_softmax )
70      model.compile( loss='categorical_crossentropy',
            ↪ optimizer = Adadelta(), metrics=['accuracy'] )
71      model.summary()
72
73      return model
```

```
74
75 def classifier_2( z, y, x, in_channel ):
76
77     _tensor_ = input_tensor = Input( (z, y, x, in_channel)
       ↪ )
78     _tensor_ = GaussianNoise( 0.3 )(_tensor_)
79     _tensor_ = Conv3D( filters = 8, kernel_size = (3,3,3),
       ↪ strides = (2,2,2), padding = 'valid' )(_tensor_)
80     _tensor_ = block_dense_conv( _tensor_ )
81     _tensor_ = Conv3D( filters = 8, kernel_size = (1,1,1),
       ↪ strides = (1,1,1), padding = 'valid' )(_tensor_)
82     _tensor_ = MaxPooling3D( pool_size = (2,2,2) )(_tensor_
       ↪ )
83     _tensor_ = block_dense_conv( _tensor_ )
84     _tensor_ = Conv3D( filters = 8, kernel_size = (1,1,1),
       ↪ strides = (1,1,1), padding = 'valid' )(_tensor_)
85     _tensor_ = MaxPooling3D( pool_size = (2,2,2) )(_tensor_
       ↪ )
86     _tensor_ = Flatten()( _tensor_ )
87     _tensor_ = Dropout( rate = 0.2 )(_tensor_)
88     _tensor_ = Dense(256,activation='relu')(_tensor_)
89     _tensor_ = Dropout( rate = 0.2 )(_tensor_)
90     _tensor_ = Dense(128,activation='relu')(_tensor_)
91     _tensor_ = Dropout( rate = 0.2 )(_tensor_)
92     output_softmax = Dense(2,activation='softmax',name='
       ↪ output_softmax')(_tensor_)
93
94     model = Model( inputs=input_tensor, outputs=
       ↪ output_softmax )
```

```
95      model.compile( loss='categorical_crossentropy',
            ↪ optimizer = SGD( lr = 0.01, momentum = 0.9, decay
            ↪  = 0.0005 ), metrics=['accuracy'] )
96      model.summary()
97
98      return model
99
100 def classifier_3( z, y, x, in_channel ):
101
102     _tensor_ = input_tensor = Input( (z, y, x, in_channel)
            ↪ )
103     _tensor_ = GaussianNoise( 0.3 )(_tensor_)
104     _tensor_ = Conv3D( filters = 32, kernel_size = (3,3,3),
            ↪  strides = (2,2,2), padding = 'valid' )(_tensor_)
105     _tensor_ = block_dense_conv( _tensor_, blocks = 3)
106     _tensor_ = Conv3D( filters = 32, kernel_size = (1,1,1),
            ↪  strides = (1,1,1), padding = 'valid' )(_tensor_)
107     _tensor_ = MaxPooling3D( pool_size = (2,2,2) )(_tensor_
            ↪ )
108     _tensor_ = block_dense_conv( _tensor_, blocks = 4)
109     _tensor_ = Conv3D( filters = 32, kernel_size = (1,1,1),
            ↪  strides = (1,1,1), padding = 'valid' )(_tensor_)
110     _tensor_ = MaxPooling3D( pool_size = (2,2,2) )(_tensor_
            ↪ )
111     _tensor_ = Conv3D( filters = 16, kernel_size = (1,1,1),
            ↪  strides = (1,1,1), padding = 'valid' )(_tensor_)
112     _tensor_ = MaxPooling3D( pool_size = (2,2,2) )(_tensor_
            ↪ )
113     _tensor_ = Flatten()( _tensor_ )
```

```
114     _tensor_ = Dropout( rate = 0.4 )(_tensor_)
115     _tensor_ = Dense(1024,activation='relu')(_tensor_)
116     _tensor_ = Dropout( rate = 0.4 )(_tensor_)
117     _tensor_ = Dense(512,activation='relu')(_tensor_)
118     _tensor_ = Dropout( rate = 0.4 )(_tensor_)
119     output_softmax = Dense(2,activation='softmax',name='
          ↪ output_softmax')(_tensor_)
120
121     model = Model( inputs=input_tensor, outputs=
          ↪ output_softmax )
122     model.compile( loss='categorical_crossentropy',
          ↪ optimizer = SGD( lr = 0.01, momentum = 0.9, decay
          ↪  = 0.0005 ), metrics=['accuracy'] )
123     model.summary()
124
125     return model
126
127 def classifier_4( z, y, x, in_channel ):
128
129     _tensor_ = input_tensor = Input( (z, y, x, in_channel)
          ↪ )
130     #_tensor_ = GaussianNoise( 0.3 )(_tensor_)
131
132     _tensor_ = Conv3D( filters = 32, kernel_size = (3,3,3),
          ↪  strides = (2,2,2), padding = 'valid' )(_tensor_)
133
134     _tensor_ = block_dense_conv( _tensor_, blocks = 5)
135
```

```
136     _tensor_ = Conv3D( filters = 16, kernel_size = (1,1,1),
          ↪  strides = (1,1,1), padding = 'valid' )(_tensor_)
137     _tensor_ = MaxPooling3D( pool_size = (2,2,2) )(_tensor_
          ↪ )
138
139     _tensor_ = block_dense_conv( _tensor_, blocks = 5)
140
141     _tensor_ = Conv3D( filters = 8, kernel_size = (1,1,1),
          ↪ strides = (1,1,1), padding = 'valid' )(_tensor_)
142     _tensor_ = MaxPooling3D( pool_size = (2,2,2) )(_tensor_
          ↪ )
143
144     _tensor_ = Flatten()( _tensor_ )
145     _tensor_ = Dropout( rate = 0.4 )(_tensor_)
146     _tensor_ = Dense(1000,activation='relu')(_tensor_)
147     _tensor_ = Dropout( rate = 0.4 )(_tensor_)
148     _tensor_ = Dense(100,activation='relu')(_tensor_)
149     _tensor_ = Dropout( rate = 0.4 )(_tensor_)
150     output_softmax = Dense(2,activation='softmax',name='
          ↪ output_softmax')(_tensor_)
151
152     model = Model( inputs=input_tensor, outputs=
          ↪ output_softmax )
153     model.compile( loss='categorical_crossentropy',
          ↪ optimizer = SGD( lr = 0.01, momentum = 0.9, decay
          ↪  = 0.0005 ), metrics=['accuracy'] )
154     model.summary()
155
156     return model
```

```
157
158 def classifier_5( z, y, x, in_channel ):
159
160     _tensor_ = input_tensor = Input( (z, y, x, in_channel)
        ↪ )
161     _tensor_ = Conv3D( filters = 32, kernel_size = (5,5,5),
        ↪  strides = (2,2,2), padding = 'valid' )(_tensor_)
162     _tensor_ = BN()(_tensor_)
163
164     _tensor_ = Conv3D( filters = 32, kernel_size = (3,3,3),
        ↪  strides = (1,1,1), padding = 'valid' )(_tensor_)
165     _tensor_ = BN()(_tensor_)
166     _tensor_ = MaxPooling3D( pool_size = (2,2,2) )(_tensor_
        ↪ )
167
168     _tensor_ = Conv3D( filters = 32, kernel_size = (3,3,3),
        ↪  strides = (1,1,1), padding = 'valid' )(_tensor_)
169     _tensor_ = BN()(_tensor_)
170     _tensor_ = MaxPooling3D( pool_size = (2,2,2) )(_tensor_
        ↪ )
171
172     _tensor_ = Conv3D( filters = 32, kernel_size = (3,3,3),
        ↪  strides = (1,1,1), padding = 'valid' )(_tensor_)
173     _tensor_ = BN()(_tensor_)
174     _tensor_ = MaxPooling3D( pool_size = (2,2,2) )(_tensor_
        ↪ )
175
176     _tensor_ = Flatten()( _tensor_ )
177     _tensor_ = Dense(1000,activation='relu')(_tensor_)
```

```
178     _tensor_ = Dense(100,activation='relu')(_tensor_)
179     output_softmax = Dense(2,activation='softmax',name='
          ↪ output_softmax')(_tensor_)
180
181     model = Model( inputs=input_tensor, outputs=
          ↪ output_softmax )
182     model.compile( loss='categorical_crossentropy',
          ↪ optimizer = Adadelta(), metrics=['accuracy'] )
183     model.summary()
184
185     return model
186
187 def classifier_6( z, y, x, in_channel ):
188
189     _tensor_ = input_tensor = Input( (z, y, x, in_channel)
          ↪ )
190     _tensor_edad_ = Input( (1,) )
191
192     _tensor_ = Conv3D( filters = 32, kernel_size = (5,5,5),
          ↪  strides = (2,2,2), padding = 'valid' )(_tensor_)
193     _tensor_ = BN()(_tensor_)
194
195     _tensor_ = Conv3D( filters = 32, kernel_size = (3,3,3),
          ↪  strides = (1,1,1), padding = 'valid' )(_tensor_)
196     _tensor_ = BN()(_tensor_)
197     _tensor_ = MaxPooling3D( pool_size = (2,2,2) )(_tensor_
          ↪ )
198
```

