Universidad de Valladolid

ESCUELA DE INGENIERÍA INFORMÁTICA
DE SEGOVIA

Grado en Ingeniería Informática
de Servicios y Aplicaciones

Soluciones Deep Learning
para el aprendizaje con datos desbalanceados

Alumno: Lucía Olmos Vela

Tutores: Anibal Bregón Bregón
Paula Mielgo Martín

Fecha: 25 de junio de 2024

Soluciones Deep Learning para el aprendizaje con datos desbalanceados

Lucía Olmos Vela

25 de junio de 2024

Todas las cosas son imposibles,
mientras lo parecen.

Concepción Arenal

Resumen

Conseguir un buen rendimiento de los algoritmos de clasificación cuando se trabaja con datos desbalanceados es una labor difícil. En los últimos años ha sido creciente la investigación en este campo, debido a la relevancia que adquiere conseguir clasificar correctamente elementos anómalos o poco comunes, en especial en entornos médicos, financieros o industriales.

Este trabajo explora distintas técnicas y enfoques, centrándose posteriormente en aquellas particularizadas a conjuntos de datos compuestos por imágenes. Se estudian diferentes algoritmos de balanceo dentro del ámbito del *Deep Learning* que buscan hacer frente a esta problemática del desbalanceo de clases mejorando así el rendimiento en las tareas de clasificación.

Las técnicas empleadas son DeepSMOTE y BAGAN-GP. DeepSMOTE combina SMOTE, método de resampling, con el *DeepLearning* generando imágenes de calidad. Por otra parte, BAGAN-GP deriva del método BAGAN, ambos son modelos de balanceo basados en redes generativas adversarias. Tras un proceso de análisis, este trabajo muestra que el empleo tanto de BAGAN-GP como de DeepSMOTE mejora la tarea de clasificación, siendo esta última la técnica que arroja los mejores resultados.

Palabras claves: Aprendizaje profundo, Desbalanceo de clases, Datos sintéticos.

Abstract

Achieving good performance of classification algorithms when working with imbalanced data is a challenging task. In recent years, research in this field has been growing due to the importance of correctly classifying anomalous or uncommon elements, especially in medical, financial, or industrial environments.

This work explores various techniques and approaches, focusing later on those specific to datasets composed of images. Different balancing algorithms within the field of Deep Learning are studied, aiming to address the issue of class imbalance and consequently improve performance in classification tasks.

The techniques employed are DeepSMOTE and BAGAN-GP. DeepSMOTE combines SMOTE, a resampling method, with Deep Learning to generate high-quality images. BAGAN-GP, on the other hand, is derived from the BAGAN method; both are balancing models based on generative adversarial networks. After an analysis process, this work shows that the use of both BAGAN-GP and DeepSMOTE improves the classification task, with DeepSMOTE achieving the best results.

Keywords: Deep Learning, Unbalanced classes, Syntetic data.

Índice general

Índice de figuras

Índice de tablas

Parte I

Descripción del proyecto

Capítulo 1

Introducción

El aprendizaje automático o *Machine Learning* es la rama de la Inteligencia Artificial que se preocupa del diseño y desarrollo de algoritmos que permiten a los ordenadores mejorar su desempeño en la realización de una tarea a partir de la experiencia [69]. Estos algoritmos son capaces de identificar la distribución de los datos, aprender de ellos y realizar predicciones o tomar decisiones. Es por esto que el *Machine Learning* ha transformado radicalmente la forma en la cual interactuamos con la tecnología y cómo abordamos los desafíos del mundo moderno desde su aparición en la década de los años cincuenta. Actualmente se encuentra como base de infinidad de aplicaciones como por ejemplo de reconocimiento de voz [2], visión computacional [7], procesamiento de lenguaje natural [60], robótica [64], conducción autónoma [5], ... Por otra parte, un área al que da solución este campo y que va a ser de interés en este trabajo es al de, a partir de un conjunto de datos que pertenecen a distintas clases, conseguir asignar la clase o grupo correspondiente a elementos nuevos que no la posean. Los algoritmos que desempeñan esta labor se denominan algoritmos de *clasificación*.

El primer paso para la elaboración de un algoritmo de clasificación eficiente es el entrenamiento. En esta etapa se prepara el conjunto de datos inicial con el que se trabaja y se separa en al menos dos subconjuntos: conjunto de entrenamiento y prueba, siendo el de entrenamiento el que posee el mayor porcentaje de elementos. El objetivo del entrenamiento es que el modelo aprenda la distribución de los datos de para así posteriormente, ser capaz de generalizar el aprendizaje y clasificar elementos nunca vistos (pertenecientes al conjunto de prueba) en una de las clases existentes. Es por esto que tanto el proceso de entrenamiento de un modelo como el conjunto de datos introducido juegan una labor fundamental en el correcto rendimiento del mismo. Trabajar con datos reales supone tener que hacer frente a inconsistencias, valores faltantes o escasez de elementos. El objetivo por lo tanto se basa en ser capaces de construir modelos robustos, capaces de clasificar con el menor error posible cada uno de los elementos requeridos. Un problema común para estos modelos de clasificación sucede cuando el conjunto de datos original no tiene un número de elementos semejante para cada una de las clases existentes. Si esto ocurre se habla de conjunto de datos desbalanceado. Los modelos de clasificación no serán capaces de asignar la clase correcta a los elementos que pertenecen a las clases minoritarias. Esta situación es frecuente en entornos médicos para el diagnóstico y pronóstico [57] o predicción de fracasos en implantes dentales [27], industriales en la detección de incidencias [55], digitales como la detección de fraude [21] o ciberbullying [16] o financieros como el hecho de detectar los clientes que se darán de baja [38].

Por ejemplificar esta situación, se plantea el escenario en el cual una empresa quiere descubrir

con antelación si un determinado cliente se va a dar de baja y qué situaciones influyen en la decisión. Por norma general, el número de clientes que se da de baja es notablemente menor (9 %) que el número de clientes que no se dan de baja (91 %, datos obtenidos de [38]). Aplicando a este conjunto de datos un algoritmo de *Machine Learning*, como por ejemplo una regresión logística, se obtiene una alta precisión, del 91 % evidenciando que ha sido capaz de asignar la clase correcta a la mayoría de los elementos. Pero al analizar el número de clientes que el algoritmo ha predicho que se han dado de baja y que realmente se han dado de baja, éste es del 0 % significando que no ha sido capaz de detectar ni un solo cliente de este tipo. Luego no es un modelo robusto que consiga prever las pérdidas futuras para una empresa.

Modelos que no sepan hacer frente a un conjunto de datos desbalanceados suponen un perjuicio económico como en el caso anterior, o referente a la salud como ocurre en entornos médicos, donde en muchas ocasiones resulta crucial ser capaces de detectar una enfermedad como puede ser un tumor. Esta dificultad ha constituido un importante campo de estudio durante las dos últimas décadas, siendo a día de hoy aún un área de constantes avances.

1.1. Planteamiento del problema (*Problem Statement*)

Es común encontrarse con situaciones en las cuales queramos ser capaces de detectar comportamientos, elementos o patrones poco usuales. Conseguir detectar estos elementos anómalos constituye un problema para los algoritmos de clasificación puesto que éstos asumen que el número de elementos presentes en cada una de las clases es similar. El caso idílico que contemplan estos algoritmos de clasificación generalmente no se encuentra en la vida real, donde es frecuente encontrarse frente a un conjunto de datos donde predomina el número de elementos de una o más clases en contraposición con la escasez de datos presentes en otras clases minoritarias y donde las características de interés se encuentran en estas clases minoritarias.

Se dice que un conjunto de datos se encuentra desbalanceado si las clases existentes no presentan un tamaño similar. En estos casos existe un conjunto de clases con un número notablemente mayor de elementos con respecto a las restantes. Las primeras se denominan clases mayoritarias, y las segundas, minoritarias. Los algoritmos de clasificación ante un conjunto de datos desbalanceado se inclinan por la correcta clasificación de los elementos pertenecientes a la clase o clases mayoritarias, sin ser capaces de clasificar los elementos pertenecientes a las clases minoritarias. El objetivo en estos casos es paliar el coste de clasificar incorrectamente un elemento de la clase minoritaria, puesto que generalmente es mayor que el coste opuesto, el de clasificar incorrectamente un elemento perteneciente a la clase mayoritaria.

Las soluciones propuestas son diversas pero generalmente se engloban en dos grandes grupos, por un lado aquellas en las cuales se pretende balancear el conjunto de datos mediante una técnica de *resampling*, eliminando elementos o generando datos sintéticos. En este grupo se encuentran técnicas como *Edited Nearest Neighbours* [85], SMOTE [14] o ADASYN [34], entre otras muchas. Por otra parte se tiene aquellas técnicas capaces de construir una función de pérdida o asignar diversos costes a un error en la clasificación en función de la clase a la que pertenezca el elemento. De esta manera se asignan costes elevados a la incorrecta clasificación de los elementos de la clase o clases minoritarias. El propósito del algoritmo es por lo tanto minimizar el coste.

Este trabajo se centra fundamentalmente el empleo de conjuntos de datos compuesto por imágenes, estudiando diferentes algoritmos de balanceo dentro del ámbito del *Deep Learning* que buscan hacer frente a la problemática del desbalanceo de clases para mejorar el rendimiento en

las tareas de clasificación. Se busca priorizar la obtención de un alto porcentaje en la detección de la clase minoritaria junto con el menor rango de error que se pueda obtener en la detección de la clase mayoritaria.

1.2. Objetivos del trabajo

Los principales objetivos que abarca este proyecto se describen a continuación:

- **OBJ-01** Explorar distintas técnicas de balanceo de datos.
- **OBJ-02** Profundizar en las técnicas de balanceo de datos dentro del ámbito del *Deep Learning* particularizando al caso en el cual se trabaja con conjuntos de datos de imágenes.
- **OBJ-03** Trabajar con DeepSMOTE y BAGAN-GP comparando y evaluando el desempeño y rendimiento de cada una.
- **OBJ-04** Adquirir un conocimiento sólido sobre redes convolucionales para labores de clasificación.

1.2.1. Restricciones

Posee a su vez un conjunto de restricciones:

- **R-01** El alcance del proyecto se encuentra limitado por la carga de trabajo establecida de 12 ECTS.

1.3. Estructura de la memoria

Todo lo expuesto se va a estructurar en un total de seis capítulos y tres apéndices.

- **Capítulo 1. Introducción**: expone de forma clara el problema de desbalanceo de clases al que pretende buscar solución este TFG. Se acompaña de los objetivos y las restricciones existentes.
- **Capítulo 2. Planificación**: para la correcta consecución de este proyecto es necesario seguir una metodología. Este capítulo desglosa todo lo referente a la metodología ASAP escogida, la planificación inicial en tiempo y presupuestos y el balance real.
- **Capítulo 3. Antecedentes**: se detalla cuidadosamente el contexto científico-técnico donde se abordan los conceptos de *Machine Learning* y *Deep Learning*. Por otra parte se muestra el estado del arte, algoritmos que en mayor o menor medida dan solución a la problemática tratada.
- **Capítulo 4. Desarrollo de la propuesta y experimentación**: muestra el proceso de entrenamiento y generación de imágenes de las dos técnicas escogidas: DeepSMOTE y BAGAN-GP. Queda a su vez reflejado el proceso de ajuste de hiperparámetros mediante la técnica *GridSearch*, para conseguir la correcta combinación que minimice la función de pérdida del entrenamiento de los modelos.

- **Capítulo 5. Evaluación**: detalla la evaluación llevada a cabo con distintas redes convolucionales, mostrando de forma exhaustiva el rendimiento de los modelos generativos estudiados.
- **Capítulo 6. Conclusiones y trabajo futuro**: este capítulo realiza un balance tanto a nivel técnico como a personal del desarrollo llevado a cabo.
- **Apéndice A. Manual de Instalación**
- **Apéndice B. Contenido adjunto**

Capítulo 2

Planificación

2.1. Metodología de trabajo

La metodología empleada es ASAP (*Agile Student Academic Projects*) [54]. Es una metodología para la organización de Trabajos Fin de Estudios surgida recientemente y englobada en el marco de las metodologías ágiles. Su principal objetivo es la consecución de los objetivos definidos en el proyecto, asegurando durante el proceso la implicación y comunicación de todos los agentes implicados a fin de maximizar la calidad del producto construido, y en los términos establecidos por la normativa de la titulación para los Trabajos Fin de Grado (TFG).

ASAP plantea un conjunto de objetivos que estructuran de forma genérica el trabajo necesario para completar cualquier proyecto de TFG en los términos indicados más arriba, y proporcionan una visión concisa y de alto nivel del proyecto. Los objetivos de aprendizaje, a su vez, se dividen en un conjunto de historias de aprendizaje, que proporcionan una especificación más detallada de las tareas a realizar, y permiten una mayor subdivisión del trabajo requerido para completar el proyecto. A su vez, las historias poseen uno o varios criterios de aceptación, que describen los resultados concretos que la realización de la historia de aprendizaje debería arrojar. Para considerarse completada, todos sus criterios de aceptación deben haber sido satisfechos y verificados.

Esta metodología propone cinco objetivos de aprendizaje:

- **Proyecto**: consiste en definir el planteamiento del problema y la planificación necesaria para llevarlo a cabo. Es un proceso iterativo-incremental para conseguir ajustar el ritmo de trabajo y el progreso del estudiante a las horas establecidas para la realización de un TFG.

- **Antecedentes**: abarca las historias referentes a comprender y profundizar en el aprendizaje del contexto en el cual se enmarca el proyecto. Ser capaces de asimilar tanto el entorno de negocio como el contexto científico-técnico permite entender la necesidad específica para la cual se desarrolla el proyecto y las soluciones ya existentes, así como poseer una visión global de los conceptos que subyacen a los empleados en el desarrollo.

