ESCUELA POLITÉCNICA SUPERIOR DE CÓRDOBA
Universidad de Córdoba

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Clasificación ordinal mediante redes neuronales convolucionales para el diagnóstico de la enfermedad de Alzheimer

Autor
Miguel Ángel Contreras Córdoba

Directores
Pedro Antonio Gutiérrez Peña
Víctor Manuel Vargas Yun

Córdoba, septiembre de 2024

UNIVERSIDAD DE CÓRDOBA

Índice general

Índice general I

Índice de figuras VII

Índice de tablas IX

I Introducción 1

1 Introducción 3

2 Definición del problema 7

2.1 Definición del problema real 7

2.2 Problema técnico 8

2.2.1 Funcionamiento 8

2.2.2 Entorno 9

2.2.3 Vida Esperada 9

2.2.4 Ciclo de mantenimiento 10

2.2.5 Competencia 10

2.2.6 Aspecto externo 10

2.2.7 Estandarización 11

2.2.8 Calidad y fiabilidad 11

2.2.9 Programa de tareas 12

2.2.10 Pruebas 12

2.2.11 Seguridad 13

3 Objetivos 15

3.1 Objetivo principal 15

3.2 Objetivos específicos 15
3.3 Objetivos de formación 16

4 Antecedentes **19**
4.1 Detección temprana de la enfermedad de Alzheimer 19
4.2 Uso de redes neuronales convolucionales para el análisis de imágenes de resonancia magnética 21

5 Fases de desarrollo del proyecto **25**
5.1 Investigación 25
5.2 Análisis y especificación de requisitos 26
5.3 Diseño 26
5.4 Codificación 26
5.5 Pruebas 27
5.6 Obtención y preparación de los datos 27
5.7 Experimentación 28
5.8 Documentación 28

6 Recursos **29**
6.1 Recursos Humanos 29
6.2 Recursos Hardware 29
6.3 Recursos Software 30

7 Restricciones **33**
7.1 Factores Dato 33
7.2 Factores Estratégicos 34

II Gestión y Preprocesamiento de Datos **37**

8 Conjuntos de datos **39**
8.1 ADNI 41

9 Preparación de los datos **45**
9.1 Selección de capas útiles 45
9.2 Recorte de fondo de capas 50

9.3 Redimensionado de las imágenes 52
9.4 Data augmentation 53

III Diseño del sistema 55

10 Modelos de aprendizaje 57
10.1 Modelo CNN 57
10.1.1 Modelo de El-Assy 58
10.1.2 Modelo GoogleNet 59
10.2 Loss Function 60
10.3 Hiperparámetros 62
10.3.1 Hiperparámetros del modelo 62
10.3.2 Hiperparámetros del algoritmo 64
10.4 Data Agumentation 66

11 Métricas de evaluación 69
11.1 Accuracy 70
11.2 Precision y Recall 71
11.3 F1 Score 73

12 Definición de los experimentos 75

13 Organización del software 79

IV Pruebas y Resultados 83

14 Pruebas software 85

15 Resultados experimentales 87
15.1 Resultados de los modelos 87
15.2 Experimentos extra 90
15.3 Tiempo de ejecución de los modelos 91

V Conclusiones **95**

16 Conclusiones del autor **97**
16.1 Objetivos específicos alcanzados 98
16.2 Objetivos didácticos alcanzados 98

17 Futuras mejoras **101**

VI Apéndices **109**

A Manual de usuario **111**
A.1 Descripción del producto 111
A.2 Requisitos del sistema 112
A.2.1 Requisitos mínimos 112
A.2.2 Requisitos recomendados 112
A.2.3 Requisitos software 113
A.3 Instalación del software 113
A.3.1 Instalación de Python 113
A.3.2 Instalación de librerías 114
A.4 Desinstalación del Software 116
A.4.1 Desinstalación de Python 116
A.4.2 Desinstalación de Librerías 116
A.5 Ejecución del software 116
A.5.1 Parámetros de ejecución 117
A.6 Formato de las salidas 118
A.6.1 Salidas del entrenamiento 118
A.6.2 Salidas del generalización 121
A.6.3 Salidas del predicción 125

B Manual de código **127**
B.1 Características de la implementación 128
B.1.1 Notación de la implementación 128
B.1.2 Estructura de la documentación 128
B.2 Documentación del código 128
B.2.1 Fichero cnn.py 129

B.2.2 Fichero cnn3d/loading.py 136
B.2.3 Fichero cnn3d/predicting.py 138
B.2.4 Fichero cnn3d/testing.py 141
B.2.5 Fichero cnn3d/training.py 148
B.2.6 Fichero cnn3d/models/cnn3d_elassy.py 161
B.2.7 Fichero cnn3d/models/cnn3d_googlenet.py 164
B.2.8 Fichero cnn3d/models/cnn3d_own.py 169
B.2.9 Fichero tools/graph_results.py 172
B.2.10 Fichero tools/metrics.py 194
B.2.11 Fichero tools/preprocess_image.py 200
B.2.12 Ejecución de los experimentos 209

Índice de figuras

1.1 Detección de Aβ en MRI [6] 5

4.1 Detección proteínas beta amiloide en MRI [24] 20
4.2 Estructura básica de una CNN [25] 21
4.3 Estructura básica de una CNN tridimensional [28] 22

8.1 Ejemplo imagen I38726 del conjunto ADNI 42
8.2 Ejemplo imagen I119138 del conjunto brain2020 42
8.3 Gráfico de distribución de etiquetas antes del balance. 44
8.4 Gráfico de distribución de etiquetas después del balance. . . 44

9.1 Ejemplo capas no útiles en imagen I3344 46
9.2 Análisis de la Suma de los valores de los píxeles 47
9.3 Análisis de la Media de los valores de los píxeles 47
9.4 Análisis de la Mediana de los valores de los píxeles 48
9.5 Análisis del Umbral de Otsu 49
9.6 Análisis de la Entropía de la imagen 50
9.7 Análisis de la Varianza de los valores de los píxeles 50
9.8 Análisis de la Suma de los bordes de Sobel 51
9.9 Análisis de la oblicuidad del histograma de los valores de los píxeles 51
9.10 Resultado Imagen I63008 con Umbral: 55 % 52
9.11 Gráficas para el análisis del corte, Imagen I33444, Threshold Row: 15 %, Threshold Column: 17 % 53
9.12 Imagen I33444 cortada 53

10.1 Ejemplo del valor de la *loss function* durante el entrenamiento [52] 60

10.2 Ejemplo de *dropout* [59] 64

11.1 Ejemplo de la curva *Precision-Recall* [65] 73

13.1 Esquema de la organización del proyecto 80

15.1 Comparación de la evolución de la *accuracy* de los distintos modelos durante el entrenamiento para el mejor *fold* de la semilla con mejores resultados en generalización para cada modelo. 88

15.2 Comparación de la evolución de la puntuación F1 de los distintos modelos durante el entrenamiento para el mejor *fold* de la semilla con mejores resultados en generalización. 89

15.3 Matriz de confusión para el conjunto de generalización del modelo GoogleNet 10 de paciencia y 10^{-4} de coeficiente de entrenamiento. 89

15.4 Comparación de la evolución de la *accuracy* durante el entrenamiento de los experimentos extra realizados. 91

15.5 Comparación de la evolución de la puntuación F1 durante el entrenamiento de los experimentos extra realizados. 92

15.6 Matriz de confusión para el conjunto de generalización del modelo GoogleNet para 10 de paciencia y 10^{-5} de coeficiente de entrenamiento. 92

15.7 Matriz de confusión para el conjunto de generalización del modelo GoogleNet para 30 de paciencia y 10^{-4} de coeficiente de entrenamiento. 93

Índice de tablas

6.1 Recursos Hardware del alumno 30

15.1 Tabla comparativa de la media y desviación típica de las métricas *Accuracy* y F1 en *test* para 5 semillas. 90
15.2 Tabla comparativa de los tiempos de entrenamiento para 5 semillas de los distintos modelos. 93

B.1 Especificación de los parámetros de la función split_labels . . 129
B.2 Especificación de los parámetros de la función main 131
B.3 Especificación de los parámetros de la función load_model . . 137
B.4 Especificación de los parámetros y salida de la función predict_image 138
B.5 Especificación de los parámetros y salida de la función predict 139
B.6 Especificación de los datos y métodos de la clase TestDatasetFromNii 141
B.7 Especificación de los parámetros de la función generalization_test 142
B.8 Especificación de los datos y métodos de la clase DatasetFromNii 149
B.9 Especificación de los parámetros de la función train_network 150
B.10 Especificación de los métodos de la clase CNN3D_Model2 . . 162
B.11 Especificación de los métodos de la clase Inception3D 165
B.12 Especificación de los métodos de la clase CNN3D_Model3 . . 165
B.13 Especificación de los métodos de la clase CNN3D_Model1 . . 170
B.14 Especificación de los parámetros y salidas de la función get_subtitle_metrics_from_file 173

B.15 Especificación de los parámetros y salidas de la función parse_cm_file 173
B.16 Especificación de los parámetros de la función create_cm_heatmat 174
B.17 Especificación de los parámetros de la función create_general_cm175
B.18 Especificación de los parámetros de la función generate_graphs_folds175
B.19 Especificación de los parámetros de la función generate_graphs_averages176
B.20 Especificación de los parámetros de la función generate_graphs_by_class177
B.21 Especificación de los parámetros de la función main 177
B.22 Especificación de los parámetros y salidas de la función confusion_matrix 195
B.23 Especificación de los parámetros y salidas de la función calculate_accuracy 195
B.24 Especificación de los parámetros y salidas de la función calculate_precision_recall 196
B.25 Especificación de los parámetros y salidas de la función calculate_f1 196
B.26 Especificación de los parámetros y salidas de la función get_crop_bundaries 200
B.27 Especificación de los parámetros y salidas de la función get_front_back_limits 201
B.28 Especificación de los parámetros y salidas de la función sobel_edges_front_back 202
B.29 Especificación de los parámetros de la función preprocess_image203
B.30 Especificación de los parámetros de la función preprocess_folder203

Índice de Código

B.1 Archivo cnn.py 131
B.2 Archivo cnn3d/loading.py 137
B.3 Archivo cnn3d/predicting.py 139
B.4 Archivo cnn3d/testing.py 142
B.5 Archivo cnn3d/training.py 150
B.6 Archivo cnn3d/models/cnn3d_elassy.py 162
B.7 Archivo cnn3d/models/cnn3d_googlenet.py 165
B.8 Archivo cnn3d/models/cnn3d_own.py 170
B.9 Archivo tools/graph_results.py 177
B.10 Archivo tools/metrics.py 196
B.11 Archivo tools/preprocess_image.py 203
B.12 Archivo submit.sh 209

Parte I

Introducción

Capítulo 1

Introducción

La demencia hace referencia a una alteración de la capacidad para recordar, pensar o tomar decisiones, que interfiere en la realización de las actividades de la vida diaria [1]. La enfermedad de Alzheimer (EA) es el tipo más común de demencia. Es una enfermedad progresiva que empieza con una leve perdida de memoria y puede evolucionar a la perdida de la habilidad para tener conversaciones o reaccionar al entorno. La EA envuelve partes del cerebro que controlan los pensamientos, la memoria y el lenguaje, afectando seriamente en la habilidad de la persona para realizar tareas de la vida diaria [2].

Es difícil hacer una estimación precisa de la prevalencia de la EA, ya que a menudo es complicado identificar el subtipo exacto de demencia. Sin embargo, en general se acepta que la EA es responsable de alrededor de dos tercios de todos los diagnósticos de demencia. Algunos estudios que han usado modelos matemáticos para estimar la prevalencia futura de la demencia, teniendo en cuenta factores como el aumento de la esperanza de vida, los cambios en la moralidad y la incidencia de las enfermedades cardiovasculares, sugieren que una proporción cada vez mayor de la sociedad se verá afectada [3].

LA EA preclínica, que es la fase más temprana de la EA, comprende una larga fase asintomática en la que las personas presentan indicios de la patología, pero no evidencias de deterioro cognitivo o funcional, y su vida cotidiana no se ve afectada. La duración de la EA preclínica puede variar de

un individuo a otro, pero suele durar entre 6 y 10 años en función de la edad de inicio. Un diagnóstico precoz de la EA puede brindar a los pacientes la oportunidad de colaborar en el desarrollo de planes de cuidados avanzados con su familia, cuidadores, médicos y otros miembros del equipo de apoyo más amplio. Es importante destacar que también permite a los pacientes buscar una intervención temprana con tratamiento sintomático, cambios en el estilo de vida para mantener la calidad de vida y estrategias de reducción de riesgos que pueden proporcionar reducciones clínicamente significativas en el deterioro cognitivo, funcional y de conducta [4].

La agregación anormal de especies tóxicas de beta-amiloide (Aβ) en el cerebro se cree que causa patologías posteriores que llevan a la muerte neuronal en la EA; por lo tanto, Aβ ha sido un objetivo principal en el desarrollo de terapias para la EA. Se han hallado algunas anomalías de imagen relacionadas con el amiloide (ARIA) que pueden observarse mediante imagen por resonancia magnética [5] (MRI por sus siglas en inglés) (Figura 1.1) [6].

En la actualidad, los sistemas asistidos por ordenador son usados para la detección precisa y temprana de la EA. En la detección temprana, las técnicas tradicionales de aprendizaje automático dependen de la extracción manual de características, lo que se sirve en gran medida de la experiencia técnica y la repetición [7]. Las redes neuronales convolucionales, o CNNs por sus siglas en inglés, [8] son herramientas que están diseñadas específicamente para extraer características relevantes de las imágenes y aprender patrones complejos que pueden ser utilizados para tareas de clasificación, entre otras alternativas. La capacidad de las CNNs para reconocer y extraer características significativas de imágenes, tales como MRI, ha sido definitiva para su aplicación en el diagnóstico del Alzheimer.

La clasificación ordinal [9] es un enfoque prometedor en el campo de la inteligencia artificial, siendo un caso concreto de los problemas de clasificación en el que las categorías siguen un orden natural, permitiendo así evitar cometer errores de clasificación en clases lejanas. Además, a diferencia de la clasificación binaria, que divide a los pacientes en dos categorías (enfermos y no enfermos), la clasificación ordinal permite una mayor granularidad al distinguir entre diferentes etapas o grados de la enfermedad. En

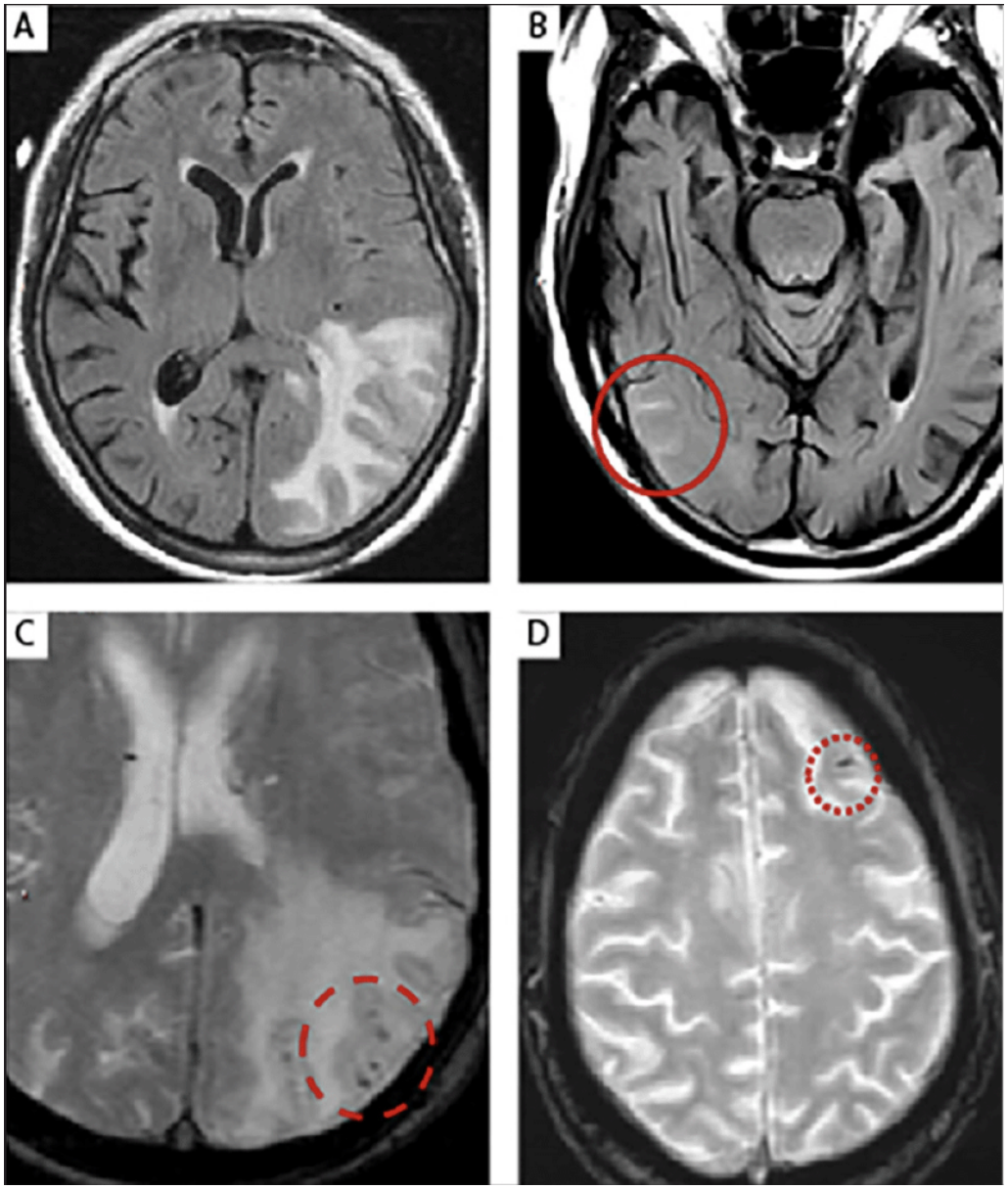

Figura 1.1: Detección de Aβ en MRI [6]

el caso del Alzheimer, esto implica clasificar a los pacientes en categorías como "Alzheimer's disease" (AD), "Mild cognitive impairment" (MCI) y "Cognitively normal" (CN).

El objetivo principal de este proyecto es desarrollar y evaluar un modelo de CNN para la clasificación ordinal de la EA utilizando MRI. Para lograr

esto, se procederá a la obtención y preprocesamiento de datos, la implementación de un modelo CNN adaptado a la clasificación ordinal y la evaluación de su rendimiento comparándolo con métodos existentes. Los datos que se van a utilizar para el desarrollo de este proyecto han sido obtenidos del proyecto *The Alzheimer's Disease Neuroimaging Initiative* (ADNI) [10], que recopila y proporciona a investigadores acceso a una amplia base de datos de pacientes de Alzheimer para sus estudios.

Este documento se estructura de la siguiente manera: en esta primera parte se hace una introducción al proyecto; en la segunda parte se explicará cómo se ha realizado la gestión y el procesamiento de los datos usados; en la tercera parte se detallará el diseño del sistema; en la cuarta parte se mostrarán y analizarán las pruebas y los resultados obtenidos; por último en la quinta parte se expondrán las conclusiones finales obtenidas durante la realización de este proyecto.

Capítulo 2

Definición del problema

El propósito de este capítulo es ofrecer una definición clara y precisa del problema que este proyecto busca resolver. Para lograrlo, se identificarán las necesidades y se establecerán los objetivos correspondientes. Con el fin de simplificar el análisis, se distinguirán dos enfoques del problema: el problema real, que refleja la perspectiva del usuario final, y el problema técnico, que representa la visión desde el punto de vista de la ingeniería.

2.1. Definición del problema real

El principal problema que se tendrá que afrontar en el desarrollo de este proyecto es la detección temprana de la enfermedad de Alzheimer. La intención es poder detectar si un paciente tiene o no Alzheimer y, en especial, poder detectar cuándo se está empezando a desarrollar. Esto le permitiría a los especialistas poder tratar al paciente para poder ralentizar el avance de la enfermedad. Para ello se pretende implementar un modelo de red neuronal convolucional que sea capaz de predecir si un paciente tiene o no Alzheimer, o está en un estado intermedio, a partir del análisis de MRI.

Para resumir, en este proyecto se tratará de hacer una detección temprana de la enfermedad de Alzheimer usando un modelo de red neuronal convolucional.

2.2. Problema técnico

Una vez ya hemos definido el problema real, vamos a pasar a una definición más técnica del problema que se pretende resolver.

2.2.1. Funcionamiento

La herramienta de software que se desarrollará contará con dos partes. La primera nos permitirá hacer un preprocesamiento de las imágenes MRI, preparándolas para que puedan ser usadas por el modelo. La segunda parte nos permitirá usar la CNN para realizar tres funciones principales. Primero se podrá usar para entrenar el modelo con unos datos de entrada proporcionados por el usuario. También se permitirá usar un conjunto de datos de generalización, diferente al utilizado durante el proceso de entrenamiento y validación, para confirmar el correcto entrenamiento del modelo. Por último, se permitirá que el usuario introduzca una imagen como entrada del modelo para obtener un diagnóstico usando el modelo previamente entrenado.

Todas estas funciones podrán ser accedidas mediante la interfaz de línea de comandos (CLI por sus siglas en inglés) permitiendo la especificación de diversos parámetros para configurar la ejecución de las distintas partes.

Resumiendo, las funciones más importantes que tendrá este software serán:

- **Preprocesamiento de los datos:** prepara las imágenes para que puedan ser usadas por el modelo.
- **Entrenamiento de la CNN:** utiliza las imágenes preprocesadas para el entrenamiento del modelo.
- **Evaluación aislada:** se utiliza un conjunto de datos completamente diferente al empleado durante el proceso de entrenamiento para evaluar el rendimiento del modelo.
- **Predicción de imagen:** se utiliza el modelo entrenado para realizar una predicción de una MRI concreta proporcionada por el usuario.

2.2.2. Entorno

El entorno de programación utilizado en este proyecto es el habilitado por el grupo de investigación AYRNA[11], grupo al que pertenecen los directores de este proyecto. Este es uno de los disponibles en el clúster del grupo, en concreto, el que estamos usando para este proyecto usa Debian 11.3 (bullseye).

La implementación del modelo CNN se realizará mediante una de las librerías de Python [12] destinada a la implementación de modelos de *deep learning*, PyTorch [13]. También se usará Matplotlib [14] para la generación de gráficas.

Por último, cabe destacar que la aplicación desarrollada no está destinada a un público general sino a la comunidad científica. Para poder ser usada, es necesario tener unos conocimientos básicos sobre las partes que conforman la aplicación, así como conocimientos sobre ejecución de programas mediante CLI.

2.2.3. Vida Esperada

La vida esperada de la solución propuesta es difícil de estimar ya que el uso de CNN es relativamente nuevo en comparación con otros modelos y está en constante evolución. La utilidad futura de este trabajo dependerá de varios factores, incluyendo los avances en la tecnología de redes neuronales y las técnicas de imagen médica. Se espera que la solución tenga una relevancia significativa durante los próximos 5 a 10 años. Esta suposición se debe al hecho de que estamos trabajando sobre un campo de la investigación que está en continuo desarrollo y, durante este periodo, se espera una continua evolución en el campo de la detección temprana de la enfermedad de Alzheimer.

Todos los algoritmos y procesos usados en el desarrollo de este proyecto, así como los conceptos en los que se basan, podrán ser modificados, actualizados o incluso desechados a medida que la investigación en este campo avance y se desarrollen nuevas teorías o se perfeccionen las actuales.

En conclusión, este trabajo no es una solución definitiva, sino una base que se podrá ir perfeccionando a lo largo de los años para adaptase a las nuevas tendencias e investigaciones en la detección temprana de la enfermedad de Alzheimer y el uso de CNN para el análisis de imágenes médicas.

2.2.4. Ciclo de mantenimiento

Como se ha comentado, la vida esperada de esta aplicación no es fácilmente determinable y, debido al constante avance de la tecnología y la ciencia, es probable que haga falta realizar mejoras y cambios a no muy largo plazo. Es por eso que no se ha realizado una planificación del ciclo de mantenimiento para esta aplicación.

2.2.5. Competencia

Los temas tratados en este Trabajo están en continua evolución y cada vez se realizan más investigaciones en relación a este tema. Durante el desarrollo de este documento, se comentan diferentes artículos que investigan el uso de redes neuronales convolucionales y su aplicación en imágenes médicas.

Esto nos lleva a determinar que existe una competencia cada vez mayor en este ámbito ya que es un área nueva y en crecimiento y trata de solucionar problemas que pueden dar resultar beneficiosos para todo el mundo.

2.2.6. Aspecto externo

La aplicación se pretende presentar al usuario como una aplicación de escritorio sin necesidad de instalaciones, lo que facilitaría su portabilidad.

Junto a ella se entregará el manual de usuario redactado de forma clara, que permitiría al usuario entender y hacer uso de la aplicación de manera sencilla.

Además, se incluirá el manual de código que permitirá al usuario enten-

der el código desarrollado y realizar modificaciones o mejoras en caso de ser necesarias. Todos estos documentos se presentarán en formato PDF junto al código.

2.2.7. Estandarización

En cuanto a la estandarización de la programación, se procurarán seguir distintos estándares que se han estudiado durante la carrera para definir las clases, funciones y variables. Así pues, con el objetivo de realizar un código limpio y legible se ha tratado de seguir las siguientes pautas durante el desarrollo:

- Los nombres de las variables y funciones deben tratar de ser explicativos de su contenido.
- Los nombres clases deberán seguir la convención *Pascal Case*. Esto es que todas las palabras del nombre comiencen con mayúscula y vayan todo junto.
- Los nombres de las variables y funciones deberán seguir la convención *Snake Case*. Esto es que todas las palabras vayan en minúscula y separadas con guión bajo (_).

2.2.8. Calidad y fiabilidad

La calidad del Trabajo será evaluada por el alumno, los directores del Trabajo y, en última instancia, por el Tribunal que lo evalúe. Aun así, en la medida de lo posible, se intentará seguir los estándares de Ingeniería del Software.

En cuanto a la fiabilidad, el software está desarrollado para, en caso de no introducir los parámetros mínimos necesarios, lanzar un mensaje de error avisando al usuario, aún así, hay parámetros que admiten parámetros por defecto en caso de no ser especificados. Asimismo, se ha tratado de controlar todos los errores que puedan surgir durante la ejecución para informar al usuario.

2.2.9. Programa de tareas

A grandes rasgos, el desarrollo de este Trabajo se compone de las siguientes etapas:

1. **Fase de preparación:** la fase de preparación consisten en realizar una investigación sobre el problema propuesto. Tras esto, se realizará una especificación de los requisitos del software que se va a desarrollar, así como un análisis de las distintas partes del proceso de desarrollo para detectar posibles problemas.

2. **Fase de diseño:** en esta fase se realizará el diseño de la estructura principal del código (funciones, clases, organización, etc.).

3. **Fase de codificación:** en esta fase se realizará la implementación del software diseñado.

4. **Fase de pruebas:** para esta fase se usará el software desarrollado para probar la correcta ejecución de las funciones, clases y opciones de ejecución.

5. **Fase de recolección de datos:** en esta fase se obtendrán las imágenes de la base de datos de ADNI [10], y se realizará cualquier preprocesamiento necesario sobre ellas.

6. **Fase de experimentación:** una vez tenemos todo el código y las imágenes, se procederá a realizar los experimentos oportunos para probar la eficacia de nuestro modelo.

7. **Fase de documentación:** por último se realizará la documentación sobre todo el proceso de desarrollo que se ha llevado a cabo.

2.2.10. Pruebas

Durante la implementación de nuestro modelo, y al final del desarrollo del mismo, se realizarán las pruebas necesarias para asegurar que nuestro modelo satisface los requisitos establecidos en este documento. Además, nos

permitiría detectar cualquier error en el desarrollo para poder solventarlo. Para asegurar que se realizan todas las pruebas necesarias, se han establecido las siguientes:

- Pruebas de las funciones y clases creadas de manera independiente.
- Pruebas de integración de los distintos módulos.
- Pruebas para asegurar el funcionamiento del software como conjunto.
- Pruebas para comprobar la validez de los algoritmos con las imágenes de la base de datos de ADNI.

2.2.11. Seguridad

La temática del software desarrollado no permite la realización de acciones que salgan de los objetivos determinados y ponga en riesgo la seguridad de la máquina.

En cuanto a la privacidad, debido a que se están usando datos sensibles cuyo acceso solo es posible si la entidad así lo permite, solo se ha limitado a compartir el número de identificación de las imágenes para que otros usuarios con acceso a la base de datos puedan obtener las mismas imágenes. Una vez el usuario tiene las imágenes, la privacidad de las mismas depende enteramente del usuario, ya que la aplicación no dispone de ninguna función que pueda llegar a hacer públicas la imágenes.

Capítulo 3

Objetivos

3.1. Objetivo principal

El objetivo principal de este Trabajo de Fin de Grado es la aplicación de CNNs para el análisis de MRI con la intención de detectar la presencia de la enfermedad de Alzheimer en pacientes, incluso antes de que muestren síntomas notables.

La idea es que estas redes neuronales sean capaces de extraer la características y diferencias que tienen las MRI de distintos pacientes en fases diferentes de la enfermedad para que puedan predecir en qué estado de la enfermedad está un paciente usando solo su MRI. Para ello vamos a usar clasificación ordinal, esto permitirá a nuestro modelo clasificar las imágenes de entrada en 3 categorías diferentes: AD (tiene Alzheimer), MCI (tiene un leve impedimento cognitivo) o CN (no tiene Alzheimer).

Para cumplir este objetivo principal es necesario que se cumplan una serie de objetivos específicos. Estos objetivos son los que se exponen en el siguiente apartado.

3.2. Objetivos específicos

Para alcanzar el objetivo principal del Trabajo de Fin de Grado, es necesario cumplir los siguientes objetivos específicos:

1. Estudio de imágenes MRI y su procesamiento usando Python y la librería NiBabel [15].
2. Estudio y análisis de la base de datos ADNI.
3. Estudio y compresión de la librería de Python PyTorch.
4. Estudio y comprensión de la redes neuronales convolucionales, específicamente las centradas en imágenes 3D [16].
5. Obtención y preprocesamiento de los datos para el entrenamiento del modelo.
6. Implementación del modelo de CNN 3D para la clasificación de las imágenes preprocesadas.
7. Análisis de los resultados obtenidos para la calibración de los hiperparámetros.
8. Experimentación con el conjunto de generalización para confirmar los resultados obtenidos con el modelo entrenado.
9. Interpretación de los análisis obtenidos.

3.3. Objetivos de formación

Durante la realización de este Trabajo de Fin de Grado se pretende adquirir, a la vez que ampliar, conocimiento sobre:

1. Conceptos y tipos de CNNs
2. MRI y cómo trabajar con ellas.
3. Comprensión e interpretación de artículos científicos.
4. Redacción de proyectos de Ingeniería Informática.
5. Uso del sistema de composición de textos LaTeX[17] y el editor en la nube Overleaf [18].

6. Lenguaje de programación Python y las distintas librerías usadas en el desarrollo de este proyecto (NiBabel, Pandas [19], PyTorch, ...).

7. Clúster del grupo de investigación AYRNA y la suite de HTCondor [20] para la planificación de los experimentos.

8. Mejora del modelo CNN para perfeccionar los resultados finales.

9. Métricas para el análisis del rendimiento y la precisión del modelo en problemas de clasificación ordinal.

Capítulo 4

Antecedentes

Este trabajo está basado en la aplicación combinada de los conceptos que se van a desarrollar a continuación:

- Detección temprana de la enfermedad de Alzheimer.
- Uso de redes neuronales convolucionales para el análisis de imágenes de resonancia magnética.