```
199     _tensor_ = Conv3D( filters = 32, kernel_size = (3,3,3),
          ↪  strides = (1,1,1), padding = 'valid' )(_tensor_)
200     _tensor_ = BN()(_tensor_)
201     _tensor_ = MaxPooling3D( pool_size = (2,2,2) )(_tensor_
          ↪ )
202
203     _tensor_ = Conv3D( filters = 32, kernel_size = (3,3,3),
          ↪  strides = (1,1,1), padding = 'valid' )(_tensor_)
204     _tensor_ = BN()(_tensor_)
205     _tensor_ = MaxPooling3D( pool_size = (2,2,2) )(_tensor_
          ↪ )
206
207     _tensor_ = Flatten()( _tensor_ )
208     _tensor_ = Concatenate()([_tensor_, _tensor_edad_])
209     _tensor_ = Dense(1000,activation='relu')(_tensor_)
210     _tensor_ = Dense(100,activation='relu')(_tensor_)
211     output_softmax = Dense(2,activation='softmax',name='
          ↪ output_softmax')(_tensor_)
212
213     model = Model( inputs=[input_tensor, _tensor_edad_],
          ↪ outputs=output_softmax )
214     keras.optimizers.RMSprop(lr=0.001, rho=0.9, epsilon=
          ↪ None, decay=0.0)
215     model.compile( loss='categorical_crossentropy',
          ↪ optimizer = Adadelta(), metrics=['accuracy'] )
216     model.summary()
217
218     return model
219
```

```
220
221 def block_convolv_7( _input_tensor_, length=3, filters=32,
      ↪ batch_normalization=True ):
222     _tensor_ = _input_tensor_
223     if batch_normalization: _tensor_ = BN()(_tensor_)
224     tensors = [_tensor_]
225     for i in range(length):
226         _tensor_ = Conv3D( filters = filters, kernel_size
            ↪ =(3,3,3), strides=(1,1,1), padding='same',
            ↪ activation='linear' )(_tensor_)
227         _tensor_ = LeakyReLU(alpha=0.3)(_tensor_)
228         if batch_normalization: _tensor_ = BN()(_tensor_)
229         tensors.append( _tensor_ )
230     _tensor_ = Concatenate()( tensors )
231     _tensor_ = Conv3D( filters = filters, kernel_size
        ↪ =(3,3,3), strides=(1,1,1), padding='same',
        ↪ activation='linear' )(_tensor_)
232     _tensor_ = LeakyReLU(alpha=0.3)(_tensor_)
233     if batch_normalization: _tensor_ = BN()(_tensor_)
234     return _tensor_
235
236 def classifier_7( z, y, x, in_channels ):
237     batch_normalization_flag=False
238     _tensor_ = input_tensor = Input( (z,y,x,in_channels) )
239     #_tensor_edad_ = Input( (7,) )
240     #
241     _tensor_ = Conv3D( filters = 32, kernel_size = (3,3,3),
        ↪  strides = (2,2,2), padding='valid', activation='
        ↪ linear' )(_tensor_)
```

```
242     #_tensor_ = LeakyReLU(alpha=0.0)(_tensor_)
243     #if batch_normalization_flag: _tensor_ = BN()(_tensor_)
244     #
245     for i in range(4):
246         _tensor_ = block_dense_conv( _tensor_, filters=16,
            ↪ blocks=5 )
247         _tensor_ = Conv3D( filters = 16, kernel_size =
            ↪ (1,1,1), strides = (1,1,1), padding='valid',
            ↪ activation='relu' )(_tensor_)
248         _tensor_ = MaxPooling3D( pool_size = (2,2,2) )(
            ↪ _tensor_)
249     #
250     _tensor_ = Flatten()( _tensor_ )
251     for num_neurons in [1000,100]:
252         _tensor_ = Dropout( rate=0.3 )(_tensor_)
253         _tensor_ = Dense( num_neurons, activation='relu' )(
            ↪ _tensor_)
254         #_tensor_ = LeakyReLU(alpha=0.3)(_tensor_)
255     #
256     if batch_normalization_flag:
257         _tensor_ = BN()(_tensor_)
258     else:
259         _tensor_ = Dropout( rate=0.3 )(_tensor_)
260     output_softmax = Dense(2,activation='softmax',name='
        ↪ output_softmax')(_tensor_)
261     #
262     model = Model( inputs=input_tensor, outputs=
        ↪ output_softmax )
```

```
263     #model.compile( loss='categorical_crossentropy',
            ↪ optimizer = Adadelta(), metrics=['accuracy'] )
264     model.compile( loss='categorical_crossentropy',
            ↪ optimizer = SGD( lr=1.0e-2, momentum=0.9, decay
            ↪ =0.005 ), metrics=['accuracy'] )
265     #model.compile( loss='mse', optimizer = Adadelta(),
            ↪ metrics=['accuracy'] )
266     model.summary()
267     #
268     return model
269
270 def classifier_8( z, y, x, in_channels ):
271     alpha_relu=0.2
272     batch_normalization_flag=False
273     _tensor_ = input_tensor = Input( (z,y,x,in_channels) )
274     #
275     _tensor_ = Conv3D( filters = 32, kernel_size = (5,5,5),
            ↪  strides = (2,2,2), padding='valid', activation='
            ↪ linear' )(_tensor_)
276     _tensor_ = BN()(_tensor_)
277     #_tensor_ = relu( _tensor_, alpha=alpha_relu )#(
            ↪ _tensor_)
278     #_tensor_ = Activation( activation='relu', alpha=
            ↪ alpha_relu )(_tensor_)
279     #
280     _tensor_ = Conv3D( filters = 32, kernel_size = (3,3,3),
            ↪  strides = (1,1,1), padding='valid', activation='
            ↪ linear' )(_tensor_)
281     _tensor_ = BN()(_tensor_)
```

```
282     #_tensor_ = relu( _tensor_, alpha=alpha_relu )#(
            ↪ _tensor_)
283     _tensor_ = Conv3D( filters = 32, kernel_size = (3,3,3),
            ↪  strides = (1,1,1), padding='valid', activation='
            ↪ linear' )(_tensor_)
284     _tensor_ = BN()(_tensor_)
285     #_tensor_ = relu( _tensor_, alpha=alpha_relu )#(
            ↪ _tensor_)
286     #
287     _tensor_ = AveragePooling3D( pool_size = (2,2,2) )(
            ↪ _tensor_)
288     #
289     _tensor_ = Conv3D( filters = 32, kernel_size = (3,3,3),
            ↪  strides = (1,1,1), padding='valid', activation='
            ↪ linear' )(_tensor_)
290     _tensor_ = BN()(_tensor_)
291     #_tensor_ = Activation( activation='relu', alpha=
            ↪ alpha_relu )(_tensor_)
292     _tensor_ = Conv3D( filters = 32, kernel_size = (3,3,3),
            ↪  strides = (1,1,1), padding='valid', activation='
            ↪ linear' )(_tensor_)
293     _tensor_ = BN()(_tensor_)
294     #_tensor_ = Activation( activation='relu', alpha=
            ↪ alpha_relu )(_tensor_)
295     #
296     _tensor_ = AveragePooling3D( pool_size = (2,2,2) )(
            ↪ _tensor_)
297     #
```

```
298     _tensor_ = Conv3D( filters = 32, kernel_size = (3,3,3),
          ↪  strides = (1,1,1), padding='valid', activation='
          ↪ linear' )(_tensor_)
299     _tensor_ = BN()(_tensor_)
300     #_tensor_ = Activation( activation='relu', alpha=
          ↪ alpha_relu )(_tensor_)
301     _tensor_ = Conv3D( filters = 32, kernel_size = (3,3,3),
          ↪  strides = (1,1,1), padding='valid', activation='
          ↪ linear' )(_tensor_)
302     _tensor_ = BN()(_tensor_)
303     #_tensor_ = Activation( activation='relu', alpha=
          ↪ alpha_relu )(_tensor_)
304     #
305     _tensor_ = AveragePooling3D( pool_size = (2,2,2) )(
          ↪ _tensor_)
306     #
307     _tensor_ = Conv3D( filters = 32, kernel_size = (3,3,3),
          ↪  strides = (1,1,1), padding='valid', activation='
          ↪ linear' )(_tensor_)
308     _tensor_ = BN()(_tensor_)
309     # #_tensor_ = Activation( activation='relu', alpha=
          ↪ alpha_relu )(_tensor_)
310     # _tensor_ = Conv3D( filters = 32, kernel_size =
          ↪ (3,3,3), strides = (1,1,1), padding='valid',
          ↪ activation='linear' )(_tensor_)
311     # _tensor_ = BN()(_tensor_)
312     # #_tensor_ = Activation( activation='relu', alpha=
          ↪ alpha_relu )(_tensor_)
313     #
```