- **Desarrollo**: se enfoca en la elaboración del producto propiamente dicha. Las historias de aprendizaje de este objetivo son acordes a la naturaleza de proyecto realizado, investigación o desarrollo.

- **Aceptación**: se encarga de validar si el producto cumple con los objetivos establecidos. A su vez, el estudiante también se encarga de hacer un análisis crítico de las técnicas empleadas en relación al cumplimiento de los objetivos del proyecto.
- **Comunicación**: las historias de aprendizaje que abarca este objetivo son dos, la memoria técnica y el acto de defensa. La memoria técnica aúna el aprendizaje y las conclusiones obtenidos en los objetivos previos. Por su parte, en el acto de defensa el estudiante expone el trabajo desarrollado a un tribunal evaluador. Tanto la memoria como la defensa se realizan de forma iterativa a lo largo del desarrollo del proyecto, obteniendo retroalimentación por parte del tutor o tutores, para conseguir paulatinamente perfilar el producto y conseguir un resultado de alta calidad.

ASAP fundamenta sus herramientas en las ya definidas por Scrum [75]. De esta forma cuenta con una serie de roles, eventos y artefactos.

2.1.1. Roles

La sinergia entre los roles (estudiante, tutor, comunidad y tribunal) proporciona un mayor control durante la elaboración del producto a entregar, dotándolo así de mayor excelencia y precisión.

- **Estudiante**: es el rol fundamental, encargado de realizar cada una de las tareas específicas para la consecución de las historias y objetivos. Su proceso de aprendizaje se ve facilitado y ampliado con la generación de *feedback* por parte de los roles restantes. A su vez debe ser capaz de llevar a cabo tareas de planificación, manteniendo actualizado tanto el tablero del proyecto como el cuaderno de trabajo.
- **Tutor**: es el encargado de supervisar al estudiante durante su proceso formativo, brindando soporte y el *feedback* oportuno. Cuando mayor relevancia tiene es en las primeras fases del proyecto, puesto que se encarga de definir, junto con el estudiante, el planteamiento del problema y el alcance de los objetivos, así como de proporcionar referencias bibliográficas para encauzar la línea de avance del estudiante. Además va constatando el ritmo de avance del estudiante, redefiniendo los objetivos acordes al progreso realizado.
- **Comunidad**: está formada por todas las personas (estudiantes, profesores, expertos, ...) que posean la capacidad de agregar valor al proyecto durante su desarrollo. Conseguir una interacción entre todos los miembros fomenta el establecimiento de un ambiente de aprendizaje, en el cual se valoran los distintos puntos de vistas aportados. Donde mayor presencia toman es en los canales de comunicación y en las comunicaciones de progresos.
- **Tribunal**: toman relevancia en el acto de defensa del proyecto y en la posterior evaluación acorde al grado de satisfacción que observen tanto en los elementos entregados como del acto de defensa realizada por parte del estudiante.

2.1.2. Eventos

La organización temporal del trabajo se distribuye en un conjunto de *sprints* de corta duración. Cada *sprint* establece las historias de aprendizaje que deben ser cumplidas durante su trascurso. De esta forma consigue ir estableciendo objetivos paulatinamente y a corto plazo,

favoreciendo la capacidad de encauzar y reelaborar los mismos al inicio de cada *sprint*. Con el mismo propósito, al final de cada *sprint* se realiza una entrega de la cual se obtendrá una retroalimentación. Cada uno de estos *sprints* está formado por una serie de eventos:

- **Reunión de inicio**: es la primera de las reuniones realizadas durante el *sprint*. Es tarea del tutor definir el alcance para posteriormente consensuar con el estudiante la carga de trabajo, es decir, los objetivos e historias de aprendizaje tratados. Ya es función del estudiante definir las tareas específicas que posee cada historia de aprendizaje junto con su duración temporal.
- **Reunión de sincronización**: se realiza de forma semanal con una duración aproximada de quince minutos. En ella el estudiante da a conocer al tutor los avances realizados en la semana previa y detalla el trabajo que prevé abordar en la próxima semana. También tiene la función de que el estudiante plantee dudas y bloqueos para obtener consejo y solución a ellos.
- **Comunicación de progresos**: se desarrolla al final del *sprint*. El estudiante debe realizar una exposición del proyecto mostrando el trabajo consolidado hasta el sprint que se finaliza; es decir, el producto parcial construido hasta ese momento. Se asemeja al acto final de defensa, sirviendo de preparación al estudiante para el mismo. Tras la exposición, la comunidad es la encargada de valorar el trabajo presentado, aportando sugerencias y correcciones. En esta misma reunión generalmente otros estudiantes en la misma situación exponen su trabajo. Por lo tanto, un mismo estudiante adopta la posición de orador de su trabajo y de miembro de la comunidad de otros compañeros.
- **Retrospectiva**: constituye el último evento, en el que de forma anónima se realiza una evaluación de los aspectos positivos y negativos del *sprint* y posibles acciones para su mejora.

2.1.3. Artefactos

- **Incremento**: está constituido por un conjunto de entregables que representan el trabajo realizado hasta ese momento.
- **Retroalimentación**: es la corrección o *feedback* que recibe el estudiante por parte del tutor al acabar cada *sprint*. También forma parte de la retroalimentación las contribuciones de la comunidad en las ceremonias de Comunicación de progresos. De esta forma el estudiante tiene un punto de partida para ampliar o rectificar las tareas realizadas.

2.1.4. Entorno de trabajo

- **Espacio de trabajo compartido**: es un espacio de comunicación que permite tanto al tutor y al estudiante comunicarse de forma privada. Posee además un canal general para el intercambio de información con la comunidad. Debe albergar a su vez espacio de almacenamiento común para la compartición de los resultados alcanzados por el estudiante y el feedback del profesor, después. El repositorio está dividido en cuatro secciones: "documentación" la cual contiene información sobre la gestión académica del TFG, "incrementos" para almacenar el trabajo realizado al finalizar cada *sprint*, "personal" donde el estudiante

posee su espacio de trabajo para gestionarlo como desee y "recursos" que albergan referencias bibliográficas o información útil para el desarrollo del proyecto. Es de uso habitual emplear la plataforma *Teams* para este propósito.

- **Tablero de proyecto**: es un tablero tipo Kanban [66]. Este tipo de tablero posee tres columnas: en la primera se colocan las tareas que aún no se han realizado (To Do), en la segunda aquellas que se encuentran en progreso (Doing) y en la última las ya finalizadas (Done). Esta herramienta permite entonces mostrar el progreso y planificación de cada uno de los objetivos, puntos de historia y tareas específicas a realizar en cada uno de los *sprints*. Una herramienta útil para esta labor es *Trello* (ver Figura 2.1).

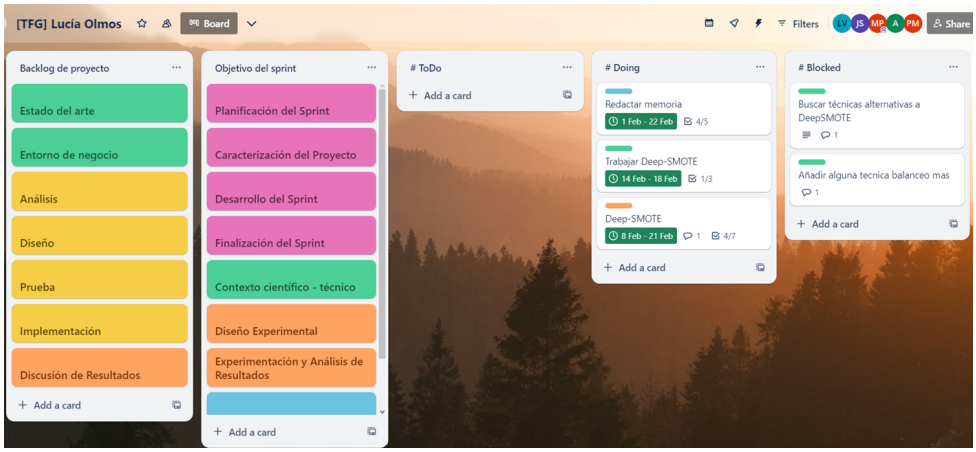


Figura 2.1: Espacio de trabajo: Trello

- **Cuaderno de trabajo**: es una herramienta de utilidad para el estudiante, en la cual registra el tiempo invertido en cada una de la tareas programadas. Favorece la implantación de una rutina de trabajo.

Alumno	Lucía Olmos Vela		TFG	Soluciones Deep Learning para el aprendizaje con datos desbalanceados
Fecha	Sprint	Actividad		Tiempo	Descripción
		Tipo	Tarea		
dd/mm/aa	Sprint #1	Dinámica de Trabajo	Preparar Incremento	hh:mm:ss	Descripción de la actividad realizada
01-12-23	Sprint #1	Dinámica de Trabajo	Otra Reunión	0:30:00	Reunión de inicio
02-12-23	Sprint #1	Lectura de Documentación		0:20:00	Lectura metodología de trabajo
03-12-23	Sprint #1	Lectura de Documentación		1:10:00	Lectura metodología de trabajo
04-12-23	Sprint #1	Lectura de Documentación		1:30:00	Búsqueda artículos y lectura técnicas de balanceo de datos
06-12-23	Sprint #1	Lectura de Documentación		0:45:00	Búsqueda artículos y lectura técnicas de balanceo de datos
07-12-23	Sprint #1	Lectura de Documentación		3:00:00	Búsqueda artículos y lectura técnicas de balanceo de datos
07-12-23	Sprint #1	Redacción de Memoria		0:20:00	Redacción memoria técnicas de balanceo de datos
08-12-23	Sprint #1	Redacción de Memoria		3:30:00	Lectura y redacción técnicas remuestreo
09-12-23	Sprint #1	Redacción de Memoria		3:30:00	Lectura y redacción técnicas remuestreo
11-12-23	Sprint #1	Dinámica de Trabajo	Reunión Semanal	0:15:00	Reunión sincronización
11-12-23	Sprint #1	Lectura de Documentación		1:30:00	Lectura SMOTE
11-12-23	Sprint #1	Redacción de Memoria		2:30:00	Lectura y redacción técnicas remuestreo
12-12-23	Sprint #1	Redacción de Memoria		1:30:00	Lectura y redacción SMOTE
13-12-23	Sprint #1	Redacción de Memoria		0:45:00	Lectura y redacción SMOTE y ADASYN
13-12-23	Sprint #1	Redacción de Memoria		2:30:00	Añadir ejemplos a las técnicas de balanceo
14-12-23	Sprint #1	Redacción de Memoria		0:45:00	Lectura y redaccíión de ADASYN
19-12-23	Sprint #1	Dinámica de Trabajo	Reunión Semanal	0:10:00	Reunión sincronización
21-12-23	Sprint #1	Lectura de Documentación		2:00:00	Lectura BAGAN
22-12-23	Sprint #1	Lectura de Documentación		1:00:00	Lectura BAGAN
23-12-23	Sprint #1	Redacción de Memoria		2:00:00	Redacción BAGAN

Figura 2.2: Cuaderno de trabajo

2.2. Planificación temporal

La planificación inicial abarca el periodo comprendido entre el 4 de diciembre de 2023 y el 15 de mayo de 2024. Se divide en cuatro *sprints*, cada uno de ellos planificado para que abarque entre 75 y 90 horas de trabajo:

- ***Sprint* #1**: 4 de diciembre - 26 de enero
- ***Sprint* #2**: 22 de enero - 1 de marzo
- ***Sprint* #3**: 26 de febrero - 12 de abril
- ***Sprint* #4**: 8 de abril - 15 de mayo

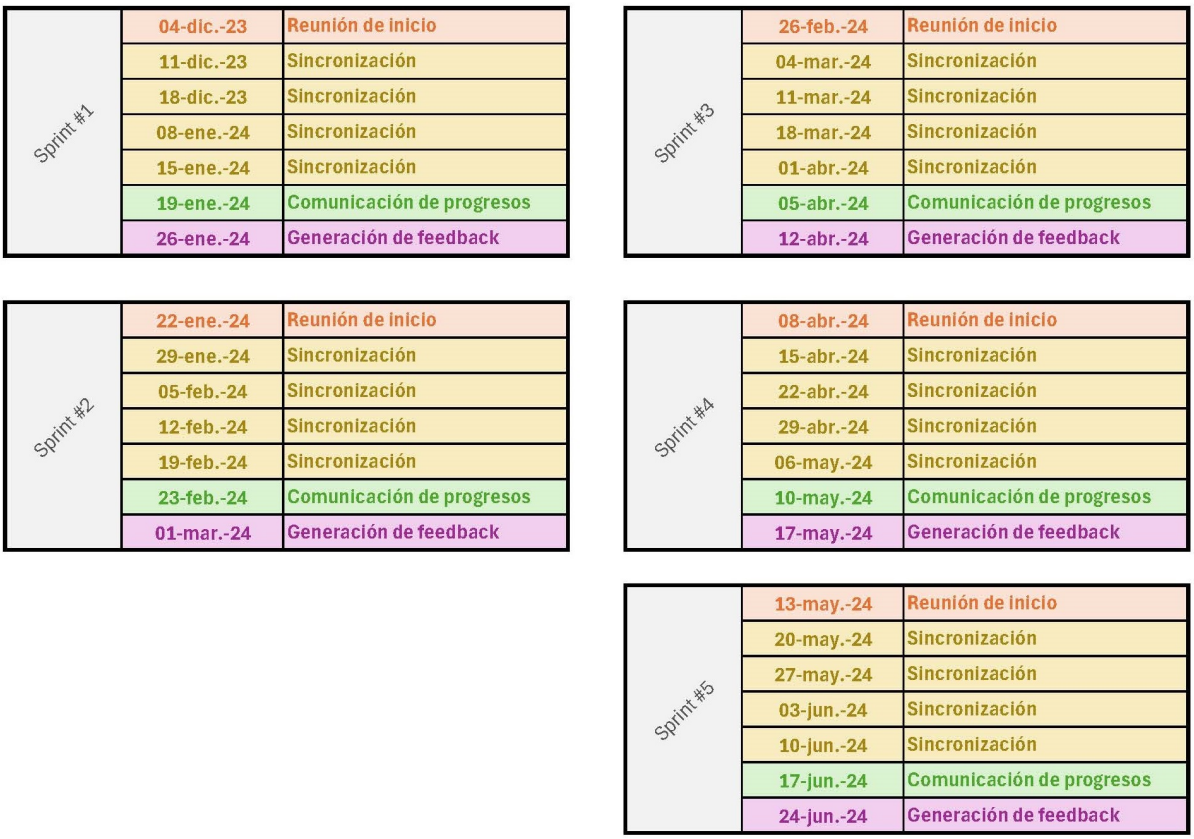


Figura 2.3: Planificación inicial

Aunque inicialmente se planifica para abordarlo en cuatro *sprints*, en la Figura 2.3 se observan cinco. El quinto sprint aparece como un *sprint* auxiliar de menor duración enfocado en la finalización de los objetivos pendientes y hasta completar las 300-360 horas establecidas.