4.1. Detección temprana de la enfermedad de Alzheimer

En 2019 se publicó en la revista *Degenerative Neurological and Neuromuscular Disease* un artículo llamado *Alzheimer's Disease – Why We Need Early Diagnosis* [3]. En este artículo, se habla de la importancia que tiene para el paciente y su entorno, detectar la enfermedad antes de que empiece a manifestarse físicamente. Se afirma que la EA se puede detectar hasta 8 años antes de que el paciente empiece a mostrar síntomas de demencia [21], este estado se conoce como deterioro cognitivo leve. Además, según los autores, se estima que entre los pacientes en este estado, el 30 % acaba desarrollando demencia a causa de la EA, otro 30 % permanece en ese estado y otro 30 % vería una mejora en los síntomas del deterioro cognitivo leve [22].

La pregunta ahora es, ¿cómo se pueden detectar estas fases previas a la enfermedad del Alzheimer si no muestran síntomas claros? En 2022 se publicó *Detection and Management of Amyloid-Related Imaging Abnormalities in Patients with Alzheimer's Disease Treated with Anti-Amyloid Beta Therapy* en *The Journal of Prevention of Alzheimer's Disease* [6]. Los autores de este artículo afirman que en su investigación pudieron comprobar que en la primera fase de la enfermedad, conocida como Alzheimer preclínico, se aprecian agregaciones anormales de amiloides beta tóxicos [23]. Estas agregaciones han sido detectadas mediante el análisis de imágenes de resonancia magnética [24], donde se aprecian manchas inusuales (Figura 4.1) que nos apunta a un posible caso de Alzheimer, lo que permite empezar a tratarlo antes de que cause daños mayores.

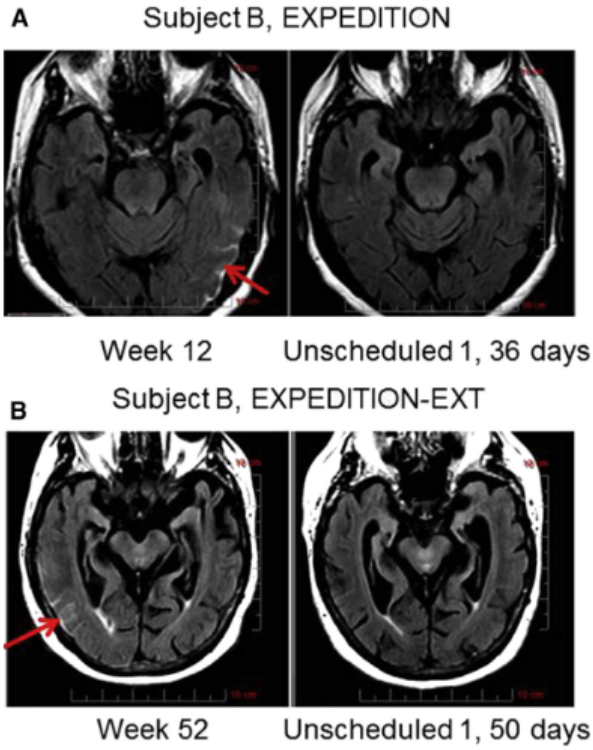

Figura 4.1: Detección proteínas beta amiloide en MRI [24]

Esto lleva a plantear la posibilidad de usar aprendizaje automático para el análisis de estas imágenes, para ser más precisos a la hora de detectar estas anomalías.

4.2. Uso de redes neuronales convolucionales para el análisis de imágenes de resonancia magnética

Las redes neuronales convencionales son buenas para la clasificación y predicción de datos pero cuando estos datos son imágenes, la solución usando estas redes aumenta el coste computacional enormemente. Para resolver este problema existen las redes neuronales convolucionales, que permiten extraer características relevantes a partir de las imágenes y aprender a partir de ellas.

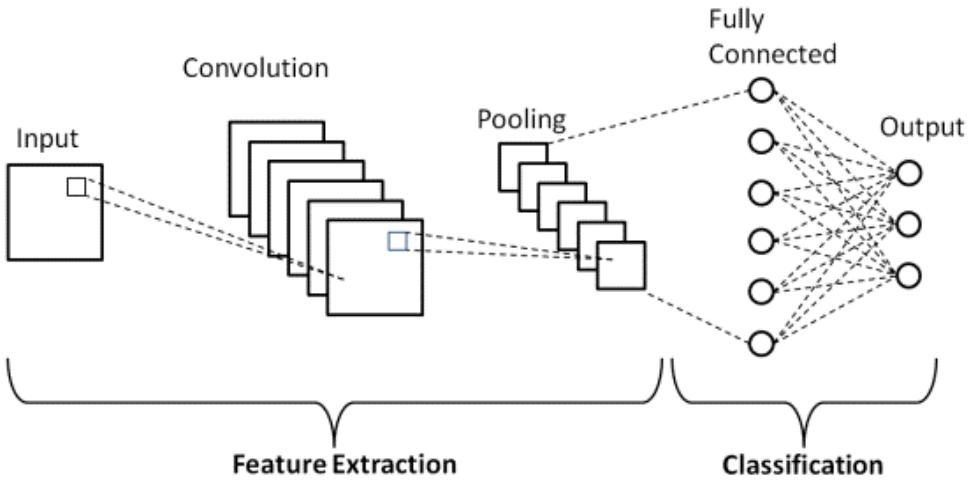

Figura 4.2: Estructura básica de una CNN [25]

En 2015, *Keiron O'Shea* y *Ryan Nash* de *Cornell University* publicaron *An Introduction to Convolutional Neural Networks* [26]. Aquí, los autores definen cómo funcionan las CNN. Explican que estas redes cuentan con tres tipos de capas, además de la capa de entrada (Figura 4.2):

- **Capa de entrada:** contiene los valores de los píxeles de entrada.
- **Capa convolucional:** determina la salida de las neuronas conectadas

a secciones determinadas de la imagen de entrada. Esto lo consigue realizando el cálculo del producto escalar entre sus pesos y la región conectada a la entrada. Esto lo hace mediante la operación de convolución [27] la cuál permite convertir los datos de entrada, que puede ser la imagen o la salida de la capa anterior, en características de más alto nivel. Es común que, además, cuenten con una función de activación, siendo la ReLU (rectified linear unit) la más usada.

- **Capa agrupamiento (pooling):** esta capa se encarga de hacer un muestreo descendente de la entrada, reduciendo el número de parámetros en la activación, y, por lo tanto, reduciendo el coste computacional.

- **Capa completamente conectada:** esta capa hace las mismas operaciones que se hacen en las redes neuronales convencionales, con la intención de obtener una salida para clasificación. Aquí también es común usar funciones de activación para mejorar la eficiencia.

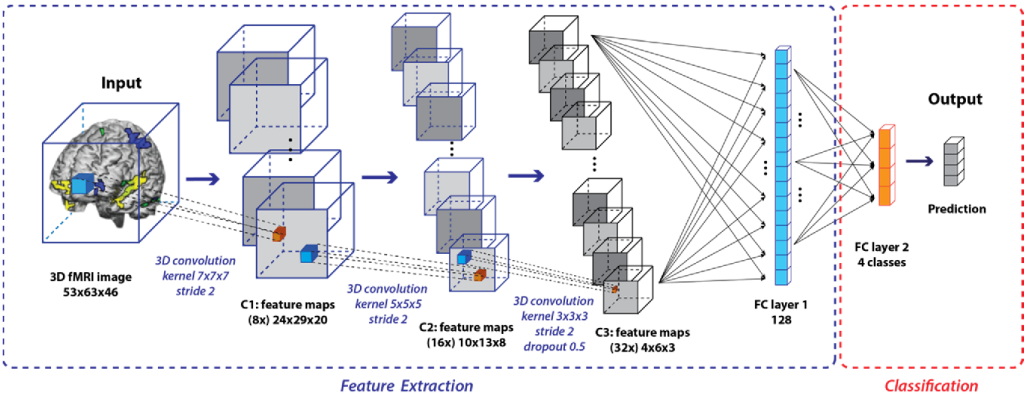

Figura 4.3: Estructura básica de una CNN tridimensional [28]

Además, en el artículo de *Journal of Big Data* llamado *Deep convolutional neural network based medical image classification for disease diagnosis* [29], los autores muestran los resultados que se obtienen al aplicar este tipo de redes con imágenes de rayos X.

4.2. USO DE REDES NEURONALES CONVOLUCIONALES PARA EL ANÁLISIS DE IMÁGENES DE RESONANCIA MAGNÉTICA

En el problema que se plantea para este proyecto, se trabaja con imágenes de resonancia magnética. Estas, a diferencia de las rayos X, son imágenes en 3 dimensiones, es decir, si la imagen tiene unas dimensiones (X, Y, Z), se interpreta como Z imágenes de dimensión (X, Y) apiladas.

Para las imágenes 3D se usan filtros convolucionales de 3 dimensiones que sean capaces de interpretar esta dimensión extra. En el artículo Detección de acciones a partir de información de profundidad mediante redes neuronales convolucionales [30], muestran cómo funcionan esta tipo de redes. En su caso las aplican para el análisis de vídeos pero, aunque parezcan diferentes, los vídeos se interpretan como bloques donde cada frame es una capa de la entrada, por lo que en la práctica ambas entradas son imágenes apiladas formando bloques tridimensionales. En la Figura 4.3 se puede encontrar un ejemplo visual de este concepto. Por tanto, la principal diferencia entre imágenes volumétricas y vídeo es que en el vídeo existe una temporalidad. Es decir, el vídeo es como una serie temporal de imágenes.

Capítulo 5

Fases de desarrollo del proyecto

En este capítulo vamos a exponer las distintas partes de las que consta el proceso de desarrollo de este Trabajo de Fin de Grado.

5.1. Investigación

Para empezar, se estudiarán distintos artículos sobre la detección del Alzheimer, especialmente aquellos que usen imágenes de resonancia magnética. También se investigará tanto la importancia como las formas de realizar una detección temprana de la enfermedad. Esto ayudará a enfocar de manera correcta la aplicación del modelo de red convolucional propuesto.

Una vez realizado este estudio, se pasará al análisis de la imágenes de resonancia magnética. Se verá cómo están formadas, cómo se pueden representar y, también, se experimentará con las distintas formas de preprocesar estas imágenes para reducir el coste computacional del entrenamiento del modelo sin afectar a la calidad de las imágenes y, por tanto, a la efectividad del entrenamiento.

Por último, se pretende ampliar el conocimiento sobre las redes neuronales convolucionales, en especial en la aplicación de estas redes con entradas tridimensionales. Se investigará cómo crear este tipo de redes, cómo leen y

procesan los datos y de qué forma son capaces de obtener resultados. En esta fase también se ampliará conocimiento sobre las librerías de Python que puedan ser útiles para el desarrollo de este proyecto como NiBabel [15], para trabajar con las imágenes de resonancia magnética, o PyTorch [13], para la implementación de la red neuronal convolucional tridimensional.

5.2. Análisis y especificación de requisitos

En esta fase, una vez terminada la investigación, se pasará a hacer una especificación de requisitos del software que se quiere desarrollar. Se determinará cual es resultado deseado y esperado de este software, asegurando que este cumple con el objetivo establecido para este Trabajo de Fin de Grado.

Además, se deberán analizar las distintas partes del proceso de desarrollo para detectar distintos problemas que puedan surgir durante el mismo. Esto nos permitirá examinar estos problemas para poder evitarlos o, en caso de que no sea posible, poder establecer medidas para solucionarlos en el momento en el que aparezcan.

5.3. Diseño

En esta fase se procederá a diseñar la estructura del código, funciones, clases, estructura, etc. Para hacer esto, es necesario usar todos los conceptos aprendidos durante las dos fases anteriores para asegurar que el diseño cumple todos los requisitos necesarios y evitar posibles problemas de organización y conflictos entre distintas partes del código durante la siguiente fase.

5.4. Codificación

A continuación se procederá a la implementación. En esta fase se usarán todos los conceptos aprendidos durante la fase de investigación con el ob-

jetivo de obtener un software que cumpla con lo propuesto. También es necesario asegurar que el software cumple con todos los requisitos establecidos durante la segunda fase, siendo aquí donde se pondrán en práctica esas soluciones a los problemas previstos que no se podían evitar. Por último, es importante que el código siga el diseño propuesto en la tercera fase.

5.5. Pruebas

Tras realizar la implementación del código es importante realizar una fase de pruebas. En esta fase se probarán las funciones codificadas y las opciones de ejecución para asegurar que la implementación se ha realizado correctamente.

Durante esta fase se pretenden detectar errores en la codificación del modelo que puedan ser perjudiciales para el desarrollo del proyecto y puedan causar graves problemas en su ejecución.

5.6. Obtención y preparación de los datos

En esta fase se pretende obtener los datos (imágenes MRI [5]) de la base de datos de ADNI [10]. Además se debe realizar una separación entre imágenes que serán usadas durante el entrenamiento (conjunto de entrenamiento) e imágenes que serán usadas durante el proceso de generalización (conjunto de generalización), práctica que es común cuando se entrenan redes neuronales.

También se realizará un preprocesamiento de estas imágenes que consistirá en la eliminación de partes sin información correspondientes al fondo y el redimensionado de las imágenes, usando interpolación tricúbica [31] para conseguir que todas las imágenes tengan el mismo tamaño y este coincida con el aceptado por la red neuronal, afectando en la menor medida posible a la calidad de los datos.

5.7. Experimentación

Una vez ya se han realizado todas las pruebas al código, funcionan conforme a lo esperado, y se han obtenido y preprocesado los datos, empieza la fase de experimentación. En esta fase se usarán los datos separados en los conjuntos de entrenamiento y generalización para entrenar y evaluar el modelo, modificando los hiperparámetros del mismo para obtener distintos resultados y, mediante el uso de métricas de rendimiento, obtener una conclusiones sobre el desempeño del modelo.

5.8. Documentación

Esta es la fase final donde se documenta todo el proceso de desarrollo.

Consiste en la redacción de un documento donde se muestra todo lo relativo al proyecto, desde el problema que se plantea, el objetivo que se intenta alcanzar, la solución propuesta para ese problema hasta la explicación sobre la codificación y uso del software propuesto para alcanzar esos objetivos planteados.

Capítulo 6

Recursos

En este capítulo se expondrán los recursos humanos, hardware y software que vamos a ocupar para la realización de este Trabajo de Fin de Grado.

6.1. Recursos Humanos

Este trabajo será desarrollado por el alumno del Grado de Ingeniería Informática Miguel Ángel Contreras Córdoba. Además, contará con el apoyo de los directores del proyecto, Dr. Pedro Antonio Gutiérrez Peña y Dr. Víctor Manuel Vargas Yun, siendo ambos profesores del departamento de Informática y Análisis Numérico de la Universidad de Córdoba e integrantes del grupo de investigación AYRNA.

6.2. Recursos Hardware

En un principio los equipos que se iban a utilizar para el desarrollo del proyecto serían los del alumno, esto incluye desde la investigación hasta la experimentación y documentación. Estos equipos son los siguientes:

Durante el proceso de desarrollo se descubrió que los recursos hardware del alumno no contaban con suficiente potencia para la realización de la fase de experimentación. Para solucionar este problema, los directores de este proyecto dieron al alumno la posibilidad de realizar los experimentos

	Sobremesa	Portátil
CPU	Intel Core i5-6600U, Dual-Core a 3.30GHz	Intel Core i3-7020U, Quad-Core a 2.30GHz
Tarjeta Gráfica	Gibabyte GeForce GTX 1060 3 GB	Intel HD Graphics 620 (KBL GT2)
RAM	16 GB	8GB
Almacenamiento	SSD: 240GB, HDD: 1TB, SSD: 1TB	HDD: 512GB

Tabla 6.1: Recursos Hardware del alumno

usando el clúster del grupo AYRNA. En concreto, el alumno está usando una máquina que cuenta con 192 GB de memoria RAM, 10 GPUs y 40 hilos de ejecución, haciendo posible continuar con la experimentación.

6.3. Recursos Software

Para el desarrollo de este proyecto se hizo uso de los siguientes recursos software:

- **Lenguaje de programación:** el código de este software ha sido desarrollado usando Python 3.
- **Sistema Operativo:** para el desarrollo del proyecto se ha usado Ubuntu 24.04 como plataforma para desarrollo de software, Windows 10 para el desarrollo de la documentación y Debian 11.3 (bullseye) como entorno presente en el clúster de AYRNA.
- **Librerías:** durante el desarrollo del código se han usando numerosas librerías de Python, entre ellas se encuentran: NiBabel, para cargar y trabajar con las MRI, PyTorch, para el desarrollo de la CNN, Click [32], para el tratamiento de parámetros, y Pandas, para la lectura y tratamiento de los archivos CSV [33].
- **Entorno de desarrollo:** para el desarrollo del código se ha usando Visual Studio Code como entorno principal y también se ha usado Vi [34] y Nano [35] para pequeños cambios que había que hacer en el clúster a través de SSH [36].

- **Control de versiones:** para el control de versiones se ha decidido usar un backup local ya que los datos proporcionados por ADNI no son enteramente públicos, teniendo ellos que darte acceso. Por ello se ha realizado un script usando Bash [37] que al ser ejecutado crea una copia de aquellos archivos modificados, manteniendo siempre las versiones anteriores y una versión completa, identificadas con la fecha y hora en la que realizó ese backup.

- **Documentación:** la documentación de este proyecto se ha realizado usando el sistema de composición de textos LaTeX dentro del sistema en la nube Overleaf. Además, las referencias de este proyecto han sido obtenidas principalmente usando Google Scholar [38].

- **Gráficas:** para la creación de gráficas se han usando dos métodos distintos. Por un lado se ha usado la librería de Python Matplotlib que permite generar gráficas con los datos obtenidos mediante *scripts*. Por otro lado, se han usado las hojas de cálculo de Google para generar algunas gráficas más simples, como las de distribución de etiquetas que se pueden ver en el capítulo 8.

Capítulo 7

Restricciones

En esta parte se van a exponer todos aquellos factores que limitan o restringen el desarrollo del Trabajo de Fin de Grado y condicionan su resultado. Estos factores son determinados por fuentes externas o por el desarrollador, siendo estos Factores Dato y Factores Estratégicos, respectivamente.

7.1. Factores Dato

Los Factores Dato son aquellos que están determinados por la propia naturaleza del proyecto y que, por tanto, no pueden ser ni modificados ni elegidos. Es este Trabajo de Fin de Grado se encuentran presentes los siguientes factores dato:

- **Horas y volumen de trabajo:** este proyecto tiene unas horas y volumen de trabajo limitado por el valor de los créditos que corresponden a este Trabajo de Fin de Grado. En concreto, para el grado de Ingeniería Informática de la Universidad de Córdoba, el Trabajo de Fin de Grado tiene asignados un total de 12 créditos. Cada crédito corresponde a 25 horas de trabajo por lo que el proyecto no puede ocupar más de 300 horas de trabajo realizadas por el alumno.

- **Hardware de desarrollo y ejecución:** al empezar este proyecto el hardware estaba limitado a los ordenadores de los que disponía el alumno

cuyas características están reflejadas en la Tabla 6.1. Durante el desarrollo del proyecto se amplió esta limitación ya que se concedió acceso al alumno al clúster del grupo de investigación AYRNA, cuyas especificaciones están reflejadas en el capítulo 6, ya que con las limitaciones anteriores el alumno no era capaz de realizar la fase de experimentación.

- **Tiempo empleado por los algoritmos de aprendizaje:** este es un factor muy a tener en cuenta al estar trabajando con CNN tridimensionales. Este tipo de redes son muy complejas y reciben una cantidad de datos enorme. Esto hace que el tiempo empleado en el entrenamiento del modelo, aunque pueda variar algo cambiando algunos hiperparámetros del mismo, sea muy grande.

7.2. Factores Estratégicos

Los Factores Estratégicos son aquellos que pueden ser elegidos. Esta elección determina en gran medida los resultados que obtenemos para el proyecto desarrollado. En este Trabajo de Fin de Grado se han tenido en cuenta los siguientes factores:

- **Sistema Operativo:** a la hora de elegir sistema operativo, se contaba con las opciones de Ubuntu 24.04, o Windows 10. Entre estas opciones se ha elegido Ubuntu para el desarrollo y ejecución del software, ya que cuenta con una alta facilidad a la hora de programar y una amplia selección de herramientas para facilitar el desarrollo. En cambio, para la documentación, se ha elegido usar el sistema Windows ya que el editor usado está en la nube, por lo que no requiere de herramientas locales, y ese el sistema operativo principal y más usado del ordenador personal del alumno, lo que lo hará más fácil de usar al estar habituado a este.

- **Lenguaje de Programación:** existen numerosos lenguajes de programación, unos de más bajo nivel, como C++, y otros de más alto nivel,

como Python. Para el desarrollo de este proyecto se ha determinado que la mejor opción para el desarrollo es Python. Esto se debe a que este lenguaje es altamente usado en la industria para el análisis de datos, esto implica una amplia comunidad y numerosas librería dedicadas a esto. En concreto, para nuestro proyecto, existe la librería PyTorch que nos permite crear redes neuronales convolucionales con una gran facilidad usando funciones predefinidas en la librería.

- **Preprocesamiento de las imágenes:** para este proyecto se ha realizado un proceso de preprocesamiento de las imágenes antes de introducirlas en el modelo, con el objetivo de reducir su tamaño y la complejidad computacional del modelo. En nuestro caso hemos realizado los siguientes procesos:

  - **Reducción fondo:** las imágenes tienen una zona exterior que rodea a toda la imagen y que consta de únicamente píxeles negros. Se ha eliminado este marco, dejando el borde de las imágenes cerca del borde del órgano, eliminando así información innecesaria y reduciendo el coste computacional del modelo.
  - **Redimensionado de las imágenes:** se ha considerado necesario reducir el tamaño de las imágenes por dos motivos. El primero es que las imágenes al ser más pequeñas reducen el coste computacional ya que se reduce el número de parámetros del modelo. El segundo es que hay conjuntos de imágenes que tienen distintos tamaños y es necesario que todas las imágenes cuenten con el mismo tamaño a la hora de entrar al modelo, siendo este tamaño determinado por el modelo. Por ello se reduce el tamaño de las imágenes con el objetivo de alcanzar el tamaño deseado.
  - **Data augmentation:** para el proceso de entrenamiento se ha usado la técnica de *data augmentation* que consiste en generar nuevas imágenes a través de las ya existentes mediante el uso de distintas técnicas. En nuestro caso hemos usado giros, transformaciones afines y ruido. Esto hace que mejore el entrenamiento del modelo al tener una mayor cantidad de datos de entrada.

Parte II

Gestión y Preprocesamiento de Datos

Capítulo 8

Conjuntos de datos

Una de las partes más importantes del desarrollo de modelos de inteligencia artificial es la obtención y uso de un conjunto de datos amplio y de calidad. Esta selección de datos es muy importante ya que de ella depende en gran parte el correcto entrenamiento del modelo y la obtención de predicciones que sean correctas. Además, es importante que sean un conjunto de datos grande ya que para poder hacer un modelo que sea útil hace falta realizar una separación de los datos en subgrupos que reducirían el tamaño útil de los datos.

A la hora de desarrollar modelos de inteligencia artificial es necesario disponer de distintos grupos de datos que sean independientes entre sí, los cuales serán usados en las distintas partes del entrenamiento del modelo. Primero se debe hacer una separación del conjunto de datos completo en dos subgrupos. Estos subgrupos son los llamados conjuntos de entrenamiento (o *train*) y de generalización (o *test*). Lo más común es hacer una separación 80-20 o 70-30, es decir, 80 % del total de los datos pertenecen al conjunto de entrenamiento y el 20 % restante pertenece al de generalización o 70 % a entrenamiento y 30 % a generalización en el otro caso, aunque esto puede variar en cada caso dependiendo del conjunto de datos al que se tenga acceso.

El conjunto de entrenamiento se usa durante la fase de entrenamiento del modelo, como se puede intuir por su nombre. Sin embargo, en la fase de entrenamiento se suele realizar otra separación de los datos, en este caso

de los de entrenamiento. Esta separación, al igual que la anterior, es común hacerla 80-20 o 70-30. Esto genera otro conjunto de entrenamiento más pequeño que el anterior, que se corresponde con el porcentaje más alto, y otro conjunto nuevo que se llama de validación.

Durante el proceso de entrenamiento se calcula el error tras cada iteración para determinar la variación que van a recibir los pesos de la red de cara a la siguiente iteración. Sin embargo, si solo nos guiamos por el error en el conjunto de entrenamiento, se corre el peligro de que el modelo acabe sobreentrenando, ya que imitaría los datos en vez de generalizar sobre ellos. Aquí es donde entra el conjunto de validación. Este conjunto se usa durante este proceso de pruebas, para realizar una validación de los parámetros, que se puede usar para estrategias como el *early stopping* o la reducción del *learning rate* progresiva, eliminando así ese riesgo que tenía el modelo de sobreentrenamiento. Por ello, es muy importante que el conjunto de entrenamiento nuevo y el de validación sean completamente independientes.

Por último, tenemos el conjunto de generalización. Este conjunto debe estar completamente aislado del proceso de entrenamiento. Su función es ser usado en una fase de *test* que evalúa la precisión del modelo, asegurando que el modelo sea capaz de generalizar para conjuntos de datos ajenos al entrenamiento, confirmando así la validez del mismo.

Como se puede ver, es muy importante seleccionar un buen conjunto de datos y que este sea amplio, ya que un conjunto de datos pequeño, tras las separaciones en subgrupos, resultaría en un conjunto de entrenamiento pequeño y esto podría provocar que el modelo no sea capaz de entrenar correctamente al tener pocos datos sobre los que basar el entrenamiento.

Para este Trabajo de Fin de Grado, y como ya se comentó en el inicio de este documento (Capítulo 1), el problema que estamos tratando es sobre análisis de imágenes de resonancia magnética, por tanto lo conjuntos que usaremos contendrán este tipo de imágenes. Como el objetivo de este trabajo es la detección temprana de la enfermedad de Alzheimer, es necesario que nuestras MRI sean del cerebro y de pacientes en los tres estadios que se van a predecir, mencionados en el Capítulo 3. Además, para poder realizar un entrenamiento y posterior prueba del modelo, es necesario que los datos

estén etiquetados de alguna forma.

A continuación, tras esta explicación de los elementos más importantes a tener en cuenta a la hora de elegir los datos que vamos a usar, se va a realizar una explicación de los conjuntos de datos que se han elegido y utilizado para el desarrollo de este proyecto. En concreto, en nuestro caso, los datos usados provienen todos de la base de datos de ADNI [10].

8.1. ADNI

The Alzheimer's Disease Neuroimaging Initiative (ADNI) es un estudio longitudinal multicentro diseñado para desarrollar biomarcadores clínicos, de imagen, genéticos y bioquímicos para la detección temprana de la enfermedad de Alzheimer [10]. Además, como bien explican los promotores de la iniciativa, uno de los principales objetivos es el de continuar administrando la innovadora política de acceso a datos de ADNI, que proporciona acceso a estos datos, sin restricción, a todos los científicos del mundo. Esto facilita mucho el estudio de esta enfermedad ya que cualquier científico puede solicitar el acceso a estos datos e iniciar un proceso de investigación.

La base de datos que nos proporciona ADNI es muy grande y, como no podemos usar todos los datos, necesitamos seleccionar una parte de ellos. Para realizar esta selección en vez de analizar todos los posibles datos que hay y coger los mejores, lo cual nos supondría un gran esfuerzo, hemos decidido usar datos que han usado otros investigadores y los cuales podemos conocer mediante sus repositorios de Github [39]. Además, de esta forma, podremos comparar los resultados obtenidos. En concreto los repositorios que hemos elegido como fuente de donde obtener nuestros conjuntos son los siguientes:

1. **ADNI [40]:** este repositorio hace uso de un conjunto de 1179 imágenes (ejemplo en la Figura 8.1).

2. **brain2020 [41]:** este repositorio hace uso de 417 imágenes (ejemplo en la Figura 8.2).

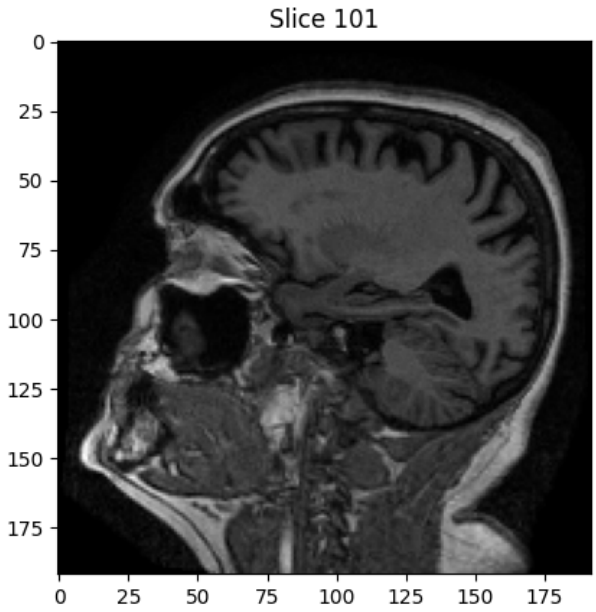

Figura 8.1: Ejemplo imagen I38726 del conjunto ADNI

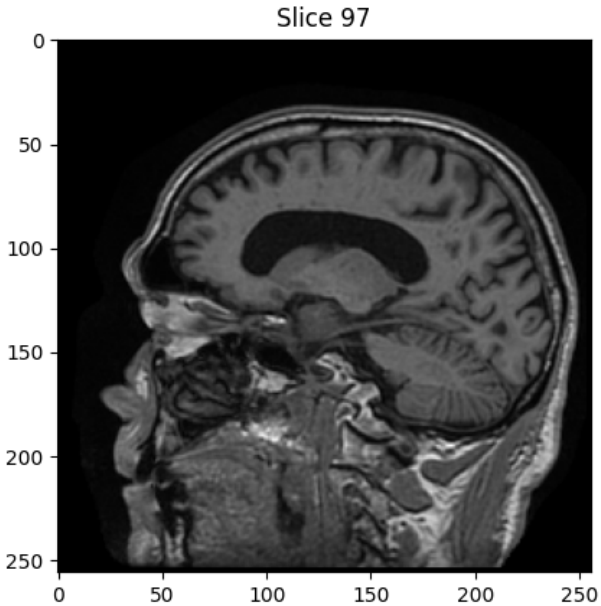

Figura 8.2: Ejemplo imagen I119138 del conjunto brain2020

La intención es combinar las imágenes de ambos proyectos para obtener un conjunto de imágenes más amplio, con el objetivo de mejorar así la efectividad del modelo.

Para obtener las etiquetas de las imágenes, se han creado colecciones en la plataforma de ADNI que representan estos dos conjuntos. Dentro de estas colecciones puedes obtener un archivo CSV [33] con toda la información acerca de las imágenes. Usando este CSV, hemos creado un *script* que extrae la ID de la imagen y su etiqueta. Esa ID la usa para buscar la imagen y concatenar la ruta de la imagen con su etiqueta, ya que esta información es la que luego usa nuestro modelo para acceder a las imágenes y sus categorías. Además las etiquetas, que vienen en texto, son tranformadas a formato numérico:

- **AD:** indica que un paciente tiene la enfermedad de Alzheimer (*Alzheimer's Disease*). A esta etiqueta se le ha asignado el valor **0**.
- **MCI:** indica que un paciente tiene un deterioro cognitivo leve (*Mild Cognitive Impairment*). A esta etiqueta se le ha asignado el valor **1**.
- **CN:** indica que un paciente no tiene la enfermedad (*Cognitive Normal*). A esta etiqueta se le ha asignado el valor **2**.

Este paso de las etiquetas a formato numérico permite al modelo poder clasificar las imágenes correctamente en su grupo.