```
314     _tensor_ = MaxPooling3D( pool_size = (2,2,2) )(_tensor_
            ↪ )
315     #
316     # _tensor_ = Conv3D( filters = 32, kernel_size =
            ↪ (2,3,2), strides = (1,1,1), padding='valid',
            ↪ activation='linear' )(_tensor_)
317     # _tensor_ = BN()(_tensor_)
318     # #_tensor_ = Activation( activation='relu', alpha=
            ↪ alpha_relu )(_tensor_)
319     # _tensor_ = Conv3D( filters = 32, kernel_size =
            ↪ (3,3,3), strides = (1,1,1), padding='valid',
            ↪ activation='linear' )(_tensor_)
320
321     _tensor_ = Flatten()( _tensor_ )
322     """
323     if batch_normalization_flag:
324         _tensor_ = BN()(_tensor_)
325     else:
326         _tensor_ = Dropout( rate=0.5 )(_tensor_)
327     """
328     _tensor_ = Dense(100,activation='linear')(_tensor_)
329     _tensor_ = BN()(_tensor_)
330     #_tensor_ = Activation( activation='relu', alpha=
            ↪ alpha_relu )(_tensor_)
331     """
332     if batch_normalization_flag:
333         _tensor_ = BN()(_tensor_)
334     else:
335         _tensor_ = Dropout( rate=0.5 )(_tensor_)
```

```
336     """
337     #_tensor_ = Dense(500,activation='linear')(_tensor_)
338     #_tensor_ = BN()(_tensor_)
339     #_tensor_ = Activation( activation='relu', alpha=
          ↪ alpha_relu )(_tensor_)
340     """
341     if batch_normalization_flag:
342         _tensor_ = BN()(_tensor_)
343     else:
344         _tensor_ = Dropout( rate=0.5 )(_tensor_)
345     """
346     #_tensor_ = Dense(100,activation='linear')(_tensor_)
347     #_tensor_ = BN()(_tensor_)
348     #_tensor_ = Activation( activation='relu', alpha=
          ↪ alpha_relu )(_tensor_)
349     #
350     """
351     if batch_normalization_flag:
352         _tensor_ = BN()(_tensor_)
353     else:
354         _tensor_ = Dropout( rate=0.5 )(_tensor_)
355     #
356     """
357     output_softmax = Dense(2,activation='softmax',name='
          ↪ output_softmax')(_tensor_)
358     #
359     model = Model( inputs=input_tensor, outputs=
          ↪ output_softmax )
```

```
360     #opt=keras.optimizers.RMSprop(lr=0.001, rho=0.9,
            ↪ epsilon=None, decay=1e-6)
361     opt=keras.optimizers.Adadelta(lr=1.0, rho=0.95, epsilon
            ↪ =None, decay=1e-6)
362     model.compile( loss='categorical_crossentropy',
            ↪ optimizer = opt, metrics=['accuracy'] )
363     #model.compile( loss='mse', optimizer = SGD( lr=1.0e-5,
            ↪  momentum=0.9 ), metrics=['accuracy'] )
364     model.summary()
365     #
366     return model
367
368 def classifier_9( z, y, x, in_channels ):
369     _tensor_ = input_tensor = Input( (z,y,x,in_channels) )
370     #
371     _tensor_ = Conv3D( filters = 32, kernel_size = (5,5,5),
            ↪  strides = (2,2,2), padding='valid', activation='
            ↪ linear' )(_tensor_)
372     _tensor_ = BN()(_tensor_)
373     for _ in range(3):
374         _tensor_ = Conv3D( filters = 32, kernel_size =
                ↪ (5,5,5), strides = (2,2,2), padding='valid',
                ↪ activation='linear' )(_tensor_)
375         _tensor_ = BN()(_tensor_)
376     _tensor_ = Conv3D( filters = 32, kernel_size = (3,3,3),
            ↪  strides = (1,1,1), padding='valid', activation='
            ↪ linear' )(_tensor_)
```

```
377     _tensor_ = Conv3D( filters = 32, kernel_size = (3,3,3),
          ↪  strides = (1,1,1), padding='valid', activation='
          ↪ linear' )(_tensor_)
378     _tensor_ = Conv3D( filters = 32, kernel_size = (3,3,3),
          ↪  strides = (1,1,1), padding='valid', activation='
          ↪ linear' )(_tensor_)
379     _tensor_ = BN()(_tensor_)
380     #
381     _tensor_ = Flatten()( _tensor_ )
382     #
383     for _ in range(3):
384         _tensor_ = Dense(100,activation='linear')(_tensor_)
385         _tensor_ = BN()(_tensor_)
386     #
387     output_softmax = Dense(2,activation='sigmoid',name='
          ↪ output_softmax')(_tensor_)
388     #
389     model = Model( inputs=input_tensor, outputs=
          ↪ output_softmax )
390     #model.compile( loss='categorical_crossentropy',
          ↪ optimizer = Adadelta(), metrics=['accuracy'] )
391     model.compile( loss='mse', optimizer = Adadelta(),
          ↪ metrics=['accuracy'] )
392     #model.compile( loss='mse', optimizer = SGD( lr=1.0e-5,
          ↪  momentum=0.9 ), metrics=['accuracy'] )
393     model.summary()
394     #
395     return model
396 100
```

```
397 def autoencoder_10( z, y, x, in_channels ):
398     _tensor_ = input_tensor = Input( (z,y,x,in_channels) )
        ↪ # 91, 109, 91, 1
399
400     # Encoder
401     _tensor_ = Conv3D( filters = 100, kernel_size = (3,3,3)
        ↪ , strides = (1,1,1), padding = "same", activation
        ↪  = "relu" )(_tensor_) # 91, 109, 91, 128
402     _tensor_ = MaxPooling3D( pool_size = (2,2,2), padding =
        ↪  "same" )(_tensor_) # 46, 55, 46, 64
403
404     _tensor_ = Conv3D( filters = 128, kernel_size = (3,3,3)
        ↪ , strides = (1,1,1), padding = "same", activation
        ↪  = "relu" )(_tensor_) # 46, 55, 46, 128
405     _tensor_ = MaxPooling3D( pool_size = (2,2,2), padding =
        ↪  "same" )(_tensor_) # 23, 28, 23, 128
406
407     _tensor_ = Conv3D( filters = 128, kernel_size = (3,3,3)
        ↪ , strides = (1,1,1), padding = "same", activation
        ↪  = "relu" )(_tensor_) # 23, 28, 23, 128
408     _tensor_ = MaxPooling3D( pool_size = (2,2,2), padding =
        ↪  "same" )(_tensor_) # 12, 14, 12, 128
409
410     _tensor_ = Conv3D( filters = 256, kernel_size = (3,3,3)
        ↪ , strides = (1,1,1), padding = "same", activation
        ↪  = "relu" )(_tensor_) # 12, 14, 12, 256
411     _tensor_ = MaxPooling3D( pool_size = (2,2,2), padding =
        ↪  "same" )(_tensor_) # 6, 7, 6, 256
412
```

```
413     _tensor_ = Conv3D( filters = 256, kernel_size = (3,3,3)
          ↪ , strides = (1,1,1), padding = "same", activation
          ↪  = "relu" )(_tensor_) # 6, 7, 6, 256
414
415     #Decoder
416     _tensor_ = UpSampling3D( size = (2,2,2) )(_tensor_) #
          ↪ 12, 14, 12, 256
417     _tensor_ = Conv3D( filters = 256, kernel_size = (3,3,3)
          ↪ , strides = (1,1,1), padding = "same", activation
          ↪  = "relu" )(_tensor_) # 12, 14, 12, 256
418
419     _tensor_ = UpSampling3D( size = (2,2,2) )(_tensor_) #
          ↪ 24, 28, 24, 256
420     _tensor_ = Conv3D( filters = 128, kernel_size = (3,3,3)
          ↪ , strides = (1,1,1), padding = "same", activation
          ↪  = "relu" )(_tensor_) # 24, 28, 24, 128
421
422     _tensor_ = UpSampling3D( size = (2,2,2) )(_tensor_) #
          ↪ 48, 56, 48, 128
423     _tensor_ = Conv3D( filters = 128, kernel_size = (3,3,3)
          ↪ , strides = (1,1,1), padding = "same", activation
          ↪  = "relu" )(_tensor_) # 48, 56, 48, 128
424
425     _tensor_ = UpSampling3D( size = (2,2,2) )(_tensor_) #
          ↪ 96, 112, 96, 128
426     _tensor_ = Cropping3D( cropping = ((2,3),(1,2), (2,3))
          ↪ )(_tensor_) # 91, 109, 91, 128
427
```