A continuación se detalla la planificación y el alcance individual de cada uno de los *sprints* haciendo uso de una EDT (Estructura de Desglose de Trabajo) y reflejando la relación entre las actividades, su secuenciación y duración mediante un cronograma, en este caso, un diagrama de Gantt.

Los objetivos a abordar son Proyecto, Antecedentes, Desarrollo y Comunicación, representados en la EDT en color rosa, verde, naranja y azul respectivamente. Cada objetivo abarca un conjunto de historias concretas:

- **Proyecto**
  - **Planificación del *sprint***: abarca las tareas relacionadas con la organización del *sprint*, definiendo el alcance y los plazos temporales de cada una de las tareas concretas existentes.
  - **Caracterización del Proyecto**: engloba la gestión del propio proyecto, poseyendo de esta forma las tareas relacionadas con la definición de la motivación, los objetivos, la metodología, la planificación, y las conclusiones del proyecto.
  - **Desarrollo del *sprint***: de esta historia se derivan las tareas relativas al seguimiento y actualización del *sprint*. Forman parte de este punto las reuniones semanales y la actualización del tablero del proyecto.
  - **Finalización del *sprint***: abarca aquellas tareas referentes a la entrega de los materiales generados durante el *sprint*, la realización de una comunicación de progresos y la posterior retrospectiva. Además los tutores generan un *feedback* para orientar al estudiante y solventar los errores.
- **Antecedentes**
  - **Estado del arte**: esta historia recoge las tareas referentes a la investigación de trabajos relacionados con el área de estudio, para de esta forma conocer las ventajas y limitaciones de cada una de ellas, y justificar la necesidad de investigar y elaborar este trabajo.
  - **Contexto científico - técnico**: proporciona el marco teórico general sobre el cual estamos trabajando. Las tareas están enfocadas en desarrollar los contenidos teóricos necesarios para el entendimiento del trabajo expuesto.
- **Desarrollo**
  - **Diseño Experimental**: esta historia se encarga de establecer pruebas con las que verificar si la solución desarrollada cumple con los objetivos deseados. Para ello las tareas de este punto abarcan la definición de los parámetros del experimento, de los conjuntos de prueba y las métricas necesarias.
  - **Construcción**: abarca las tareas referentes a la implementación del producto.
  - **Experimentación y Análisis de Resultados**: engloba las tareas referentes a la evaluación del cumplimiento o no de los objetivos definidos.
- **Comunicación**
  - **Presentación**: abarca la realización de una presentación, que evidencie los avances realizados en el *sprint*, como si se tratara del acto de defensa del TFG. Debe poseer una contextualización del proyecto, seguida del progreso realizado y una valoración crítica de acuerdo a la planificación y a los objetivos establecidos.
  - **Memoria**: esta historia las tareas relacionadas con la elaboración de una documentación del proyecto precisa y de calidad.

2.2.1. Sprint #1

En este *sprint* se tratan las historias y tareas pertenecientes a los objetivos: Proyecto, Antecedentes, Desarrollo y Comunicación. Se puede evidenciar en la EDT (Figura 2.4) y en el diagrama de Gantt (Figura 2.5). En general este primer *sprint* está enfocado en investigar el marco sobre el que se encuadra el proyecto, adquiriendo conocimiento sobre las soluciones existentes al problema que se plantea.

2.2.2. Sprint #2

El desglose de tareas e historias se encuentra representado en la EDT (Figura 2.4) y los intervalos de tiempo y secuencialidad de cada tarea se puede consultar en el diagrama de Gantt correspondiente (Figura 2.7). Este *sprint* ya presenta una primera aproximación al hecho de trabajar con modelos concretos de balanceo de datos, consiguiendo evaluar su rendimiento en distintos conjuntos de datos.

2.2.3. Sprint #3

Este *sprint* tiene previsto ahondar en otra de las técnicas de balanceo de datos para poder construir una comparativa y arrojar conclusiones significativas a favor o en contra de cada técnica. Las historias y tareas se encuentran recogidas en la EDT (Figura 2.8) y la planificación en el diagrama de Gantt (Figura 2.9).

2.2.4. Sprint #4

En este *sprint* se prevé el refinamiento de los métodos para conseguir mejorar las métricas obtenidas. Las tareas que se van a desempeñar se encuentran recogidas en la EDT (Figura 2.10) y la planificación en el diagrama de Gantt (Figura 2.11) correspondiente.

2.2.5. Sprint #5

Este *sprint* posee una planificación (Figura 2.13) con un menor número de horas puesto que las tareas involucradas están basadas en la finalización de aquellas que habían quedado pendientes en el *sprint* anterior. Esto se puede constatar en la EDT (Figura 2.12).