Además, analizando las etiquetas de las imágenes obtenidas, se detectó que las etiquetas no estaban balanceadas. Se ha podido determinar que en la combinación de estos juntos de datos había 835 imágenes para CN, 501 para AD y 12 para MCI (Figura 8.3). Para corregir este desequilibrio se han buscado en la base de datos oficial de ADNI, todas las imágenes para el grupo MCI y se han encontrado 327 en total, que han sido añadidas a nuestro data set. Por último, con intención de equilibrarlo del todo, se han eliminado 162 imágenes del grupo AD y 496 del grupo CN. Esto deja a todos los grupos con el mismo número de imágenes (339) y con un total de 1017 imágenes (Figura 8.4).

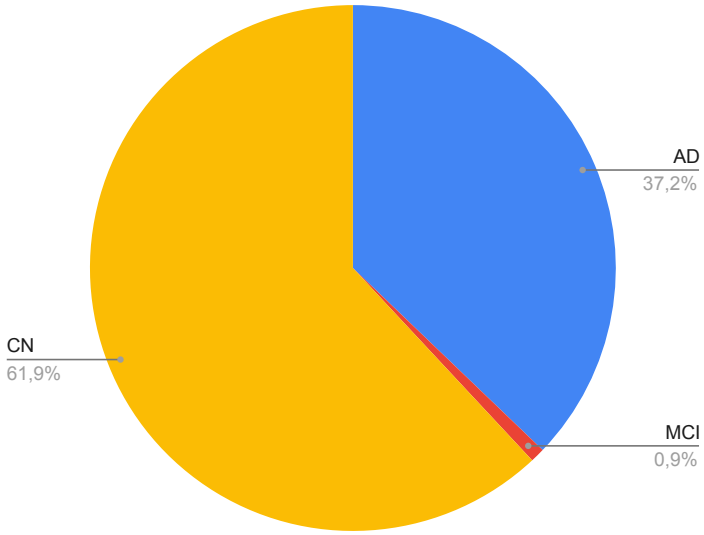

Figura 8.3: Gráfico de distribución de etiquetas antes del balance.

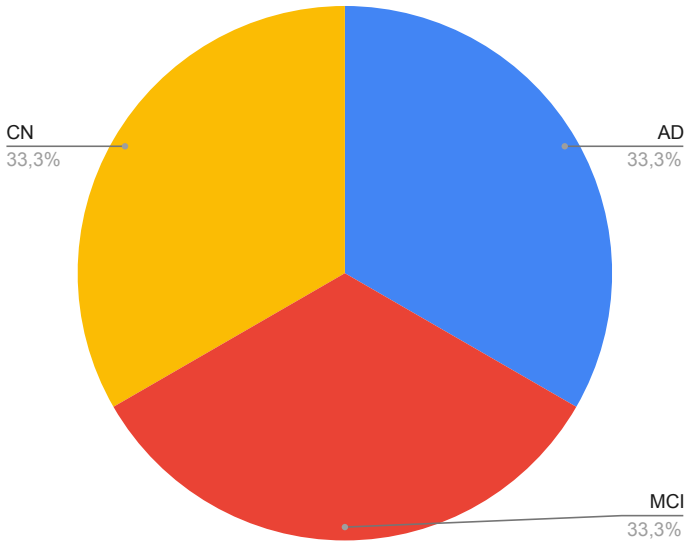

Figura 8.4: Gráfico de distribución de etiquetas después del balance.

Capítulo 9

Preparación de los datos

En este capítulo vamos a tratar el preprocesamiento que se ha realizado sobre los datos. Para obtener modelos de buena calidad, eficientes y que sean capaces de predecir resultados correctamente, es necesario que se realice una buena preparación de los datos. Esto nos asegura que las imágenes son de la mejor calidad posible y se elimina posible ruido e información innecesaria para que el modelo pueda realizar un mejor entrenamiento.

A continuación se muestran los procedimientos que se han realizado sobre las imágenes usadas para este Trabajo de Fin de Grado.

9.1. Selección de capas útiles

Como se ha comentado en otros capítulos, en este proyecto estamos trabajando con MRI. Estas imágenes son tridimensionales y a la hora de interpretarlas se comportan como capas de imágenes bidimensionales apiladas. En nuestro caso, como se puede ver en las Figuras 8.1 y 8.2, las imágenes son del lateral de la cabeza. Si analizamos todas las capas, podemos ver cómo hay capas que ni siquiera muestran el cerebro o solo muestran fondo (Figura 9.1).

Una vez hemos detectado esto, tenemos que definir un algoritmo que sea capaz de encontrar los límites a partir de los cuáles las capas empiezan a ser útiles para nuestro modelo. Para ello hemos buscado distintos métodos de

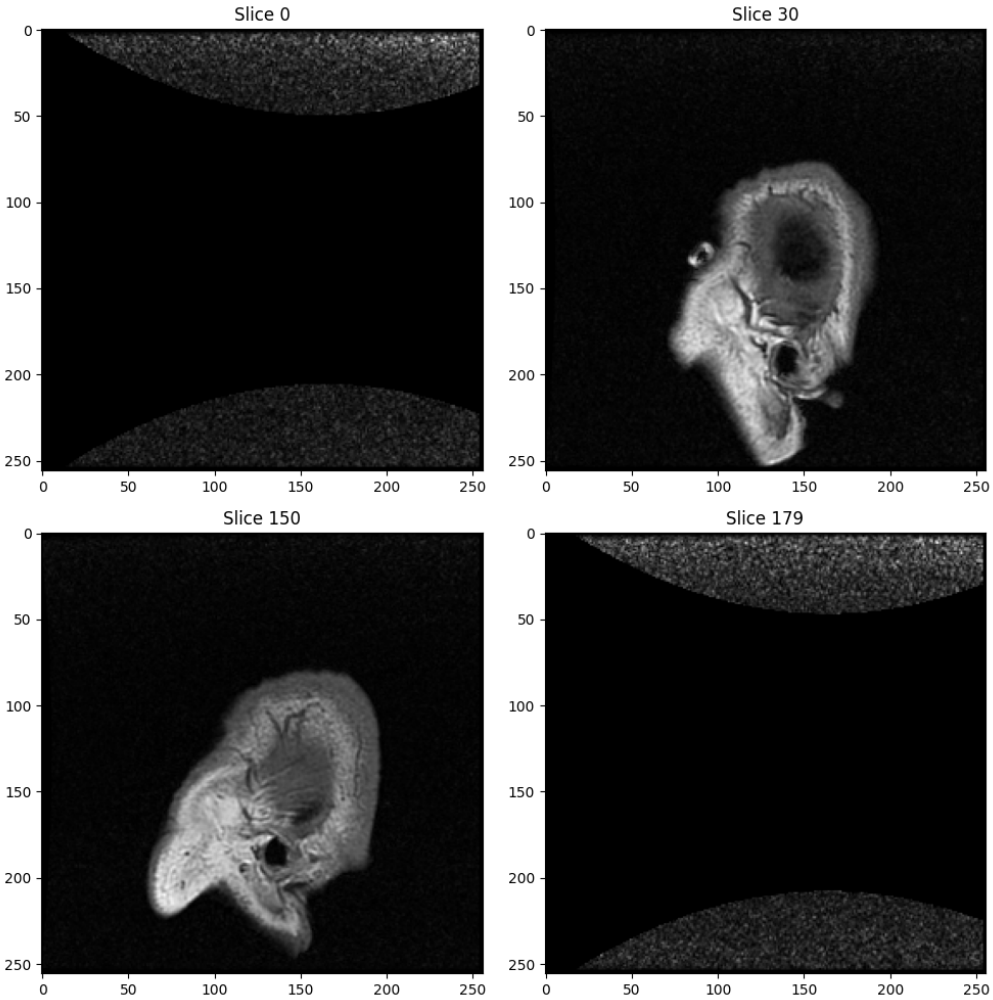

Figura 9.1: Ejemplo capas no útiles en imagen I3344

análisis de imágenes. Primero, y como más básicos, usamos la suma, media y mediana del valor de los píxeles de cada capa y obtuvimos una gráfica con los resultados y las capas que daba como límites según un umbral establecido por el investigador. Para estos métodos hemos obtenidos los resultados de las Figuras 9.2, 9.3 y 9.4.

Viendo las gráficas obtenidas, consideramos que la mejor opción entre las vistas es la de la mediana, ya que las otras aumentan de manera muy progresiva y es difícil encontrar el umbral a partir del cual seleccionar las capas de manera consistente. Aun así, se ha tomado la decisión de probar

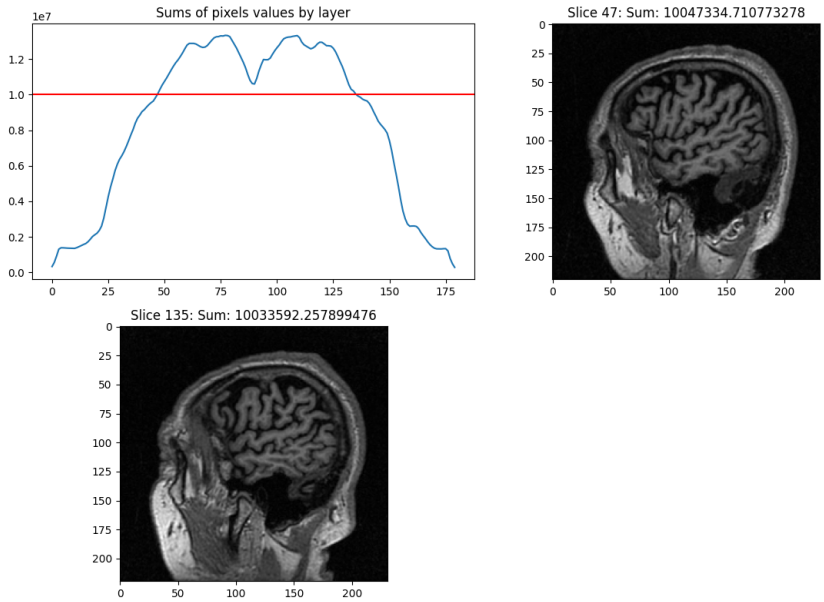

Figura 9.2: Análisis de la Suma de los valores de los píxeles

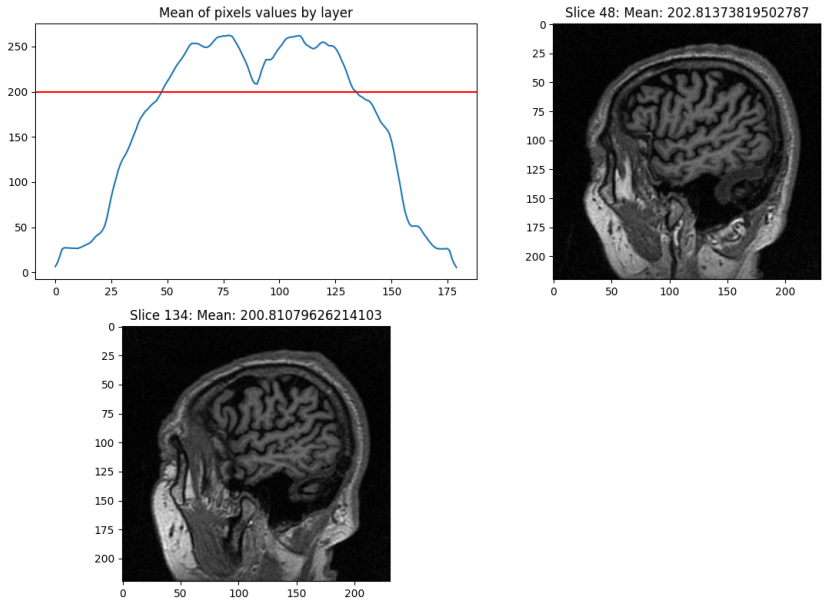

Figura 9.3: Análisis de la Media de los valores de los píxeles

más técnicas antes de seleccionar una. Además de las ya mencionadas se han probado las siguientes:

- **Umbral Otsu:** el algoritmo determina el umbral de intensidad único

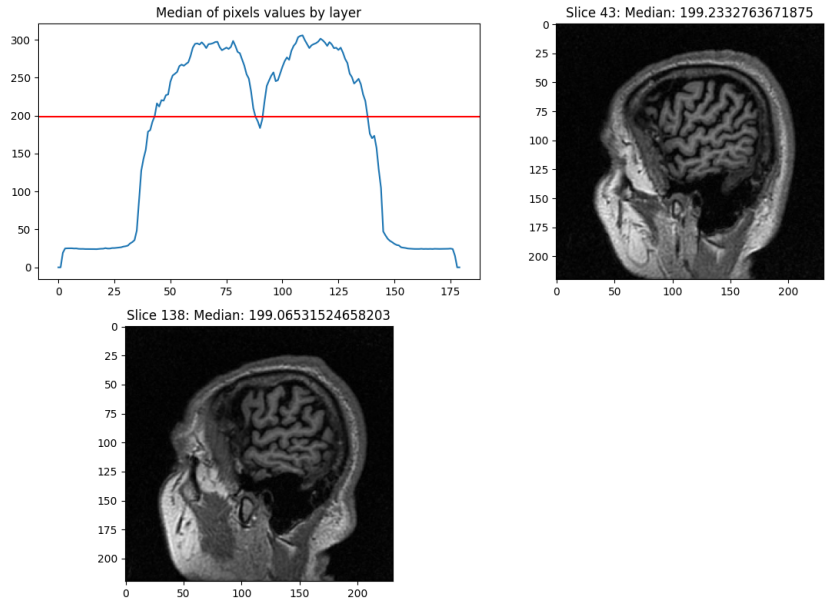

Figura 9.4: Análisis de la Mediana de los valores de los píxeles

que separa los píxeles en dos clases, primer plano y fondo [42] (Figura 9.5).

- **Entropía:** es una medida estadística de aleatoriedad que se puede utilizar para caracterizar la textura de la imagen de entrada [43] (Figura 9.6).

- **Varianza [44]:** esta métrica, aplicada en imágenes, indica el nivel de variación en los valores de los píxeles en cada capa. Capas con mayor varianza son propensas a contener estructuras más significativas (Figura 9.7).

- **Detección de bordes Sobel:** el operador de Sobel realiza una medición del gradiente espacial bidimensional en una imagen, enfatizando las regiones de alta frecuencia espacial que corresponden a los bordes [45] (Figura 9.8).

- **Skewness [46]:** es una medida de asimetría que, cuando se aplica en imágenes, mide la asimetría del histograma de la intensidad de los píxeles para encontrar capas con contenido significativo (Figura 9.9).

A continuación se muestran los resultados obtenidos usando estas métricas:

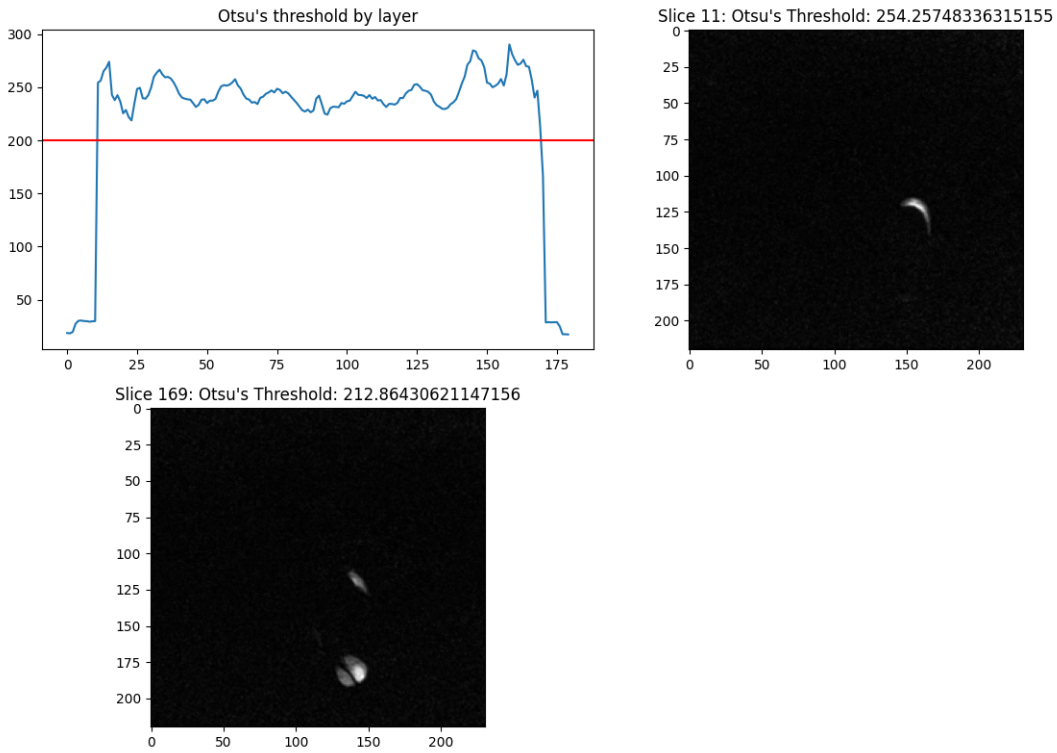

Figura 9.5: Análisis del Umbral de Otsu

Teniendo en cuenta estos resultados, se determinó que los métodos más eficientes son los de la media y la detección de bordes de Sobel. Para poder tomar una decisión sobre cuál de estos dos era mejor, primero se tomó el umbral usado en la imagen de prueba y se extrajo a qué porcentaje del valor máximo correspondía ese valor y se usó ese porcentaje para las siguientes pruebas, ya que al usar un valor fijo, si la imagen variaba un poco, el valor podía quedar fuera de la gráfica y nunca encontrar un borde.

Tras hacer esto y ejecutar ambos métodos en 10 imágenes diferentes y comparar los resultados variando el umbral para encontrar el punto óptimo, se pudo observar que Sobel nos da los resultados más consistentes, por lo que se ha decidido usar este método para seleccionar las capas útiles (Figura 9.10).

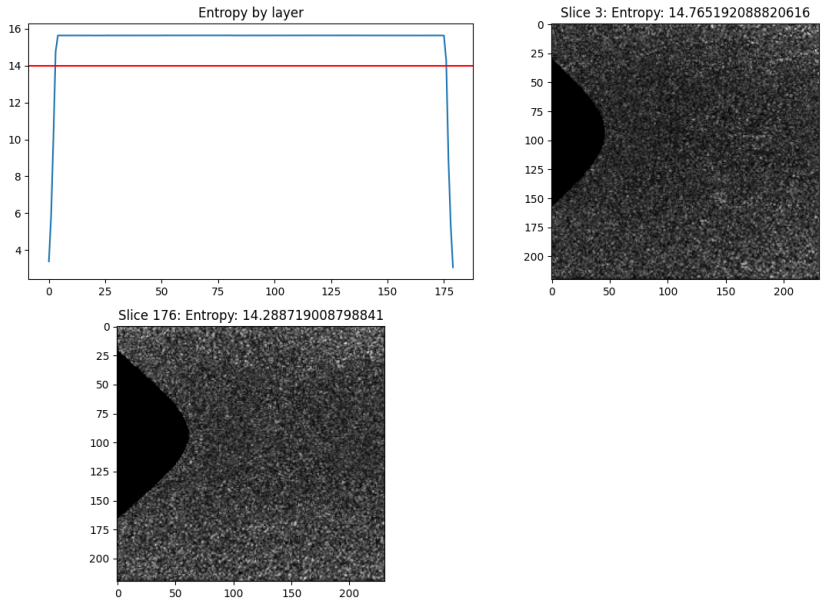

Figura 9.6: Análisis de la Entropía de la imagen

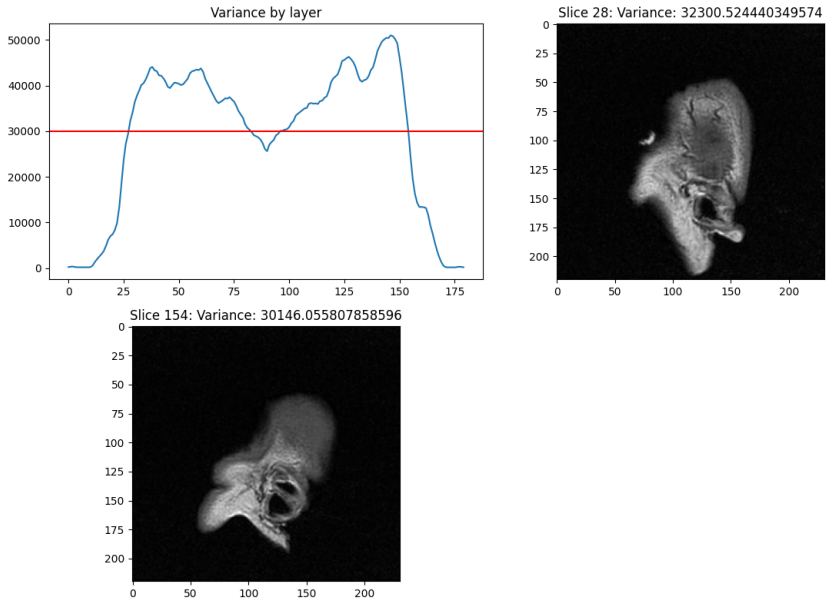

Figura 9.7: Análisis de la Varianza de los valores de los píxeles

9.2. Recorte de fondo de capas

Una vez determinadas las capas útiles de la imagen, ahora tenemos que recortar la imagen para eliminar todo el fondo posible. El método que hemos

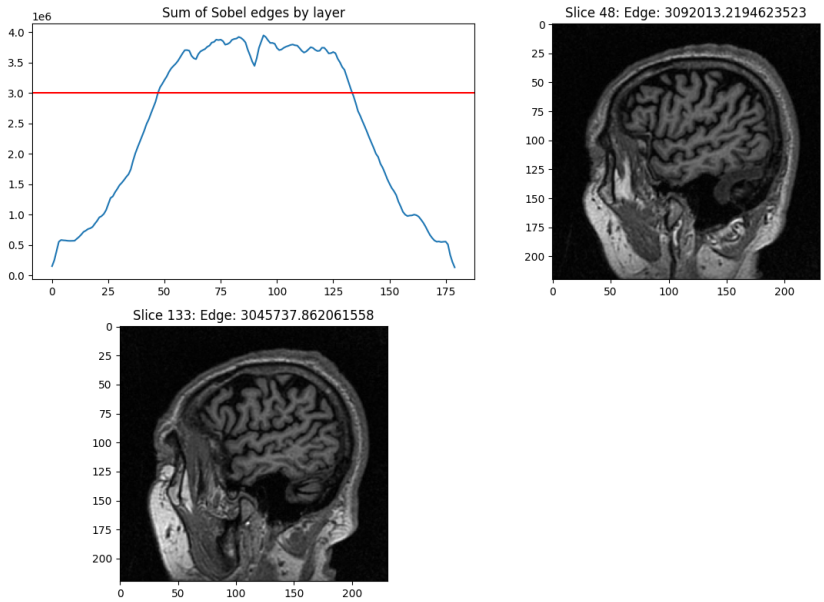

Figura 9.8: Análisis de la Suma de los bordes de Sobel

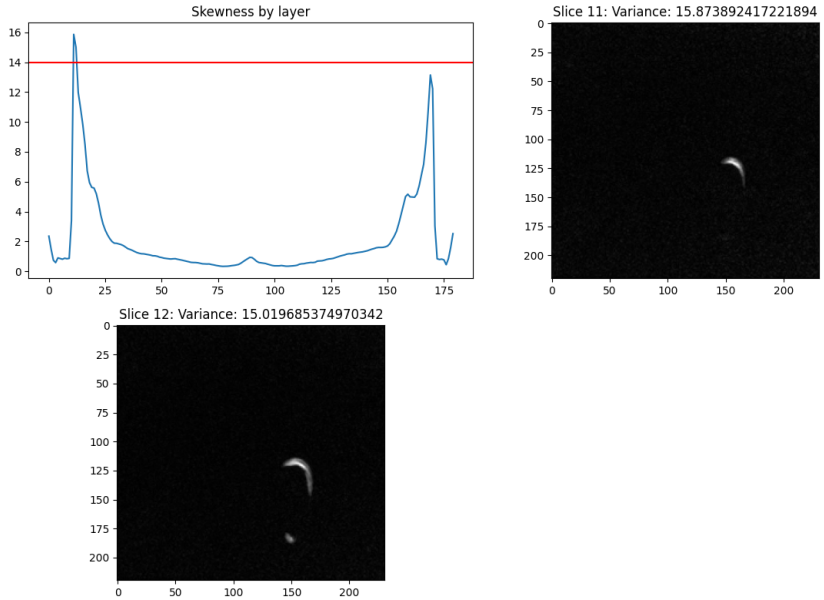

Figura 9.9: Análisis de la oblicuidad del histograma de los valores de los píxeles

seguido consiste buscar en qué punto empieza a haber píxeles de color y establecer ese punto como punto de corte. Esto se hace por arriba, abajo, izquierda y derecha. Además, esto no puede realizarse en todas las capas, ya

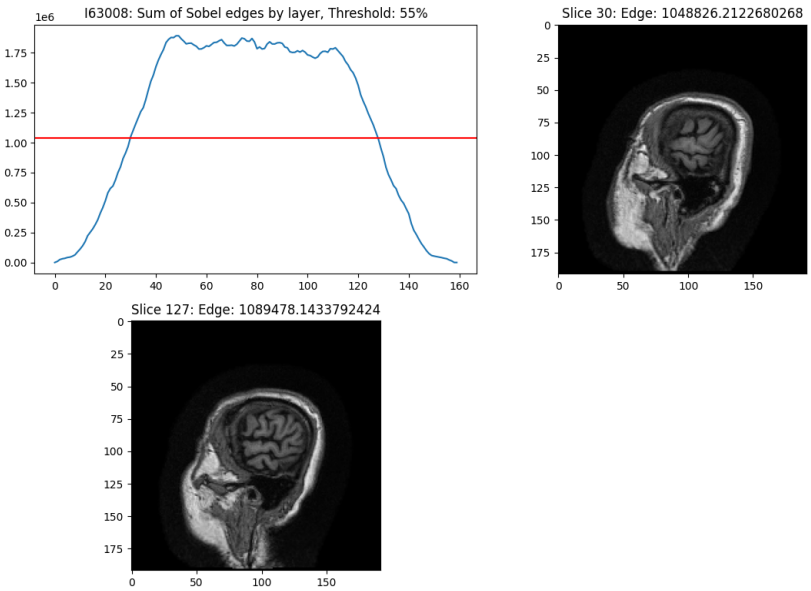

Figura 9.10: Resultado Imagen I63008 con Umbral: 55 %

que todas las capas de la imagen deben tener el mismo tamaño y, como se ha visto en la sección anterior, hay capas con mucho más fondo que otras. Para determinar sobre qué capa hacer los cortes, se ha decidido usar la capa que más cantidad de color muestre haciendo la suma de los valores de todos los píxeles de la imagen. Luego de esa capa se realizan unos gráficos de la media de los valores de los píxeles por columna y fila para determinar en qué valor vamos a establecer el umbral y después, igual que antes, obtener el porcentaje. A continuación se pueden ver las gráficas usadas para determinar el umbral (Figura 9.11) y la capa cortada (Figura 9.12).

9.3. Redimensionado de las imágenes

Por último, una vez ya hemos realizado todos estos procesos sobre las imágenes, es necesario redimensionarlas. Esto se debe a que no todas la imágenes tienen el mismo tamaño y el mismo número de capas y es necesario que todas las imágenes que entren al modelo tengan el tamaño que el modelo acepta. Esto se ha realizado usando una técnica conocida como interpolación tricúbica [31], siendo esta una técnica que trata de variar en la menor medida

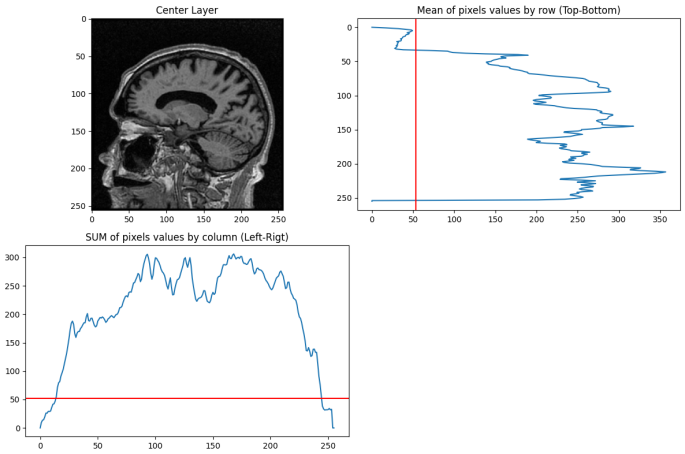

Figura 9.11: Gráficas para el análisis del corte, Imagen I33444, Threshold Row: 15 %, Threshold Column: 17 %

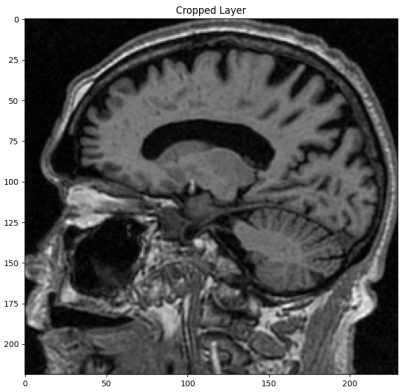

Figura 9.12: Imagen I33444 cortada

posible a la calidad de los datos. Esto se ha realizado con un *script* que recibe todas las imágenes y las procesa guardando los resultados en un directorio diferente al original.

9.4. Data augmentation

Además, y solo para el proceso de entrenamiento, se ha usado la técnica de ampliación de los datos conocida como *Data Augmentation*. Lo que se

hace es, a la hora de entrenar, se generan nuevas imágenes a partir de las que ya tenemos aplicando giros, transformaciones afines y ruido. Esto incrementa las posibilidades de obtener mejores resultados tras el entrenamiento del modelo al tener una mayor cantidad de datos de entrada. Este proceso lo realizamos haciendo uso de las funciones ya predefinidas de la librería TorchIO [47], que forma parte del ecosistema de PyTorch.

Parte III

Diseño del sistema

Capítulo 10

Modelos de aprendizaje

En este capítulo se van a explicar de manera más extensa todas aquellas partes que componen el modelo de red desarrollado para la detección temprana de la enfermedad de Alzheimer mediante el análisis de imágenes de resonancia magnética.

10.1. Modelo CNN

Para este proyecto se van a usar Redes Neuronales Convolucionales o CNN [26] para analizar las MRI. En concreto, se necesita usar CNN que sean tridimensionales [16] ya que las MRI son imágenes en tres dimensiones del cerebro, por lo que las CNN convencionales que usan imágenes en dos dimensiones no serían útiles. En la Figura 4.3 que aparece en el capítulo 4 se puede ver un ejemplo básico.

En esa imagen se puede apreciar cómo la red recibe una representación en tres dimensiones del cerebro y va extrayendo mapas de características de la misma, a la vez que avanza a través de las distintas capas del modelo. Esto se realiza en las llamadas capas convolucionales. Entremedias de estas capas convoluciones se puede realizar un proceso de *pooling* a la salida de la capa para reducir el tamaño de estos mapas y también se puede aplicar una función de activación, siendo la ReLU la más habitual.

Una vez se ha pasado por todas las capas convolucionales definidas, el

mapa de características se pasa a una capa completamente conectada que recibe esas características como entrada y clasifica finalmente la imagen. Este modelo nos devuelve como resultado unos valores que, aplicándole una función de activación *softmax*, transformamos en probabilidades de pertenencia a una clase. En nuestro caso, nuestro modelo va a devolver una lista con 3 probabilidades que indicarán con qué seguridad clasifica el modelo la imagen de entrada en AD, MCI o CN. Para determinar la clase predicha se cogería la clase a la que el modelo le haya asignado mayor probabilidad.

La estructura concreta de este modelo, número de capas, tamaño de la entrada, etc., lo definirán los hiperparámetros que se determinarán mediante la experimentación.