```
428     output_sigmoid = Conv3D( filters = 1, kernel_size =
          ↪ (3,3,3), strides = (1,1,1), padding = "same",
          ↪ activation = "sigmoid" )(_tensor_) # 91, 108, 91,
          ↪  1
429
430     model = Model( inputs = input_tensor, outputs =
          ↪ output_sigmoid )
431     model.compile( loss = 'mse', optimizer = Adadelta(),
          ↪ metrics = ['accuracy'] )
432     model.summary()
433
434     return model
435
436 def autoencoder_11( z, y, x, in_channels ):
437     _tensor_ = input_tensor = Input( (z,y,x,in_channels) )
          ↪ # 91, 109, 91, 1
438
439     # Encoder
440     _tensor_ = Conv3D( filters = 100, kernel_size = (3,3,3)
          ↪ , strides = (1,1,1), padding = "same", activation
          ↪  = "relu", kernel_regularizer = regularizers.l2
          ↪ (0.01) )(_tensor_) # 91, 109, 91, 128
441     _tensor_ = MaxPooling3D( pool_size = (2,2,2), padding =
          ↪  "same" )(_tensor_) # 46, 55, 46, 128
442
443     _tensor_ = Conv3D( filters = 128, kernel_size = (3,3,3)
          ↪ , strides = (1,1,1), padding = "same", activation
          ↪  = "relu", kernel_regularizer = regularizers.l2
          ↪ (0.01) )(_tensor_) # 46, 55, 46, 128
```

```
444     _tensor_ = MaxPooling3D( pool_size = (2,2,2), padding =
          ↪  "same" )(_tensor_) # 23, 28, 23, 128
445
446     _tensor_ = Conv3D( filters = 128, kernel_size = (3,3,3)
          ↪ , strides = (1,1,1), padding = "same", activation
          ↪  = "relu", kernel_regularizer = regularizers.l2
          ↪ (0.01) )(_tensor_) # 23, 28, 23, 128
447     _tensor_ = MaxPooling3D( pool_size = (2,2,2), padding =
          ↪  "same" )(_tensor_) # 12, 14, 12, 128
448
449     _tensor_ = Conv3D( filters = 256, kernel_size = (3,3,3)
          ↪ , strides = (1,1,1), padding = "same", activation
          ↪  = "relu", kernel_regularizer = regularizers.l2
          ↪ (0.01) )(_tensor_) # 12, 14, 12, 256
450     _tensor_ = MaxPooling3D( pool_size = (2,2,2), padding =
          ↪  "same" )(_tensor_) # 6, 7, 6, 256
451
452     _tensor_ = Conv3D( filters = 256, kernel_size = (3,3,3)
          ↪ , strides = (1,1,1), padding = "same", activation
          ↪  = "relu", kernel_regularizer = regularizers.l2
          ↪ (0.01) )(_tensor_) # 6, 7, 6, 256
453
454     #Decoder
455     _tensor_ = UpSampling3D( size = (2,2,2) )(_tensor_) #
          ↪ 12, 14, 12, 256
456     _tensor_ = Conv3D( filters = 256, kernel_size = (3,3,3)
          ↪ , strides = (1,1,1), padding = "same", activation
          ↪  = "relu", kernel_regularizer = regularizers.l2
          ↪ (0.01) )(_tensor_) # 12, 14, 12, 256
```

```
457
458     _tensor_ = UpSampling3D( size = (2,2,2) )(_tensor_) #
            ↪ 24, 28, 24, 256
459     _tensor_ = Conv3D( filters = 128, kernel_size = (3,3,3)
            ↪ , strides = (1,1,1), padding = "same", activation
            ↪  = "relu", kernel_regularizer = regularizers.l2
            ↪ (0.01) )(_tensor_) # 24, 28, 24, 128
460
461     _tensor_ = UpSampling3D( size = (2,2,2) )(_tensor_) #
            ↪ 48, 56, 48, 128
462     _tensor_ = Conv3D( filters = 128, kernel_size = (3,3,3)
            ↪ , strides = (1,1,1), padding = "same", activation
            ↪  = "relu", kernel_regularizer = regularizers.l2
            ↪ (0.01) )(_tensor_) # 48, 56, 48, 128
463
464     _tensor_ = UpSampling3D( size = (2,2,2) )(_tensor_) #
            ↪ 96, 112, 96, 128
465     _tensor_ = Cropping3D( cropping = ((2,3),(1,2), (2,3))
            ↪ )(_tensor_) # 91, 109, 91, 128
466
467     output_sigmoid = Conv3D( filters = 1, kernel_size =
            ↪ (3,3,3), strides = (1,1,1), padding = "same",
            ↪ activation = "sigmoid", kernel_regularizer =
            ↪ regularizers.l2(0.01) )(_tensor_) # 91, 108, 91,
            ↪ 1
468
469     model = Model( inputs = input_tensor, outputs =
            ↪ output_sigmoid )
```

```
470     model.compile( loss = 'mse', optimizer = Adadelta(),
          ↪ metrics = ['accuracy'] )
471     model.summary()
472
473     return model
474
475 def classifier_12( z, y, x, in_channels ):
476     _tensor_ = input_tensor = Input( (z,y,x,in_channels) )
477     #
478     _tensor_ = Conv3D( filters = 32, kernel_size = (5,5,5),
          ↪  strides = (2,2,2), padding='valid', activation='
          ↪ relu' )(_tensor_)
479     _tensor_ = BN()(_tensor_)
480     for _ in range(3):
481         _tensor_ = Conv3D( filters = 32, kernel_size =
              ↪ (5,5,5), strides = (2,2,2), padding='valid',
              ↪ activation='relu' )(_tensor_)
482         _tensor_ = BN()(_tensor_)
483     _tensor_ = Conv3D( filters = 32, kernel_size = (3,3,3),
          ↪  strides = (1,1,1), padding='valid', activation='
          ↪ relu' )(_tensor_)
484     _tensor_ = Conv3D( filters = 32, kernel_size = (3,3,3),
          ↪  strides = (1,1,1), padding='valid', activation='
          ↪ relu' )(_tensor_)
485     _tensor_ = Conv3D( filters = 32, kernel_size = (3,3,3),
          ↪  strides = (1,1,1), padding='valid', activation='
          ↪ relu' )(_tensor_)
486     _tensor_ = BN()(_tensor_)
487     #
```

```
488     _tensor_ = Flatten()( _tensor_ )
489     #
490     for _ in range(3):
491         _tensor_ = Dense(100,activation='relu')(_tensor_)
492         _tensor_ = BN()(_tensor_)
493     #
494     output_softmax = Dense(2,activation='softmax',name='
          ↪ output_softmax')(_tensor_)
495     #
496     model = Model( inputs=input_tensor, outputs=
          ↪ output_softmax )
497     model.compile( loss='categorical_crossentropy',
          ↪ optimizer = Adadelta(), metrics=['accuracy'] )
498     #model.compile( loss='mse', optimizer = Adadelta(),
          ↪ metrics=['accuracy'] )
499     #model.compile( loss='mse', optimizer = SGD( lr=1.0e-5,
          ↪  momentum=0.9 ), metrics=['accuracy'] )
500     model.summary()
501     #
502     return model
503
504 def classifier_13( z, y, x, in_channels ):
505     _tensor_ = input_tensor = Input( (z,y,x,in_channels) )
506     #
507     _tensor_ = Conv3D( filters = 32, kernel_size = (5,5,5),
          ↪  strides = (2,2,2), padding='valid', activation='
          ↪ relu', kernel_regularizer = regularizers.l2(0.01)
          ↪  )(_tensor_)
508     _tensor_ = BN()(_tensor_)
```

```
509     for _ in range(2):
510         _tensor_ = Conv3D( filters = 32, kernel_size =
              ↪ (5,5,5), strides = (2,2,2), padding='valid',
              ↪ activation='relu', kernel_regularizer =
              ↪ regularizers.l2(0.01) )(_tensor_)
511         _tensor_ = BN()(_tensor_)
512     _tensor_ = Conv3D( filters = 32, kernel_size = (3,3,3),
          ↪  strides = (1,1,1), padding='valid', activation='
          ↪ relu', kernel_regularizer = regularizers.l2(0.01)
          ↪  )(_tensor_)
513     _tensor_ = Conv3D( filters = 32, kernel_size = (3,3,3),
          ↪  strides = (1,1,1), padding='valid', activation='
          ↪ relu', kernel_regularizer = regularizers.l2(0.01)
          ↪  )(_tensor_)
514     _tensor_ = Conv3D( filters = 32, kernel_size = (3,3,3),
          ↪  strides = (1,1,1), padding='valid', activation='
          ↪ relu', kernel_regularizer = regularizers.l2(0.01)
          ↪  )(_tensor_)
515     _tensor_ = BN()(_tensor_)
516     #
517     _tensor_ = Flatten()( _tensor_ )
518     #
519     for _ in range(3):
520         _tensor_ = Dense(100,activation='relu')(_tensor_)
521         _tensor_ = BN()(_tensor_)
522     #
523     output_softmax = Dense(2,activation='softmax',name='
          ↪ output_softmax')(_tensor_)
524     #
```