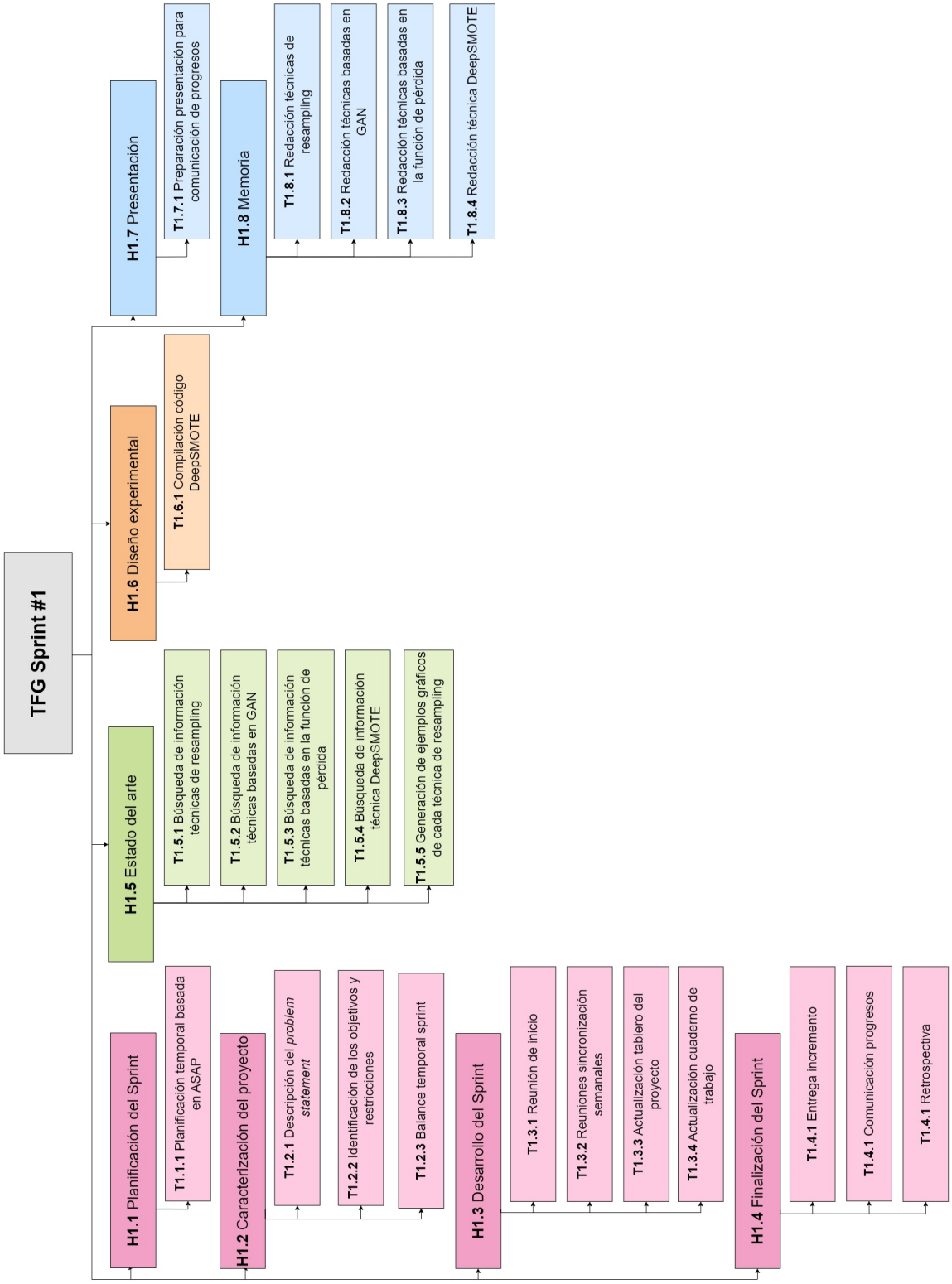


Figura 2.4: EDT *Sprint* #1

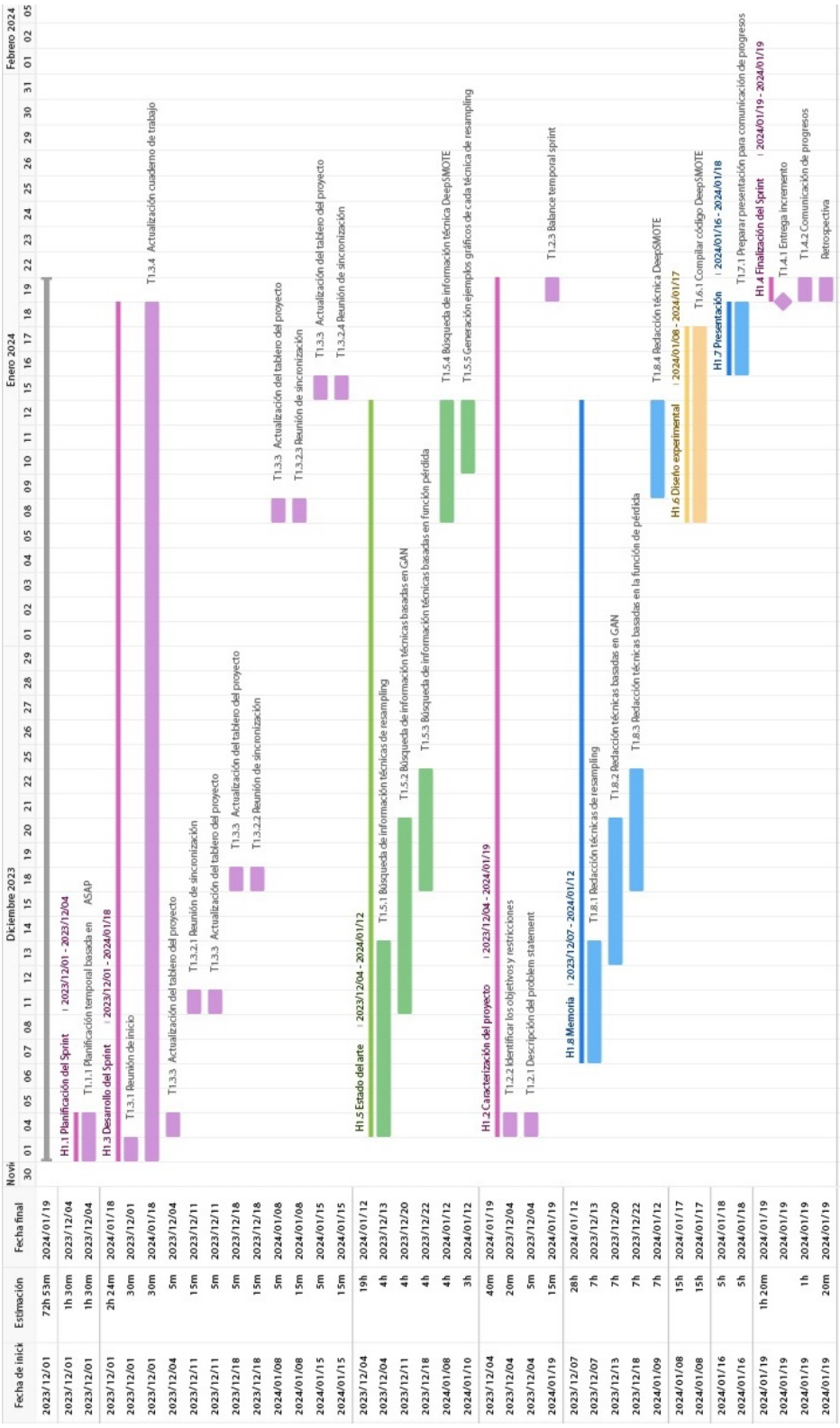


Figura 2.5: Diagrama Gantt *Sprint* #1

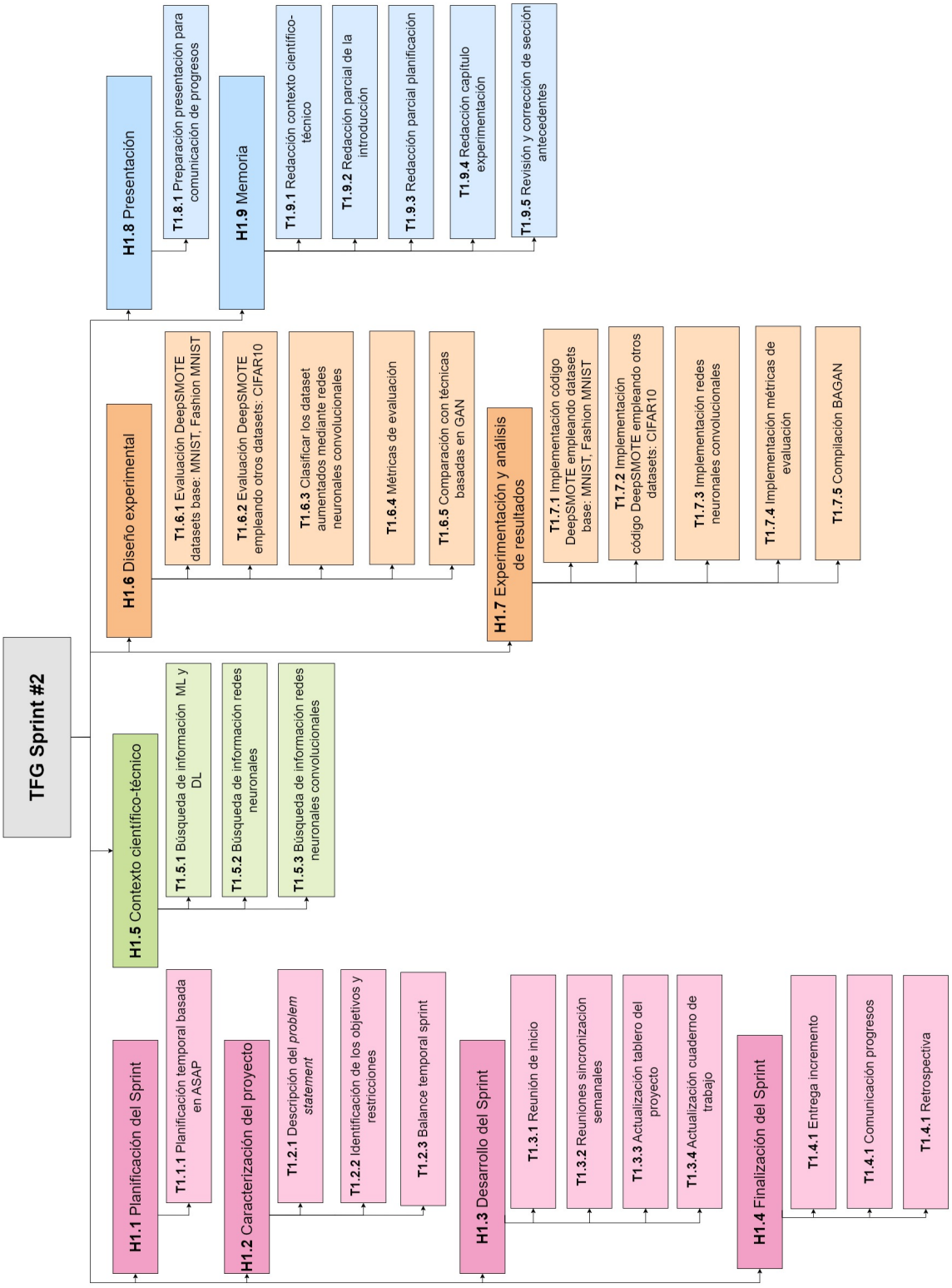


Figura 2.6: EDT *Sprint* #2

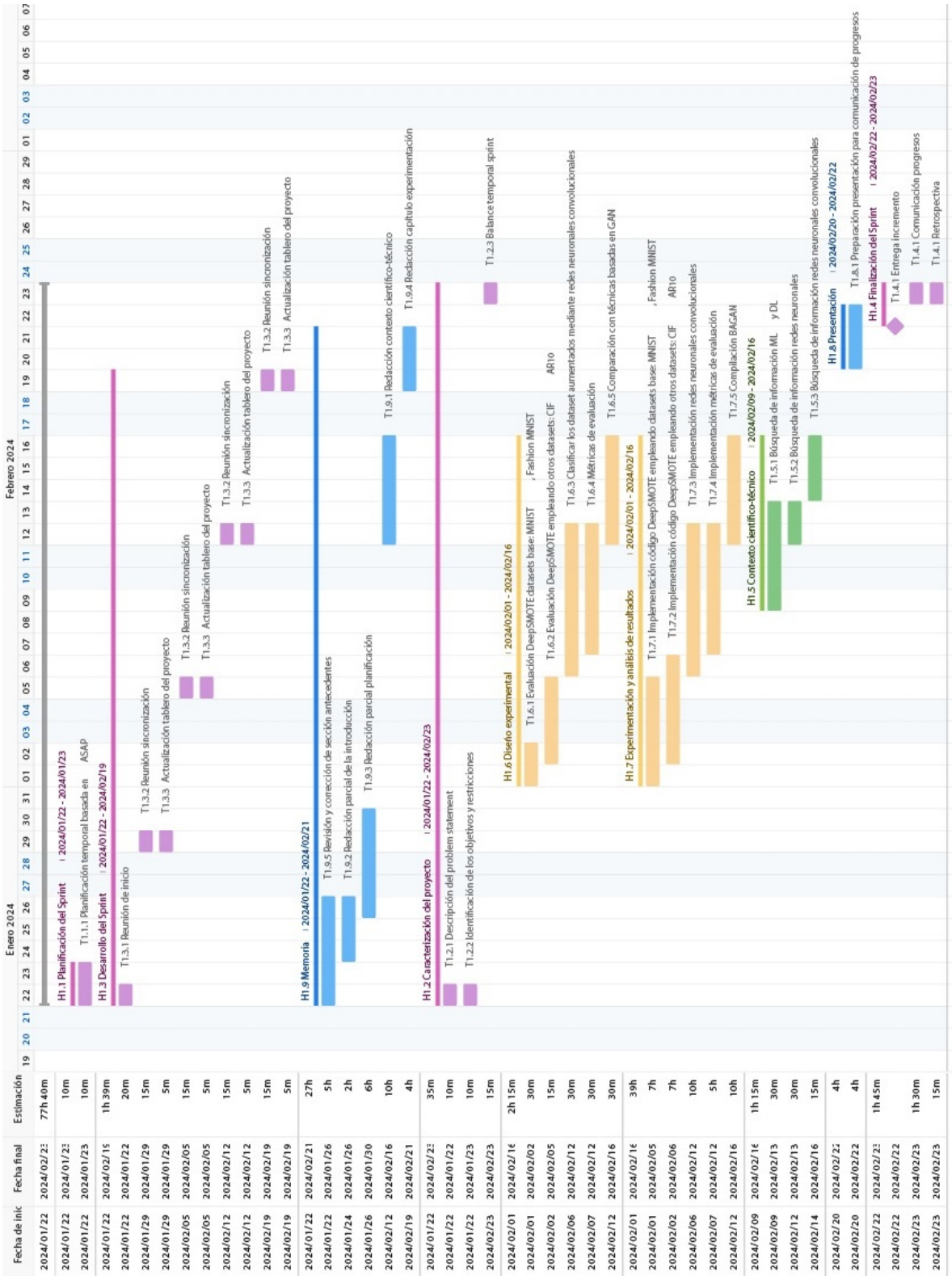


Figura 2.7: Diagrama Gantt *Sprint* #2

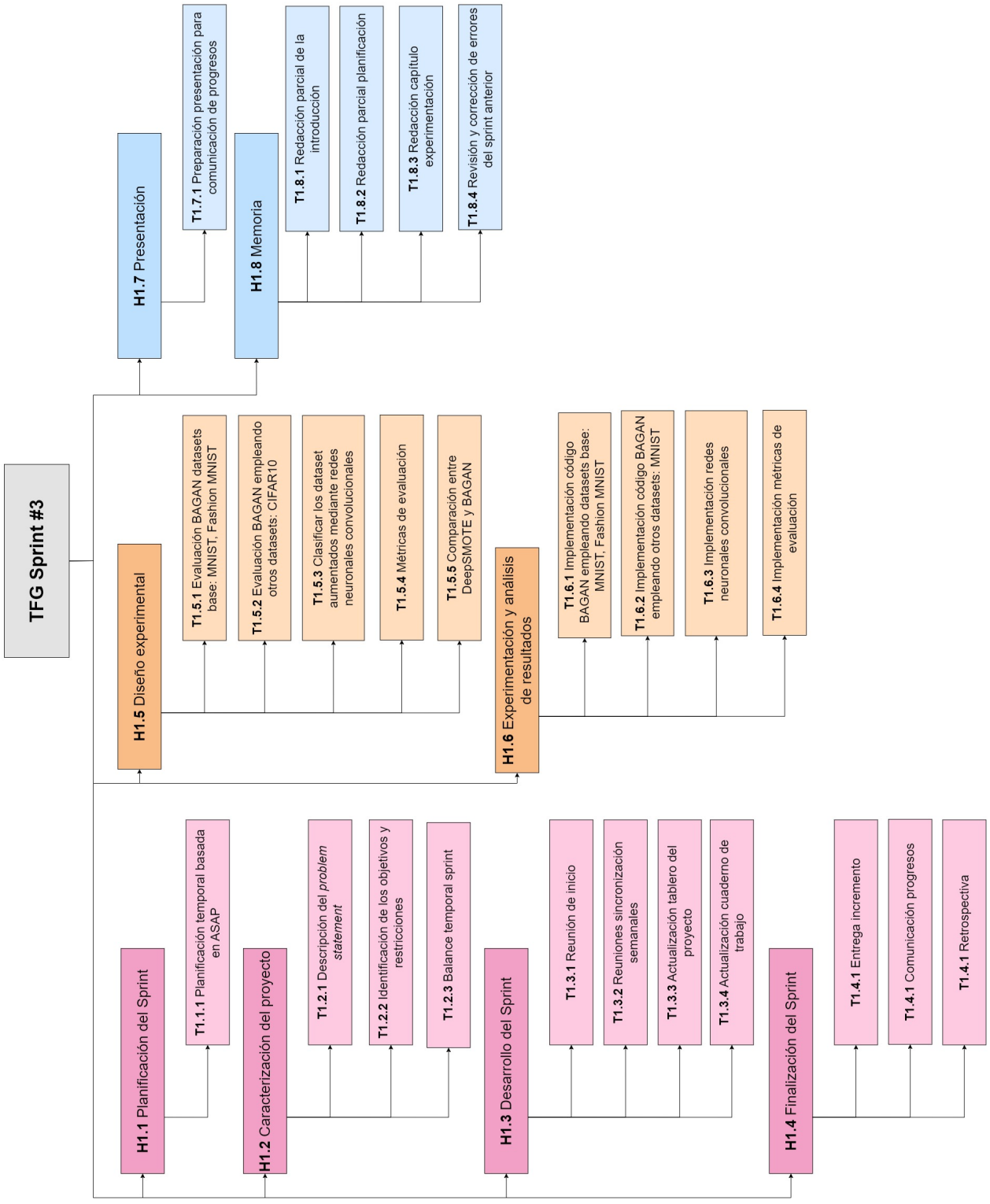


Figura 2.8: EDT *Sprint* #3

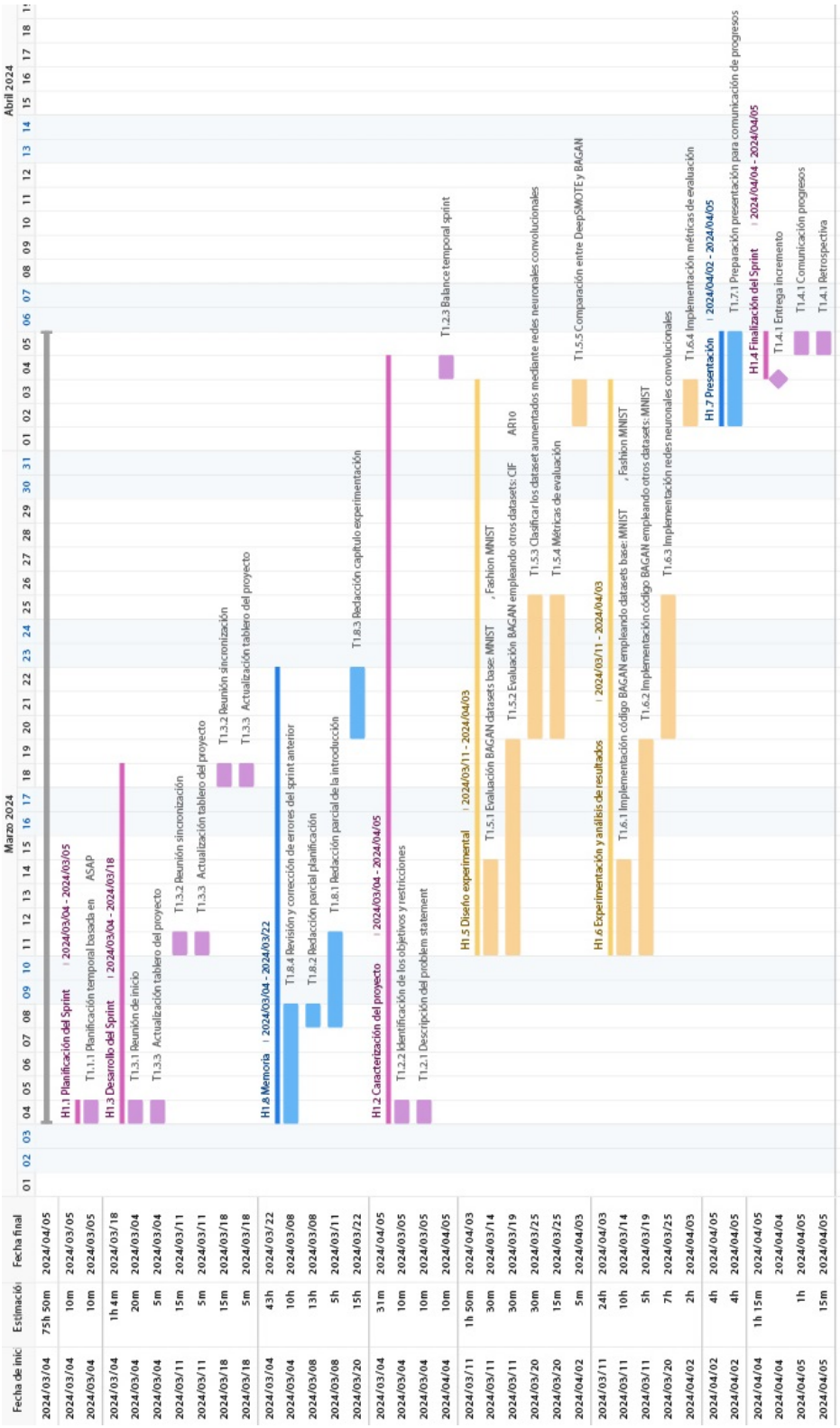


Figura 2.9: Diagrama Gantt *Sprint* #3

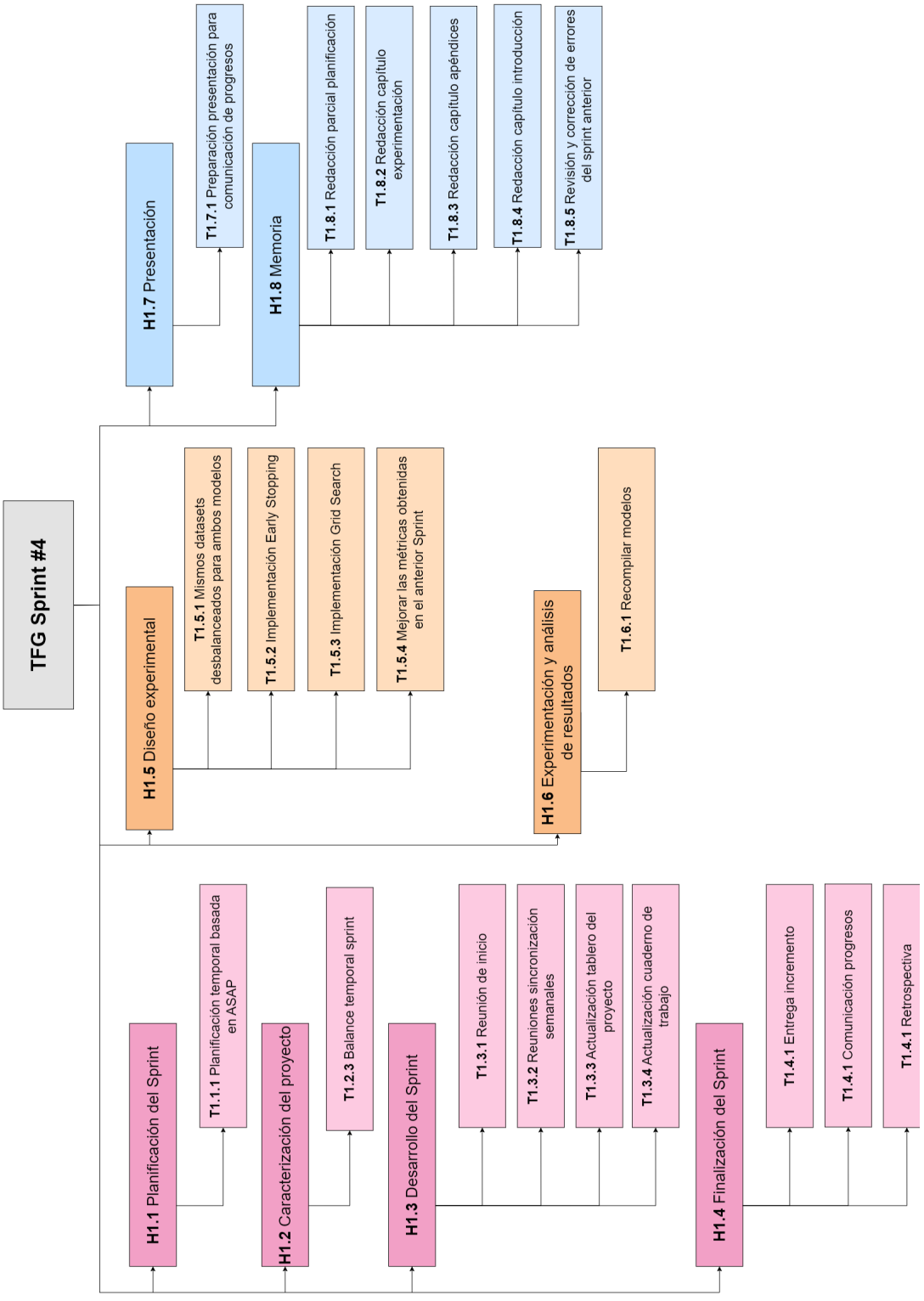


Figura 2.10: EDT *Sprint* #4

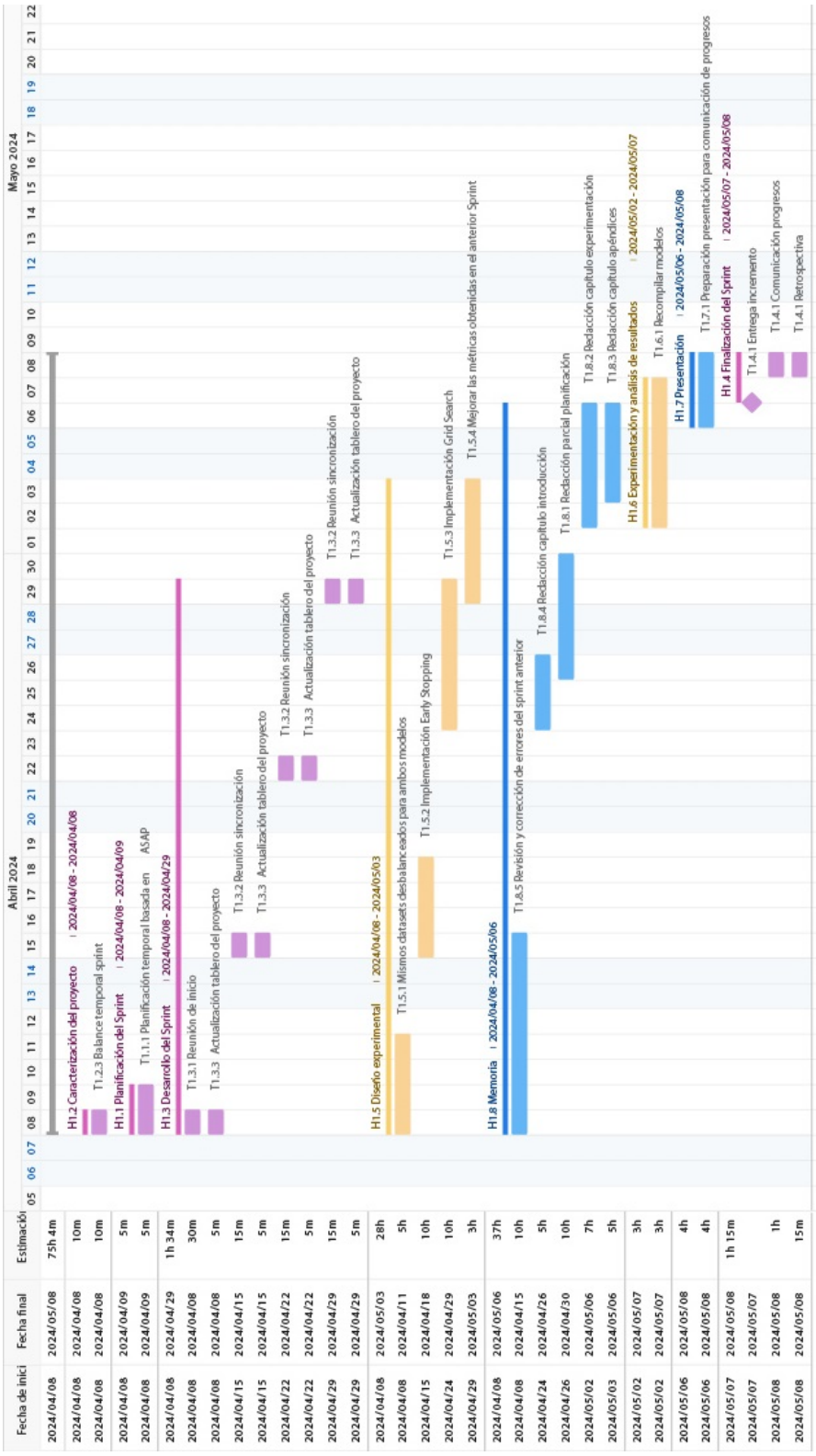


Figura 2.11: Diagrama Gantt *Sprint* #4

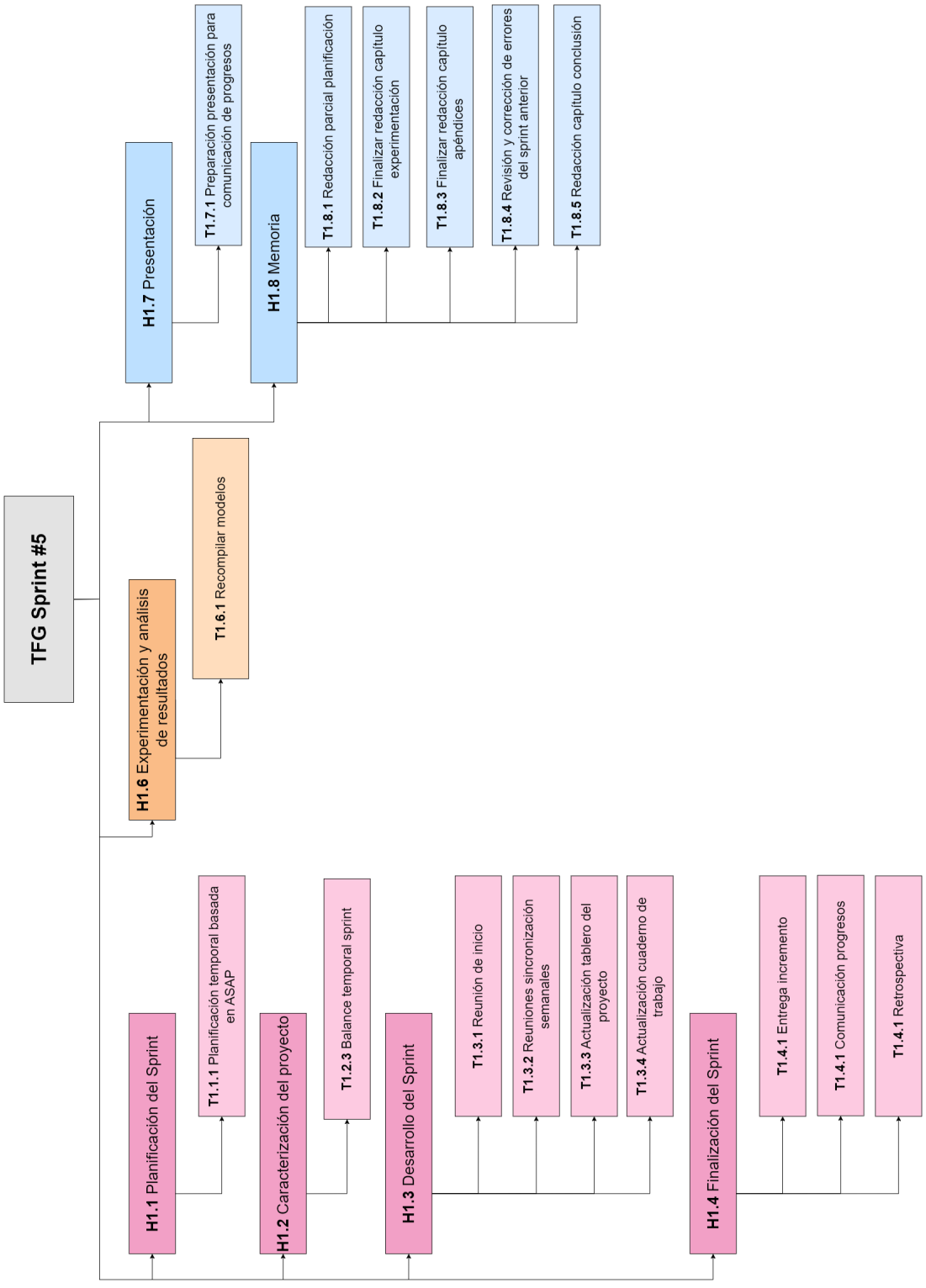


Figura 2.12: EDT *Sprint* #5

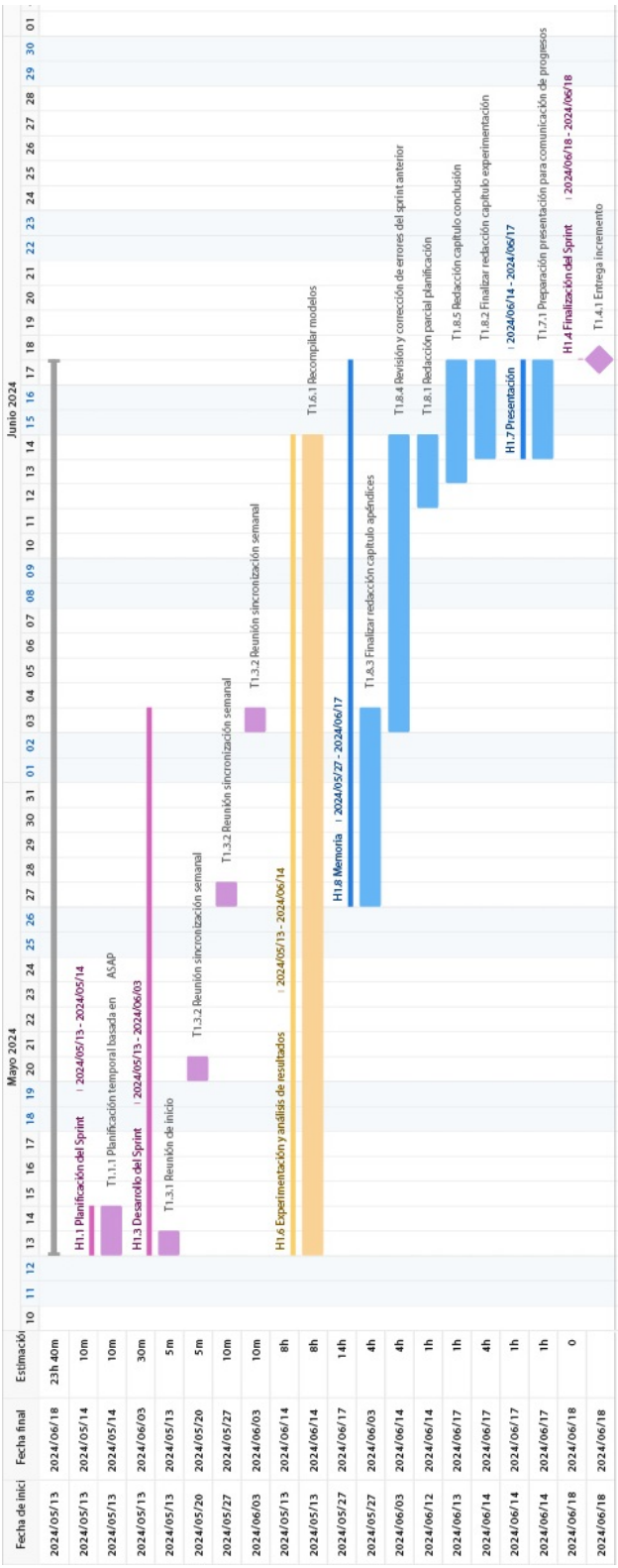


Figura 2.13: Diagrama Gantt *Sprint* #5

2.3. Presupuestos

En esta sección se presenta un análisis detallado de los presupuestos necesarios para llevar a cabo el trabajo. El presupuesto es una parte fundamental en la planificación y ejecución de cualquier proyecto, ya que permite estimar los recursos requeridos para alcanzar los objetivos planteados. En este contexto, se detallan los costos asociados a los recursos *hardware*, *software* y humanos.

Hardware

El recurso *hardware* más empleado ha sido el ordenador personal HP con procesador Intel i7-10750H, 16GB de memoria RAM y 500GB de disco. Para hacer uso de las herramientas software, en su mayoría es necesario hacer uso de una conexión Wi-Fi estable y rápida. Se va a trabajar con una zona Wi-Fi creada a partir de los datos móviles del teléfono personal, luego el coste se puede ajustar de forma más precisa conociendo el valor de los GigaBytes consumidos cada mes. Se hace una estimación promedio de 55GB al mes, teniendo en cuenta que el proyecto posee una duración de 5 meses. Previendo una limitación de recursos computacionales, para la compilación total del proyecto se va a recurrir al empleo de un servidor que alberga una GPU NVIDIA A40 de 48GB. El porcentaje de uso de cada elemento, y por tanto el coste que acarrea se encuentra resumido en la Tabla 2.1.

Componente	**Coste**	**% uso**	**Total**
Ordenador personal	1200 €	17 %	204 €
Conexión Wi-Fi	0,30 €/GB	55 GB/mes	82,5 €
Servidor	7500 €	0.05 %	375 €
GPU adicional	6004 €	0.07 %	420,28 €
			1081,78 €

Tabla 2.1: Costes *hardware*

Software

Los componentes software que se van a emplear, o bien son de libre acceso o bien poseen licencias gratuitas, en un principio suficientes para el correcto desarrollo del proyecto. Overleaf será la herramienta para redactar la memoria, *Microsoft Teams* y *Trello* se emplearán para favorecer el seguimiento, comunicación y organización. Por otra parte, *Google Colab* abastecerá al proyecto de una GPU Tesla T4, convirtiéndose así en el espacio de trabajo idóneo para construir y compilar el código necesario. *Draw.io* facilitará la labor de realizar esquemas, creando elementos vistosos y clarificadores. Para realizar los diagramas de Gantt se hará uso de *Gantt PRO*. Por otra parte, para facilitar el ajuste de hiperparámetros de un modelo se empleará *Weights and Biases*. Además, aunque se prevé usar *Python* como lenguaje de programación, *R Studio* se encuentra disponible por si fuera necesario realizar alguna compilación en R. El ordenador personal cuenta con *Windows* 10 como sistema operativo. Todo lo descrito se muestra en la Tabla 2.2.

Componente	Coste	Total
Overleaf	0 €	0 €
Trello	0 €	0 €
Microsoft Teams	0 €	0 €
Google Colab	0 €	0 €
R Studio	0 €	0 €
Draw.io	0 €	0 €
Gantt PRO	0 €	0 €
Weights and Biases	0 €	0 €
Windows 10	0 €	0 €
Ubuntu	0 €	0 €
		0 €

Tabla 2.2: Costes *software*

Recursos humanos