Además de un modelo creado por el alumno, con el objetivo de comparar y buscar los mejores resultados se han implementado dos modelos que ya había sido previamente definidos por otros investigadores.

10.1.1. Modelo de El-Assy

En el artículo *"A novel CNN architecture for accurate early detection and classification of Alzheimer's disease using MRI data"* [48] A.M. El-Assy, junto con otros autores, propone un modelo de red con dos CNN de las cuales se va a usar solo una. Para implementar esta red, se va a usar la descripción que se da sobre la misma en el artículo y se va a adaptar a nuestros datos de entrada y salida. Según los autores, la CNN tiene las siguientes características:

1. Dos capas convolucionales con 16 filtros de 3×3 cada uno.
2. Una capa de *max-pooling* con un salto (o *stride*) de 2.
3. Otras dos capas convolucionales, esta vez con 64 filtros, también de 3×3, cada uno.
4. Otra capa de *max-pooling* igual a la anterior.
5. Una capa de convolución con 256 filtros de 3×3.

6. Una capa de *dropout* con una probabilidad del 20 % de eliminación y aplicando normalización por lotes (o *batch normalization*).

7. Por último, una capa completamente conectada con 128 neuronas.

10.1.2. Modelo GoogleNet

GoogleNet (o InceptionV1) fue propuesto por Google en una investigación junto con varias universidades [49]. Esta arquitectura ha proporciona una disminución significativa de la tasa de error en comparación con otras redes del estado del arte como AlexNet [50]. Este modelo presenta la siguiente arquitectura que cuenta con 22 capas de profundidad.

- Una serie de módulos Inception que combinan capas convolucionales con filtros de diferentes tamaños. Cada módulo tiene una combinación de tres capas con filtros de 1×1, 3×3 y 5×5 convoluciones, respectivamente, junto con una capa de *max-pooling* de 3×3.

- Se utiliza una capa de *average pooling* global antes de la capa completamente conectada, lo que reduce significativamente el número de parámetros.

- La red incluye varias capas de *dropout* con una probabilidad de desactivación del 40 % para prevenir el sobreajuste.

- Tres clasificadores auxiliares para combatir el desvanecimiento del gradiente, con dos de ellos en posiciones intermedias para ayudar en la retropropagación durante el entrenamiento.

- Finalmente, una capa completamente conectada con 1000 neuronas seguida de una capa *softmax* para la clasificación de las imágenes.

Debido a que GoogleNet fue creada para imágenes bidimensionales, en este proyecto se ha usado su arquitectura para adaptarla a las entradas tridimensionales que acepta nuestro modelo.

10.2. Loss Function

Durante el entrenamiento, el modelo se optimiza mediante un algoritmo de descenso de gradiente, el cual utiliza una **función de pérdida** (***loss function***) para guiar el proceso de ajuste de los parámetros, con el objetivo de minimizar el error y mejorar progresivamente los resultados que devuelve.

Esta función devuelve un valor que indica al modelo cómo de buenos son los resultados que está devolviendo y le permite corregir en función de este valor. El objetivo del modelo siempre va a ser obtener el mínimo error posible. En la Figura 10.1 se puede ver un ejemplo del valor de la *loss function* durante el entrenamiento para la conocida base de datos de números escritos a mano MNIST [51].

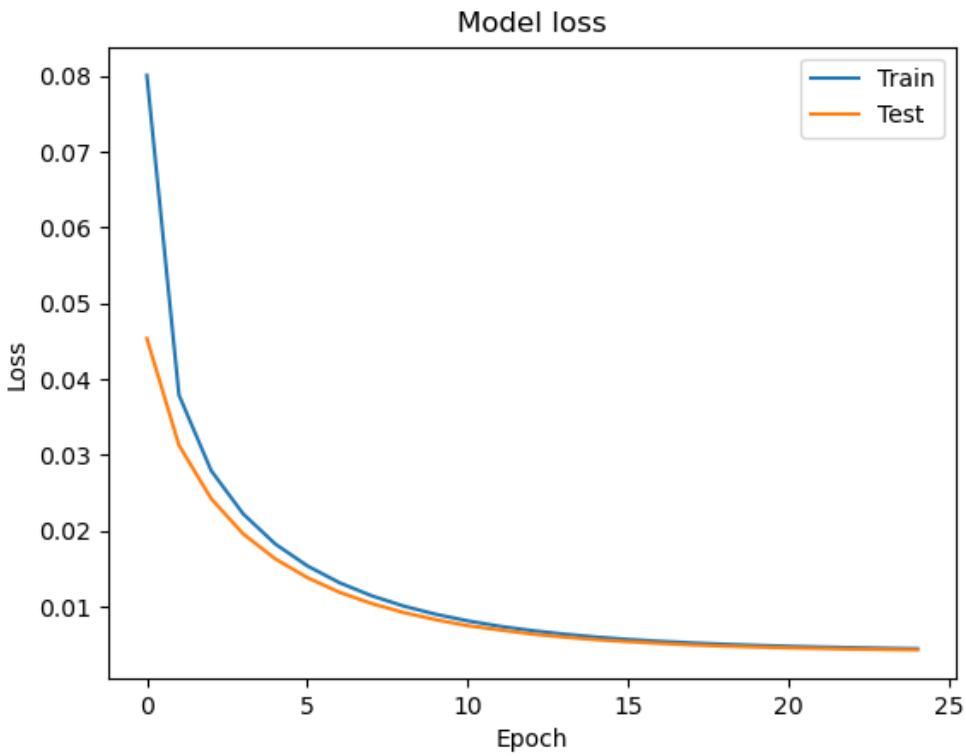

Figura 10.1: Ejemplo del valor de la *loss function* durante el entrenamiento [52]

Hay numerosas funciones de pérdida que se pueden usar y su elección

depende del problema que se intenta resolver. Para este proyecto se ha decidido usar la función de **Entropía Cruzada**.

La función de pérdida Entropía Cruzada o *Cross-Entropy Loss* es una medida usada para identificar qué tan bien la categorización de un modelo está funcionando. En concreto cuantifica la desviación entre las probabilidades predichas y los valores reales. En concreto, para problemas multiclase como el que se nos presenta, mide la habilidad del modelo para distinguir entre las distintas clases. Por simplificar, *Cross-Entropy* premia al modelo cuando predice bien y lo penaliza en caso contrario. En su forma matemática, se expresa de forma diferente para problemas de clasificación binarios y multiclase [53] [54].

En problemas de clasificación binarios la función de pérdida de entropía cruzada para cada entrada es:

$$L(x, y) = -[y * log(P(\mathrm{y} = y|x)) + (1 - y) * log(1 - P(\mathrm{y} = y|x))]$$

Donde y representa el valor real de la etiqueta de la entrada y P es la probabilidad de que la clase sea y (la clase real) dado x (los datos de entrada). Tambien se puede calcular la media del error de la siguiente forma:

$$\overline{L}(x, y) = -\frac{1}{N}\left[\sum_{j=1}^{N}[y_j * log(P(\mathrm{y} = y_j|x_j) + (1 - y_j) * log(1 - P(\mathrm{y} = y_j|x_j)]\right]$$

Donde N es el número de patrones. Para problemas de clasificación multiclase cambia un poco la función. Para problemas con C clases donde y y P son listas de tamaño C de valores reales, o esperados, y probabilidades respectivamente, la función de entropía cruzada para cada entrada quedaría de la siguiente forma:

$$L(x, y) = -\sum_{i=1}^{C}[y_i * log(P(\mathrm{y} = y_i|x))]$$

En este caso también se podría calcular la media para todas las entradas así:

$$\overline{L}(x,y) = -\frac{1}{N}\left[\sum_{j=1}^{N}\left(\sum_{i=1}^{C}[y_{ij} * log(P(\mathrm{y} = y_i|x))]\right)\right]$$

10.3. Hiperparámetros

Los **hiperparámetros** son parámetros cuyo valor no puede ser ajustado durante el proceso de entrenamiento. Estos valores no son conocidos a priori y son obtenidos mediante experimentos que se realizan con el objetivo de buscar el valor óptimo para cada caso, es decir, aquellos valores para los cuales el modelo nos proporcione mejores resultados. Los hiperparámetros pueden clasificarse como **hiperparámetros del modelo**, que no pueden deducirse mientras se ajusta la máquina al conjunto de entrenamiento porque se refieren a la tarea de selección del modelo, por ejemplo la topología de la red, o **hiperparámetros del algoritmo**, que pueden afectar a la convergencia del modelo, por ejemplo la tasa de aprendizaje [55].

10.3.1. Hiperparámetros del modelo

En esta sección se muestran todos los hiperparámetros que se tienen en cuenta para nuestro modelo.

Topología

La topología hace referencia a la estructura de la red neuronal. Esto es el número de capas convolucionales, de capas de *pooling* y de capas completamente conectadas y las funciones de activación usadas. Por tanto, la topología no es un único hiperparámetro sino un conjunto de ellos:

- **Número de capas convolucionales:** es importante determinar cuántas capas convolucionales se van a usar. Un mayor número de estas aumen-

ta la efectividad del modelo pero, a su vez, aumenta la complejidad computacional de este. El objetivo es encontrar el punto en el que el modelo sea efectivo y no muy exigente computacionalmente.

- **Capas de *Pooling***: estas capas reducen el tamaño de los mapas de características, reduciendo así el coste computacional. Aparecen siempre después de una capa convolucional pero no tienen por qué aparecer después de todas y cada una de ellas. Existen muchos modelos que introducen una capa de *pooling* cada 2 convolucionales o en otros casos como en el de la red AlexNet [50] en las que la estructura no es simétrica.

- **Capas completamente conectadas:** una vez terminada la fase de convolución, el mapa de características resultante entra a la segunda fase de capas completamente conectadas. La cantidad y estructura de estas capas, al igual que con las convolucionales, depende del equilibrio entre eficacia y coste computacional, por lo que también se busca la mejor eficacia sin que sea costoso computacionalmente.

- **Funciones de activación:** después de cada capa de convolución y completamente conectada es común usar funciones de activación que permiten que el modelo se ajuste a datos que contengan estructuras no lineales. En redes convolucionales la función que se usa más comúnmente es la función ReLU [56] debido a su simplicidad pero hay redes como la red LeNet [57] que usan funciones de activación sigmoides. Además, en la salida del modelo, se pueden usar funciones de activación, como *softmax* en el caso de este proyecto, para obtener probabilidades de pertenencia a clases a partir de la salida del modelo.

Dropout

Durante la fase de las capas complemente conectadas aparece el concepto de *dropout*. Este hiperparámetro hace referencia a la probabilidad que tiene el modelo de "apagar" nodos durante el proceso de entrenamiento, como se puede ver en la Figura 10.2. Se trata de una técnica de regularización y está demostrado que ayuda a que el modelo sea más robusto [58].

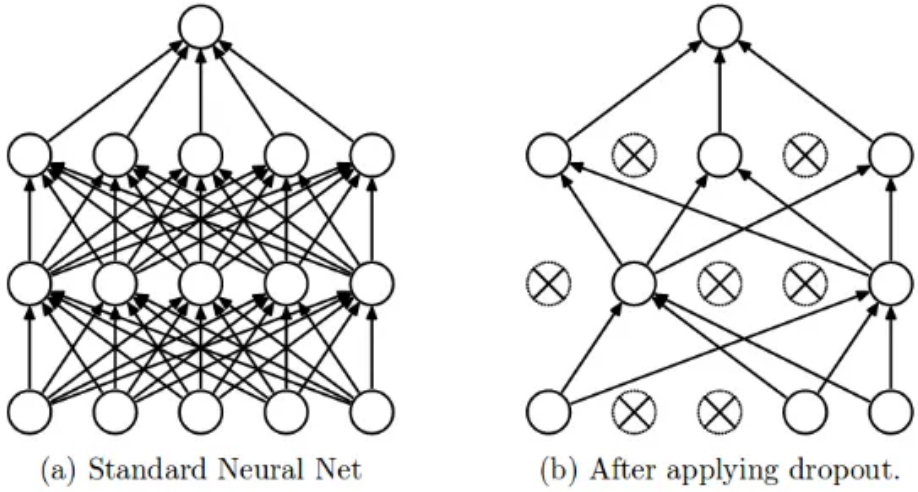

Figura 10.2: Ejemplo de *dropout* [59]

10.3.2. Hiperparámetros del algoritmo

Tasa de aprendizaje

La **tasa de aprendizaje** o ***learning rate*** es un parámetro que determina el tamaño del paso que da el algoritmo de optimización en cada iteración mientras minimiza la función de perdida.

Controlar adecuadamente la tasa de aprendizaje es esencial para garantizar una convergencia rápida. Un valor muy grande puede dar lugar a oscilaciones y provocar que nunca se alcance el óptimo buscado. Por otro lado, un valor muy pequeño puede alargar en bastante el tiempo de entrenamiento y hacer que se estanque en mínimos locales. Por ello es muy importante elegir bien el valor que se desea usar.

Optimizador

Los optimizadores dentro de las redes neuronales, son los encargados de optimizar los pesos del modelo para minimizar el valor de la función de pérdida. Básicamente se encargan de variar el peso de las conexiones del

modelo con la intención de reducir el error de la red. Existen numerosos tipos de optimizadores y el objetivo es encontrar aquel que funcione mejor para el problema propuesto. Por lo general, los optimizadores para redes neuronales se basan en el algoritmo de descenso de gradiente y existen ciertas variantes que incorporan mejoras con respecto a este algoritmo básico. Uno de los más conocidos en *deep learning* es el optimizador Adam [60].

Tamaño de *batch*

El **tamaño del lote** o ***batch size*** en redes neuronales convolucionales (CNN) se refiere al número de muestras de entrenamiento que se procesan juntas antes de que el modelo actualice sus pesos. En lugar de actualizar los pesos del modelo después de cada muestra (entrenamiento en línea) o después de procesar todo el conjunto de datos (entrenamiento en *batch* completo), se utilizan *batches* para equilibrar entre estos dos extremos.

El uso de *batches* es importante por varias razones. Primero, mejora la eficiencia computacional y el aprovechamiento del hardware. Segundo, proporciona un balance entre la actualización frecuente de los pesos, lo que puede acelerar el entrenamiento, y la estabilidad en el cálculo de los gradientes. Utilizar *batches* también ayuda a suavizar la estimación de los gradientes, haciendo que las actualizaciones sean más estables que en el entrenamiento en línea.

La elección del tamaño del *batch* puede afectar significativamente el rendimiento del modelo y la velocidad de entrenamiento. Un tamaño de *batch* muy pequeño puede llevar a estimaciones ruidosas de los gradientes, lo que puede resultar en una convergencia más lenta y potencialmente inestable. Por otro lado, un tamaño de *batch* muy grande puede requerir más memoria y recursos computacionales, además de poder llevar a una convergencia hacia mínimos locales menos generalizables. Encontrar el tamaño de *batch* adecuado implica un balance entre estos factores, y suele depender de la arquitectura del modelo, el conjunto de datos y los recursos disponibles.

Número de iteraciones y valor de espera

Por último, el **número de iteraciones** determina, por lo general, el número de veces que el modelo va a actualizar los pesos de las conexiones, lo que afecta en gran medida al resultado del entrenamiento del modelo. En este caso, al usar *batches*, el número de iteraciones se puede asociar con el número de veces que se pasará el conjunto de entrenamiento completo por la red. El número de actualizaciones de pesos será el número de iteraciones multiplicado por el número de *batches* por iteración. Un número de iteraciones muy bajo hace que los pesos se actualicen poco, por lo que no da tiempo al modelo a "aprender" de las entradas. En cambio, un número de iteraciones muy grande puede hacer que el modelo tarde mucho en entrenar y acabe produciendo sobreentrenamiento, haciendo que el modelo se "memorize" las entradas.

En este proyecto se va a utilizar una técnica denominada ***Early Stopping***. En esta técnica también hay que indicar un número de iteraciones, la diferencia es que también hace falta indicar un **valor de espera**. Lo que hace esta técnica es comprobar si el modelo ha mejorado con respecto a la anterior mejora, si no lo ha hecho sigue comprobándolo durante el número de iteraciones especificadas en el valor de espera y tras esto, si sigue sin producirse mejora, se corta el entrenamiento independientemente del número de iteraciones que queden por realizar. Por último, se restaura el modelo identificado como mejor antes de la parada de *early stopping*.

Esto convierte al **valor de espera** en otro hiperparámetro que hay que estudiar ya que un valor muy pequeño puede hacer que el entrenamiento termine antes de tiempo y un valor muy grande puede hacer volver la técnica inútil provocando que llegue antes el final de las iteraciones que el del *early stopping*.

10.4. Data Agumentation

La técnica de ***Data Augmentation***, o **Aumento de Datos**, es un conjunto de métodos utilizados para aumentar el tamaño y la variabilidad de un

conjunto de datos de entrenamiento, especialmente cuando se trabaja con imágenes. Al generar nuevas imágenes a partir de las existentes mediante transformaciones como rotaciones, traslaciones, escalados y cambios de color, se puede mejorar la capacidad del modelo para generalizar y hacer frente a nuevas imágenes que no ha visto antes. Esto es particularmente útil cuando se tiene un conjunto de datos limitado, ya que ayuda a evitar el sobreajuste al proporcionar más ejemplos de entrenamiento variados.

Al aplicar *data augmentation*, se mejora la robustez del modelo y su capacidad para reconocer patrones independientemente de variaciones menores en las imágenes. Además, este enfoque puede simular situaciones del mundo real, como diferentes orientaciones, iluminaciones y posiciones de los objetos dentro de una imagen, lo cual es crucial para el rendimiento de modelos de visión por computadora en aplicaciones prácticas.

- El **desenfoque** o ***blur*** es una técnica de *data augmentation* que aplica un filtro de suavizado a las imágenes, reduciendo los detalles finos y haciendo que las características principales sean más prominentes. Esto puede ayudar al modelo a aprender representaciones más robustas al hacer que se enfoque en las estructuras globales en lugar de los detalles específicos que pueden variar entre las imágenes. El desenfoque es particularmente útil para simular condiciones de captura de imágenes en las que las fotos pueden no ser perfectamente nítidas, como en movimientos rápidos o en ambientes con poca luz.

- El **volteo** o *flip* de imágenes implica invertir las imágenes horizontal o verticalmente. Esta técnica es sencilla pero efectiva, ya que muchas características de los objetos en las imágenes son asimétricas y pueden aparecer en diferentes orientaciones en situaciones reales. El *flip* ayuda a que el modelo sea invariante a estas orientaciones, mejorando su capacidad para generalizar.

- La **rotación de imágenes** consiste en girarlas en torno a su centro por un ángulo específico, ya sea en incrementos fijos o aleatorios dentro de un rango determinado. Esta técnica es útil para entrenar modelos que deben reconocer objetos independientemente de su orientación en

el plano de la imagen. Al rotar las imágenes, se puede aumentar la diversidad del conjunto de datos y enseñar al modelo a ser insensible a la orientación de los objetos.

Capítulo 11

Métricas de evaluación

Durante el desarrollo de este documento, se ha hablado de la importancia de seleccionar correctamente los datos que vamos a usar para el modelo, la preparación de estos y el entrenamiento del modelo como las claves para su éxito. Esto sin duda es importante pero también es muy importante medir el rendimiento del modelo durante el proceso de generalización. Si no se realiza un medición apropiada pueda dar lugar a desarrollar un modelo que realice malas predicciones.

Cuando se alimenta como entrada una MRI a nuestro modelo, este nos devuelve 3 valores denominado *logits* [61], uno para cada clase. Estos son los valores que devuelve un modelo multiclase por defecto y representa la salida lineal de un modelo antes de ser transformado a probabilidades. A estos *logits* le aplicamos la función *softmax* [62] para convertir cada uno de esos 3 valores en probabilidades de pertenencia a una clase. Luego de estos 3 cogemos el mayor de ellos que nos indica que entre las 3 clases posibles tiene mayor probabilidad de pertenecer a esa.

Antes de empezar a describir las métricas que se van a usar, es necesario conocer los resultados que podemos obtener de las predicciones que realice nuestro modelo. Como es un problema multiclase, se definirá para cada clase i:

- **Verdadero Positivo**: indica el número imágenes correctamente clasificadas como pertenecientes a la clase i. También se conoce como ***True***

Positive o ***TP***.

- **Verdadero Negativo**: indica el número imágenes correctamente clasificadas como **no** pertenecientes a la clase *i*. También se conoce como ***True Negative*** o ***TN***.

- **Falso Positivo**: indica el número imágenes incorrectamente clasificadas como pertenecientes a la clase *i* y que en realidad pertenecen a otra clase. También se conoce como ***False Positive*** o ***FP***.

- **Falso Negativo**: indica el número imágenes incorrectamente clasificadas como pertenecientes a otra clase y que en realidad pertenecen a la clase *i*. También se conoce como ***False Negative*** o ***FN***.

11.1. Accuracy

El *accuracy* [63] es una métrica de rendimiento muy popular en los problemas de clasificación cuya idea principal puede ser extraída de los problemas de clasificación binaria y ser usada en problemas multiclase de la misma forma.

Esta métrica mide la proporción de imágenes correctamente clasificadas en relación al total de las predicciones.

$$\text{Accuracy} = \frac{\text{Predicciones Correctas}}{\text{Todas las Predicciones}}$$

Si aplicamos este concepto a nuestro problema, las predicciones correctas serían la suma de los TP de cada clase divididos entre la suma de todos los valores (TP, TN, FP, FN) para cada clase. Esto quedaría de la siguiente forma:

$$\text{Accuracy} = \frac{\sum_{i=0}^{C} TP_i}{\sum_{i=0}^{C} (TP_i + TN_i + FP_i + FN_i)}$$

El *accuracy* es fácil de interpretar ya que nos da el porcentaje de valores que han sido clasificados correctamente. Sin embargo, no tiene en cuenta el balance entre clases y puede estar dando un buen resultado porque clasifica bien para todas las clases menos una clase concreta y puede parecer que el modelo es bueno cuando en realidad no lo es. Por ello, es recomendable usar otras métricas para complementar a esta.

11.2. Precision y Recall

Para complementar la métrica anterior, dos de las métricas que se usan en clasificación binaria son *Precision* y *Recall*. La *precision* representa el ratio de observaciones correctamente predichas como positivas del total de observaciones predichas como positivas. Por otro lado, el *recall* es el ratio de observaciones correctamente predichas como positivas del total de observaciones que de verdad pertenecen a la clase positiva. [64]

Estos mismos conceptos se pueden aplicar a problemas multiclase de dos formas, calculado la *precision* y el *recall* para cada clase individualmente o haciendo la media estos resultados para todas las clases.

- ***Precision***: para calcularlo, dividimos los TP de la clase i entre la suma de los TP y FP de la clase i dando como resultado el la *precision* para la clase i:

$$\text{Precision}_i = \frac{TP_i}{TP_i + FP_i}$$

- ***Mean Precision***: para calcularlo, dividimos la suma de la *precision* para cada clase entre el número de clases:

$$\text{Average Precision} = \frac{\sum_{i=0}^{C} \text{Precision}_i}{C}$$

- ***Recall***: para calcularlo, dividimos los TP de la clase i entre la suma de los TP y FN de la clase i dando como resultado el *recall* para la

clase i:

$$\text{Recall}_i = \frac{TP_i}{TP_i + FN_i}$$

- ***Mean Recall***: para calcularlo, dividimos la suma del *recall* para cada clase entre el número de clases:

$$\text{Average Recall} = \frac{\sum_{i=0}^{C} \text{Recall}_i}{C}$$

En resumen, podemos decir que la *precision* mide la relevancia del resultado y el *recall* mide cuántos resultados verdaderamente relevantes nos ha devuelto el modelo. Estas métricas están relacionadas de manera que el aumentar una implica la disminución de la otra. Con un valor alto para *recall* y bajo para *precision* el modelo se vuelve sensible y acepta imágenes que no debería, obteniendo muchas predicciones positivas pero con predicciones casi aleatorias. En el caso contrario, el modelo se volvería muy preciso, y nos asegura que las imágenes que clasifica como positivas son positivas pero devuelve muy pocas predicciones positivas dejando muchas positivas como negativas.

Esto se suele representar de forma gráfica con la llamada Curva PR. La curva PR (o curva *Precision-Recall*) es un gráfico que muestra el equilibrio entre la precisión (*precision*) y la exhaustividad (*recall*) de un modelo según un umbral de decisión. Esta curva es útil para evaluar el rendimiento de un clasificador en problemas con clases desbalanceadas. En la Figura 11.1 se puede ver un ejemplo de esta gráfica.

Esta curva nos permite ver la relación entre estos parámetros visualmente. Además existe el concepto de ***Average Precision*** que representa el área bajo la curva PR, el cuál nos sirve para evaluar y comparar el rendimiento de los modelos, cuanto más se acerque este valor a 1, mejor será nuestro modelo.

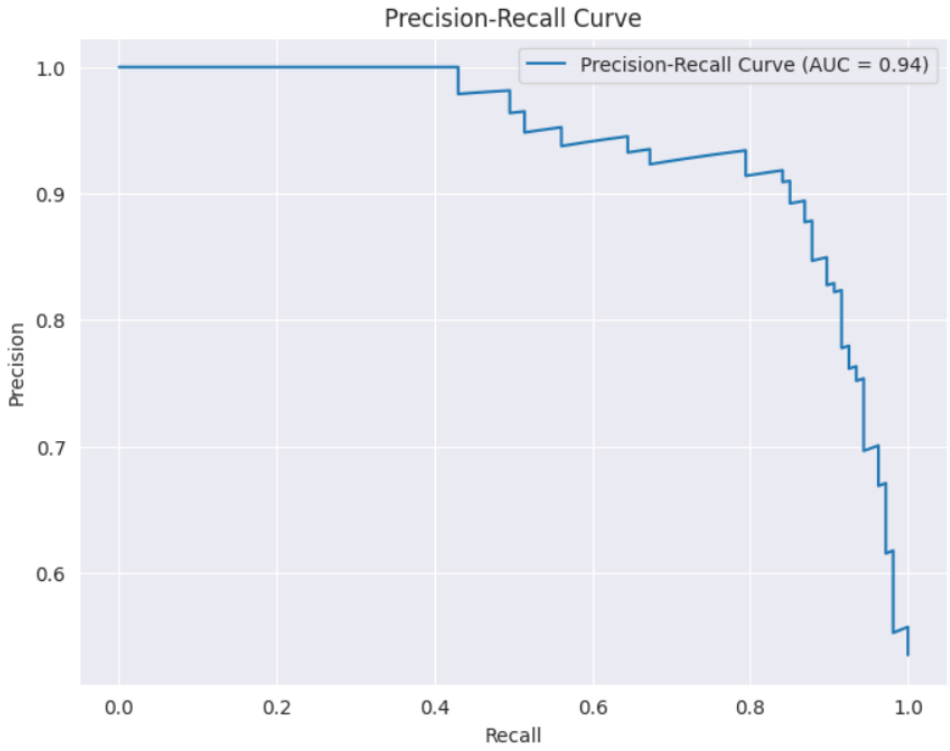

Figura 11.1: Ejemplo de la curva *Precision-Recall* [65]

11.3. F1 Score

Por último, otra métrica que se usa es el Valor F1. Este valor deriva de la *precision* y el *recall* y representa la media armónica de estas métricas. Esto quiere decir que representa simétricamente estos dos valores en uno solo. El valor máximo que se puede obtener es 1, lo que indica que ambos valores son los ideales, y el mínimo es 0, lo que significa que ambos son 0. La forma de calcularlo es la siguiente:

$$\mathrm{F1}_i = 2 \times \frac{\mathrm{Precision}_i \times \mathrm{Recall}_i}{\mathrm{Precision}_i + \mathrm{Recall}_i}$$

Capítulo 12

Definición de los experimentos

En este capítulo se describen los experimentos que se han llevado a cabo para evaluar el rendimiento del modelo propuesto para este Trabajo de Fin de Grado.

En primer lugar, para realizar los distintos experimentos y, como se ha comentado anteriormente, se va a usar la base de datos de ADNI. Al inicio de la ejecución del código, antes de que empiece la fase de entrenamiento, se realiza una separación de los datos basada en una semilla aleatoria proporcionada por el usuario, para permitir la replicación de los experimentos. La separación se realiza en tres subconjuntos, siendo esta realizada de manera estratificada para asegurar la separación homogénea de los datos:

- **Conjunto de entrenamiento o *train set*:** subconjunto de muestras de entrada que serán utilizadas a lo largo de las sucesivas épocas del proceso de aprendizaje de la red. Este subconjunto contiene un 80 % de los datos del conjunto original. Además, será almacenado en un archivo CSV cuyo nombre seguirá el formato `train_labels_{semilla}.csv`.

- **Conjunto de validación o *validation set*:** subconjunto de muestras de entrada desconocidas por el modelo que estamos entrenando y que serán utilizadas, tras finalizar cada época del entrenamiento, para evaluar si dicho modelo está aprendiendo correctamente y presenta un rendimiento óptimo. Este subconjunto se obtiene realizado una separación del conjunto de entrenamiento y se realiza mediante vali-

dación cruzada en tiempo de ejecución. El porcentaje del conjunto de entrenamiento que se va a usar como validación depende del número de *folds* que se quieren usar. En nuestro caso se van a usar 10 *folds* que corresponde a un 10 %.

- **Conjunto de generalización o *test set*:** conjunto de muestras de entrada independientes del conjunto de entrenamiento y validación que serán utilizadas, tras finalizar todo el proceso de entrenamiento, para determinar el rendimiento y capacidad de generalización de nuestro modelo final. Este subconjunto contiene un 20 % de los datos del conjunto original. Además, será almacenado en un archivo CSV cuyo nombre seguirá el formato `test_labels_{semilla}.csv`.

Como se ha comentado, el conjunto de validación se obtiene usando validación cruzada, en concreto la validación cruzada de K iteraciones o *K-fold cross-validation*. En este tipo de validación cruzada, el conjunto de datos se divide en k subconjuntos o *folds*. En cada iteración, uno de los *folds* se usa como conjunto de validación, mientras que el resto de los $k-1$ *folds* se usa como conjunto de entrenamiento. Este proceso se realiza k veces, usando cada *fold* exactamente una vez.

Este conjunto de validación se usa con dos propósitos. Primero se usa durante el entrenamiento para evaluar la precisión del modelo y hacer un ajuste de los pesos del mismo. Después, es usado por la técnica de *early stopping* para determinar cuándo el modelo ha dejado de mejorar.

Debido a que hay un gran número de hiperparámetros que pueden ser configurados, se han definido una serie de experimentos en los cuales se modifican algunos de ellos mientras que otros se dejan con unos valores preestablecidos.

Así pues, para la ejecución de los experimentos, se ha creado un *script* que genera un archivo (`task_params.txt`) con los diferentes valores que se le va a pasar al modelo en la llamada por línea de comandos para luego hacer la llamada a *HTCondor* [20] que usará ese archivo para añadir a la cola las ejecuciones de los diferentes experimentos que prueban todas las variaciones de los siguientes hiperparámetros:

- Modelos: 1 (creado por el alumno), 2 (creado por El-Assy y sus compañeros), 3 (GoogleNet).
- Paciencia de *early stopping*: 10, 15
- Tasa de aprendizaje: 10^{-2}, 10^{-3}, 10^{-4}

Además de estos se han preestablecido los siguiente parámetros:

- Número de *folds*: 10
- Tamaño de *batch*: 4
- Número de épocas: se ha establecido en 1000 ya que se está implementando la estrategia de *early stopping* que parará el entrenamiento cuando se cumplan las condiciones preestablecidas.
- La estrategia de *data augmentation*.

Finalmente, todos los modelos generados se han sometido a una fase de *test* cuyos resultados se analizarán en el capítulo 15.