```
525     model = Model( inputs=input_tensor, outputs=
          ↪ output_softmax )
526     model.compile( loss='categorical_crossentropy',
          ↪ optimizer = Adadelta(), metrics=['accuracy'] )
527     #model.compile( loss='mse', optimizer = Adadelta(),
          ↪ metrics=['accuracy'] )
528     #model.compile( loss='mse', optimizer = SGD( lr=1.0e-5,
          ↪  momentum=0.9 ), metrics=['accuracy'] )
529     model.summary()
530     #
531     return model
532
533 def classifier_14( z, y, x, in_channels ):
534     _tensor_ = input_tensor = Input( (z,y,x,in_channels) )
535     #
536     _tensor_ = Conv3D( filters = 32, kernel_size = (5,5,5),
          ↪  strides = (2,2,2), padding='valid', activation='
          ↪ relu', kernel_regularizer = regularizers.l2(0.01)
          ↪  )(_tensor_)
537     _tensor_ = BN()(_tensor_)
538     for _, filter in zip(range(3),(32,64,128)):
539         _tensor_ = Conv3D( filters = filter, kernel_size =
              ↪ (5,5,5), strides = (2,2,2), padding='valid',
              ↪ activation='relu', kernel_regularizer =
              ↪ regularizers.l2(0.01) )(_tensor_)
540         _tensor_ = BN()(_tensor_)
541     _tensor_ = Conv3D( filters = 128, kernel_size = (3,3,3)
          ↪ , strides = (1,1,1), padding='valid', activation=
```

```
        ↪ 'relu', kernel_regularizer = regularizers.l2
        ↪ (0.01) )(_tensor_)
542     _tensor_ = Conv3D( filters = 128, kernel_size = (3,3,3)
        ↪ , strides = (1,1,1), padding='valid', activation=
        ↪ 'relu', kernel_regularizer = regularizers.l2
        ↪ (0.01) )(_tensor_)
543     _tensor_ = Conv3D( filters = 128, kernel_size = (3,3,3)
        ↪ , strides = (1,1,1), padding='valid', activation=
        ↪ 'relu', kernel_regularizer = regularizers.l2
        ↪ (0.01) )(_tensor_)
544     _tensor_ = Conv3D( filters = 128, kernel_size = (3,3,3)
        ↪ , strides = (1,1,1), padding='valid', activation=
        ↪ 'relu', kernel_regularizer = regularizers.l2
        ↪ (0.01) )(_tensor_)
545     _tensor_ = BN()(_tensor_)
546     #
547     _tensor_ = Flatten()( _tensor_ )
548     #
549     for _, filter in zip(range(2),(100,100)):
550         _tensor_ = Dense(filter,activation='relu')(_tensor_
            ↪ )
551         _tensor_ = BN()(_tensor_)
552     #
553     output_softmax = Dense(2,activation='softmax',name='
        ↪ output_softmax')(_tensor_)
554     #
555     model = Model( inputs=input_tensor, outputs=
        ↪ output_softmax )
```

```
556     model.compile( loss='categorical_crossentropy',
            ↪ optimizer = Adam(), metrics=['accuracy'] )
557     #model.compile( loss='mse', optimizer = Adadelta(),
            ↪ metrics=['accuracy'] )
558     #model.compile( loss='mse', optimizer = SGD( lr=1.0e-5,
            ↪  momentum=0.9 ), metrics=['accuracy'] )
559     model.summary()
560     #
561     return model
562
563
564
565
566 def block_ConvBnRePool(tensor,filters,filter_size,pool_size
    ↪ ,doble=False):
567
568     tensor_ = Conv3D(
569         filters,
570         filter_size,
571         padding='same',
572         kernel_initializer=keras.initializers.glorot_normal
            ↪ (),
573         bias_initializer=keras.initializers.glorot_normal()
            ↪ ,
574         #kernel_regularizer=keras.regularizers.l2(),
575         #bias_regularizer=keras.regularizers.l2(),
576         activity_regularizer=keras.regularizers.l2()
577     )(tensor)
578
```

```
579     tensor_ = (BN())(tensor_)
580     tensor_ = (Activation('relu'))(tensor_)
581 
582     if doble:
583         tensor_ = Conv3D(
584             filters,
585             filter_size,
586             padding='same',
587             kernel_initializer=keras.initializers.
                ↪ glorot_normal(),
588             bias_initializer=keras.initializers.
                ↪ glorot_normal(),
589             #kernel_regularizer=keras.regularizers.l2(),
590             #bias_regularizer=keras.regularizers.l2(),
591             activity_regularizer=keras.regularizers.l2()
592         )(tensor_)
593 
594         tensor_ = (BN())(tensor_)
595         tensor_ = (Activation('relu'))(tensor_)
596     tensor_ = MaxPooling3D(pool_size=pool_size)(tensor_)
597     return tensor_
598 
599 
600 def block_dense(tensor,Neurons,activation):
601     tensor_ = Dense(
602         Neurons,
603         kernel_initializer=keras.initializers.glorot_normal
            ↪ (),
```

```
604         bias_initializer=keras.initializers.glorot_normal()
              ↪ ,
605         #kernel_regularizer=keras.regularizers.l2(),
606         #bias_regularizer=keras.regularizers.l2(),
607         activity_regularizer=keras.regularizers.l2()
608     )(tensor)
609     tensor_ = BN()(tensor_)
610     tensor_ = Activation(activation)(tensor_)
611     return tensor_
612
613
614 def block_output(tensor,Neurons):
615     tensor_ = Dense(
616         Neurons,
617         activation="softmax",
618         kernel_initializer=keras.initializers.glorot_normal
              ↪ (),
619         bias_initializer=keras.initializers.glorot_normal()
              ↪ ,
620         #kernel_regularizer=keras.regularizers.l2(),
621         #bias_regularizer=keras.regularizers.l2(),
622         activity_regularizer=keras.regularizers.l2()
623     )(tensor)
624     return tensor_
625
626 def classifier_16( z, y, x, in_channels ):
627     input_tensor = Input((z, y, x, in_channels))
628     tensor_3 = block_ConvBnRePool(input_tensor,32,(3,3,3)
          ↪ ,(2,2,2))
```

```
629     tensor_3 = block_ConvBnRePool(tensor_3,32,(3,3,3)
          ↪ ,(2,2,2))
630     tensor_3 = block_ConvBnRePool(tensor_3,32,(3,3,3)
          ↪ ,(2,2,2))
631     #tensor_3 = block_ConvBnRePool(tensor_3,32,(3,3,3)
          ↪ ,(2,2,2))
632     tensor_3 = block_ConvBnRePool(tensor_3,64,(3,3,3)
          ↪ ,(2,2,2))
633     #tensor_3 = block_ConvBnRePool(tensor_3,64,(3,3,3)
          ↪ ,(2,2,2))
634     #tensor_3 = block_ConvBnRePool(tensor_3,512,(3,3,3)
          ↪ ,(2,2,2))
635     tensor_3 = Flatten()(tensor_3)
636
637     tensor_5 = block_ConvBnRePool(input_tensor,32,(5,5,5)
          ↪ ,(2,2,2))
638     tensor_5 = block_ConvBnRePool(tensor_5,32,(5,5,5)
          ↪ ,(2,2,2))
639     tensor_5 = block_ConvBnRePool(tensor_5,32,(5,5,5)
          ↪ ,(2,2,2))
640     #tensor_5 = block_ConvBnRePool(tensor_5,32,(5,5,5)
          ↪ ,(2,2,2))
641     tensor_5 = block_ConvBnRePool(tensor_5,64,(5,5,5)
          ↪ ,(2,2,2))
642     #tensor_5 = block_ConvBnRePool(tensor_5,512,(5,5,5)
          ↪ ,(2,2,2))
643     #tensor_5 = block_ConvBnRePool(tensor_5,64,(5,5,5)
          ↪ ,(2,2,2))
644     tensor_5 = (Flatten())(tensor_5)
```

```
645
646     tensor_3_5 = Concatenate()([tensor_3,tensor_5])
647
648     tensor_3_5 = block_dense(tensor_3_5,512,"relu")
649     tensor_3_5 = block_dense(tensor_3_5,128,"relu")
650     tensor_3_5 = block_dense(tensor_3_5,64,"relu")
651     tensor_output = block_output(tensor_3_5,2)
652     model = Model(inputs=input_tensor, outputs=
          ↪ tensor_output)
653     model.compile( loss='categorical_crossentropy',
          ↪ optimizer = Adam(), metrics=['accuracy'] )
654
655     model.summary()
656     return model
657 def classifier_17( z, y, x, in_channels):
658     alpha_relu=0.2
659     batch_normalization_flag=False
660     _tensor_1 = input_tensor = Input( (z,y,x,in_channels) )
661     #
662     _tensor_1 = Conv3D( filters = 32, kernel_size = (5,5,5)
          ↪ , strides = (2,2,2), padding='valid', activation=
          ↪ 'linear' )(_tensor_1)
663     _tensor_1 = BN()(_tensor_1)
664     #_tensor_1 = relu( _tensor_1, alpha=alpha_relu )#(
          ↪ _tensor_1)
665     #_tensor_1 = Activation( activation='relu', alpha=
          ↪ alpha_relu )(_tensor_1)
666     #
```