Los recursos humanos han sido distribuidos para un proyecto de una duración de 300 horas. En este caso el trabajo lo abarca una única persona que alterna cuatro roles distintos. La tarea del gestor de proyectos es planificar y controlar las distintas tareas, éste cuenta con un salario bruto de $10,77$€por hora. Las labores de un analista consisten en la interpretación y manipulación de los datos. Su salario por hora es de 14,36€. El científico de datos se encarga de analizar los datos y extraer conclusiones valiosas de ellos, mediante técnicas que involucran las ramas de la estadística y la inteligencia artificial. El salario por hora es de 18,43€. Por último, un programador se encarga de desarrollar el código de la aplicación necesario. Su salario es de 14,70€/hora. La Tabla 2.3 muestra el desglose de horas y la justificación del salario bruto asignado a cada puesto de trabajo.

Puesto	Salario (por hora)	Horas totales	Salario total
Gestor de proyectos	$10,77$ €/hora	20h	$215,40$ €
Analista	$14,36$ €/hora	70h	$1005,20$ €
Científico de datos	$18,43$ €/hora	110h	$2027,30$ €
Programador	$14,70$ €/hora	100h	1470 €
			$4717,90$ €

Tabla 2.3: Costes Recursos Humanos

A este coste total es necesario añadir el coste que supone dar de alta a un trabajador en la Seguridad Social. Este es el $28,30\,\%$ [8] del salario bruto. Luego el coste asociado a los recursos humanos sería:

$$C_{RRHH} = Total + SS = 4717,9 + 28,30\,\% \cdot 4717{,}9 = 6053,07 \text{ euros}$$

Costes totales

La Tabla 2.4 muestra un resumen completo del costo total estimado para la realización de este proyecto.

Concepto	Coste
Hardware	1081,78 €
Software	0 €
Recursos Humanos	6053,07 €
Total	7134,85 €

Tabla 2.4: Costes totales planificados

2.4. Gestión de riesgos

Realizar una correcta gestión de riesgos permite identificarlos y analizarlos de forma que se maximice la probabilidad de eventos positivos minimizando los negativos. Para llevar a cabo este proceso es necesario identificar los riesgos, realizar un análisis cualitativo teniendo en cuenta su probabilidad de ocurrencia e impacto, realizar un análisis cuantitativo, desarrollar planes de contingencia para mitigar las amenazas y realizar un seguimiento y control de los riesgos [42].

2.4.1. Identificar los factores de riesgo

	Riesgo
R-01	El proyecto se retrasa con respecto a la planificación inicial
R-02	Las técnicas de balanceo de datos no muestran mejoras significativas
R-03	La GPU existente no es lo suficientemente potente
R-04	Se posee poca experiencia con el *Machine Learning* y las técnicas de balanceo de datos
R-05	Los conjuntos de datos empleados no poseen suficientes elementos
R-06	Las librerías empleadas por las técnicas de balanceo se han quedado obsoletas

Tabla 2.5: Listado riesgos

2.4.2. Estimación de los riesgos

En las Tablas 2.6 y 2.7 se analiza el valor de probabilidad de ocurrencia de cada riesgo y el efecto que produce si ese riesgo ocurre (Impacto). Ambos factores poseen valores en el rango $[0, 5]$. La probabilidad de ocurrencia es la probabilidad de que ocurra si no se lleva a cabo ninguna acción preventiva. Tiene relación directa con el valor de probabilidad cuantificando de forma más precisa en una escala porcentual desde 0 % hasta 100 %. Por otra parte, la pérdida operacional hace referencia a las pérdidas económicas directas que supondría no hacer frente a un riesgo, mientras que el impacto en los costes cuantifica el incremento de los costes totales si el riesgo ocurre.

Riesgo	Probabilidad de ocurrencia	Valor Probabilidad	Justificación
R-01	20 %	2	Es asumible que en un año académico exista algún retraso puntual debido a exámenes o trabajos
R-02	40 %	3	Al ser una investigación sobre métodos de balanceo novedosos en determinadas condiciones, puede ocurrir que no muestren mejora evidente
R-03	85 %	4	Es muy común que las técnicas empleadas requieran una mayor capacidad computacional para ser ejecutadas en un tiempo razonablemente finito