Capítulo 13

Organización del software

Durante el desarrollo de este capítulo se ha presenta como estar organizado el proyecto con la intención de facilitar la lectura y comprensión de este documento. Este software contiene todos los procesos que se han usado durante el desarrollo de este Trabajo de Fin de Grado, desde el preprocesado de las imágenes hasta las predicciones usando el modelo entrenado.

En la figura 13.1 se puede ver un esquema de la organización por directorios del software que se detallará a continuación:

- El fichero `cnn.py` es el fichero principal del modelo y el que controla el funcionamiento de software llamando, cuando sea necesario a las distintas funciones creadas.
- El fichero `requeriments.txt` es usado para la instalación de las librerias de Python en su versión correspondiente.
- El fichero `submit.sh` es un *script* de *bash* que genera un archivo `task_params.txt` con los datos de todos los experimentos y hace la llamada a HTCondor que hace uso de este archivo para crear la cola de experimentos.
- En el directorio `cnn3d` se encuentran todos los ficheros que llama el archivo `cnn.py` y contienen el desarrollo de cada una de las fases del proceso de entrenamiento y experimentación del modelo. Dentro de este fichero se encuentran los siguientes ficheros:

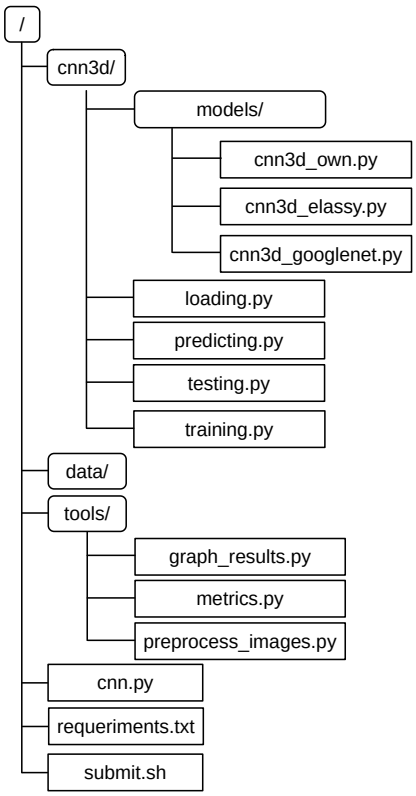

Figura 13.1: Esquema de la organización del proyecto

- El fichero `loading.py` contiene la función encargada de cargar el modelo de red neuronal, en caso de no realizar el entrenamiento y usar un modelo previamente entrenado.
- El fichero `predicting.py` contiene las funciones necesarias para la predicción del estado de un paciente dado su MRI. Hace uso de fichero `preprocess_image.py` para ajustar la imagen a lo requerido por la red neuronal.
- El fichero `testing.py` contiene las funciones necesarias para realizar las pruebas del modelo entrenado y obtener las méticas especificadas en el capítulo 11.
- El fichero `training.py` contiene todos los procesos para realizar el entrenamiento del modelo. Hace uso del fichero `cnn3.py` para obtener la estructura de la red.

- En el directorio `models` se encuentran todos los modelos que se usan para el entrenamiento. Dentro de este fichero se encuentran los siguientes ficheros:
    - El fichero `cnn3d_own.py` contiene la definición del modelo de red propuesto por el alumno, identificado como modelo 1.
    - El fichero `cnn3d_elassy.py` contiene la definición del modelo de red propuesto por El-Assy y otros autores, identificado como modelo 2.
    - El fichero `cnn3d_googlenet.py` contiene la definición del modelo de red GoogleNet, identificado como modelo 3.

- En el directorio `data` se encuentra todo lo relacionado con los datos, tanto las imágenes ya preprocesadas como el CSV que contiene las etiquetas y luego será usado por el modelo.

- Por último, tenemos el directorio `tools` que contiene herramientas extras usadas para el desarrollo del proyecto. Dentro se encuentran los siguientes fichero:
    - El fichero `graphs_results.py` es un script que genera los gráficos, tablas y mapas de calor a partir de los resultados del modelo.
    - El fichero `metrics.py` contiene las funciones que calculan las métricas con los datos obtenidas tras la fase de generalización.
    - El fichero `preprocess_image.py` contiene todo el proceso de preprocesado de las imágenes, desde que se reciben hasta que se guardan ya preprocesadas.

Parte IV

Pruebas y Resultados

Capítulo 14

Pruebas software

Durante el desarrollo de nuestro software, los distintos componentes y funciones que se han ido desarrollando se han probando de forma independiente, con el objetivo de asegurar que todas las partes funcionan correctamente a la hora de unirlas, evitando así problemas inesperados.

A continuación se comentarán algunas de las distintas pruebas realizadas para garantizar la correcta ejecución del software:

- En general, todos los *scripts* y funciones desarrolladas han sido probados de manera independiente. Las funciones han sido creadas y probadas en un cuaderno de *Jupyter*, por su facilidad de uso. Además, como todo esto se realiza en la máquina local, los *scripts* eran luego probados en el entorno de grupo AYRNA para asegurar que el código seguía funcionando correctamente. Por último, se incluían los *scripts* y funciones creadas en el código principal y se realizaba una ejecución simple con pocos datos para comprobar que todo ejecutaba correctamente.

- Para el caso del preprocesamiento de las imágenes, se realizaron pruebas de las distintas fases del preprocesamiento en un cuaderno de *Jupyter* sobre un pequeño conjunto de imágenes. Luego, estas fases se iban unificando hasta conseguir el *script* que realiza el procesamiento completo de las imágenes.

- Para el caso de el *script* de entrenamiento, se ejecutó de manera in-

dependiente en las máquinas del clúster con un número reducido de imágenes para probar que la red se creaba correctamente y no había problemas con el tamaño de las imágenes de entrada ni las salidas del modelo.

- Para el caso del *script* de *test* y predicción se usó un modelo previamente entrenado para probar que todos los datos se obtenían correctamente.

- También se probaron los *DataGenerators*. La forma de hacerlo fue comprobar que las rutas a las imágenes `.nii` era correctas y el *script* era capaz de abrir las imágenes, así como de aplicar la estrategia de *data augmentation*.

- Por último, se unieron todas las partes, juntando las partes repetidas en un solo *script* y organizándolas en carpetas para una mejor comprensión. Todo esto fue probado hasta asegurar que todo funcionaba correctamente.

Capítulo 15

Resultados experimentales

En este capítulo se mostrarán e interpretarán los resultados obtenidos en los experimentos que se han realizado.

Primero se mostrará cuál de las tres redes propuestas (explicadas en el capítulo 10) nos ha proporcionado mejores resultados y seleccionará las tres combinaciones de hiperparámetros con mejores resultados. Esto nos permitirá identificar la red que clasifica mejor nuestros datos para luego poder indagar en ella. Posteriormente, se pasará a analizar la influencia de la variación de los otros hiperparámetros en los resultados devueltos por esta red. Por último, se probará a modificar los hiperparámetros más influyentes para tratar de mejorar el modelo y obtener mejores resultados.

15.1. Resultados de los modelos

Como se comentó en el capítulo 10, para estos experimentos se han probado tres modelos de red distintos y se han variado algunos de sus hiperparámetros, con el objetivo de obtener el mejor modelo. Tras realizar los experimentos, hemos podido comprobar como tanto el modelo creado por alumno como el creado por El-Assy nos proporcionan peores resultados que el modelo GoogleNet. Para ello se puede apreciar la gráfica de la Figura 15.1 que muestra los resultados para la mejor combinación de hiperparámetros de cada modelo.

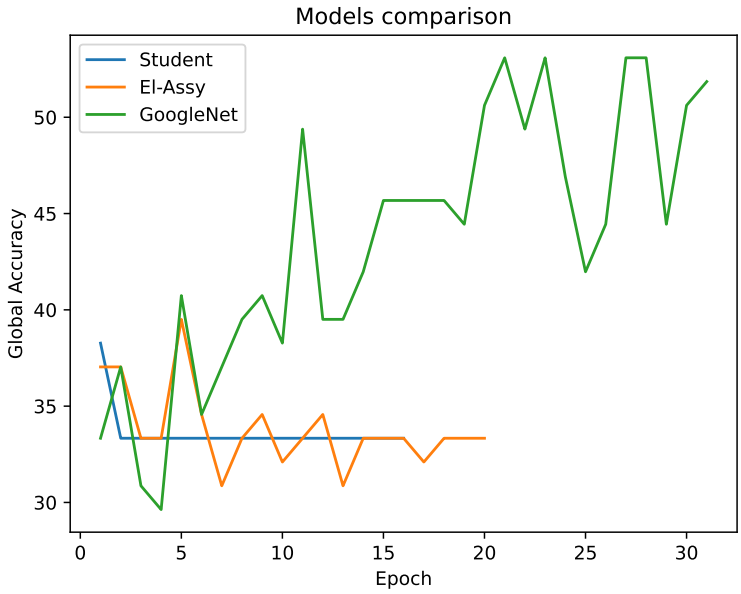

Figura 15.1: Comparación de la evolución de la *accuracy* de los distintos modelos durante el entrenamiento para el mejor *fold* de la semilla con mejores resultados en generalización para cada modelo.

En esta gráfica se muestra el mejor resultado de entre todos los experimentos realizados con cada uno de los modelos. Se puede apreciar claramente como los resultados obtenidos por el modelo GoogleNet son mucho más prometedores que los de los otros dos modelos. Si comparamos el valor de la puntuación F1 se puede ver como el modelo GoogleNet también es notablemente mejor con respecto a los otros dos modelos, llegando a superar 0.50, como se puede ver en la Figura 15.2.

También, si miramos la matriz de confusión de GoogleNet para estos hiperparámetros (10 de paciencia y 10^{-4} de coeficiente de entrenamiento) (Figura 15.3), se aprecia como, sin ser perfecta debido a la alta complejidad del problema, los resultados parecen indicar que el modelo va por buen camino y nos hace pensar que se podría llegar a obtener un modelo más preciso.

Aun así se puede apreciar como el modelo tiene dificultades para clasificar para la clase CN. Esto puede deberse a que la técnica de *early stopping* ha parado el entrenamiento antes de que hubiera terminado de clasificar co-

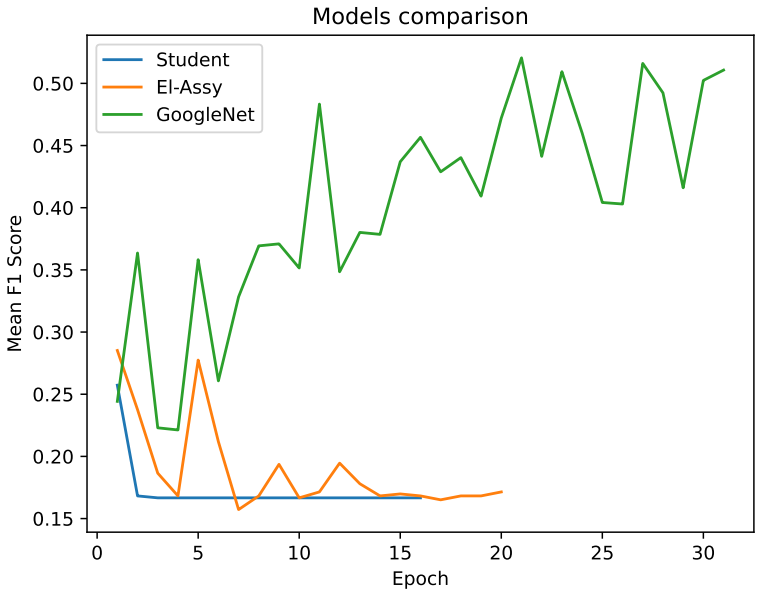

Figura 15.2: Comparación de la evolución de la puntuación F1 de los distintos modelos durante el entrenamiento para el mejor *fold* de la semilla con mejores resultados en generalización.

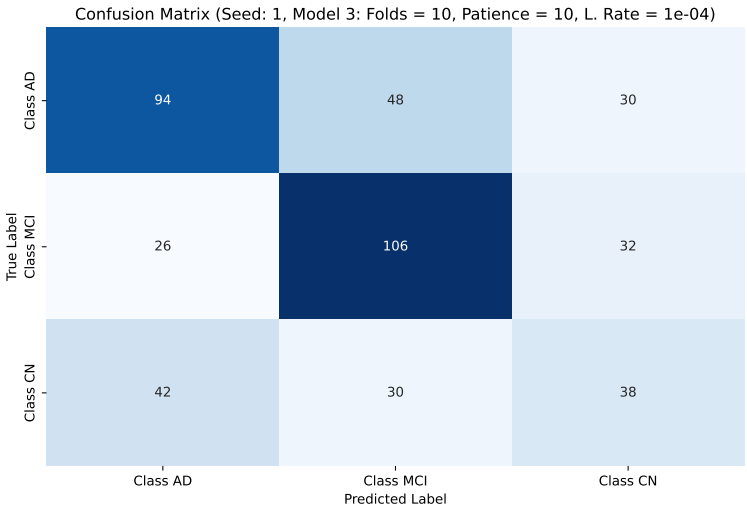

Figura 15.3: Matriz de confusión para el conjunto de generalización del modelo GoogleNet 10 de paciencia y 10^{-4} de coeficiente de entrenamiento.

rrectamente o que el valor del coeficiente de entrenamiento fuese demasiado alto provocando que, al momento de minimizar el error, "saltase" alrededor del mínimo.

Por último, en la tabla 15.1 se puede ver una comparación de los valores medios y su desviación típica para 5 semillas de las métricas *accuracy* y puntuación F1 en cada uno de los modelos para el conjunto de generalización.

Modelo	$\overline{\textbf{Accuracy}}$	$\sigma_{\textbf{Accuracy}}$	$\overline{\textbf{F1}}$	$\sigma_{\textbf{F1}}$
Alumno	0.3932	0.0865	0.1994	0.0655
El-Assy	0.4069	0.0824	0.2064	0.0449
GoogleNet	0.4864	0.0488	0.4053	0.0523

Tabla 15.1: Tabla comparativa de la media y desviación típica de las métricas *Accuracy* y F1 en *test* para 5 semillas.

15.2. Experimentos extra

Debido a esta observación, a continuación se van a realizar dos experimentos extra que probarán a ejecutar este mismo modelo cambiando el coeficiente de entrenamiento, disminuyéndolo, y después, el valor de la paciencia, aumentándolo.

Tras realizar estos experimentos hemos podido determinar que la variación de estos hiperparámetros puede llegar a mejorar los resultados como se puede ver en las siguientes gráficas (Figuras 15.4 y 15.5). En concreto se puede ver cómo al aumentar la paciencia, dejando que el modelo entrene durante más tiempo, los valores de *accuracy* y F1 aumentan. Aun así, si miramos sus matrices de confusión se puede apreciar que en una de ellas (Figura 15.6) aumenta la tendencia del modelo hacia clasificar en la clase central. Esto puede ser porque, debido al reducido número de épocas por el cambio de coeficiente de entrenamiento, no ha podido llegar a aprender suficiente. Mientras tanto, en la otra (Figura 15.7) la clasificación parece baste similar a la obtenida anteriormente (Figura 15.3). Si sumamos esto a

la tendencia alcista que se puede ver en las gráficas, podemos llegar a pensar que modelo todavía necesitaría más entrenamiento para poder converger correctamente.

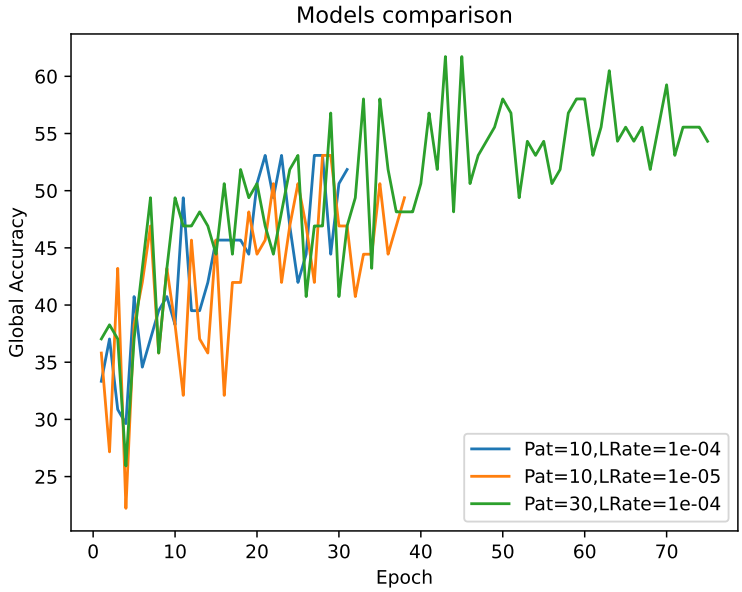

Figura 15.4: Comparación de la evolución de la *accuracy* durante el entrenamiento de los experimentos extra realizados.

15.3. Tiempo de ejecución de los modelos

Para terminar se ha guardado cuánto ha durado la ejecución de cada uno de los modelos. En la tabla 15.2 se puede ver la comparativa entre los modelos y los distintos tiempos que se han extraído. De cada modelo se ha extraído el tiempo de ejecución del entrenamiento (t) menor, mayor, medio y su desviación típica.

Como se puede ver en general el modelo que más tarda en realizar el entrenamiento es el modelo GoogleNet. Aunque estos resultados pueden no ser exactos debido a que al usar *early stopping* no todos los modelos entrenan durante el mismo número de épocas, tiene sentido que este modelo sea el más lento debido a la alta complejidad del mismo.

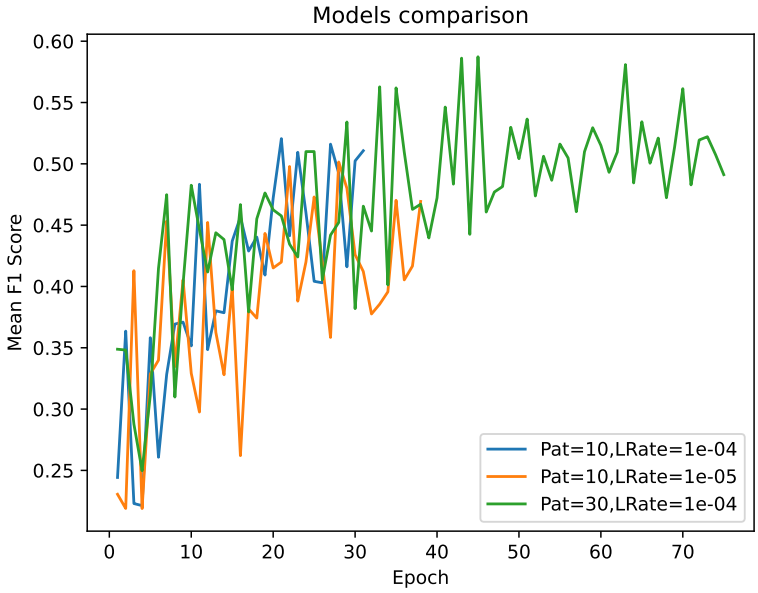

Figura 15.5: Comparación de la evolución de la puntuación F1 durante el entrenamiento de los experimentos extra realizados.

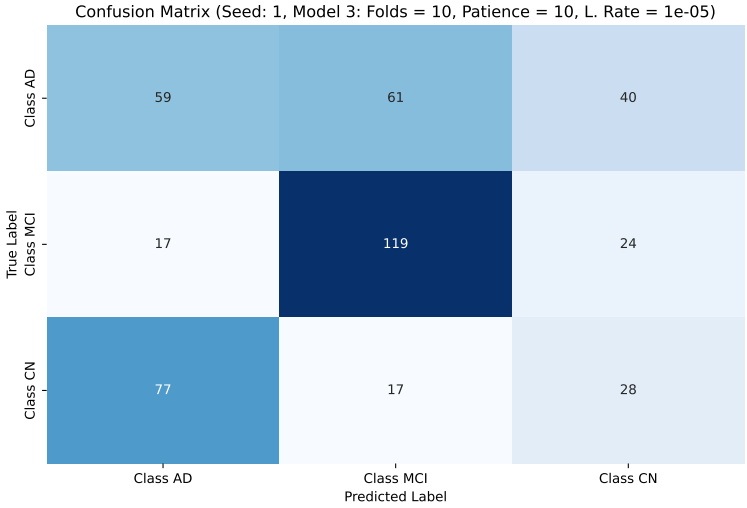

Figura 15.6: Matriz de confusión para el conjunto de generalización del modelo GoogleNet para 10 de paciencia y 10^{-5} de coeficiente de entrenamiento.

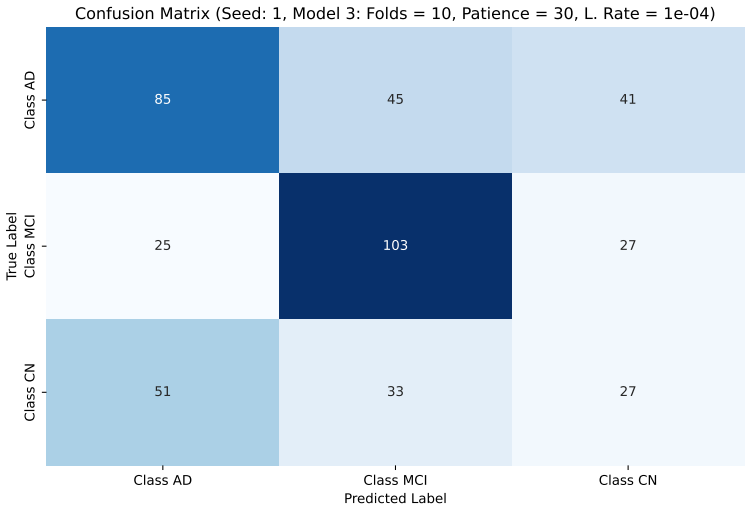

Figura 15.7: Matriz de confusión para el conjunto de generalización del modelo GoogleNet para 30 de paciencia y 10^{-4} de coeficiente de entrenamiento.

Modelo	**Menor t**	**Mayor t**	$\bar{\mathbf{t}}$	$\sigma_{\mathbf{t}}$
Alumno	4h 56m	8h 50m	7h 20m 48s	1h 24m 6s
El-Assy	9h 10m	12h 35m	11h 19m	1h 22m 40s
GoogleNet	5h 25m	12h 31m	8h 49m 36s	2h 34m 12s

Tabla 15.2: Tabla comparativa de los tiempos de entrenamiento para 5 semillas de los distintos modelos.

Parte V

Conclusiones

Capítulo 16

Conclusiones del autor

El problema que se planteó al inicio de este Trabajo de Fin de Grado fue la detección temprana de la enfermedad de Alzheimer.

Con este estudio hemos podido resolver este problema haciendo uso de redes neuronales convolucionales. Aunque este estudio no nos proporciona una solución perfecta al problema se puede ver cómo los resultados obtenidos son prometedores y, con algo más de investigación y experimentación, se pueden llegar a obtener unos modelos que puedan servir en entornos reales para resolver el problema propuesto.

Además, la realización de este Trabajo de Fin de Grado ha permitido entender la importancia de obtener un conjunto de datos bueno y equilibrado para entrenar modelos de red, así como la alta complejidad que conlleva trabajar con redes neuronales convolucionales y tridimensionales, tanto de desarrollo como de entrenamiento. Tambien, se ha podido aprender la importancia de realizar una separación estratificada de los datos, ya que una separación desproporcionada puede llevar a modelos mal entrenados que no sean capaces de clasificar correctamente.

En resumen, se puede decir que se ha cumplido el objetivo principal de este Trabajo de Fin de Grado, que se detalló en el capítulo 3 de este documento, ya que se ha podido realizar una red neuronal convolucional capaz de detectar indicios de la enfermedad de Alzheimer en pacientes que no muestran síntomas. Igualmente, para que se cumpla el objetivo principal

es necesario que se cumplan los objetivos específicos y didácticos.

16.1. Objetivos específicos alcanzados

Durante el desarrollo de este Trabajo y para poder cumplir el objetivo principal, se ha realizado exitosamente:

- El estudio de imágenes MRI y su preprocesamiento con Python.
- El estudio y análisis de la base de datos ADNI.
- El estudio y comprensión de la librería de Python PyTorch.
- El estudio y comprensión de la redes neuronales convoluciones tridimensionales.
- La obtención y preprocesamiento de los datos para el entrenamiento del modelo.
- La implementación del modelo de CNN 3D para la clasificación de las imágenes MRI.
- El análisis de los resultados obtenidos y la calibración de los hiperparámetros.
- La experimentación con el conjunto de generalización para confirmar los resultados obtenidos con el modelo entrenado.
- La interpretación de los análisis obtenidos.

16.2. Objetivos didácticos alcanzados

Además, se han podido adquirir y ampliar conocimientos sobre los siguientes temas:

- Redes neuronales convolucionales y sus tipos.

- Imágenes de resonancia magnética, en concreto Nifti, y cómo trabajar con ellas usando Python.
- La comprensión e interpretación de artículos científicos.
- La redacción de proyectos de Ingeniería Informática.
- El uso del sistema de composición de textos LaTeX[17] y el editor en la nube Overleaf [18].
- El lenguaje de programación Python y sus distintas librerías para análisis de datos y redes neuronales.
- El clúster del grupo de investigación AYRNA y la suite de HTCondor para la planificación de los experimentos.
- Mejorar un modelo CNN para perfeccionar los resultados obtenidos.
- Las distintas métricas para el análisis del rendimiento y la precisión del modelo en problemas de clasificación ordinal.

Capítulo 17

Futuras mejoras

Para finalizar con este Trabajo de Fin de Grado se van a realizar una serie de propuestas, para la realización de cambios en el futuro, con el objetivo de mejorar los resultados de la solución propuesta.

En primer lugar se propone el estudio de diferentes métodos de preprocesamiento de imágenes. Esto podría mejorar los resultados obtenidos, ya que se podría aislar mejor la información relevante de las imágenes, obteniendo así datos de mejor calidad.

Tambien se plantea probar más estructuras de red diferentes con el objetivo de encontrar modelos que mejoren los resultados obtenidos por el propuesto en este proyecto.

Otra mejora que se propone es el uso de redes neuronales que sean capaces de optimizar de manera automática los hiperparámetros de un modelo. Entre ellas, se sugiere el uso de las Redes Neuronales Evolutivas (*Evolutionary Neural Networks*). Estas redes son un conjunto de estrategias adaptativas inspiradas en los mecanismos de selección natural con el objetivo de optimizar un aspecto de la red neuronal artificial [66]. Se ajustan utilizando metaheurísticas. Por lo tanto, se pueden optimizar tanto los pesos de la red como los hiperparámetros. Esto nos permitiría obtener un modelo más óptimo del que podríamos llegar a obtener de forma manual.

Por último, se plantea la búsqueda y obtención de un número mayor de datos bien diferenciados entre ellos para mejorar el entrenamiento del

modelo y obtener resultados más precisos.

Bibliografía

[1] *About Dementia*. U.S. Centers for Disease Control y Prevention. 2019. URL: https://www.cdc.gov/aging/dementia/index.html. [En Línea. Última consulta: 08-07-2024].

[2] *Alzheimer's Disease and Related Dementias*. U.S. Centers for Disease Control y Prevention. 2020. URL: https://www.cdc.gov/aging/aginginfo/alzheimers.htm. [En Línea. Última consulta: 08-07-2024].

[3] Jill Rasmussen y Haya Langerman. «Alzheimer's Disease – Why We Need Early Diagnosis». En: *Degenerative Neurological and Neuromuscular Disease* (2019), págs. 123-130. DOI: 10.2147/DNND.S228939.

[4] A.P. Porsteinsson et al. «Diagnosis of Early Alzheimer's Disease: Clinical Practice in 2021». En: *The Journal of Prevention of Alzheimer's Disease* (2021), 371–386. DOI: 10.14283/jpad.2021.23.

[5] *Magnetic Resonance Imaging (MRI)*. U.S. National Institute of Biomedical Imaging y Bioengineering. URL: https://www.nibib.nih.gov/science-education/science-topics/magnetic-resonance-imaging-mri. [En Línea. Última consulta: 10-07-2024].

[6] Jerome Barakos et al. «Detection and Management of Amyloid-Related Imaging Abnormalities in Patients with Alzheimer's Disease Treated with Anti-Amyloid Beta Therapy». En: *The Journal of Prevention of Alzheimer's Disease* (2022), 211–220. DOI: 10.14283/jpad.2022.21.

[7] Hadeer A. Helaly, Mahmoud Badawy y Amira Y. Haikal. «Deep Learning Approach for Early Detection of Alzheimer's Disease». En: *Cog-*

nitive Computation (2022), 1711–1727. DOI: 10.1007/s12559-021-09946-2.

[8] Jiuxiang Gu et al. «Recent advances in convolutional neural networks». En: *Pattern recognition* (2028), págs. 354-377. DOI: 10.1016/j.patcog.2017.10.013.

[9] Pedro Antonio Gutiérrez et al. «Ordinal regression methods: survey and experimental study». En: *IEEE Transactions on Knowledge and Data Engineering* (2015), págs. 127-146. DOI: 10.1109/TKDE.2015.2457911.

[10] *The Alzheimer's Disease Neuroimaging Initiative*. URL: https://adni.loni.usc.edu/about/.

[11] *AYRNA*. URL: https://www.uco.es/ayrna/.

[12] *Python*. URL: https://www.python.org/.

[13] *PyTorch*. URL: https://pytorch.org/.

[14] *Matplotlib*. URL: https://matplotlib.org/.

[15] *NiBabel*. URL: https://nipy.org/nibabel/.

[16] Shivam Bansal. *3D Convolutions : Understanding + Use Case*. Kaggle. 2019. URL: https://www.kaggle.com/code/shivamb/3d-convolutions-understanding-use-case. [En Línea. Última consulta: 10-07-2024].

[17] *LaTeX*. URL: https://www.latex-project.org/.

[18] *Overleaf*. URL: https://es.overleaf.com/.

[19] *Pandas*. URL: https://pandas.pydata.org/.

[20] *Condor*. URL: https://htcondor.org/.

[21] J. Saxton et al. «Preclinical Alzheimer disease». En: *Neurology* (2004). DOI: 10.1212/01.WNL.0000147470.58328.50.

[22] A. J. Mitchell y M. Shiri-Feshki. «Rate of progression of mild cognitive impairment to dementia – meta-analysis of 41 robust inception cohort studies». En: *Acta Psychiatrica Scandinavica* (2009). DOI: 10.1111/acp.2009.119.issue-4.

[23] Dennis J Selkoe y John Hardy. «The amyloid hypothesis of Alzheimer's disease at 25 years». En: *EMBO Molecular Medicine* (2016). DOI: 10.15252/emmm.201606210.

[24] Christopher Carlson et al. «Amyloid-related imaging abnormalities from trials of solanezumab for Alzheimer's disease». En: *Diagnosis, Assessment Disease Monitoring* (2016). DOI: 10.1016/j.dadm.2016.02.004.

[25] M.K. Gurucharan. *Basic CNN Architecture: Explaining 5 Layers of Convolutional Neural Network*. upGrad. 2024. URL: https://www.upgrad.com/blog/basic-cnn-architecture/. [En Línea. Última consulta: 13-07-2024].

[26] Keiron O'Shea y Ryan Nash. *An Introduction to Convolutional Neural Networks*. 2015. DOI: 10.48550/arXiv.1511.08458.

[27] *Convolución*. URL: https://es.wikipedia.org/wiki/Convoluci%C3%B3n.

[28] Hanh Vu, Hyun-Chul Kim y Jong-Hwan Lee. «3D convolutional neural network for feature extraction and classification of fMRI volumes». En: *2018 International Workshop on Pattern Recognition in Neuroimaging (PRNI)* (2018). DOI: 10.1109/PRNI.2018.8423964.

[29] Samir S. Yadav y Shivajirao M. Jadhav. «Deep convolutional neural network based medical image classification for disease diagnosis». En: *Journal of Big Data* (2019). DOI: 10.1186/s40537-019-0276-2.