```
667     _tensor_1 = Conv3D( filters = 32, kernel_size = (3,3,3)
          ↪ , strides = (1,1,1), padding='valid', activation=
          ↪ 'linear' )(_tensor_1)
668     _tensor_1 = BN()(_tensor_1)
669     #_tensor_1 = relu( _tensor_1, alpha=alpha_relu )#(
          ↪ _tensor_1)
670     _tensor_1 = Conv3D( filters = 32, kernel_size = (3,3,3)
          ↪ , strides = (1,1,1), padding='valid', activation=
          ↪ 'linear' )(_tensor_1)
671     _tensor_1 = BN()(_tensor_1)
672     #_tensor_1 = relu( _tensor_1, alpha=alpha_relu )#(
          ↪ _tensor_1)
673     #
674     _tensor_1 = AveragePooling3D( pool_size = (2,2,2) )(
          ↪ _tensor_1)
675     #
676     _tensor_1 = Conv3D( filters = 32, kernel_size = (3,3,3)
          ↪ , strides = (1,1,1), padding='valid', activation=
          ↪ 'linear' )(_tensor_1)
677     _tensor_1 = BN()(_tensor_1)
678     #_tensor_1 = Activation( activation='relu', alpha=
          ↪ alpha_relu )(_tensor_1)
679     _tensor_1 = Conv3D( filters = 32, kernel_size = (3,3,3)
          ↪ , strides = (1,1,1), padding='valid', activation=
          ↪ 'linear' )(_tensor_1)
680     _tensor_1 = BN()(_tensor_1)
681     #_tensor_1 = Activation( activation='relu', alpha=
          ↪ alpha_relu )(_tensor_1)
682     #
```

```
683     _tensor_1 = AveragePooling3D( pool_size = (2,2,2) )(
            ↪ _tensor_1)
684     #
685     _tensor_1 = Conv3D( filters = 32, kernel_size = (3,3,3)
            ↪ , strides = (1,1,1), padding='valid', activation=
            ↪ 'linear' )(_tensor_1)
686     _tensor_1 = BN()(_tensor_1)
687     #_tensor_1 = Activation( activation='relu', alpha=
            ↪ alpha_relu )(_tensor_1)
688     _tensor_1 = Conv3D( filters = 32, kernel_size = (3,3,3)
            ↪ , strides = (1,1,1), padding='valid', activation=
            ↪ 'linear' )(_tensor_1)
689     _tensor_1 = BN()(_tensor_1)
690     #_tensor_1 = Activation( activation='relu', alpha=
            ↪ alpha_relu )(_tensor_1)
691     #
692     _tensor_1 = AveragePooling3D( pool_size = (2,2,2) )(
            ↪ _tensor_1)
693     #
694     # _tensor_1 = Conv3D( filters = 32, kernel_size =
            ↪ (3,3,3), strides = (1,1,1), padding='valid',
            ↪ activation='linear' )(_tensor_1)
695     # _tensor_1 = BN()(_tensor_1)
696     # #_tensor_1 = Activation( activation='relu', alpha=
            ↪ alpha_relu )(_tensor_1)
697     # _tensor_1 = Conv3D( filters = 32, kernel_size =
            ↪ (3,3,3), strides = (1,1,1), padding='valid',
            ↪ activation='linear' )(_tensor_1)
698     # _tensor_1 = BN()(_tensor_1)
```

```
699     # #_tensor_1 = Activation( activation='relu', alpha=
            ↪ alpha_relu )(_tensor_1)
700     # #
701     # #_tensor_1 = MaxPooling3D( pool_size = (2,2,2) )(
            ↪ _tensor_1)
702     # #
703     # _tensor_1 = Conv3D( filters = 32, kernel_size =
            ↪ (2,3,2), strides = (1,1,1), padding='valid',
            ↪ activation='linear' )(_tensor_1)
704     # _tensor_1 = BN()(_tensor_1)
705     # #_tensor_1 = Activation( activation='relu', alpha=
            ↪ alpha_relu )(_tensor_1)
706     # #_tensor_1 = Conv3D( filters = 32, kernel_size =
            ↪ (3,3,3), strides = (1,1,1), padding='valid',
            ↪ activation='linear' )(_tensor_1)
707     #
708     _tensor_1 = Flatten()( _tensor_1 )
709
710
711
712     _tensor_2 = input_tensor
713     #
714     _tensor_2 = Conv3D( filters = 32, kernel_size = (5,5,5)
            ↪ , strides = (2,2,2), padding='valid', activation=
            ↪ 'linear' )(_tensor_2)
715     _tensor_2 = BN()(_tensor_2)
716     #_tensor_2 = relu( _tensor_2, alpha=alpha_relu )#(
            ↪ _tensor_2)
```

```
717     #_tensor_2 = Activation( activation='relu', alpha=
            ↪ alpha_relu )(_tensor_2)
718     #
719     _tensor_2 = Conv3D( filters = 32, kernel_size = (3,3,3)
            ↪ , strides = (1,1,1), padding='valid', activation=
            ↪ 'linear' )(_tensor_2)
720     _tensor_2 = BN()(_tensor_2)
721     #_tensor_2 = relu( _tensor_2, alpha=alpha_relu )#(
            ↪ _tensor_2)
722     _tensor_2 = Conv3D( filters = 32, kernel_size = (3,3,3)
            ↪ , strides = (1,1,1), padding='valid', activation=
            ↪ 'linear' )(_tensor_2)
723     _tensor_2 = BN()(_tensor_2)
724     #_tensor_2 = relu( _tensor_2, alpha=alpha_relu )#(
            ↪ _tensor_2)
725     #
726     _tensor_2 = AveragePooling3D( pool_size = (2,2,2) )(
            ↪ _tensor_2)
727     #
728     _tensor_2 = Conv3D( filters = 32, kernel_size = (3,3,3)
            ↪ , strides = (1,1,1), padding='valid', activation=
            ↪ 'linear' )(_tensor_2)
729     _tensor_2 = BN()(_tensor_2)
730     #_tensor_2 = Activation( activation='relu', alpha=
            ↪ alpha_relu )(_tensor_2)
731     _tensor_2 = Conv3D( filters = 32, kernel_size = (3,3,3)
            ↪ , strides = (1,1,1), padding='valid', activation=
            ↪ 'linear' )(_tensor_2)
732     _tensor_2 = BN()(_tensor_2)
```

```
733     #_tensor_2 = Activation( activation='relu', alpha=
            ↪ alpha_relu )(_tensor_2)
734     #
735     _tensor_2 = AveragePooling3D( pool_size = (2,2,2) )(
            ↪ _tensor_2)
736     #
737     _tensor_2 = Conv3D( filters = 32, kernel_size = (3,3,3)
            ↪ , strides = (1,1,1), padding='valid', activation=
            ↪ 'linear' )(_tensor_2)
738     _tensor_2 = BN()(_tensor_2)
739     #_tensor_2 = Activation( activation='relu', alpha=
            ↪ alpha_relu )(_tensor_2)
740     _tensor_2 = Conv3D( filters = 32, kernel_size = (3,3,3)
            ↪ , strides = (1,1,1), padding='valid', activation=
            ↪ 'linear' )(_tensor_2)
741     _tensor_2 = BN()(_tensor_2)
742     #_tensor_2 = Activation( activation='relu', alpha=
            ↪ alpha_relu )(_tensor_2)
743     #
744     _tensor_2 = AveragePooling3D( pool_size = (2,2,2) )(
            ↪ _tensor_2)
745     # #
746     # _tensor_2 = Conv3D( filters = 32, kernel_size =
            ↪ (3,3,3), strides = (1,1,1), padding='valid',
            ↪ activation='linear' )(_tensor_2)
747     # _tensor_2 = BN()(_tensor_2)
748     # #_tensor_2 = Activation( activation='relu', alpha=
            ↪ alpha_relu )(_tensor_2)
```

```
749     # _tensor_2 = Conv3D( filters = 32, kernel_size =
            ↪ (3,3,3), strides = (1,1,1), padding='valid',
            ↪ activation='linear' )(_tensor_2)
750     # _tensor_2 = BN()(_tensor_2)
751     # #_tensor_2 = Activation( activation='relu', alpha=
            ↪ alpha_relu )(_tensor_2)
752     # #
753     # #_tensor_2 = MaxPooling3D( pool_size = (2,2,2) )(
            ↪ _tensor_2)
754     # #
755     # _tensor_2 = Conv3D( filters = 32, kernel_size =
            ↪ (2,3,2), strides = (1,1,1), padding='valid',
            ↪ activation='linear' )(_tensor_2)
756     # _tensor_2 = BN()(_tensor_2)
757     # #_tensor_2 = Activation( activation='relu', alpha=
            ↪ alpha_relu )(_tensor_2)
758     # #_tensor_2 = Conv3D( filters = 32, kernel_size =
            ↪ (3,3,3), strides = (1,1,1), padding='valid',
            ↪ activation='linear' )(_tensor_2)
759     # #
760     _tensor_2 = Flatten()( _tensor_2 )
761
762     _tensor_1_2 = Concatenate()([_tensor_1,_tensor_2])
763
764
765     """
766     if batch_normalization_flag:
767         _tensor_ = BN()(_tensor_)
768     else:
```