R-04	70 %	4	Únicamente una asignatura de la carrera aborda conceptos sobre sistemas inteligentes, luego es común que no se pueda profundizar en todos los campos
R-05	5 %	1	Existen numerosos conjuntos de datos, luego es muy posible encontrar alguno acorde a las características buscadas
R-06	25 %	2	Es un campo en constante avance, luego las librerías son actualizadas constantemente

Tabla 2.6: Probabilidad de cada riesgo

Riesgo	Pérdida operacional	Impacto en los costes	Impacto	Justificación
R-01	82,21€por cada día de retraso	5,8 %	3	Provoca no entregar en la fecha planificada el proyecto
R-02	-	-	1	Repercute en la obtención de conclusiones. No es evaluable numéricamente
R-03	6004€	75 %	5	No poseer una GPU adecuada puede retrasar o imposibilitar el correcto desarrollo del proyecto
R-04	82,21€por cada día de retraso	2,5 %	3	Puede retrasar el desarrollo del proyecto al ser necesario un aprendizaje previo
R-05	15€	12, 4 %	5	No se puede realizar un correcto entrenamiento de los modelos
R-06	82,21€por cada día empleado en modificarlo	1,3 %	3	Puede provocar un esfuerzo adicional y tiempo extra su modificación

Tabla 2.7: Impacto de cada riesgo

2.4.3. Matriz de Probabilidad × Impacto

Tras realizar el producto Exposición = Probabilidad × Impacto, se obtienen diferentes valores que permiten clasificar los riesgos en función de su prioridad:

- Prioridad alta: $10 \leq$ Exposición
- Prioridad media: $5 \leq$ Exposición ≤ 10
- Prioridad baja: $5 \leq$ Exposición

Riesgo	Probabilidad	Impacto	Exposición
R-01	2	3	6
R-02	3	1	3
R-03	4	5	20
R-04	4	3	12
R-05	1	5	5
R-06	2	3	6

Tabla 2.8: Matriz de Probabilidad × Impacto

Atendiendo a la Tabla 2.8, los riesgos que poseen alta prioridad son **R-03** y **R-04**, son aquellos en los que se requiere una actuación correctiva.

2.4.4. Plan de contingencia

Se muestra en la Tabla2.9 el plan de contingencia para cada uno de los riesgos ordenados de mayor a menor Exposición.

Riesgo	Plan de contingencia
R-03	Emplear una GPU A40 proporcionada por los tutores de TFG
R-04	Dedicar tiempo y esfuerzo a la adquisición de los conocimientos necesarios
R-01	Establecer una planificación más realista y ajustada al ritmo de trabajo
R-06	Actualizar el código con la sintaxis propia de la versión más nueva de la librería o buscar otros códigos alternativos
R-05	Buscar otro conjunto de datos que sí posea esas características requeridas
R-02	Mostrar un análisis detallado de los resultados y las conclusiones, con independencia de si existe o no mejora

Tabla 2.9: Plan de contingencia

2.5. Balance temporal y económico

Esta sección muestra los balances reales, ofreciendo así una visión más precisa de cómo se ha desarrollado el proyecto y cuánto se ha ajustado a la planificación inicial.

2.5.1. Balance temporal

Nuevamente se hace uso de los diagramas de Gantt, esta vez evidenciando el transcurso temporal real de las tareas desarrolladas. Queda reflejado a su vez, en color rojo, aquellas tareas cuya finalización se ha retrasado con respecto a lo previsto, y en verde manzana aquellas tareas cuyo inicio ha sido anterior al marcado. En color gris se encuentran las tareas que habían sido planificadas y que no han sido realizadas.

Sprint #1

Este sprint está caracterizado por la búsqueda de soluciones al problema planteado e investigación del estado del arte. Abarca un mayor número de semanas puesto incluye el periodo navideño no lectivo (23 diciembre - 7 enero). Para cumplir con los objetivos del *sprint* ha sido necesario emplear los primeros días de las vacaciones para completar una tarea que había comenzado con retraso respecto a lo establecido. La planificación inicial contaba con una dedicación de 72h 53m, mientras que realmente han empleado ocho horas más de lo previsto, realizando un total de 80h 44m de trabajo en este *sprint*. Se puede evidenciar el balance detallado en el diagrama de Gantt (Figura 2.14).