[30] Sergio de Lopez et al. *Detección de acciones a partir de información de profundidad mediante redes neuronales convolucionales*. 2019. URL: https://www.researchgate.net/publication/340023601_Deteccion_de_acciones_a_partir_de_informacion_de_profundidad_mediante_redes_neuronales_convolucionales.

[31] F. Lekien y J. Marsden. «Tricubic interpolation in three dimensions». En: *Internation Journal for Numerical Methods in Engineering* (2005). DOI: 10.1002/nme.1296.

[32] *Click*. URL: https://click.palletsprojects.com/en/8.1.x/.

[33] *CSV. Valores separados por comas.* URL: https://es.wikipedia.org/wiki/Valores_separados_por_comas.

[34] *Vi.* URL: https://ex-vi.sourceforge.net/.

[35] *Nano.* URL: https://www.nano-editor.org/.

[36] *SSH.* URL: https://es.wikipedia.org/wiki/Secure_Shell.

[37] *Bash.* URL: https://www.gnu.org/software/bash/.

[38] *Google Scholar.* URL: https://scholar.google.es/.

[39] *Github.* URL: https://github.com/.

[40] *ADNI, Repositorio de bsearchinger.* URL: https://github.com/bsearchinger/ADNI.

[41] *brain202, Repositorio de vkola-lab.* URL: https://github.com/vkola-lab/brain2020.

[42] *Otsu's Thresholding.* URL: https://en.wikipedia.org/wiki/Otsu%27s_method. [En Línea. Última consulta: 22-07-2024].

[43] *Entropía.* URL: https://es.mathworks.com/help/images/ref/entropy.html. [En Línea. Última consulta: 22-07-2024].

[44] *Varianza.* URL: https://es.wikipedia.org/wiki/Varianza. [En Línea. Última consulta: 22-07-2024].

[45] *Sobel Edge Detector.* URL: https://homepages.inf.ed.ac.uk/rbf/HIPR2/sobel.htm. [En Línea. Última consulta: 22-07-2024].

[46] *Skewness.* URL: https://en.wikipedia.org/wiki/Skewness. [En Línea. Última consulta: 22-07-2024].

[47] *TorchIO.* URL: https://torchio.readthedocs.io/.

[48] A.M. El-Assy et al. «A novel CNN architecture for accurate early detection and classification of Alzheimer's disease using MRI data». En: *Scientific Reports* (2024). DOI: 10.1038/s41598-024-53733-6.

[49] Christian Szegedy et al. «Going deeper with convolutions». En: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* 2015. DOI: 10.1109/CVPR.2015.7298594.

[50] *AlexNet*. URL: https://en.wikipedia.org/wiki/AlexNet. [En Línea. Última consulta: 26-07-2024].

[51] *The MNIST Database of Handwritten Digits*. URL: https://yann.lecun.com/exdb/mnist/.

[52] Francis Benistant. *Training a Simple Autoencoder on the MNIST Dataset: A Hand-on-tutorial*. 2023. URL: https://2020machinelearning.medium.com/training-a-simple-autoencoder-on-the-mnist-dataset-a-hand-on-tutorial-46d8a024604c. [En Línea. Última consulta: 25-07-2024].

[53] Isra Sheikh. *Understanding Cross-Entropy Loss and Its Role in Classification Problems*. 2023. URL: https://medium.com/@l228104/understanding-cross-entropy-loss-and-its-role-in-classification-problems-d2550f2caad5. [En Línea. Última consulta: 25-07-2024].

[54] Kurtis Pykes. *Cross-Entropy Loss Function in Machine Learning: Enhancing Model Accuracy*. 2024. URL: https://www.datacamp.com/tutorial/the-cross-entropy-loss-function-in-machine-learning. [En Línea. Última consulta: 25-07-2024].

[55] *Hiperparámetro (aprendizaje automático)*. URL: https://es.wikipedia.org/wiki/Hiperpar%C3%A1metro_(aprendizaje_autom%C3%A1tico). [En Línea. Última consulta: 25-07-2024].

[56] Vinod Nair y Geoffrey E Hinton. «Rectified linear units improve restricted boltzmann machines». En: *Proceedings of the 27th international conference on machine learning (ICML-10)*. 2010, págs. 807-814.

[57] *LeNet*. URL: https://en.wikipedia.org/wiki/LeNet. [En Línea. Última consulta: 26-07-2024].

[58] Nitish Srivastava et al. «Dropout: a simple way to prevent neural networks from overfitting». En: *The journal of machine learning research* (2014).

[59] Harsh Yadav. *Dropout in Neural Networks*. 2022. URL: https://towardsdatascience.com/dropout-in-neural-networks-47a162d621d9. [En Línea. Última consulta: 26-07-2024].

[60] *Adam*. Interactive Chaos. URL: https://interactivechaos.com/es/manual/tutorial-de-machine-learning/adam. [En Línea. Última consulta: 15-08-2024].

[61] Aina Temiloluwa. *Understanding Logits: A Fundamental Concept in Machine Learning*. 2024. URL: https://www.linkedin.com/pulse/understanding-logits-fundamental-concept-machine-aina-temiloluwa-7d3ff/. [En Línea. Última consulta: 24-07-2024].

[62] *Función SoftMax*. URL: https://es.wikipedia.org/wiki/Funci%C3%B3n_SoftMax.

[63] *Accuracy, precision, and recall in multi-class classification*. URL: https://www.evidentlyai.com/classification-metrics/multi-class-metrics. [En Línea. Última consulta: 24-07-2024].

[64] *Evaluation Metrics For Multi-class Classification*. Kaggle. URL: https://www.evidentlyai.com/classification-metrics/multi-class-metrics. [En Línea. Última consulta: 24-07-2024].

[65] *Precision-Recall Curve — ML*. Geeks For Geeks. URL: https://www.geeksforgeeks.org/precision-recall-curve-ml/. [En Línea. Última consulta: 15-08-2024].

[66] Fernando Javier Aguilar Canto. «Redes neuronales evolutivas con modelos de Lotka-Volterra». En: *Research in Computing Science* (2020).

[67] *Revo Uninstaller*. URL: https://www.revouninstaller.com/es/.

Parte VI

Apéndices

Apéndice A

Manual de usuario

Este documento contiene el manual de usuario del código realizado en Python para el Trabajo de Fin de Grado ***Clasificación ordinal mediante redes neuronales convolucionales para el diagnóstico de la enfermedad de Alzheimer.***

A.1. Descripción del producto

El producto que se va definir en este manual se usa mediante una interfaz en línea de comandos. Esto nos permite ejecutar tanto las fases de entrenamiento y generalización como el modo de predicción, permitiendo seleccionar algunos de los hiperparámetros de entrenamiento o usar modelos previamente entrenados para estas ejecuciones.

Básicamente, esta interfaz nos permite entrenar el modelo o usar los modelos ya entrenados, permitiendo la especificación de hiperparámetros y la obtención resultados.

Entre las características del producto desarrollado, podemos encontrar las siguientes:

- Desarrollado usando Python como lenguaje de programación.
- Gran control sobre la ejecución mediante su interfaz en línea de comandos.

- Ejecución y entrenamiento de un modelo de red neuronal convolucional.
- Generalización de los resultados devueltos por el modelo.
- Extracción de los resultados a ficheros externos.

A.2. Requisitos del sistema

Como se ha comentado anteriormente el producto se ha desarrollado usando el lenguaje de programación Python, el cual es un lenguaje multiplataforma. Esto permite que la ejecución de este producto sea posible en cualquier plataforma siempre y cuando tenga incorporado Python junto con las librerías que se especificarán más adelante. Aun así, ya que el modelo es bastante exigente computacionalmente hablando, es necesario que la máquina donde se ejecute cumpla los siguientes requisitos:

A.2.1. Requisitos mínimos

- Tarjeta gráfica NVIDIA con soporte para CUDA 12.0 o superior y 25GB de memoria de vídeo.
- 128GB de RAM.
- 20GB de espacio disponible en el disco duro para almacenar las imágenes.

A.2.2. Requisitos recomendados

- Tarjeta gráfica NVIDIA con soporte para CUDA 12.0 o superior y 50GB de memoria de vídeo.
- 192GB de RAM.
- 30GB de espacio disponible en el disco duro para almacenar las imágenes y los modelos que se generen.

A.2.3. Requisitos software

- Cualquier sistema operativo que permita la instalación de Python y sus librerías (*GNU/Linux, Windows o Mac OS X*). Además deberá tener soporte para CUDA 12.0 o superior de NVIDIA.

A.3. Instalación del software

A.3.1. Instalación de Python

Windows

Para instalar Python en Windows hay que descargar el instalador de la página web oficial (`https://www.python.org/`) y ejecutarlo marcando la opción de añadir Python al PATH para facilitar el uso de Python a través de la línea de comandos.

Linux

En la mayoría de instalaciones de Linux, Python es instalado junto al sistema operativo por lo que no haría falta instalarlo. En caso de que no esté instalado por defecto, tendríamos que instalarlo usando el gestor de paquetes que use la distribución de Linux que se este usando. En el caso de distribuciones basadas en Debian, el gestor de paquetes más utilizado es apt. Usando este gestor, Python se instalaría de la siguiente forma:

```
sudo apt install python3
```

Junto con Python, es necesario instalar su gestor de paquetes Pip:

```
sudo apt install python3-pip
```

Mac OS X

Igual que pasa con Linux, Python suele venir instalado por defecto con Mac OS X. En caso de no estarlo, podemos usar su gestor de paquetes Homebrew para instalarlo:

```
brew install python
```

A.3.2. Instalación de librerías

Este proyecto usa numerosas librerías para su correcto funcionamiento. Con el objetivo de facilitar el proceso de instalación, se ha creado un archivo `requeriments.txt` que se puede usar junto con pip para instalar todas las librerías con sus versiones correspondientes. Es recomendable que se cree un entorno virtual para evitar incompatibilidades de versiones o modificaciones de las que tenga el usuario actualmente instaladas en su máquina. A continuación se explica como crear el entorno virtual e instalar las librerías dentro de él:

Windows

1. Crea el entorno virtual desde la consola dentro de la carpeta del proyecto:
   `python -m venv <nombre_del_entorno_virtual>`

2. Activa el entorno virtual:
   `venv\Scripts\activate`

3. Instala las libererías:
   `pip install -r requirements.txt`

4. Para desactivar el entorno virtual, escribe el siguiente comando en la consola:
   `deactivate`

Linux

1. Instala el módulo venv si es necesario usando el gestor de paquetes que se desee (sudo en este caso):
   `sudo apt install python3-venv`

2. Crea el entorno virtual desde la consola dentro de la carpeta del proyecto:
   `python -m venv <nombre_del_entorno_virtual>`

3. Activa el entorno virtual:
   `source venv/bin/activate`

4. Instala las librerías:
   `pip install -r requirements.txt`

5. Para desactivar el entorno virtual, escribe el siguiente comando en la consola:
   `deactivate`

Mac OS X

1. Crea el entorno virtual desde la consola dentro de la carpeta del proyecto:
   `python -m venv <nombre_del_entorno_virtual>`

2. Activa el entorno virtual:
   `source venv/bin/activate`

3. Instala las librerías:
   `pip install -r requirements.txt`

4. Para desactivar el entorno virtual, escribe el siguiente comando en la consola:
   `deactivate`

A.4. Desinstalación del Software

La desinstalación de este software es baste sencilla, ya que al no requerir ser instalado, basta con eliminar todos los archivos relacionados con el mismo. En caso de haber instalado Python y sus librerías, podemos desinstalarlos como se explica a continuación.

A.4.1. Desinstalación de Python

Windows

Bastaría con eliminar el programa haciendo uso de la herramienta de desinstalación de Windows que se encuentra en: Inicio > Configuración > Aplicaciones > Aplicaciones & funciones. Tambien se pueden usar aplicaciones de terceros como [67] que se encargar de escanear tu computadora para eliminar cualquier archivo que se haya quedado tras la desinstalación.

Linux y Mac OS X

En estos sistemas operativos Python es una herramienta esencial para su funcionamiento por lo que su desinstalación completa no es posible.

A.4.2. Desinstalación de Librerías

Como hemos usado entornos virtuales en la instalación de librerías, su desinstalación es muy sencilla. Basta con eliminar la carpeta del entorno virtual (`venv`).

A.5. Ejecución del software

La ejecución del software se realizará desde la terminal y será igual para todos los sistemas operativos. El comando para realizar la ejecución, y que se deberá hacer desde dentro del *root* del proyecto, es:

```
python cnn.py [parámetro [valor]]
```

Si se realiza una ejecución sin seleccionar ningún modo (entrenamiento, generalización o predicción), el programa mostrará el mensaje de ayuda para indicar al usuario los parámetros disponibles y cortará la ejecución del mismo.

A continuación se explican los diferentes parámetros que acepta nuestro modelo.

A.5.1. Parámetros de ejecución

- `-s|--seed INT`**:** semilla aleatoria que se usa durante distintos procesos para facilitar la replicación de los resultados.
- `-d|--date INT`**:** fecha de ejecución del programa para identificar los resultados. Por defecto usa la fecha actual y el formato `yyyymmddHHMM`.
- `-D|--data_path TEXT`**:** ruta al archivo CSV que contiene la ruta a las imágenes y sus etiquetas.
- `-T|--train`**:** *flag* que indica al programa que ejecute la fase de entrenamiento.
- `-M|--selected_model INT`**:** indica qué modelo se quiere usar para el entrenamiento de entre los 3 disponibles.
- `-m|--trained_model_path TEXT`**:** ruta al archivo que contiene un modelo previamente entrenado que será usado en las fases de generalización y validación, siempre y cuando no se ejecute la fase de entrenamiento.
- `-t|--test`**:** *flag* que indica al programa que ejecute la fase de generalización.
- `-p|--predict_path TEXT`**:** permite especificar la ruta a una imagen sobre la que se quiere realizar una predicción. Cuando se incluye indica a su vez al programa que ejecute la fase de predicción.

- `-b|--batch_size INT`: tamaño del *batch* que se va a usar durante el entrenamiento y la generalización del modelo.
- `-f|--num_folds INT`: número de *folds* que se van a usar en la fase de entrenamiento.
- `-w|--patience INT`: número de épocas que va a esperar el modelo antes de parar por *early stopping*.
- `-l|--learning_rate FLOAT`: velocidad a la que el modelo es capaz de "aprender".
- `-r|--results_path TEXT`: ruta donde se almacenarán los resultados del modelo. Si no existe se creará.

A.6. Formato de las salidas

A continuación se va a explicar el formato de la salida de los diferentes modos:

A.6.1. Salidas del entrenamiento

Las salidas de entrenamiento se guardan de dos formas:

- Guardado en archivos dentro de la carpeta `training` en la ruta especificada por el usuario con el parámetro `--results_path`. Se saca el valor de cada una de las métricas en cada una de las épocas de cada *fold*, los valores de las métricas de la época con menor *loss error* para cada *fold* y la media de estos valores.
- Impresión por línea de comandos.
    - Al inicio del entrenamiento se imprimen los hiperparámetros del modelo para indicar el inicio. Ejemplo:

```
Hiperparametros:

 Batch Size: 4
 Learning Rate: 0.0001
 Number of Epochs: 1000
 Number of Folds: 10
 Patience: 10
 Model: GoogleNet



----------------------------------------------------
```

- Al inicio de cada fold se imprime:
  `Fold {fold + 1}/{num_folds}`
- En cada *batch* de cada época se imprime el *running loss*:
  `Epoch {epoch + 1}, Batch {i + 1}, Loss: {running_loss / 10}`
- Al final de cada época se imprime el *validation loss* y la *accuracy* global y la media de la *accuracy*, *precision*, *recall* y *f1_score* de cada clase en validación:
  `Epoch {epoch + 1}, Validation Loss: {val_loss}, Global Accuracy: {global_accuracy}%`
  `Mean Accuracy: {accuracy}%`
  `Mean Accuracy: {accuracy}%`
  `Mean Precision: {precision}`
  `Mean Recall: {recall}`
  `Mean F1 Score: {f1_score}\n`

Para los resultados que se almacenan en disco, hay dos archivos para entrenamiento

```
folds-{fecha}_{semilla}-{modelo}_{fold}_{paciencia}_
{learning rate}.csv
```

`averages-{fecha}_{semilla}-{modelo}_{fold}_{paciencia}_{learning rate}.csv`

Archivo `folds`

El archivo `folds` contiene los valores por época y la mejor época de cada *fold*. Cada época está identificada con su número de época y a la mejor se le ha puesto época 1001, ya que el modelo entrena un máximo de 1000 épocas y lo hace fácilmente identificable.

El archivo cuenta con las siguientes columnas:

`Fold`

`Epoch`

`Validation Loss`

`Global Accuracy`

`Mean Accuracy`

`Mean Precision`

`Mean Recall`

`Mean F1 Score`

Archivo `averages`

El archivo `averages` La media de la mejor época de cada *fold* para cada una de las métricas.

El archivo cuanta con las siguientes columnas:

`Validation Loss`

`Global Accuracy`

`Mean Accuracy`

```
Mean Precision

Mean Recall

Mean F1 Score
```

A.6.2. Salidas del generalización

Las salidas de generalización también se guardan de dos formas:

- Guardado en archivos dentro de la carpeta `testing` en la ruta especificada por el usuario con el parámetro `--results_path`. Se saca el valor de cada una de las métricas en cada clase, los valores medios de cada una y la matriz de confusión para cada clase.
- Impresión por línea de comandos.
  - Al inicio se imprime una línea para avisar que ha empezado la fase de generalización.
  - Al final se escriben todos los resultados.
  - Ejemplo:

```
Testing the model...


Confusion Matrices:

Class 0:
[[30 53]
 [38 83]]

Class 1:
[[33 55]
 [35 81]]
```

```
Class 2:
[[ 15  18]
 [ 53 118]]

Test Loss: 1.091213615501628 | Test Global Accuracy:
38.23529411764706%


Accuracy by Class:

Class 0 Accuracy: 14.705882352941178%
Class 1 Accuracy: 16.176470588235293%
Class 2 Accuracy: 7.352941176470589%

Mean Accuracy: 12.745098039215685%


Precision by Class:

Class 0 Precision: 0.3614457831325301
Class 1 Precision: 0.375
Class 2 Precision: 0.45454545454545453

Mean Precision: 0.3969970792259949


Recall by Class:

Class 0 Recall: 0.4411764705882353
Class 1 Recall: 0.4852941176470588
Class 2 Recall: 0.22058823529411764

Mean Recall: 0.38235294117647056
```

```
F1 Score by Class:

Class 0 F1 Score: 0.3973509933774834
Class 1 F1 Score: 0.423076923076923
Class 2 F1 Score: 0.29702970297029707

Mean F1 Score: 0.37248587314156784
```

Para los resultados que se almacenan en archivos, hay dos archivos para entrenamiento

`folds-{fecha}_{semilla}-{modelo}_{fold}_{paciencia}_{learning rate}.csv`

`averages-{fecha}_{semilla}-{modelo}_{fold}_{paciencia}_{learning rate}.csv`

Las salidas de generalización se guardan dentro de la carpeta `testing` en la ruta especificada por el usuario con el parámetro `--results_path`. Para generalización sacamos el valor de cada una de las métricas en cada clase, los valores medios de cada una y la matriz de confusión para cada clase.

Esto se guarda en tres archivos distintos:

`by_class-{fecha}_{semilla}-{modelo}_{fold}_{paciencia}_{learning rate}.csv`

`averages-{fecha}_{semilla}-{modelo}_{fold}_{paciencia}_{learning rate}.csv`

`cms-{fecha}_{semilla}-{modelo}_{fold}_{paciencia}_{learning rate}.txt`

Archivo `by_class`

El archivo `by_class` contiene los valores de cada una de las métricas para cada clase.

El archivo cuenta con las siguientes columnas:

`Class`

`Accuracy`

`Precision`

`Recall`

`F1 Score`

Archivo `averages`

El archivo `averages` contiene la media de los valores de las métricas.

El archivo cuenta con las siguientes columnas:

`Accuracy`

`Precision`

`Recall`

`F1 Score`

Archivo `cms`

El archivo `cms` contiene las matrices de confusión de cada clase.

El archivo cms tiene el siguiente formato:

```
Class 0:
[[ 29  21]
 [ 39 115]]
```

```
Class 1:
[[ 19  34]
 [ 49 102]]

Class 2:
[[51 50]
 [17 86]]
```

A.6.3. Salidas del predicción

Los resultados del modo de predicción solo se imprimen por línea de comandos. Primero imprime una línea indicando la imagen sobre la que se ha hecho la predicción y después el vector de probabilidades de pertenencia a las clases 0, 1 o 2. Por ejemplo:

```
Prediction for I31107.nii is:
[0.375457763671875, 0.3740262985229492, 0.250515878200531]
```

Apéndice B

Manual de código

En este Manual de Código del Trabajo Fin de Grado ***Clasificación ordinal mediante redes neuronales convolucionales para el diagnóstico de la enfermedad de Alzheimer***, se incluye una descripción detallada de todos los elementos que componen el sistema desarrollado. La función de este documento es la explicación del *software* desarrollado para mejorar el entendimiento del mismo y servir de apoyo en el desarrollo de futuras mejoras.

Es importante remarcar que este manual de código es solo un apoyo. Para entender por completo el funcionamiento se requiere leer el Manual Técnico.

Como lenguaje de programación se ha usado Python, ya que es un lenguaje potente como una amplia variedad de módulos dedicados al desarrollo de redes neuronales y análisis de datos.

En este manual se explicarán cada una de las funciones de la aplicación, centrándose en la explicación y objetivo de cada una de ellas.

B.1. Características de la implementación

B.1.1. Notación de la implementación

Con el objetivo de realizar un código limpio y legible se ha tratado de seguir algunas pautas durante el desarrollo:

- Los nombres de las variables y funciones deben tratar de ser explicativos de su contenido.
- Los nombres clases deberán seguir la convención *Pascal Case*. Esto es que todas las palabras del nombre comiencen con mayúscula y vayan todo junto.
- Los nombres de las variables y funciones deberán seguir la convención *Snake Case*. Esto es que todas las palabras vayan en minúscula y separadas con guión bajo (_).

B.1.2. Estructura de la documentación

En esta documentación se irá describiendo detalladamente la función de cada uno de los ficheros de código del programa, empezando por aquellos que se encuentren en la raíz del proyecto y continuando por los distintos directorios, siempre siguiente el orden alfabético.

Para cada un de los ficheros se explicará su cometido y cada una de las variables que contiene.

B.2. Documentación del código

En este capítulo se va a explicar en detalle como se han implementado cada una de las funciones que se han desarrollado para este Trabajo de Fin de Grado.

B.2.1. Fichero cnn.py

Este fichero es la entrada al programa. Recibe los parámetros y llama a las distintas funciones de los modos del programa.

Especificación de la función split_labels

Función: *split_labels*	
Esta función divide el fichero de las etiquetas en dos para entrenamiento y generalización.	
Parámetros	
- data_path	Incluye la ruta al fichero de etiquetas.
- seed	Semilla que se usara en la división de datos para facilitar su replicación.

Tabla B.1: Especificación de los parámetros de la función split_labels

Especificación de la función main

Función: *main*	
Esta es la función principal del programa. Recibe los parámetros que indique el usuario y hace la llama a las funciones necesarias.	
Parámetros	
- seed	Semilla que se usará para facilitar la replicación de las ejecuciones.

- date	Fecha de ejecución del programa. Se usa para identificar las distintas ejecuciones.
- data_path	Incluye la ruta al fichero de etiquetas.
- train	Es una *flag* y se usa para indicar si se quiere realizar entrenamiento.
- selected_model	Sirve para indicar cuál de los modelos propuestos se va a usar en el entrenamiento.
- trained_model_path	Sirve para seleccionar el modelo previamente entrenado que se quiere usar en caso de no realizar entrenamiento.
- test	Es una *flag* y se usa para indicar si se quiere realizar la fase de generalización.
- predict_path	Incluye la ruta a la imagen sobre la que se quiere realizar la predicción.
- batch_size	Permite indicar el tamaño de *batch* que se quiere usar.
- num_folds	Permite indicar el número de *folds* que quiere usar para el entrenamiento.
- patience	Permite indicar el número de epocas que se quiere esperar antes de realizar *early stopping*.
- learning_rate	Permite indicar el coeficiente de entrenamiento que se quiere usar.

- results_path	Incluye la ruta donde se almacenarán los ficheros de resultados. Se creará en caso de no existir.

Tabla B.2: Especificación de los parámetros de la función main

Fichero cnn.py

```
#!/home/i92cocom/miniconda3/envs/tfg/bin/python3

import click
import torch
import os

import pandas as pd

from datetime import datetime
from sklearn.model_selection import
    train_test_split

# Self-Created Libreries
from cnn3d.training import train_network
from cnn3d.loading import load_model
from cnn3d.testing import generalization_test
from cnn3d.predicting import predict


def split_labels(data_path, seed):

    # Load the base CSV file
    labels_df = pd.read_csv(data_path, header=None)
```

```
    # Split the data into training (80%) and
testing (20%)
    train_df, test_df = train_test_split(labels_df,
    test_size=0.2, random_state=seed, stratify=
labels_df[1])

    base_folder = '/'.join(data_path.split('/')
[:-1])

    train_labels_path = os.path.join(base_folder, f
'train_labels_{seed}.csv')
    test_labels_path = os.path.join(base_folder, f'
test_labels_{seed}.csv')

    # Check if the file exists
    if os.path.isfile(train_labels_path):
        try:
            # Delete the file
            os.remove(train_labels_path)
        except Exception as e:
            print(f"An error occurred while trying
to delete the file: {e}")

    # Check if the file exists
    if os.path.isfile(test_labels_path):
        try:
            # Delete the file
            os.remove(test_labels_path)
        except Exception as e:
            print(f"An error occurred while trying
to delete the file: {e}")

    # Save the resulting dataframes to CSV files
```

```
    train_df.to_csv(train_labels_path, index=False,
    header=False)
    test_df.to_csv(test_labels_path, index=False,
    header=False)

@click.command()
@click.option('--seed', '-s', default=1, required=
    False, help=u'Seed to be used.')
@click.option('--date', '-d', default=datetime.now
    ().strftime("%y%m%d%H%M"), required=False, help=u
    'Date of execution.')
@click.option('--data_path', '-D', default='data/
    labels.csv', required=False, help="CSV with all
    the labels for the images combined")
@click.option('--train', '-T', is_flag=True, help="
    Training Flag")
@click.option('--selected_model', '-M', default=1,
    required=False, help="Model selected to use on
    training")
@click.option('--trained_model_path', '-m', default
    ='cnn3d_model.pth', required=False, help="
    Pretained model to use in case of no training")
@click.option('--test', '-t', is_flag=True, help="
    Testing Flag")
@click.option('--predict_path', '-p', default=None,
     required=False, help="Path of the image to
    predict")
@click.option('--batch_size', '-b', default=4,
    required=False, help="Size of the batches")
@click.option('--num_folds', '-f', default=5,
    required=False, help="Number of folds")
@click.option('--patience', '-w', default=10,
    required=False, help="Number of epochs to wait
```

```
    for improvement before stopping")
@click.option('--learning_rate', '-l', default
    =0.0001, required=False, help="Learning rate")
@click.option('--results_path', '-r', default='
    results', required=False, help='Path where the
    results are stored')
@click.pass_context
def main(ctx, seed, date, data_path, train,
    selected_model, trained_model_path, test,
    predict_path, batch_size, num_folds, patience,
    learning_rate, results_path):

    if not train and not test and predict_path ==
    None:
        click.echo(ctx.get_help())
        ctx.exit()

    file_diferentiator = f'{date}_{seed}-{
    selected_model}_{num_folds}_{patience}_{
    learning_rate}'

    print(f'Start Date: {datetime.now().strftime("[
    %d/%m/%y  %H:%M ]")}')

    # Use GPU if available
    device = torch.device('cuda' if torch.cuda.
    is_available() else 'cpu')

    model = None

    split_labels(data_path, seed)

    selected_model = int(selected_model)
    batch_size = int(batch_size)
```

```
    num_folds = int(num_folds)
    patience = int(patience)
    learning_rate = float(learning_rate)

    print(f'Start Training: {datetime.now().
strftime("[ %d/%m/%y  %H:%M ]")}')
    if train:
        model = train_network(seed, date, device,
selected_model, batch_size, num_folds, patience,
learning_rate, file_diferentiator, results_path)
        print(f'End Training: {datetime.now().
strftime("[ %d/%m/%y  %H:%M ]")}')

    if test:
        if not train:
            model = load_model(trained_model_path,
device)

        generalization_test(model, device, seed,
batch_size, file_diferentiator, results_path)

    if not predict_path == None:
        if not train and not test:
            model = load_model(trained_model_path,
device)

        probabilities = predict(model, device,
predict_path)
        # prediction_label = ''
        # if prediction == 0: #AD
        #     prediction_label = 'AD'
        # elif prediction == 1:
        #     prediction_label = 'MCI'
        # elif prediction == 2:
```

```
        #     prediction_label = 'CN'

        print(f"Prediction for {predict_path.split
    ('/')[-1]} is:\n{probabilities}")
116
     print(f'End Date: {datetime.now().strftime("[ %
    d/%m/%y  %H:%M ]")}')


if __name__ == "__main__":
121     main()
```

Código B.1: Archivo cnn.py

B.2.2. Fichero cnn3d/loading.py

Este fichero contiene la función que carga el modelo preentrenado especificado por el usuario.

Especificación de la función load_model

Función: *load_model*	
Esta función carga un modelo preentrenado y lo devuelve.	
Parámetros	
- model_path	Incluye la ruta al modelo preentrenado.
- device	CPU/GPU donde se cargará el modelo.
Salida	
- model	Modelo cargado.

Tabla B.3: Especificación de los parámetros de la función load_model

Fichero cnn3d/loading.py

```
import torch
from cnn3d.models.cnn3d_own import Cnn3DOwn
from cnn3d.models.cnn3d_elassy import Cnn3DElAssy
from cnn3d.models.cnn3d_googlenet import Cnn3DGoogleNet


# Function to load the trained model
def load_model(model_path, device):

    selected_model = int(model_path.split('/')[-1].split('-')[1].split('_')[0])

    if selected_model == 1:
        model = Cnn3DOwn().to(device)
    elif selected_model == 2:
        model = Cnn3DElAssy().to(device)
    elif selected_model == 3:
        model = Cnn3DGoogleNet().to(device)

    model.load_state_dict(torch.load(model_path, map_location=device))
    model.eval()
    return model
```

Código B.2: Archivo cnn3d/loading.py

B.2.3. Fichero cnn3d/predicting.py

Este fichero contiene las funciones necesarias para realizar una predicción.

Especificación de la función predict_image

Función: *predict_image*	
Esta función es la encargada de realizar las predicciones sobre la imagen.	
Parámetros	
- model	Modelo previamente cargado/entrenado.
- image_tensor	Imagen que ha sido previamente procesada y cargada en *tensor*.
Salida	
- probabilities	Lista con las probabilidades de pertenencia a cada una de las clases.

Tabla B.4: Especificación de los parámetros y salida de la función predict_image

Especificación de la función predict

Función: *predict*
Esta función llama a las funciones de preprocesado de imagen, la carga en *tensor* y llama a la función que realiza las predicciones.

Parámetros	
- model	Modelo previamente cargado/entrenado.
- device	CPU/GPU donde se cargará el modelo.
- image_path	Ruta donde se encuentra la imágen sobre la que se quiere realizar la predicción.
Salida	
- probabilities	Lista con las probabilidades de pertenencia a cada una de las clases.