```
769         _tensor_ = Dropout( rate=0.5 )(_tensor_)
770     """
771     _tensor_ = Dense(100,activation='linear')(_tensor_1_2)
772     _tensor_ = BN()(_tensor_)
773     #_tensor_ = Activation( activation='relu', alpha=
            ↪ alpha_relu )(_tensor_)
774     """
775     if batch_normalization_flag:
776         _tensor_ = BN()(_tensor_)
777     else:
778         _tensor_ = Dropout( rate=0.5 )(_tensor_)
779     """
780     #_tensor_ = Dense(500,activation='linear')(_tensor_)
781     #_tensor_ = BN()(_tensor_)
782     #_tensor_ = Activation( activation='relu', alpha=
            ↪ alpha_relu )(_tensor_)
783     """
784     if batch_normalization_flag:
785         _tensor_ = BN()(_tensor_)
786     else:
787         _tensor_ = Dropout( rate=0.5 )(_tensor_)
788     """
789     #_tensor_ = Dense(100,activation='linear')(_tensor_)
790     #_tensor_ = BN()(_tensor_)
791     #_tensor_ = Activation( activation='relu', alpha=
            ↪ alpha_relu )(_tensor_)
792     #
793     """
794     if batch_normalization_flag:
```

```
795         _tensor_ = BN()(_tensor_)
796     else:
797         _tensor_ = Dropout( rate=0.5 )(_tensor_)
798     #
799     """
800     output_softmax = Dense(2,activation='softmax',name='
          ↪ output_softmax')(_tensor_)
801     #
802     model = Model( inputs=input_tensor, outputs=
          ↪ output_softmax )
803     model.compile( loss='categorical_crossentropy',
          ↪ optimizer = Adadelta(), metrics=['accuracy'] )
804     #model.compile( loss='mse', optimizer = SGD( lr=1.0e-5,
          ↪  momentum=0.9 ), metrics=['accuracy'] )
805     model.summary()
806     #
807     return model
```

Código 7.4: Libreria de modelos 3D

7.6 entrenamiento cnn 3D freesurfer

```
1  import tensorflow as tf
2  import keras
3  import os
4  import sys
5  os.environ["CUDA_DEVICE_ORDER"] = "PCI_BUS_ID"
6  os.environ["CUDA_VISIBLE_DEVICES"]="1"
7  import fnmatch
8  import random
9  import numpy as np
10 from keras.layers import Input, Conv2D, Conv2DTranspose,
     ↪ MaxPooling2D, UpSampling2D, Dropout, Activation,
     ↪ Flatten, Concatenate, Dense, Reshape
11 from keras.layers.normalization import BatchNormalization
     ↪ as BN
12 from keras.models import Model, Sequential, load_model
13 from keras.optimizers import RMSprop, Adam, Adagrad, SGD,
     ↪ Adadelta
14 from keras.applications import vgg16, vgg19, ResNet50
15 from tensorflow.python.client import device_lib
16 from resnet3d import Resnet3DBuilder
17 import re
18 import math
19 import pickle
20 from skimage.transform import rescale, resize,
     ↪ downscale_local_mean
21 import sklearn
22 import pandas as pd
23 from sklearn import metrics
```

```
24 from scipy import interp
25 import matplotlib.pyplot as plt
26
27 config = tf.ConfigProto()
28 config.gpu_options.allow_growth=True
29 sess = tf.Session(config=config)
30 from utils import save_metrics
31 from make_charts import make_charts
32 #from my_data_generator_3d import DataGenerator3D
33 from my_data_generator_3d_mask import DataGenerator3Dmask,
     ↪ DataGenerator3Dmask_test
34
35
36 #z, y, x, in_channel = 91, 109, 91, 1
37 z, y, x, in_channel = 200,200,200, 4
38 #z, y, x, in_channel = 182, 218, 182, 1
39 #z, y, x, in_channel = 157, 200, 173, 1
40 #z, y, x, in_channel = 256, 256, 256, 1
41 batch_size = 8
42 epochs = 100
43 is_training=True
44
45 model_number=8
46 model_version= "a_adadelta_imagemask_DA_rgb"    #'b2'
47 if is_training:
48
49
50
51     if model_number == 1:
```

```
52          #
               ↪ ----------------------------------------------------------------
               ↪
53          from adni_cnn_models_3D import classifier_1
54          model = classifier_1( z, y, x, in_channel )
55          #
               ↪ ----------------------------------------------------------------
               ↪
56      elif model_number == 2:
57          #
               ↪ ----------------------------------------------------------------
               ↪
58          from adni_cnn_models_3D import classifier_2
59          model = classifier_2( z, y, x, in_channel )
60          #
               ↪ ----------------------------------------------------------------
               ↪
61      elif model_number == 3:
62          #
               ↪ ----------------------------------------------------------------
               ↪
63          from adni_cnn_models_3D import classifier_3
64          model = classifier_3( z, y, x, in_channel )
65          #
               ↪ ----------------------------------------------------------------
               ↪
66      elif model_number == 4:
```

```
67          #
               ↪ ----------------------------------------------------------
               ↪
68          from adni_cnn_models_3D import classifier_4
69          model = classifier_4( z, y, x, in_channel )
70          #
               ↪ ----------------------------------------------------------
               ↪
71      elif model_number == 5:
72          #
               ↪ ----------------------------------------------------------
               ↪
73          from adni_cnn_models_3D import classifier_5
74          model = classifier_5( z, y, x, in_channel )
75          #
               ↪ ----------------------------------------------------------
               ↪
76      elif model_number == 6:
77          #
               ↪ ----------------------------------------------------------
               ↪
78          from adni_cnn_models_3D import classifier_6
79          model = classifier_6( z, y, x, in_channel )
80          #
               ↪ ----------------------------------------------------------
               ↪
81      elif model_number == 7:
```

```
82          #
               ↪ ----------------------------------------------------------
               ↪
83          from adni_cnn_models_3D import classifier_7
84          model = classifier_7( z, y, x, in_channel )
85          #
               ↪ ----------------------------------------------------------
               ↪
86      elif model_number == 8:
87          #
               ↪ ----------------------------------------------------------
               ↪
88          from adni_cnn_models_3D import classifier_8
89          model = classifier_8( z, y, x, in_channel )
90          #
               ↪ ----------------------------------------------------------
               ↪
91      elif model_number == 9:
92          #
               ↪ ----------------------------------------------------------
               ↪
93          from adni_cnn_models_3D import classifier_9
94          model = classifier_9( z, y, x, in_channel )
95          #
               ↪ ----------------------------------------------------------
               ↪
96      elif model_number == 10:
```

```
97          #
               ↪ ----------------------------------------------------------
               ↪
98          from adni_cnn_models_3D import autoencoder_10
99          model = autoencoder_10( z, y, x, in_channel )
100         #
               ↪ ----------------------------------------------------------
               ↪
101     elif model_number == 11:
102         #
               ↪ ----------------------------------------------------------
               ↪
103         from adni_cnn_models_3D import autoencoder_11
104         model = autoencoder_11( z, y, x, in_channel )
105         #
               ↪ ----------------------------------------------------------
               ↪
106     elif model_number == 12:
107         #
               ↪ ----------------------------------------------------------
               ↪
108         from adni_cnn_models_3D import classifier_12
109         model = classifier_12( z, y, x, in_channel )
110         #
               ↪ ----------------------------------------------------------
               ↪
111     elif model_number == 13:
```

```
112         #
               ↪ ----------------------------------------------------------
               ↪
113         from adni_cnn_models_3D import classifier_13
114         model = classifier_13( z, y, x, in_channel )
115         #
               ↪ ----------------------------------------------------------
               ↪
116     elif model_number == 14:
117         #
               ↪ ----------------------------------------------------------
               ↪
118         from adni_cnn_models_3D import classifier_14
119         model = classifier_14( z, y, x, in_channel )
120         #
               ↪ ----------------------------------------------------------
               ↪
121     elif model_number == 15:
122         #
               ↪ ----------------------------------------------------------
               ↪
123         from adni_cnn_models  import classifier_5
124         #help(Resnet3DBuilder)
125         model = Resnet3DBuilder.build_resnet_34(( z, y, x,
               ↪ in_channel ), 2)
126         model.compile( loss='categorical_crossentropy',
               ↪ optimizer = Adagrad(), metrics=['accuracy'] )
127         model.summary()
```