Sprint #2

Este *sprint* tiene una mayor dedicación a labores de entrenamiento y validación del modelo que a redacción de memoria. Al no poseer destreza en el manejo de librerías de *Machine Learning*, las tareas relacionadas con ello se demoraron en el tiempo. Tanto es así que de los dos modelos de balanceo de datos (DeepSMOTE y BAGAN) solo se ha podido completar las tareas establecidas para uno de ellos (DeepSMOTE) quedando el otro pendiente para el siguiente *sprint*. Se puede apreciar en el diagrama de Gantt (Figura 2.15). Las tareas en color gris son aquellas inicialmente planificadas y que no ha sido posible realizarlas. En este *sprint* se ha realizado un total de 92h y 39 min de trabajo, superando las 77h planificadas inicialmente.

Sprint #3

Nuevamente se presenta un *sprint* de mayor longitud en semanas puesto que abarca el periodo no lectivo de Semana Santa (25 marzo - 2 abril). El principal retraso en la planificación se debe a la incapacidad para compilar correctamente la técnica BAGAN. Finalmente, se ha optado por buscar técnicas alternativas, siendo BAGAN-GP la escogida. Para cumplir todos los objetivos y no retrasar las tareas venideras se ha hecho uso de los primeros días de vacaciones. La dedicación en horas ha sido de 89h 23min, se puede constatar en el diagrama de Gantt correspondiente (Figura 2.16).

Sprint #4

Este *sprint* se ha centrado en refinar las métricas obtenidas en los *sprints* anteriores y rematar las secciones aún sin acabar de la memoria técnica. El desglose real de tareas se encuentra en el diagrama de Gantt (Figura 2.17). La dedicación en horas ha sido de 81h 47min en contraposición con las 75h planificadas.

Sprint #5

A este *sprint* se ha dedicado un menor número de horas (27h 40min) puesto que las labores han consistido en finalizar la redacción de la memoria y la compilación de los modelos. El balance se encuentra detallado en el diagrama de Gantt (Figura 2.18).

2.5.2. Balance económico

Al abarcar el proyecto un total de 367 horas, se ha incurrido en costes adicionales referentes a los recursos humanos. Es necesario tener en cuenta también el tiempo empleado por la tutora Paula Mielgo al ser quien ha realizado el entrenamiento definitivo de los modelos generativos y las redes convolucionales empleando su GPU A40. La tabla actualizada quedaría como se muestra en la Tabla 2.10.

Puesto	Salario (por h)	h totales	Salario total	Salario Total con SS
Gestor de proyectos	10,77 €/hora	20h	215,4 €	276,4 €
Analista	14,36 €/hora	82h	1177,52 €	1510,75 €
Científico de datos	18,43 €/hora	140h	2580,2 €	3310,39 €
Programador	14,70 €/hora	125h	1837,5 €	2357,51 €
				7455,05 €

Tabla 2.10: Costes Recursos Humanos Reales

Y de esta forma los costes totales reales del proyecto serían:

Concepto	Coste
Hardware	1081,78 €
Software	0 €
Recursos Humanos	7455,05 €
Total	8536,83 €