Tabla B.5: Especificación de los parámetros y salida de la función predict

Fichero cnn3d/predicting.py

```
import torch

import numpy as np

from tools.preprocess_image import preprocess_image

# Function to make a prediction
def predict_image(model, image_tensor):
    with torch.no_grad():
        outputs = model(image_tensor)
        probabilities = torch.softmax(outputs, dim
=1) # Convert logits to probabilities
```

```
    return probabilities

# Function to predict the class of a .nii image using the trained model
def predict(model, device, image_path):

    # Preprocess the image
    image_np = preprocess_image(image_path, labels=False)

    # Check if the preprocessing function adds channel dimension
    if image_np.ndim == 3:
        image_np = np.expand_dims(image_np, axis=0)
    # Add channel dimension: [1, depth, height, width]

    # Convert the NumPy array to a PyTorch tensor and add batch dimension
    image_tensor = torch.from_numpy(image_np).float().unsqueeze(0).to(device)

    # Make a prediction
    probabilities_tensor = predict_image(model, image_tensor)

    # Convert tensor to list
    probabilities = probabilities_tensor.cpu().numpy().flatten().tolist()

    return probabilities
```

Código B.3: Archivo cnn3d/predicting.py

B.2.4. Fichero cnn3d/testing.py

Este fichero contiene las funciones necesarias para realizar la fase de generalización.

Especificación de la clase TestDatasetFromNii

Clase: *TestDatasetFromNii*	
Esta clase se encarga de cargar las imágenes Nifti para la generalización.	
Datos	
- csv_path	Ruta al fichero de etiquetas para generalización.
Métodos	
- __len__	Obtiene la cantidad de datos para generalización.
- __getitem__	Carga las imágenes en *tensor* y las devuelve junto con sus respectivas etiquetas.

Tabla B.6: Especificación de los datos y métodos de la clase TestDatasetFromNii

Especificación de la función generalization_test

Función: *generalization_test*
Esta función realiza la fase de generalización del modelo.
Parámetros

- model	Modelo previamente cargado/entrenado.
- device	CPU/GPU donde se cargará el modelo.
- seed	Semilla que permite la replicación de los datos.
- batch_size	Permite indicar el tamaño de *batch* que se quiere usar.
- file_diferentiator	Cadena que se incluye en el nombre de los ficheros para diferenciar las distintas ejecuciones.
- results_path	Incluye la ruta donde se almacenarán los ficheros de resultados. Se creerá en caso de no existir.

Tabla B.7: Especificación de los parámetros de la función generalization_test

Fichero cnn3d/testing.py

```
import os
import torch

import torch.nn as nn
import pandas as pd
import numpy as np
import nibabel as nib

from torch.utils.data import DataLoader, Dataset
from tools import metrics
from pathlib import Path
```

```
class TestDatasetFromNii(Dataset):
    def __init__(self, csv_path):
        self.data_info = pd.read_csv(csv_path,
    header=None)
        self.image_arr = np.asarray(self.data_info.
    iloc[:, 0])
        self.label_arr = np.asarray(self.data_info.
    iloc[:, 1])
        self.data_len = len(self.data_info.index)

    def __len__(self):
        return self.data_len

    def __getitem__(self, index):
        single_image_path = self.image_arr[index]
        single_image_nii = nib.load(
    single_image_path)
        single_image_arrary = single_image_nii.
    get_fdata()
        single_image_arrary = single_image_arrary.
    astype(np.float32)
        single_image_arrary = np.expand_dims(
    single_image_arrary, axis=0)

        img_as_tensor = torch.from_numpy(
    single_image_arrary)
        single_image_label = self.label_arr[index]
        label_as_tensor = torch.tensor(
    single_image_label, dtype=torch.long)
        return (img_as_tensor, label_as_tensor)

```

```
def generalization_test(model, device, seed,
    batch_size, file_diferentiator, results_path):

    # Test phase
    print("\nTesting the model...\n")

    test_csv = f'data/test_labels_{seed}.csv'

    test_dataset = TestDatasetFromNii(test_csv)
    test_loader = DataLoader(test_dataset,
    batch_size=batch_size, shuffle=False)

    criterion = nn.CrossEntropyLoss().to(device)

    model.eval()

    test_loss = 0.0
    test_correct = 0
    test_total = 0

    all_labels = []
    all_predictions = []

    with torch.no_grad():
        for data in test_loader:
            inputs, labels = data
            inputs, labels = inputs.to(device),
    labels.to(device)
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            test_loss += loss.item()
```

```
            _, predicted = torch.max(outputs.data, 1)
            test_total += labels.size(0)
            test_correct += (predicted == labels).sum().item()

            # Store labels and predictions for confusion matrix
            all_labels.extend(labels.cpu().numpy())
            all_predictions.extend(predicted.cpu().numpy())

    test_loss /= len(test_loader)
    test_accuracy = 100 * test_correct / test_total

    # Calculate confusion matrix
    cms = metrics.confusion_matrix(all_labels, all_predictions)

    # Calculate accuracy
    accuracy_list = metrics.calculate_accuracy(cms)

    # Calculate Precision and Recall
    precision_list, recall_list = metrics.calculate_precision_recall(cms)

    # Calculate F1 Score
    F1_list = metrics.calculate_f1(precision_list, recall_list)

    base_path = os.path.join(results_path, 'testing')
    Path(base_path).mkdir(parents=True, exist_ok=True)
```

```
 with open(os.path.join(base_path, f'cms-{file_diferentiator}.txt'), 'w') as cms_file:

    print(f'\nConfusion Matrices:\n')

    for label, cm in cms.items():
        print(f'Class {label}:')
        print(f'{cm}\n')

        cms_file.write(f'Class {label}:\n{cm}\n\n')




 with open(os.path.join(base_path, f'by_class-{file_diferentiator}.csv'), 'w') as clases_file:

    clases_file.write('Class,Accuracy,Precision,Recall,F1 Score')

    print(f'Test Loss: {test_loss} | Test Global Accuracy: {test_accuracy}%\n')

    sum_acc = 0
    print(f'\nAccuracy by Class:\n')
    for label, accuracy in enumerate(accuracy_list):
        print(f'Class {label} Accuracy: {accuracy}%')
        sum_acc += accuracy
```

```
        print(f'\nMean Accuracy: {sum_acc / len(accuracy_list)}%\n')

        sum_prec = 0
        print(f'\nPrecision by Class:\n')
        for label, precision in enumerate(precision_list):
            print(f'Class {label} Precision: {precision}')
            sum_prec += precision

        print(f'\nMean Precision: {sum_prec / len(precision_list)}\n')

        sum_rec = 0
        print(f'\nRecall by Class:\n')
        for label, recall in enumerate(recall_list):
            print(f'Class {label} Recall: {recall}')
            sum_rec += recall

        print(f'\nMean Recall: {sum_rec / len(recall_list)}\n')

        sum_f1 = 0
        print(f'\nF1 Score by Class:\n')
        for label, F1 in enumerate(F1_list):
            print(f'Class {label} F1 Score: {F1}')
            sum_f1 += F1

        print(f'\nMean F1 Score: {sum_f1 / len(F1_list)}\n')
```

```
        for label in range(len(accuracy_list)):
            clases_file.write(f'\n{label},{accuracy_list[label]},{precision_list[label]},{recall_list[label]},{F1_list[label]}')
146
 with open(os.path.join(base_path, f'averages-{file_diferentiator}.csv'), 'w') as averages_file:
151         averages_file.write('Accuracy,Precision,Recall,F1 Score')
        averages_file.write(f'\n{sum_acc / len(accuracy_list)},{sum_prec / len(precision_list)},{sum_rec / len(recall_list)},{sum_f1 / len(F1_list)}')
```

CÓDIGO B.4: Archivo cnn3d/testing.py

B.2.5. Fichero cnn3d/training.py

Este fichero contiene las funciones necesarias para realizar la fase de entrenamiento.

Especificación de la clase DatasetFromNii

Clase: *DatasetFromNii*	
Esta clase se encarga de cargar las imágenes Nifti para la entrenamiento y realiza el proceso de *data augmentation* si así se indica.	
Datos	
- csv_path	Ruta al fichero de etiquetas para generalización.

- augment	Booleano que indica si se realiza *data augmentation*.
Métodos	
- __len__	Obtiene la cantidad de datos para generalización.
- __getitem__	Carga las imágenes en *tensor* y las devuelve, con *data augmentation* si se especifica, junto con sus respectivas etiquetas.

Tabla B.8: Especificación de los datos y métodos de la clase DatasetFromNii

Especificación de la función train_network

Función: *train_network*	
Esta función realiza el entrenamiento del modelo.	
Parámetros	
- seed	Semilla que permite la replicación de los datos.
- date	Fecha de ejecución del programa. Se usa para identificar las distintas ejecuciones.
- device	CPU/GPU donde se cargará el modelo.
- selected_model	Sirve para indicar cuál de los modelos propuestos se va a usar en el entrenamiento.

- batch_size	Permite indicar el tamaño de *batch* que se quiere usar.
- num_folds	Permite indicar el número de *folds* que quiere usar para el entrenamiento.
- patience	Permite indicar el número de epocas que se quiere esperar antes de realizar *early stopping*.
- learning_rate	Permite indicar el coeficiente de entrenamiento que se quiere usar.
- file_diferentiator	Cadena que se incluye en el nombre de los ficheros para diferenciar las distintas ejecuciones.
- results_path	Incluye la ruta donde se almacenarán los ficheros de resultados. Se creará en caso de no existir.

Tabla B.9: Especificación de los parámetros de la función train_network

Fichero cnn3d/training.py

```
import os
import torch
from cnn3d.models.cnn3d_own import Cnn3DOwn
from cnn3d.models.cnn3d_elassy import Cnn3DElAssy
from cnn3d.models.cnn3d_googlenet import
    Cnn3DGoogleNet

import torch.nn as nn
import torch.optim as optim
import pandas as pd
```

```
import numpy as np
import nibabel as nib
import torchio as tio

from torch.utils.data import DataLoader, Dataset,
    Subset
from sklearn.model_selection import StratifiedKFold
from torch.optim.lr_scheduler import
    ReduceLROnPlateau
from pathlib import Path

from tools import metrics



# Custom dataloader
class DatasetFromNii(Dataset):
    def __init__(self, csv_path, augment=False):
        self.data_info = pd.read_csv(csv_path,
    header=None)
        self.image_arr = np.asarray(self.data_info.
    iloc[:, 0])
        self.label_arr = np.asarray(self.data_info.
    iloc[:, 1])
        self.data_len = len(self.data_info.index)
        self.augment = augment
        if self.augment:
            self.transforms = tio.Compose([
                tio.RandomFlip(axes=(0, 1, 2)),
                tio.RandomAffine(),
                tio.RandomNoise(),
            ])
```

```
    def __len__(self):
        return self.data_len

    def __getitem__(self, index):
        single_image_path = self.image_arr[index]
        single_image_nii = nib.load(
single_image_path)
        single_image_arrary = single_image_nii.
get_fdata()
        single_image_arrary = single_image_arrary.
astype(np.float32)
        single_image_arrary = np.expand_dims(
single_image_arrary, axis=0)

        if self.augment:
            single_image_arrary = self.transforms(
single_image_arrary)

        img_as_tensor = torch.from_numpy(
single_image_arrary)
        single_image_label = self.label_arr[index]
        label_as_tensor = torch.tensor(
single_image_label, dtype=torch.long)
        return (img_as_tensor, label_as_tensor)


def train_network(seed, date, device,
selected_model, batch_size, num_folds, patience,
learning_rate, file_diferentiator, results_path):

    # Parameters
    csv_path = f'data/train_labels_{seed}.csv'
    num_epochs = 1000  # Increased number of epochs
    for early stopping
```

```
66     # Total memory
       total_memory = torch.cuda.get_device_properties
      (0).total_memory
       # Cached memory
       cached_memory = torch.cuda.memory_reserved(0)
       # Allocated memory
71     allocated_memory = torch.cuda.memory_allocated
      (0)
       # Free memory
       free_memory = total_memory - (cached_memory +
      allocated_memory)

       print(f'Using device: {device} ( {free_memory /
       1024**3:.2f} GB / {total_memory / 1024**3:.2f}
      GB )\n')
76
       print('Hiperparametros:\n')

       print(f'\tBatch Size: {batch_size}')
81     print(f'\tLearning Rate: {learning_rate}')
       print(f'\tNumber of Epochs: {num_epochs}')
       print(f'\tNumber of Folds: {num_folds}')
       print(f'\tPatience: {patience}')
       if selected_model == 1:
86         print(f'\tModel: Created by Student')
       elif selected_model == 2:
           print(f'\tModel: Created by El-Assy, A. M.'
      )
       elif selected_model == 3:
           print(f'\tModel: GoogleNet')
```

```
91
    print('\n\n
    ----------------------------------------------------\
    n\n')

96  torch.manual_seed(seed)

    # Load dataset
    dataset = DatasetFromNii(csv_path, augment=True
    )
    dataset_size = len(dataset)
101 labels = dataset.label_arr

    # Initialize K-Fold cross-validation
    kf = StratifiedKFold(n_splits=num_folds,
    shuffle=True, random_state=seed)

106 base_path = os.path.join(results_path, '
    training')
    Path(base_path).mkdir(parents=True, exist_ok=
    True)

    # Cross-validation loop
    with open(os.path.join(base_path, f'folds-{
    file_diferentiator}.csv'), 'w') as folds_file:
111
        folds_file.write('Fold,Epoch,Validation
    Loss,Global Accuracy,Mean Accuracy,Mean Precision
    ,Mean Recall,Mean F1 Score')

        fold_results = []
```

```
        for fold, (train_idx, val_idx) in enumerate
(kf.split(np.arange(dataset_size), labels)):
            print(f'Fold {fold + 1}/{num_folds}')

            # Subset the data for the current fold
            train_subset = Subset(dataset,
train_idx)
            val_subset = Subset(dataset, val_idx)

            train_loader = DataLoader(train_subset,
 batch_size=batch_size, shuffle=True)
            val_loader = DataLoader(val_subset,
batch_size=batch_size, shuffle=False)

            # Model, loss function, and optimizer
            if selected_model == 1:
                model = Cnn3DOwn().to(device)
            elif selected_model == 2:
                model = Cnn3DElAssy().to(device)
            elif selected_model == 3:
                model = Cnn3DGoogleNet().to(device)

            criterion = nn.CrossEntropyLoss().to(
device)

            optimizer = optim.AdamW(model.
parameters(), lr=learning_rate, weight_decay=1e
-5)
            # Instantiate the scheduler
            scheduler = ReduceLROnPlateau(optimizer
, 'min', patience=3, factor=0.1)

            best_val_loss = float('inf')
            best_glob_accu = 0
```

```
            best_accu = 0
            best_prec = 0
            best_rec = 0
            best_f1 = 0
            patience_counter = 0
            best_model_state = None

            # Training loop
            for epoch in range(num_epochs):
                model.train()
                running_loss = 0.0
                for i, data in enumerate(
train_loader, 0):
                    inputs, labels = data
                    inputs, labels = inputs.to(
device), labels.to(device)
                    optimizer.zero_grad()
                    outputs = model(inputs)
                    loss = criterion(outputs,
labels)
                    loss.backward()
                    optimizer.step()
                    running_loss += loss.item()
                    if i % 10 == 9:
                        print(f'Epoch {epoch + 1},
Batch {i + 1}, Loss: {running_loss / 10}')
                        running_loss = 0.0

                # Validation loop
                model.eval()
                val_loss = 0.0
                correct = 0
                total = 0
```

```
                all_labels = []
                all_predictions = []

                with torch.no_grad():
                    for data in val_loader:
                        inputs, labels = data
                        inputs, labels = inputs.to(device), labels.to(device)
                        outputs = model(inputs)
                        loss = criterion(outputs, labels)
                        val_loss += loss.item()
                        _, predicted = torch.max(outputs.data, 1)
                        total += labels.size(0)
                        correct += (predicted == labels).sum().item()

                        # Store labels and predictions for confusion matrix
                        all_labels.extend(labels.cpu().numpy())
                        all_predictions.extend(predicted.cpu().numpy())

                val_loss /= len(val_loader)
                global_accuracy = 100 * correct / total
                print(f'Epoch {epoch + 1}, Validation Loss: {val_loss}, Global Accuracy: {global_accuracy}%')

                # Calculate confusion matrix
```

```
                cms = metrics.confusion_matrix(
    all_labels, all_predictions)

                # Calculate accuracy
                accuracy_list = metrics.
    calculate_accuracy(cms)

                # Calculate Precision and Recall
                precision_list, recall_list =
    metrics.calculate_precision_recall(cms)

                # Calculate F1 Score
                F1_list = metrics.calculate_f1(
    precision_list, recall_list)

                accuracy = sum(accuracy_list) / len
    (accuracy_list)
                print(f'Mean Accuracy: {accuracy}%'
    )

                precision = sum(precision_list) /
    len(precision_list)
                print(f'Mean Precision: {precision}
    ')

                recall = sum(recall_list) / len(
    recall_list)
                print(f'Mean Recall: {recall}')

                f1_score = sum(F1_list) / len(
    F1_list)
                print(f'Mean F1 Score: {f1_score}\n
    ')

```

```
                folds_file.write(f'\n{fold + 1},{epoch + 1},{val_loss},{global_accuracy},{accuracy},{precision},{recall},{f1_score}')

                # Step the scheduler based on validation loss
                scheduler.step(val_loss)

                # Early stopping check
                if val_loss < best_val_loss:
                    best_val_loss = val_loss
                    best_glob_accu = global_accuracy
                    best_accu = accuracy
                    best_prec = precision
                    best_rec = recall
                    best_f1 = f1_score
                    patience_counter = 0
                    best_model_state = model.state_dict()
                else:
                    patience_counter += 1
                    if patience_counter >= patience:
                        print(f'Early stopping at epoch {epoch + 1}')
                        break

            if best_model_state is not None:
                model.load_state_dict(best_model_state)

            fold_results.append((best_val_loss, best_glob_accu, best_accu, best_prec, best_rec,
```

```
best_f1))
            print(f'Fold {fold + 1}, Best
Validation Loss: {best_val_loss}, Global Accuracy
: {best_glob_accu}%')
            print(f'Mean Accuracy: {best_accu}%')
            print(f'Mean Precision: {best_prec}')
            print(f'Mean Recall: {best_rec}')
            print(f'Mean F1 Score: {best_f1}\n\n')

            folds_file.write(f'\n{fold +
1},{1001},{best_val_loss},{best_glob_accu},{
best_accu},{best_prec},{best_rec},{best_f1}')

 with open(os.path.join(base_path, f'averages-{
file_diferentiator}.csv'), 'w') as averages_file:

     averages_file.write('Validation Loss,Global
 Accuracy,Mean Accuracy,Mean Precision,Mean
Recall,Mean F1 Score')

     # Average results across folds
     avg_val_loss = sum([result[0] for result in
 fold_results]) / num_folds
     avg_global_accuracy = sum([result[1] for
result in fold_results]) / num_folds
     avg_mean_accuracy = sum([result[3] for
result in fold_results]) / num_folds
     avg_mean_precision = sum([result[2] for
result in fold_results]) / num_folds
     avg_mean_recall = sum([result[4] for result
 in fold_results]) / num_folds
     avg_mean_f1_score = sum([result[5] for
result in fold_results]) / num_folds
```

```
261         print(f'Average Validation Loss: {
    avg_val_loss}, Average Global Accuracy: {
    avg_global_accuracy}%')
            print(f'Average Mean Accuracy: {
    avg_mean_accuracy}%')
            print(f'Average Mean Precision: {
    avg_mean_precision}')
            print(f'Average Mean Recall: {
    avg_mean_recall}')
            print(f'Average Mean F1 Score: {
    avg_mean_f1_score}\n\n')
266
            averages_file.write(f'\n{avg_val_loss},{
    avg_global_accuracy},{avg_mean_accuracy},{
    avg_mean_precision},{avg_mean_recall},{
    avg_mean_f1_score}')

        # Save the model
        torch.save(model.state_dict(), f'{
    file_diferentiator}-cnn3d_model.pth')
271

        return model
```

Código B.5: Archivo cnn3d/training.py

B.2.6. Fichero cnn3d/models/cnn3d_elassy.py

Este fichero contiene la clase que define el modelo de El-Assy.

Especificación de la clase CNN3D_Model2

Clase: *CNN3D_Model2*

Esta clase se encarga de definir el modelo de El-Assy.	
Métodos	
+ forward	Establece el orden de ejecución de las distintas capas definidas.

Tabla B.10: Especificación de los métodos de la clase CNN3D_Model2

Fichero cnn3d/models/cnn3d_elassy.py

```
import torch
import torch.nn as nn
import torch.nn.functional as F

class Cnn3DElAssy(nn.Module):
    def __init__(self):
        super(Cnn3DElAssy, self).__init__()

        # First two convolutional layers with 16 filters
        self.conv1 = nn.Conv3d(1, 16, kernel_size=3, stride=1, padding=1)
        self.bn1 = nn.BatchNorm3d(16)
        self.conv2 = nn.Conv3d(16, 16, kernel_size=3, stride=1, padding=1)
        self.bn2 = nn.BatchNorm3d(16)

        # First max-pooling layer
        self.pool1 = nn.MaxPool3d(kernel_size=2, stride=2, padding=0)

```

```
        # Next two convolutional layers with 64 filters
        self.conv3 = nn.Conv3d(16, 64, kernel_size=3, stride=1, padding=1)
        self.bn3 = nn.BatchNorm3d(64)
        self.conv4 = nn.Conv3d(64, 64, kernel_size=3, stride=1, padding=1)
        self.bn4 = nn.BatchNorm3d(64)

        # Second max-pooling layer
        self.pool2 = nn.MaxPool3d(kernel_size=2, stride=2, padding=0)

        # Final convolutional layer with 256 filters
        self.conv5 = nn.Conv3d(64, 256, kernel_size=3, stride=1, padding=1)
        self.bn5 = nn.BatchNorm3d(256)

        # Third max-pooling layer
        self.pool3 = nn.MaxPool3d(kernel_size=2, stride=2, padding=0)

        # Dropout layer with 20% rate
        self.dropout = nn.Dropout(p=0.2)

        # Calculate the flattened feature map size after convolutions and pooling
        self._to_linear = None
        self.convs = nn.Sequential(
            self.conv1, self.bn1, nn.ReLU(),
            self.conv2, self.bn2, nn.ReLU(), self.pool1,
            self.conv3, self.bn3, nn.ReLU(),
```

```
                self.conv4, self.bn4, nn.ReLU(), self.
    pool2,
                self.conv5, self.bn5, nn.ReLU(), self.
    pool3
            )

        # Assuming input size is (1, 150, 150, 125)
        x = torch.randn(1, 1, 150, 150, 125)
        self._to_linear = self.convs(x).view(1, -1)
    .size(1)

        # Fully connected layer
        self.fc1 = nn.Linear(self._to_linear, 128)
        self.fc2 = nn.Linear(128, 3)

    def forward(self, x):
        x = self.convs(x)
        x = x.view(x.size(0), -1)  # Flatten the
    tensor
        x = self.dropout(F.relu(self.fc1(x)))
        x = self.fc2(x)
        return x
```

CÓDIGO B.6: Archivo cnn3d/models/cnn3d_elassy.py

B.2.7. Fichero cnn3d/models/cnn3d_googlenet.py

Este fichero contiene la clase que define el modelo GoogleNet.

Especificación de la clase Inception3D

Clase: *Inception3D*

Esta clase se encarga de definir las capas *inception* características del modelo GoogleNet.	
Métodos	
+ forward	Establece el orden de ejecución de las distintas capas definidas.

Tabla B.11: Especificación de los métodos de la clase Inception3D

Especificación de la clase CNN3D_Model3

Clase: *CNN3D_Model3*	
Esta clase se encarga de definir el modelo GoogleNet.	
Métodos	
+ forward	Establece el orden de ejecución de las distintas capas definidas.

Tabla B.12: Especificación de los métodos de la clase CNN3D_Model3

Fichero cnn3d/models/cnn3d_googlenet.py

```
import torch
import torch.nn as nn
import torch.nn.functional as F

class Inception3D(nn.Module):
    def __init__(self, in_channels, out1x1,
   out3x3_reduce, out3x3, out5x5_reduce, out5x5,
   out_pool_proj):
```

```
        super(Inception3D, self).__init__()

        self.branch1x1 = nn.Sequential(
            nn.Conv3d(in_channels, out1x1,
kernel_size=1),
            nn.BatchNorm3d(out1x1),
            nn.ReLU(True)
        )

        self.branch3x3 = nn.Sequential(
            nn.Conv3d(in_channels, out3x3_reduce,
kernel_size=1),
            nn.BatchNorm3d(out3x3_reduce),
            nn.ReLU(True),
            nn.Conv3d(out3x3_reduce, out3x3,
kernel_size=3, padding=1),
            nn.BatchNorm3d(out3x3),
            nn.ReLU(True)
        )

        self.branch5x5 = nn.Sequential(
            nn.Conv3d(in_channels, out5x5_reduce,
kernel_size=1),
            nn.BatchNorm3d(out5x5_reduce),
            nn.ReLU(True),
            nn.Conv3d(out5x5_reduce, out5x5,
kernel_size=5, padding=2),
            nn.BatchNorm3d(out5x5),
            nn.ReLU(True)
        )

        self.branch_pool = nn.Sequential(
            nn.MaxPool3d(kernel_size=3, stride=1,
padding=1),
```

```
            nn.Conv3d(in_channels, out_pool_proj,
    kernel_size=1),
            nn.BatchNorm3d(out_pool_proj),
            nn.ReLU(True)
        )

    def forward(self, x):
        branch1x1 = self.branch1x1(x)
        branch3x3 = self.branch3x3(x)
        branch5x5 = self.branch5x5(x)
        branch_pool = self.branch_pool(x)
        outputs = [branch1x1, branch3x3, branch5x5,
    branch_pool]
        return torch.cat(outputs, 1)


class Cnn3DGoogleNet(nn.Module):
    def __init__(self, num_classes=3):
        super(Cnn3DGoogleNet, self).__init__()

        self.conv1 = nn.Conv3d(1, 64, kernel_size
    =7, stride=2, padding=3)
        self.bn1 = nn.BatchNorm3d(64)
        self.pool1 = nn.MaxPool3d(kernel_size=3,
    stride=2, padding=1)

        self.conv2 = nn.Conv3d(64, 64, kernel_size
    =1)
        self.conv3 = nn.Conv3d(64, 192, kernel_size
    =3, padding=1)
        self.bn2 = nn.BatchNorm3d(192)
        self.pool2 = nn.MaxPool3d(kernel_size=3,
    stride=2, padding=1)
```

```
        self.inception3a = Inception3D(192, 64, 96,
    128, 16, 32, 32)
        self.inception3b = Inception3D(256, 128,
    128, 192, 32, 96, 64)
        self.pool3 = nn.MaxPool3d(kernel_size=3,
    stride=2, padding=1)

        self.inception4a = Inception3D(480, 192,
    96, 208, 16, 48, 64)
        self.inception4b = Inception3D(512, 160,
    112, 224, 24, 64, 64)
        self.inception4c = Inception3D(512, 128,
    128, 256, 24, 64, 64)
        self.inception4d = Inception3D(512, 112,
    144, 288, 32, 64, 64)
        self.inception4e = Inception3D(528, 256,
    160, 320, 32, 128, 128)
        self.pool4 = nn.MaxPool3d(kernel_size=3,
    stride=2, padding=1)

        self.inception5a = Inception3D(832, 256,
    160, 320, 32, 128, 128)
        self.inception5b = Inception3D(832, 384,
    192, 384, 48, 128, 128)

        self.pool5 = nn.AdaptiveAvgPool3d((1, 1, 1)
    )

        self.dropout = nn.Dropout(p=0.2)
        self.fc = nn.Linear(1024, num_classes)

    def forward(self, x):
        x = F.relu(self.bn1(self.conv1(x)))
        x = self.pool1(x)
```

```
        x = F.relu(self.conv2(x))
        x = F.relu(self.bn2(self.conv3(x)))
        x = self.pool2(x)

        x = self.inception3a(x)
        x = self.inception3b(x)
        x = self.pool3(x)

        x = self.inception4a(x)
        x = self.inception4b(x)
        x = self.inception4c(x)
        x = self.inception4d(x)
        x = self.inception4e(x)
        x = self.pool4(x)

        x = self.inception5a(x)
        x = self.inception5b(x)

        x = self.pool5(x)
        x = x.view(x.size(0), -1)  # Flatten the tensor
        x = self.dropout(x)
        x = self.fc(x)
        return x
```

Código B.7: Archivo cnn3d/models/cnn3d_googlenet.py

B.2.8. Fichero cnn3d/models/cnn3d_own.py

Este fichero contiene la clase que define el modelo propuesto por el alumno.

Especificación de la clase CNN3D_Model1

Clase: *CNN3D_Model1*	
Esta clase se encarga de definir el modelo propuesto por el alumno.	
Métodos	
+ forward	Establece el orden de ejecución de las distintas capas definidas.

Tabla B.13: Especificación de los métodos de la clase CNN3D_Model1

Fichero cnn3d/models/cnn3d_own.py

```
import torch
import torch.nn as nn
import torch.nn.functional as F

# Define the CNN3D model
class Cnn3DOwn(nn.Module):
    def __init__(self):
        super(Cnn3DOwn, self).__init__()
        self.conv1 = nn.Conv3d(1, 16, kernel_size
=2, stride=1, padding=1)
        self.bn1 = nn.BatchNorm3d(16)
        self.pool = nn.MaxPool3d(kernel_size=2,
stride=2, padding=0)
        self.conv2 = nn.Conv3d(16, 32, kernel_size
=2, stride=1, padding=1)
        self.bn2 = nn.BatchNorm3d(32)
        self.conv3 = nn.Conv3d(32, 64, kernel_size
=2, stride=1, padding=1)
        self.bn3 = nn.BatchNorm3d(64)
```

```
        self.dropout = nn.Dropout(p=0.5)

        # Calculate the correct size after convolutions and max pooling
        self._to_linear = None
        self.convs = nn.Sequential(
            self.conv1, self.bn1, nn.ReLU(), self.pool,
            self.conv2, self.bn2, nn.ReLU(), self.pool,
            self.conv3, self.bn3, nn.ReLU(), self.pool
        )
        x = torch.randn(1, 1, 150, 150, 125)
        self._to_linear = self.convs(x).view(1, -1).size(1)

        self.fc1 = nn.Linear(self._to_linear, 128)
        self.fc2 = nn.Linear(128, 64)
        self.fc3 = nn.Linear(64, 3)

    def forward(self, x):
        x = self.convs(x)
        x = x.view(x.size(0), -1)  # Flatten the tensor
        x = self.dropout(F.relu(self.fc1(x)))
        x = self.dropout(F.relu(self.fc2(x)))
        x = self.fc3(x)
        return x
```

Código B.8: Archivo cnn3d/models/cnn3d_own.py

B.2.9. Fichero tools/graph_results.py

Este fichero contiene las funciones que generan las gráficas, tablas y mapas de calor a partir de los resultados del modelo.

Especificación de la función get_subtitle_metrics_from_file

Función: *get_subtitle_metrics_from_file*	
Esta función obtiene los algunos de los hiperparámetros del modelo y crea un subtítulo con ellos a partir del nombre del archivo de los resultados.	
Parámetros	
- filename	Nombre del archivo de resultados del que se van a extraer los hiperparámetros.
Salida	
- model	Valor que indica cuál de los tres modelos disponibles se ha usado.
- folds	Valor del hiperparámetros número de *folds*.
- patience	Valor del hiperparámetro que indica la paciencia del *early stopping*.
- lrate	Valor del coeficiente de entrenamiento.
- subtitle	Título generado a partir de los hiperparámetros para usar en las gráficas y facilitar su identificación.