```
128         #
               ↪ ----------------------------------------------------------
               ↪
129     elif model_number == 16:
130         #
               ↪ ----------------------------------------------------------
               ↪
131         from adni_cnn_models_3D import classifier_16
132         model = classifier_16( z, y, x, in_channel )
133         #
               ↪ ----------------------------------------------------------
               ↪
134     elif model_number == 17:
135         #
               ↪ ----------------------------------------------------------
               ↪
136         from adni_cnn_models_3D import classifier_17
137         model = classifier_17( z, y, x, in_channel )
138         #
               ↪ ----------------------------------------------------------
               ↪
139     else:
140         raise Exception( 'invalid model number' + str(
               ↪ model_number) )
141
142 model_filename='lib2/adni_cnn_classifier-3D-%d-%s.hdf5' % (
      ↪ model_number, model_version)
143 log_filename='log2/adni-cnn-classifier-3D-%d-%s.log' % (
      ↪ model_number, model_version)
```

```
144
145 #model = load_model( model_filename + "-temp" )
146
147
148 if is_training:
149     gpus=0
150     parallel_model = model
151     use_gpus=False
152     if gpus > 0:
153         use_gpus=True
154         try:
155             from keras.utils.training_utils import
                ↪ multi_gpu_model
156             parallel_model = multi_gpu_model( model, gpus=
                ↪ gpus )
157             #parallel_model.compile( loss='
                ↪ categorical_crossentropy', optimizer =
                ↪ Adadelta(), metrics=['accuracy'] )
158             parallel_model.compile( loss=model.loss,
                ↪ optimizer = model.optimizer, metrics=
                ↪ model.metrics )
159             parallel_model.summary()
160         except:
161             use_gpus=False
162             parallel_model = model
163
164
165 #df_adni_3d = pd.read_csv( 'adni_opt_3d_flirt_small/
   ↪ adni_table_3D_flirt_small_balanced.tsv', sep="\t" )
```

```
166 #df_adni_3d = pd.read_csv( 'adni_opt_3d_flirt_small/
      ↪ adni_table_3D_flirt_small_3year_balanced_1.tsv', sep
      ↪ ="\t" )
167 #df_adni_3d = pd.read_csv( 'adni_opt_3d_flirt2/
      ↪ adni_table_3D_flirt_balanced.tsv', sep="\t" )
168 #df_adni_3d = pd.read_csv( 'etc/index-training-3d-balanced.
      ↪ tsv', sep="\t" )
169 df_adni_3d = pd.read_csv( 'adni_opt_3d_freemask_rgb/
      ↪ adni_table_3D_freesurfer_balanced_0.tsv', sep="\t" )
170 if is_training:
171     data_filter  = df_adni_3d['Label']=='CN'
172     data_filter |= df_adni_3d['Label']=='AD'
173     data_filter &= df_adni_3d['Partition']=='tr'
174 
175     #data_filter = df_adni_3d['Partition']=='tr'
176     ##mask = np.random.rand( len(data_filter) )
177     ##data_filter &= mask >= 0.5
178     #print( data_filter.sum() )
179     data_generator_train = DataGenerator3Dmask( df_adni_3d[
          ↪  data_filter ], target_shape=(z, y, x),
          ↪ target_channels=in_channel, indexes_output=[True,
          ↪ True,False,False,False], batch_size=batch_size,
          ↪ rotation=False, shuffle=True, do_balance = True,
          ↪ with_age = False, DA=True, autoencoder = False )
180     #
181 
182     data_filter  = df_adni_3d['Label']=='CN'
183     data_filter |= df_adni_3d['Label']=='AD'
184     data_filter &= df_adni_3d['Partition']=='dev'
```

```
185
186     #data_filter = df_adni_3d['Partition']=='dev'
187     ##data_filter &= mask >= 0.5
188     #print( data_filter.sum() )
189     data_generator_dev = DataGenerator3Dmask( df_adni_3d[
            ↪ data_filter ], target_shape=(z, y, x),
            ↪ target_channels=in_channel, indexes_output=[True,
            ↪ True,False,False,False], batch_size=batch_size,
            ↪ rotation=False, shuffle=True, do_balance = False,
            ↪  with_age = False, autoencoder = False )
190     #
191
192 data_filter  = df_adni_3d['Label']=='CN'
193 data_filter |= df_adni_3d['Label']=='AD'
194 data_filter &= df_adni_3d['Partition']=='te'
195
196 #data_filter = df_adni_3d['Partition']=='te'
197 data_generator_test = DataGenerator3Dmask_test( df_adni_3d[
    ↪  data_filter ], target_shape=(z, y, x),
    ↪ target_channels=in_channel, indexes_output=[True,True
    ↪ ,False,False,False], batch_size=batch_size, rotation=
    ↪ False, shuffle=True, do_balance = False, with_age =
    ↪ False, autoencoder = False )
198 #
199 if is_training:
200     csv_logger = keras.callbacks.CSVLogger( log_filename )
201     checkpoint1 = keras.callbacks.ModelCheckpoint( model_fi
            ↪ lename + '-lost', monitor='val_loss', verbose=1,
```

```
        ↪ save_best_only=True,  save_weights_only=False,
        ↪ mode='min',  period=1 )
202     checkpoint2 = keras.callbacks.ModelCheckpoint( model_fi
        ↪ lename + '-last', monitor='val_loss', verbose=1,
        ↪ save_best_only=False, save_weights_only=False,
        ↪ mode='auto', period=1 )
203     checkpoint3 = keras.callbacks.ModelCheckpoint( model_fi
        ↪ lename + '-acc', monitor='val_acc', verbose=1,
        ↪ save_best_only=True,  save_weights_only=False,
        ↪ mode='min',  period=1 )
204     #
205     vgg16_train = parallel_model.fit_generator(
        ↪ data_generator_train, epochs=epochs, verbose=1,
        ↪ validation_data=data_generator_dev, callbacks = [
        ↪ checkpoint1, checkpoint2, checkpoint3, csv_logger
        ↪ ] )
206     model.save( model_filename )
207
208 if not is_training:
209     parallel_model = load_model( model_filename + '-last' )
210
211 plot_file='aroc/adni-cnn-classifier-3D-%d-%s-%s-aroc.png' #
    ↪ % (model_number, model_version)
212 log_file='log2/adni-cnn-classifier-3D-%d-%s-%s-
    ↪ classification_report.txt' #% (model_number,
    ↪ model_version)
213 save_plot="models_plots/adni-cnn-classifier-3D-%d-%s-%s"
```

```
214 print("
    ↪ ##############################################################
    ↪ ")
215 predict = parallel_model.predict_generator(
    ↪ data_generator_test, verbose=1 )
216 test_Y = np.array(data_generator_test.labels_array)[:,:2]
217 evaluate = parallel_model.evaluate_generator(
    ↪ data_generator_test, verbose=1 )
218 make_charts(log_filename, save_plot % (model_number,
    ↪ model_version,"last"), evaluate[1], evaluate[0])
219
220
221 save_metrics(
222      test_Y, predict,"adni-%d-%s"%(model_number,
        ↪ model_version),
223      log_file %(model_number, model_version,"lost"),
224      plot_file %(model_number, model_version,"lost")
225
226  )
227
228
229 print("
    ↪ ##############################################################
    ↪ ")
230 parallel_model = load_model( model_filename + '-lost' )
231 predict = parallel_model.predict_generator(
    ↪ data_generator_test, verbose=1 )
232 evaluate = parallel_model.evaluate_generator(
    ↪ data_generator_test, verbose=1 )
```

```
233 make_charts(log_filename, save_plot % (model_number,
      ↪ model_version,"lost"), evaluate[1], evaluate[0])
234 save_metrics(
235       test_Y, predict,"adni-%d-%s"%(model_number,
            ↪ model_version),
236       log_file %(model_number, model_version,"lost"),
237       plot_file %(model_number, model_version,"lost")
238
239  )
240
241
242 print("
      ↪ ##############################################################
      ↪ ")
243
244 #data_generator_test = DataGenerator3Dmask_test( df_adni_3d
      ↪ [ data_filter ], target_shape=(z, y, x),
      ↪ target_channels=in_channel, indexes_output=[True,True
      ↪ ,False,False,False], batch_size=batch_size, rotation=
      ↪ False, shuffle=True, do_balance = False, with_age =
      ↪ False, autoencoder = False )
245 parallel_model = load_model( model_filename + '-acc' )
246 predict = parallel_model.predict_generator(
      ↪ data_generator_test, verbose=1 )
247 evaluate = parallel_model.evaluate_generator(
      ↪ data_generator_test, verbose=1 )
248 make_charts(log_filename, save_plot % (model_number,
      ↪ model_version,"acc"), evaluate[1], evaluate[0])
249 save_metrics(
```

```
250     test_Y, predict,"adni-%d-%s"%(model_number,
            ↪ model_version),
251     log_file %(model_number, model_version,"lost"),
252     plot_file %(model_number, model_version,"lost")
253
254 )
```

Código 7.5: Cofigo de entrenamiento para las imagenes de *FreeSurfer*