Tabla B.14: Especificación de los parámetros y salidas de la función get_subtitle_metrics_from_file

Especificación de la función parse_cm_file

Función: *parse_cm_file*	
Esta función obtiene las matrices de confusión de los archivos de resultados donde están almacenadas.	
Parámetros	
- filename	Nombre del archivo de resultados del que se van a extraer los hiperparámetros.
Salida	
- cm_matrices	Diccionario que contiene las matrices confusión de cada una de las clases.

Tabla B.15: Especificación de los parámetros y salidas de la función parse_cm_file

Especificación de la función create_cm_heatmat

Función: *create_cm_heatmat*
Esta función crea un mapa de calor a partir de una matriz de confusión para obtener una representación visual clara.
Parámetros

- ax	*Subplot* donde se dibujará la matriz de confusión.
- cm	Matriz de confusión.
- class_label	Etiqueta de la clase que representa la matriz de confusión.

Tabla B.16: Especificación de los parámetros de la función create_cm_heatmat

Especificación de la función create_general_cm

Función: *create_general_cm*	
Esta función crea la matriz de confusión general de las tres clases y su mapa calor.	
Parámetros	
- cm_matrices	Diccionario que contiene las matrices confusión de cada una de las clases.
- model_cms_path	Ruta donde se almacenará el mapa de calor.
- model	Valor que indica cuál de los tres modelos disponibles se ha usado.
- folds	Valor del hiperparámetros número de *folds*.
- patience	Valor del hiperparámetro que indica la paciencia del *early stopping*.

- lrate	Valor del coeficiente de entrenamiento.
- subtitle	Título usado en las gráficas para facilitar su identificación.

Tabla B.17: Especificación de los parámetros de la función create_general_cm

Especificación de la función generate_graphs_folds

Función: *generate_graphs_folds*	
Esta función crea gráficos a partir de los resultados almacenados en los ficheros de resultados *folds*.	
Parámetros	
- model	Valor que indica cuál de los tres modelos disponibles se ha usado.
- results_path	Ruta donde se encuentran los ficheros de resultados.
- graphs_base_path	Ruta donde se almacenará gráfico.

Tabla B.18: Especificación de los parámetros de la función generate_graphs_folds

Especificación de la función generate_graphs_averages

Función: *generate_graphs_averages*

Esta función crea tablas a partir de los resultados almacenados en los ficheros de resultados *averages*.	
Parámetros	
- model	Valor que indica cuál de los tres modelos disponibles se ha usado.
- mode	Indica si los resultados que están siendo tratados son de entrenamiento o generalización.
- results_path	Ruta donde se encuentran los ficheros de resultados.
- graphs_base_path	Ruta donde se almacenará gráfico.

Tabla B.19: Especificación de los parámetros de la función generate_graphs_averages

Especificación de la función generate_graphs_by_class

Función: *generate_graphs_by_class*	
Esta función crea tablas a partir de los resultados almacenados en los ficheros de resultados *by_class*.	
Parámetros	
- model	Valor que indica cuál de los tres modelos disponibles se ha usado.
- mode	Indica si los resultados que están siendo tratados son de entrenamiento o generalización.

- results_path	Ruta donde se encuentran los ficheros de resultados.
- graphs_base_path	Ruta donde se almacenará gráfico.

Tabla B.20: Especificación de los parámetros de la función generate_graphs_by_class

Especificación de la función main

Función: *main*	
Esta función es la principal y hace las llamadas a todas las funciones necesarias para crear los gráficos, tablas y mapas de calor, para entrenamiento y generalización, a partir de los resultados.	
Parámetros	
- results_path	Ruta donde se encuentran los ficheros de resultados.

Tabla B.21: Especificación de los parámetros de la función main

Fichero tools/graph_results.py

```
import click
import os
import re

import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sns
```

```
from pathlib import Path

def get_subtitle_metrics_from_filename(filename):

    seed = int(filename.split('-')[1].split('_')[-1])

    hyperparams = filename.split('-')[-1].split('_')
    model = hyperparams[0]
    folds = hyperparams[1]
    patience = hyperparams[2]
    lrate = '.'.join(hyperparams[3].split('.')[0:2])

    subtitle = f'Seed: {seed}, Model {model}: Folds = {folds}, Patience = {patience}, L. Rate = {float(lrate):.0e}'

    return seed, model, folds, patience, lrate, subtitle


def parse_cm_file(filename):
    with open(filename, 'r') as file:
        lines = file.readlines()

    cm_matrices = {}
    class_idx = None
    matrix_lines = []
    for line in lines:
        line = line.strip()
```

```
        if line.startswith("Class"):
            if class_idx is not None:
                # Save the previous class's confusion matrix
                cm_matrices[class_idx] = np.array(matrix_lines, dtype=int)
                matrix_lines = []
            class_idx = int(line.split()[1][:-1]) # Extract class index
        elif line.startswith("["):
            # Collect lines of the matrix
            matrix_lines.append(list(map(int, line.strip("[]").split())))

    if class_idx is not None:
        # Save the last class's confusion matrix
        cm_matrices[class_idx] = np.array(matrix_lines, dtype=int)

    return cm_matrices


def create_cm_heatmap(ax, cm, class_label):
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False, ax=ax,
                xticklabels=['Positive', 'Negative'], yticklabels=['Positive', 'Negative'])

    # Set the title based on the class label
    if class_label == 0:
        class_label_name = 'AD'
    elif class_label == 1:
        class_label_name = 'MCI'
    elif class_label == 2:
```

```
        class_label_name = 'CD'
    ax.set_title(f'Class {class_label_name}')
    ax.set_xlabel('Predicted')
    ax.set_ylabel('Actual')


def create_general_cm(cm_matrices, model_cms_path,
    seed, model, folds, patience, lrate, subtitle):
    # Initialize the 3x3 confusion matrix
    general_cm = np.zeros((3, 3), dtype=int)

    # Fill in the confusion matrix based on the
    individual class matrices

    # Class 0
    general_cm[0, 0] = cm_matrices[0][1, 1]  # True
    Positives for Class 0
    general_cm[0, 1] = cm_matrices[1][1, 0]  #
    False Positives for Class 1, from Class 0
    perspective
    general_cm[0, 2] = cm_matrices[2][1, 0]  #
    False Positives for Class 2, from Class 0
    perspective

    # Class 1
    general_cm[1, 1] = cm_matrices[1][1, 1]  # True
    Positives for Class 1
    general_cm[1, 0] = cm_matrices[0][1, 0]  #
    False Positives for Class 0, from Class 1
    perspective
    general_cm[1, 2] = cm_matrices[2][0, 1]  #
    False Positives for Class 2, from Class 1
    perspective
```

```
    # Class 2
    general_cm[2, 2] = cm_matrices[2][0, 0]  # True Positives for Class 2
    general_cm[2, 0] = cm_matrices[0][0, 1]  # False Positives for Class 0, from Class 2 perspective
    general_cm[2, 1] = cm_matrices[1][0, 1]  # False Positives for Class 1, from Class 2 perspective

    class_labels = ['Class AD', 'Class MCI', 'Class CN']

    plt.figure(figsize=(8, 6))
    sns.heatmap(general_cm, annot=True, fmt="d", cmap="Blues", cbar=False,
                xticklabels=class_labels, yticklabels=class_labels)
    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')
    plt.title(f'Confusion Matrix ({subtitle})')

    plt.tight_layout(rect=[0, 0.03, 1, 0.95])  # Add some padding for the title

    best_save_folder = os.path.join(model_cms_path, f'general_cm-{seed}-{model}_{folds}_{patience}_{lrate}.pdf')
    plt.savefig(best_save_folder)
    plt.close()


def generate_graphs_folds(model, results_path, graphs_base_path):
```

```
    pattern = fr'folds-(\d+)_(\d+)-{model}_(\d+)_(\d+)_([\d.]+)\.csv'
    files = [f for f in os.listdir(os.path.join(results_path, 'training')) if re.search(pattern, f)]

    model_folds_path = os.path.join(graphs_base_path, f'model{model}', 'folds')
    Path(model_folds_path).mkdir(parents=True, exist_ok=True)

    # Metrics list
    metrics = ['Validation Loss', 'Global Accuracy', 'Mean Accuracy', 'Mean Precision', 'Mean Recall', 'Mean F1 Score']

    for file in files:

        # Load the CSV file
        df = pd.read_csv(os.path.join(results_path, 'training', file))

        seed, model, folds, patience, lrate, subtitle = get_subtitle_metrics_from_filename(file)

        # Create a plot for each metric
        for metric in metrics:

            title = f'{metric} ({subtitle})'
            x_label = 'Epoch'
            y_label = metric
```

```
            base_folder = os.path.join(
model_folds_path, metric.lower().replace(" ", "_"
))
            folder_name = f'{metric.lower().replace
(" ", "_")}-{model}_{folds}_{patience}_{lrate}'
            save_folder = os.path.join(base_folder,
 folder_name)
            Path(save_folder).mkdir(parents=True,
exist_ok=True)

            # Trace a line for each fold
            for fold in df['Fold'].unique():
                fold_data = df[df['Fold'] == fold]
                fold_data = fold_data[fold_data['
Epoch'] != 1001]
                plt.plot(fold_data['Epoch'],
fold_data[metric], label=f'Fold {fold}')

                plt.title(title)
                plt.xlabel(x_label)
                plt.ylabel(y_label)
                plt.legend()

                # Guardar la gráfica en un archivo
PNG
                save_filename = os.path.join(
save_folder, f'Fold{fold}.pdf')
                plt.savefig(save_filename)
                plt.close()

        # Define the grid dimensions
        nrows, ncols = 2, 3

        # Create subplots: a 2x3 grid
```

```
        fig, axes = plt.subplots(nrows=nrows, ncols
    =ncols, figsize=(15, 10))

        # Flatten the axes array to easily iterate
    over it
        axes = axes.flatten()

        fig.suptitle(subtitle, fontsize=16)

        for idx, metric in enumerate(metrics):

            title = f'Best {metric}'
            x_label = 'Fold'
            y_label = metric

            best_value = df[(df['Epoch'] == 1001)]

            # Plot on the corresponding subplot
            axes[idx].plot(df['Fold'].unique(),
    best_value[metric])
            axes[idx].set_title(title)
            axes[idx].set_xlabel(x_label)
            axes[idx].set_ylabel(y_label)

        # Hide any unused subplots if there are
    fewer metrics than subplots
        for i in range(len(metrics), nrows * ncols)
    :
            fig.delaxes(axes[i])

        # Adjust layout to prevent overlap
        plt.tight_layout(rect=[0, 0.03, 1, 0.95])
    # Add some padding for the title
```

```
        best_save_folder = os.path.join(
    model_folds_path, f'best_metrics-{seed}-{model}_{
    folds}_{patience}_{lrate}.pdf')
        plt.savefig(best_save_folder)
        plt.close()


def generate_graphs_averages(model, mode,
    results_path, graphs_base_path):

    pattern = fr'averages-(\d+)_(\d+)-{model}_(\d+)
    _(\d+)_([\d.]+)\.csv'
    files = [f for f in os.listdir(os.path.join(
    results_path, mode)) if re.search(pattern, f)]

    model_averages_path = os.path.join(
    graphs_base_path, f'model{model}', 'averages')
    Path(model_averages_path).mkdir(parents=True,
    exist_ok=True)

    for file in files:

        df = pd.read_csv(os.path.join(results_path,
     mode, file))

        seed, model, folds, patience, lrate,
    subtitle = get_subtitle_metrics_from_filename(
    file)

        columns = list(df.columns)
        data = df.iloc[0].tolist()

        if mode == 'training':
```

```
            formatted_data = [
                round(data[0], 4),  # Validation
    Loss
                f"{round(data[1], 4)}%",  # Global
    Accuracy with percentage
                f"{round(data[2], 4)}%",  # Mean
    Accuracy with percentage
                round(data[3], 4),  # Mean
    Precision
                round(data[4], 4),  # Mean Recall
                round(data[5], 4)   # Mean F1 Score
            ]

        else:

            formatted_data = [
                f"{round(data[0], 4)}%",  #
    Accuracy
                round(data[1], 4),  # Precision
                round(data[2], 4),  # Recall
                round(data[3], 4)   # F1 Score
            ]

        fig, ax = plt.subplots(figsize=(8, 3))

        fig.suptitle(subtitle, fontsize=16)

        ax.axis('tight')
        ax.axis('off')

        table_data = [formatted_data]
        table = ax.table(cellText=table_data,
    colLabels=columns, cellLoc='center', loc='center'
```

```
    )

        table.auto_set_font_size(False)
        table.set_fontsize(10)
        table.scale(1.2, 1.2)

        plt.tight_layout(rect=[0, 0.03, 1, 0.95])
    # Add some padding for the title

        best_save_folder = os.path.join(
    model_averages_path, f'averages-{seed}-{model}_{
    folds}_{patience}_{lrate}.pdf')
        plt.savefig(best_save_folder)
        plt.close()


def generate_graphs_by_class(model, mode,
    results_path, graphs_base_path):

    pattern = fr'by_class-(\d+)_(\d+)-{model}_(\d+)
    _(\d+)_([\d.]+)\.csv'
    files = [f for f in os.listdir(os.path.join(
    results_path, mode)) if re.search(pattern, f)]

    model_by_class_path = os.path.join(
    graphs_base_path, f'model{model}', 'by_class')
    Path(model_by_class_path).mkdir(parents=True,
    exist_ok=True)

    for file in files:

        # Read the CSV file
        df = pd.read_csv(os.path.join(results_path,
    mode, file))
```

```
        seed, model, folds, patience, lrate,
    subtitle = get_subtitle_metrics_from_filename(
    file)

        # Extract the data
        columns = list(df.columns)
        data = df.values.tolist()

        # Format the data
        formatted_data = []
        for row in data:
            formatted_row = [
                int(row[0]),  # Class (as integer)
                f"{round(row[1], 2)}%",  # Accuracy
     with percentage
                round(row[2], 2),  # Precision
                round(row[3], 2),  # Recall
                round(row[4], 2)   # F1 Score
            ]
            formatted_data.append(formatted_row)

        # Create a figure and axis
        fig, ax = plt.subplots(figsize=(10, 4))  #
    Adjust the figure size as needed

        fig.suptitle(subtitle, fontsize=16)

        ax.axis('tight')
        ax.axis('off')

        # Create the table
        table = ax.table(cellText=formatted_data,
    colLabels=columns, cellLoc='center', loc='center'
```

```
    )

        # Style the table
        table.auto_set_font_size(False)
        table.set_fontsize(10)
        table.scale(1.2, 1.2)

        plt.tight_layout(rect=[0, 0.03, 1, 0.95])
    # Add some padding for the title

        best_save_folder = os.path.join(
    model_by_class_path, f'by_class-{seed}-{model}_{
    folds}_{patience}_{lrate}.pdf')
        plt.savefig(best_save_folder)
        plt.close()


def generate_cms(model, mode, results_path,
    graphs_base_path):

     pattern = fr'cms-(\d+)_(\d+)-{model}_(\d+)_(\d
    +)_([\d.]+)\.txt'
     files = [f for f in os.listdir(os.path.join(
    results_path, mode)) if re.search(pattern, f)]

     model_cms_path = os.path.join(graphs_base_path,
     f'model{model}', 'cms')
     Path(model_cms_path).mkdir(parents=True,
    exist_ok=True)

     for file in files:

        # Parse the confusion matrix file
```

```
        cm_matrices = parse_cm_file(os.path.join(
    results_path, mode, file))

        seed, model, folds, patience, lrate,
    subtitle = get_subtitle_metrics_from_filename(
    file)

        # Create subplots
        fig, axs = plt.subplots(2, 2, figsize=(8,
    8))  # Adjust size as needed
        axs = axs.flatten()  # Flatten the array
    for easy indexing

        fig.suptitle(subtitle, fontsize=16)

        # Plot each class confusion matrix
        for i, (class_label, cm) in enumerate(
    cm_matrices.items()):
            create_cm_heatmap(axs[i], cm,
    class_label)

        # Hide any unused subplots if there are
    fewer metrics than subplots
        fig.delaxes(axs[3])

        # Adjust layout and save the figure
        plt.tight_layout(rect=[0, 0.03, 1, 0.95])
    # Add some padding for the title

        best_save_folder = os.path.join(
    model_cms_path, f'cms-{seed}-{model}_{folds}_{
    patience}_{lrate}.pdf')
        plt.savefig(best_save_folder)
        plt.close()
```

```
        create_general_cm(cm_matrices, model_cms_path, seed, model, folds, patience, lrate, subtitle)



@click.command()
@click.option('--results_path', '-r', default='results', required=False, help='Path where the results are stored')
def main(results_path):

    ##################################
    # CHECKIN IF PROVIDED PATH EXISTS #
    ##################################
    if not os.path.exists(results_path):
        print(f'The specified path "{results_path}" does not exists')
        exit()


    ###################
    # TRAINING GRAPHS #
    ###################

    print('Generating Training Grahs...')

    # Training graphs base folder
    mode = 'training'
    graphs_base_path = os.path.join(results_path, 'graphs', mode)

    for model in range(1, 4):
```

```
        print(f'|\n|\tGenerating Model {model}
    Grahs...')

367     print('|\t|\n|\t|\tGenerating Folds Grahs
    ...')
        generate_graphs_folds(model, results_path,
    graphs_base_path)
        print('|\t|\t|\n|\t|\tFolds Grahs Generated
    ')


372     print('|\t|\n|\t|\tGenerating Averages
    Grahs...')
        generate_graphs_averages(model, mode,
    results_path, graphs_base_path)
        print('|\t|\t|\n|\t|\tAverages Grahs
    Generated')


377     print(f'|\t|\n|\tModel {model} Grahs
    Generated')

    print('|\nTraining Grahs Generated')


382 ##################
    # TESTING GRAPHS #
    ##################

    print('\nGenerating Testing Grahs...')
387
    # Training graphs base folder
    mode = 'testing'
```

```
    graphs_base_path = os.path.join(results_path, 'graphs', mode)

    for model in range(1, 4):

        print(f'|\n|\tGenerating Model {model} Grahs...')


        print('|\t|\n|\t|\tGenerating Averages Grahs...')
        generate_graphs_averages(model, mode, results_path, graphs_base_path)
        print('|\t|\t|\n|\t|\tAverages Grahs Generated')

        print('|\t|\n|\t|\tGenerating Folds Grahs...')
        generate_graphs_by_class(model, mode, results_path, graphs_base_path)
        print('|\t|\t|\n|\t|\tFolds Grahs Generated')

        print('|\t|\n|\t|\tGenerating CMs...')
        generate_cms(model, mode, results_path, graphs_base_path)
        print('|\t|\t|\n|\t|\tCMs Generated')

        print(f'|\t|\n|\tModel {model} Grahs Generated')


    print('|\nTraining Grahs Generated')
```

```
    print("\n\nGráficas creadas y guardadas
   exitosamente.")

417

if __name__ == "__main__":
    main()
```

CÓDIGO B.9: Archivo tools/graph_results.py

B.2.10. Fichero tools/metrics.py

Este fichero contiene las funciones que calculan cada una de las distintas métricas que se obtienen.

Especificación de la función confusion_matrix

Función: *confusion_matrix*	
Esta función calcula las matrices de confusión de cada clase a partir de las etiquetas reales y las predichas.	
Parámetros	
- labels	Lista que contiene las etiquetas reales de las imágenes.
- predictions	Lista que contiene las etiquetas predichas por el modelo.
Salida	
- cms	Lista que contiene las matrices de confusión de cada clase.

Tabla B.22: Especificación de los parámetros y salidas de la función confusion_matrix

Especificación de la función calculate_accuracy

Función: *calculate_accuracy*	
Esta función calcula el valor de *accuracy* para cada clase.	
Parámetros	
- cms	Lista que contiene las matrices de confusión de cada clase.
Salida	
- accuracy_list	Lista que contiene el valor de *accuracy* para cada clase.

Tabla B.23: Especificación de los parámetros y salidas de la función calculate_accuracy

Especificación de la función calculate_precision_recall

Función: *calculate_precision_recall*	
Esta función calcula el valor de *precision* y *recall* para cada clase.	
Parámetros	
- cms	Lista que contiene las matrices de confusión de cada clase.
Salida	

- precision_list	Lista que contiene el valor de *precision* para cada clase.
- recall_list	Lista que contiene el valor de *recall* para cada clase.

Tabla B.24: Especificación de los parámetros y salidas de la función calculate_precision_recall

Especificación de la función calculate_f1

Función: *calculate_f1*	
Esta función calcula el valor de la puntuación F1 para cada clase.	
Parámetros	
- precision_list	Lista que contiene el valor de *precision* para cada clase.
- recall_list	Lista que contiene el valor de *recall* para cada clase.
Salida	
- F1_list	Lista que contiene el valor de la puntuación *f1* para cada clase.

Tabla B.25: Especificación de los parámetros y salidas de la función calculate_f1

Fichero tools/metrics.py

```
import numpy as np
```

```
def confusion_matrix(labels, predictions):

  cms = {}

  for label in range(3):

    # Inicializar la matriz de confusión 2x2 para la clase actual
    cm = np.zeros((2, 2), dtype=int)

    # Iterar sobre las listas de valores reales y predichos
    for true, pred in zip(labels, predictions):
      if true == label and pred == label:
        cm[0, 0] += 1  # TP
      elif true == label and pred != label:
        cm[1, 0] += 1  # FN
      elif true != label and pred == label:
        cm[0, 1] += 1  # FP
      elif true != label and pred != label:
        cm[1, 1] += 1  # TN

    # Almacenar la matriz de confusión 2x2 en el diccionario
    cms[label] = cm

  return cms


def calculate_accuracy(cms):

  accuracy_list = []

  for _, cm in cms.items():
```

```
    numerator = cm[0, 0] # TPi
    denominator = cm[0, 0] + cm[1, 0] + cm[0, 1] +
  cm[1, 1] # TPi + FNi + FPi + TNi

    if denominator == 0:
      accuracy_list.append(0)
    else:
      accuracy_list.append((numerator / denominator
  ) * 100)

  return accuracy_list


def calculate_precision_recall(cms):

  precision_list = []
  recall_list = []

  for _, cm in cms.items():
    numerator_prec = cm[0, 0] # TPi
    denominator_prec = cm[0, 0] + cm[0, 1] # TPi +
   FPi

    if denominator_prec == 0:
      precision_list.append(0)
    else:
      precision_list.append((numerator_prec /
  denominator_prec))


    numerator_rec = cm[0, 0] # TPi
    denominator_rec = cm[0, 0] + cm[1, 0] # TPi +
  FNi
```

```
64
      if denominator_rec == 0:
        recall_list.append(0)
      else:
        recall_list.append((numerator_rec /
    denominator_rec))
69
    return precision_list, recall_list




74
  def calculate_f1(precision_list, recall_list):

    F1_list = []

79  for i in range(len(precision_list)):
      numerator = precision_list[i] * recall_list[i]
      denominator = precision_list[i] + recall_list[i
    ]

      if denominator == 0:
84      F1_list.append(0)
      else:
        F1_list.append( 2 * ( numerator / denominator
      ) )

    return F1_list
```

Código B.10: Archivo tools/metrics.py

B.2.11. Fichero tools/preprocess_image.py

Este fichero contiene las funciones que se encargan del preprocesamiento de las imágenes.

Especificación de la función get_crop_bundaries

Función: *get_crop_bundaries*	
Esta función calcula donde hay que establecer los límites de corte de las capas de la imagen.	
Parámetros	
- data	Variable que contiene la imagen cargada.
Salida	
- top_limit	Píxel donde hay que realizar el corte por arriba.
- bottom_limit	Píxel donde hay que realizar el corte por abajo.
- left_limit	Píxel donde hay que realizar el corte por la izquierda.
- right_limit	Píxel donde hay que realizar el corte por la derecha.

Tabla B.26: Especificación de los parámetros y salidas de la función get_crop_bundaries

Especificación de la función get_front_back_limits

Función: *get_front_back_limits*

Esta función calcula donde hay que establecer los límites de profundidad de la imagen basado en un límite.	
Parámetros	
- array	Lista que contiene los valores de la suma de Sobel de cada capa.
- threshold	Límite a partir del cual se ha determinado que una capa se considere válida.
- num_slices	Número de capas de la imagen.
Salida	
- front_limit	Capa donde hay que realizar el corte por delante.
- back_limit	Capa donde hay que realizar el corte por atrás.

Tabla B.27: Especificación de los parámetros y salidas de la función get_front_back_limits

Especificación de la función sobel_edges_front_back

Función: *sobel_edges_front_back*	
Esta función calcula los límites delantero y trasero usando la suma de Sobel.	
Parámetros	
- percentage	Porcentaje usado para calcular donde establecer el límite.

- data	Variable que contiene la imagen cargada.
Salida	
- front_limit	Capa donde hay que realizar el corte por delante.
- back_limit	Capa donde hay que realizar el corte por atrás.

Tabla B.28: Especificación de los parámetros y salidas de la función sobel_edges_front_back

Especificación de la función preprocess_image

Función: *preprocess_image*	
Esta función se encarga de realizar todo el preprocesado de una imágen.	
Parámetros	
- file_path	Ruta donde se encuentra la imagen Nifti.
- destination	Ruta donde se va a guardar la imagen procesada.
- labels_dest	Ruta al fichero donde se van a escribir las etiquetas.
- dataframe	*Dataframe* de Pandas que contiene el fichero CSV en el formato de ADNI

- labels	Booleano que indica si queremos que se genere el fichero de etiquetas. Por defecto es verdadero.

Tabla B.29: Especificación de los parámetros de la función preprocess_image

Especificación de la función preprocess_folder

Función: *preprocess_folder*	
Esta función se encarga de realizar todo el preprocesado de las imágenes de un directorio.	
Parámetros	
- source_folder	Ruta donde se encuentran las imágenes Nifti.
- output_folder	Ruta donde se van a guardar las imágenes procesadas.
- labels_source	Ruta del fichero CSV original de ADNI.
- labels_destination	Ruta al fichero donde se van a escribir las etiquetas.

Tabla B.30: Especificación de los parámetros de la función preprocess_folder

Fichero tools/preprocess_image.py

```
import nibabel as nib
import numpy as np
import pandas as pd
```

```
import matplotlib.pyplot as plt
from skimage.filters import sobel

import os
import glob

def get_crop_bundaries(data):

    # We assume that the brain occupies higher
   intensity values in the image
    # Summing up pixel values along the z-axis (
   depth) to find the slice with the most brain
   tissue
    sums = np.sum(np.sum(data, axis=0), axis=0)

    # Find the index of the slice with the maximum
   sum
    max_index = np.argmax(sums)

    # Select the slice with the most brain tissue
    slice_data = data[:, :, max_index]

    percentage_row = 0.15
    percentage_col = 0.17

    # Initialize limits
    top_limit = 0
    bottom_limit = slice_data.shape[0] - 1
    left_limit = 0
    right_limit = slice_data.shape[1] - 1


```

```
 means_row = np.array([np.mean(slice_data[
row_idx, :]) for row_idx in range(slice_data.
shape[0])])
 threshold_row = percentage_row * np.max(
means_row)

 # Determine the top limit
 for row_idx in range(slice_data.shape[0]):
     row_mean = np.mean(slice_data[row_idx, :])
     if not row_mean < threshold_row:
         top_limit = row_idx
         break

 # Determine the bottom limit
 for row_idx in range(slice_data.shape[0] - 1,
-1, -1):
     row_mean = np.mean(slice_data[row_idx, :])
     if not row_mean < threshold_row:
         bottom_limit = row_idx
         break



 means_col = np.array([np.mean(slice_data[:,
col_idx]) for col_idx in range(slice_data.shape
[1])])
 threshold_col = percentage_col * np.max(
means_col)

 # Determine the left limit
 for col_idx in range(slice_data.shape[1]):
     col_mean = np.mean(slice_data[:, col_idx])
     if not col_mean < threshold_col:
         left_limit = col_idx
```

```
            break

    # Determine the right limit
    for col_idx in range(slice_data.shape[1] - 1, -1, -1):
        col_mean = np.mean(slice_data[:, col_idx])
        if not col_mean < threshold_col:
            right_limit = col_idx
            break

    return top_limit, bottom_limit, left_limit, right_limit


def get_front_back_limits(array, threshold, num_slices):

  front_limit = 0
  for i in range(num_slices):
    if array[i] > threshold:
      front_limit = i
      break


  back_limit = 0
  for i in range(num_slices-1, -1, -1):
    if array[i] > threshold:
      back_limit = i
      break


  return front_limit, back_limit
```

```
def sobel_edges_front_back(percentage, data):

    plt.figure(figsize=(12, 8))

    num_slices = data.shape[2]

    # Calculate the sum of edge responses for each slice using Sobel
    edges = [np.sum(sobel(data[:, :, i])) for i in range(num_slices)]

    threshold = percentage * np.max(edges)

    front_limit, back_limit = get_front_back_limits(edges, threshold, num_slices)

    return front_limit, back_limit


def preprocess_image(file_path, destination=None, labels_dest=None, dataframe=None, labels=True):

    # Load the MRI image
    img = nib.load(file_path)
    data = img.get_fdata()

    top_limit, bottom_limit, left_limit, right_limit = get_crop_bundaries(data)
    front_limit, back_limit = sobel_edges_front_back(0.65, data)

    # Crop all layers of the 3D image using the calculated boundaries
```

```
    cropped_data = data[top_limit:bottom_limit+1,
    left_limit:right_limit+1, front_limit:back_limit
    +1]

    if not labels:
        return cropped_data

    # Create a new NIfTI image
    cropped_img = nib.Nifti1Image(cropped_data, img
    .affine, img.header)

    # Save the new cropped image
    image_id = file_path.split('/')[-1]
    output_file_path = os.path.join(destination,
    image_id)
    nib.save(cropped_img, output_file_path)

    for ind in dataframe.index:
        label = 0 #AD
        if (dataframe["Group"][ind] == "MCI"):
            label = 1
        elif (dataframe["Group"][ind] == "CN"):
            label = 2

        if os.path.exists(output_file_path):
            labels_dest.write(f"{output_file_path
    },{label}\n")


def preprocess_folder(source_folder, output_folder,
    labels_source, labels_destination):

    f = open(os.path.join(labels_destination))
```

```
    files = [f for f in os.listdir(source_folder) if f.endswith('.nii') or f.endswith('.nii.gz')]
    i=0

    df = pd.read_csv(labels_source)

    for file in files:
        file_path = os.path.join(source_folder, file)
        preprocess_image(file_path, output_folder, f, df)
        i += 1

        exists = glob.glob(f'{output_folder}/{file}')

        if len(exists) == 0:
            print(f'File {file} not found')
            break
        else:
            print(f'{file}')
```

Código B.11: Archivo tools/preprocess_image.py

B.2.12. Ejecución de los experimentos

Para facilitar la ejecución de experimentos, se ha creado una *script* de *bash* que genera un archivo `task_params.txt` con los datos de todos los experimentos y hace la llamada a HTCondor que hace uso de este archivo para crear la cola de experimentos.

```
#! /bin/bash

rm condor_output/out/*
```

```
rm condor_output/err/*
rm condor_output/log/*

rm results/training/*
rm results/testing/*

rm *.pth

# Get the current date in the format yymmddHHMM
current_date=$(date +"%y%m%d%H%M")

# Start the seed
seed=1

first=true

models=(1 2 3)
learning_rate=(0.01 0.001 0.0001)
nfolds=(5 10)
patiences=(10 15)

# Overwrite the old file
#echo "" > task_params.txt

# Append the remaining lines to the file if needed
for model in "${models[@]}"; do
  for lrate in "${learning_rate[@]}"; do
    for fold in "${nfolds[@]}"; do
      for patience in "${patiences[@]}"; do
        if [ "$first" = true ] ; then
          echo "$seed $current_date $model $lrate
   $fold $patience" > task_params.txt
          first=false
        else
```

```
            echo "$seed $current_date $model $lrate
    $fold $patience" >> task_params.txt
          fi
        done
      done
    done
done

condor_submit task.sub
```

Código B.12: Archivo submit.sh