Tabla 2.11: Costes Totales Reales

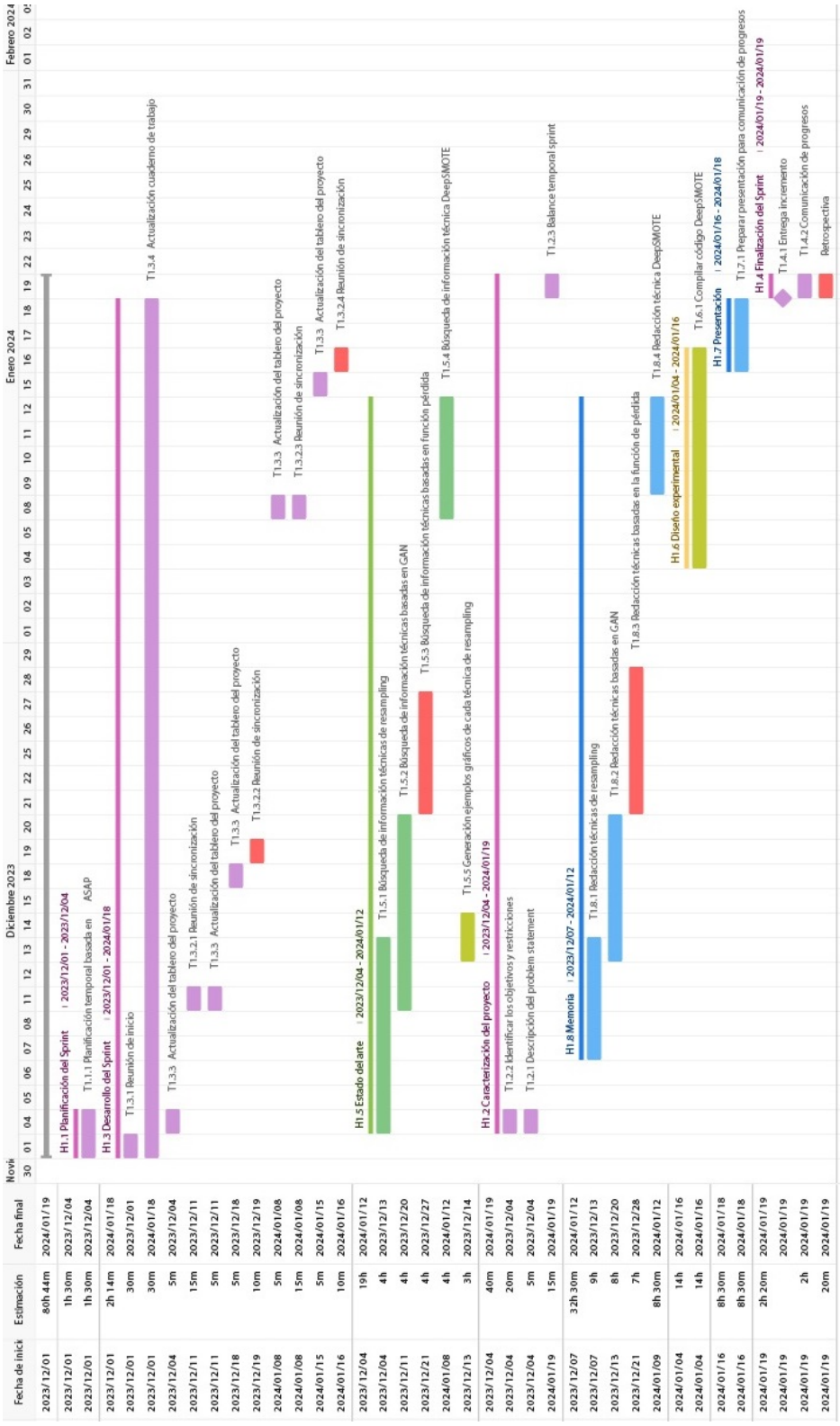


Figura 2.14: Balance *Sprint* #1

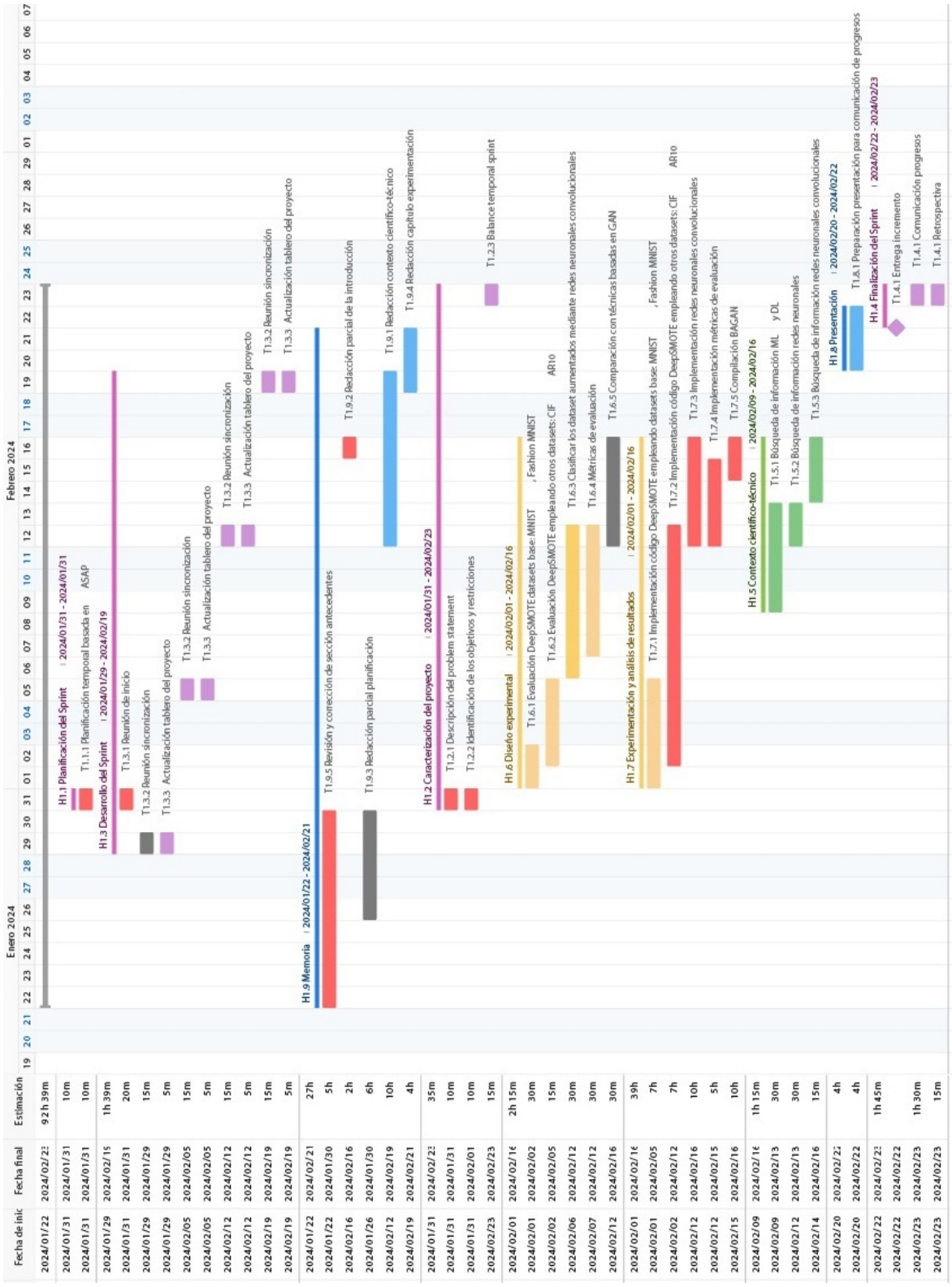


Figura 2.15: Balance *Sprint* #2

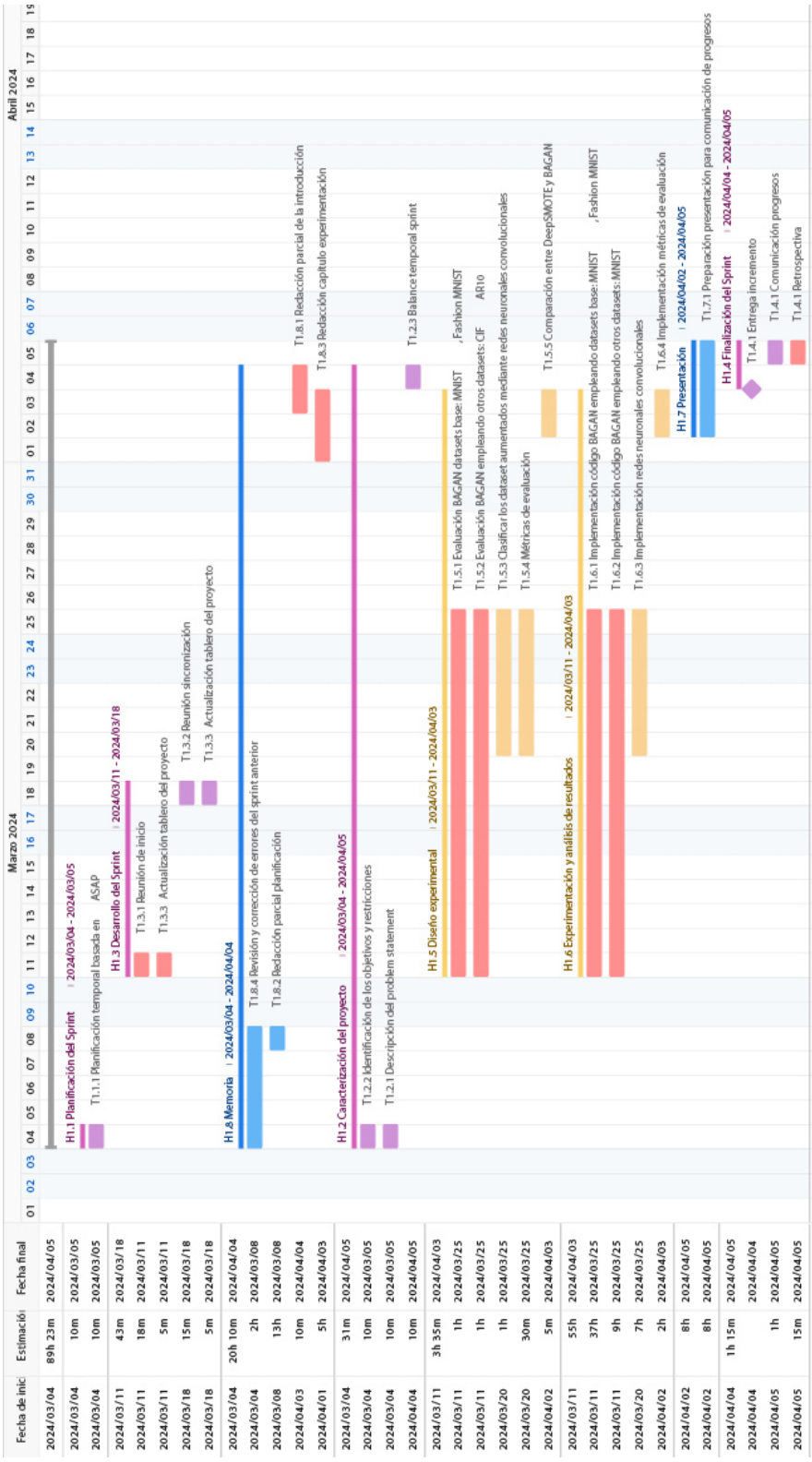


Figura 2.16: Balance *Sprint* #3

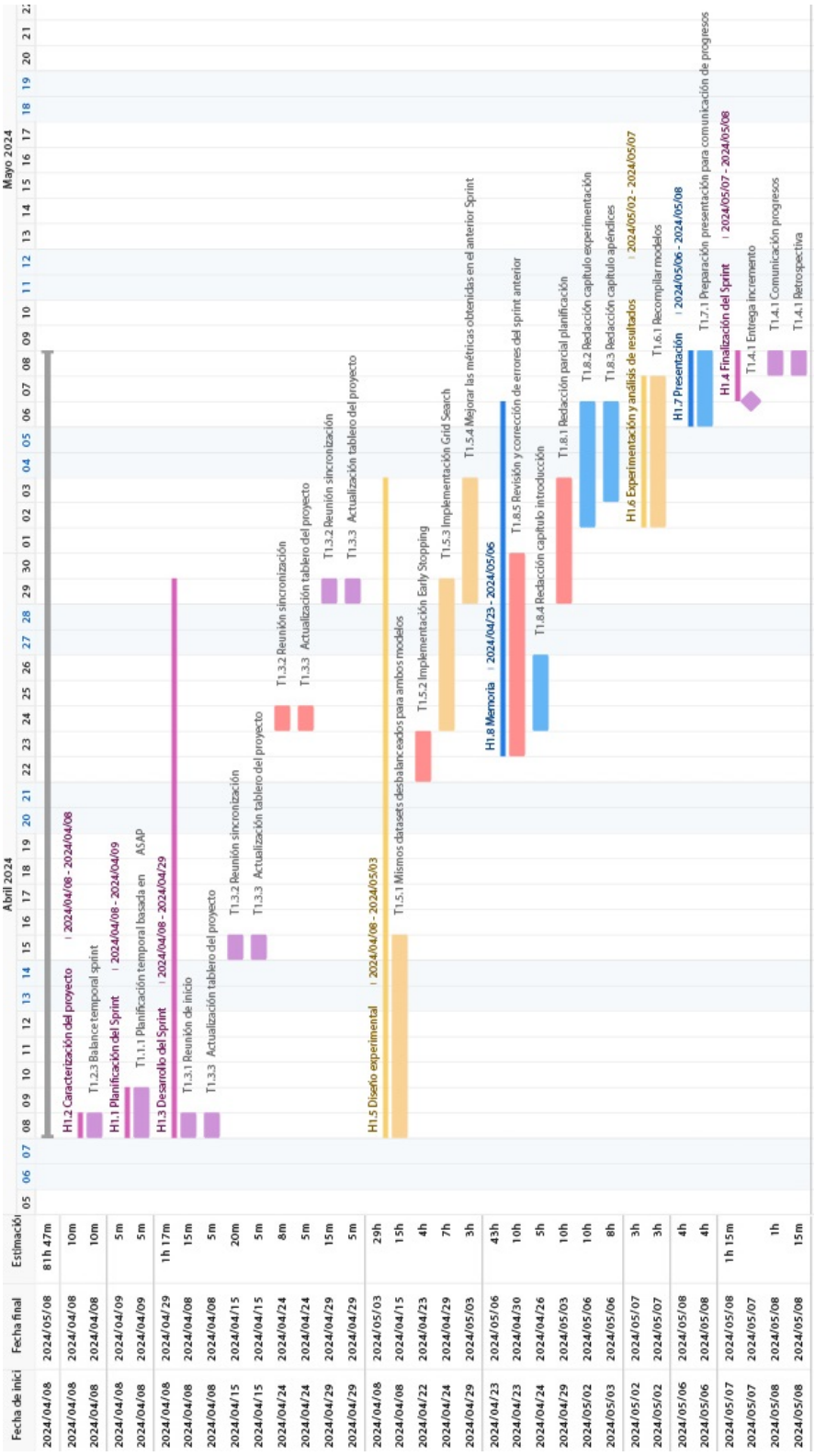


Figura 2.17: Balance *Sprint* #4

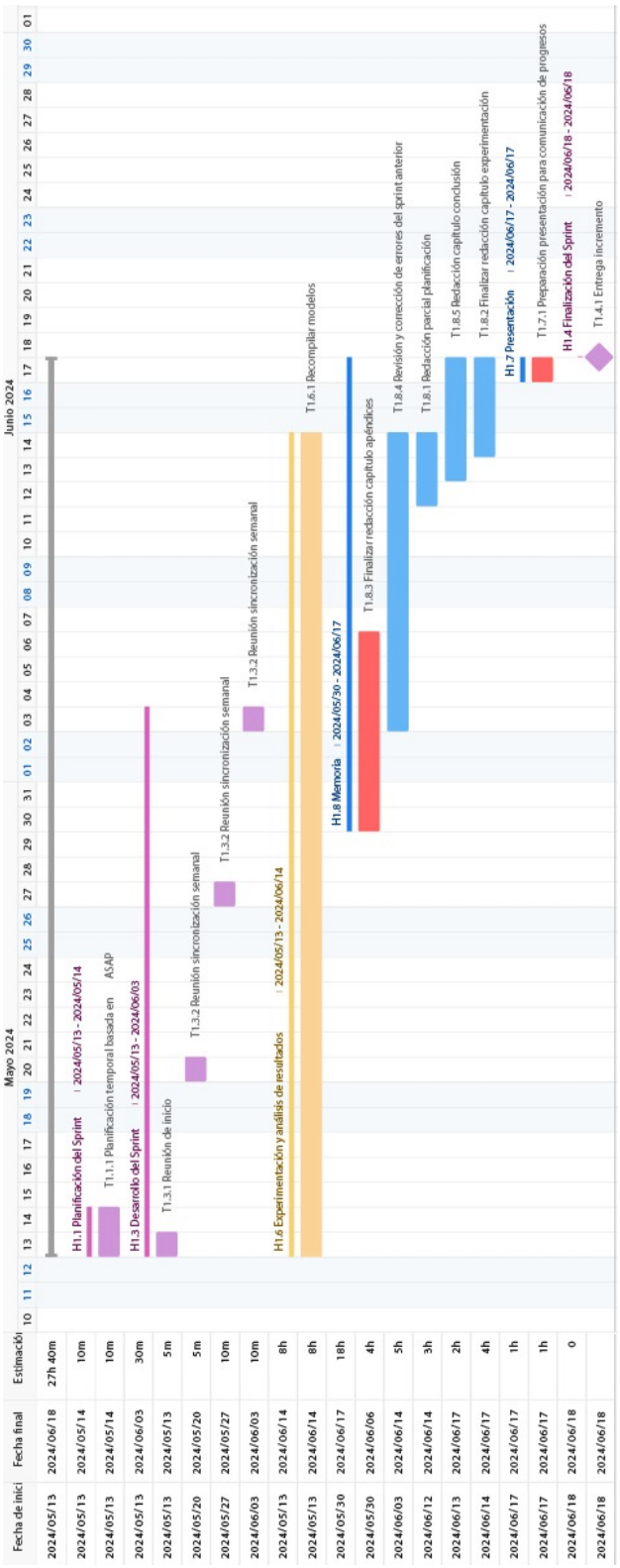


Figura 2.18: Balance *Sprint* #5

Capítulo 3

Antecedentes

Este capítulo muestra el contexto-científico técnico, es decir el marco teórico general sobre el cual se trabaja. Por una parte, en las subsecciones 3.1.1, 3.1.2, 3.1.3, 3.1.4 se describen los fundamentos del Machine Learning y DeepLearning, explicando la estructura de las redes neuronales y en particular, las convolucionales. Por otra parte, en las subsecciones 3.1.5, 3.1.6, 3.1.7, 3.1.8 se encuentran las técnicas más comunes empleadas para hacer frente a conjuntos de datos desbalanceados. Se dividen principalmente en dos grupos, definir una función de pérdida y asignar costes distintos a las distintas clases, y hacer remuestreos del conjunto de datos original. Este remuestreo puede consistir en aumentar el conjunto de datos de la clase minoritaria (*oversampling*) o reducir el conjunto de datos de la clase minoritaria (*undersampling*) de forma que todas las clases posean el mismo número de elementos.

Tal como se plantea el trabajo y la situación actual en la que se encuentran las técnicas abordadas no se considera necesario la elaboración de una sección que recoja el estado del arte. Los dos modelos de balanceo con los que se va a trabajar son técnicas novedosas y aún no se han publicado artículos que apliquen estos métodos en distintos entornos. Únicamente existen los artículos originales de cada una de las técnicas, así que no es posible realizar una comparativa.

3.1. Contexto científico-técnico

3.1.1. Machine Learning

El *Machine Learning* o Aprendizaje Automático es la rama de la Inteligencia Artificial que se preocupa del diseño y desarrollo de algoritmos que permiten a los ordenadores mejorar su desempeño en la realización de una tarea a partir de la experiencia [69]. De esta forma se dota a los ordenadores de cierta autonomía.

Dentro del *Machine Learning* se pueden distinguir tres áreas fundamentales:

- **Aprendizaje supervisado**: en el momento de entrenar el algoritmo, éste cuenta con un conjunto de datos etiquetados. Las etiquetas arrojan información sobre la clase a la que pertenece cada elemento, es decir, la salida del algoritmo esperada si tiene como entrada ese elemento. De esta forma, el algoritmo cuenta desde el inicio con información sobre la distribución de los datos, lo que facilita las posteriores labores de clasificación o regresión sobre conjuntos de datos nunca tratados. Entre los algoritmos de aprendizaje supervisado destacan las redes neuronales, el clasificador de Naive Bayes, la regresión lineal, las Má-

quinas de Vectores Soporte (SVM) [17], el K vecinos más cercanos (KNN) [72] y Bosques aleatorios [13].

Dentro del aprendizaje supervisado se pueden distinguir dos variantes principales:

- **Algoritmos de clasificación**: establece una categoría de entre las existentes a cada elemento del conjunto de datos. El algoritmo aprende con un conjunto de entrenamiento los patrones comunes de cada grupo para conseguir clasificar los elementos nunca vistos.
- **Algoritmos de regresión**: la variable objetivo la cual se pretende predecir su valor es una variable numérica continua. Es de utilidad para conocer la relación de dependencia o no entre los datos existentes.

- **Aprendizaje no supervisado**: el algoritmo toma como entrada un conjunto de datos no etiquetados. De esta forma el algoritmo debe ser capaz de abstraer las características y patrones comunes de los elementos.

  - **Algoritmos de clustering**: consiste en agrupar los diferentes elementos del conjunto de datos en distintos grupos (clústeres) en función de sus similitudes o discrepancias. Entre los algoritmos existentes se encuentran el algoritmo *k-means* o k-medias [43] y los modelos de mezcla gaussiana [68].
  - **Algoritmos de asociación**: se fundamentan en una serie de reglas para conseguir abstraer las relaciones entre los elementos.
  - **Algoritmos de reducción de la dimensionalidad**: como el propio nombre indica, consigue reducir el espacio de características o dimensión del conjunto de datos de estudio. Tiene por objetivo conseguir una mejora en el rendimiento de posteriores algoritmos de *Machine Learning* que se apliquen a los elementos. Entre las técnicas existentes, destaca el Análisis de componentes principales (PCA) [86], la descomposición en valores singulares (SVD) [71] y los codificadores automáticos [6].

- **Aprendizaje por refuerzo**: el algoritmo tiene por objetivo aprender en base a su propia experiencia. El algoritmo cuenta con el conocimiento de los posibles estados del sistema, las acciones que pueden ser realizadas y las recompensas que otorga cada acción. De esta forma el algoritmo es capaz de escoger, tras un proceso de ensayo y error, la mejor acción en diferentes escenarios, entendiendo por mejor acción, aquella que maximice las recompensas obtenidas. Un ejemplo de algoritmo dentro del aprendizaje por refuerzo es el *Q-learning* [82], empleado en juegos de toma de decisiones y robótica entre otros campos.

3.1.2. Redes neuronales

Las redes neuronales son un modelo computacional que vertebran especialmente los algoritmos de *Deep Learning*. La estructura está inspirada en las neuronas cerebrales. Se puede considerar que este estudio se inició con McCulloch y Pitts [56] cuando en 1943 publicaron un estudio acerca de cómo el cerebro humano conseguía abstraer patrones gracias a sus neuronas y lo comenzaron a relacionar con la lógica booleana. Fueron capaces de modelar una red neuronal sencilla empleando circuitos eléctricos.

De forma resumida, las neuronas biológicas poseen tres componentes diferenciados: dendritas, cuerpo y axón. Emiten impulsos eléctricos a través de su axón y los reciben de otras neuronas

gracias a las dendritas. El axón es una larga fibra que comunica una neurona con el resto. La sinapsis es el proceso por el cual las neuronas se comunican entre sí con el propósito de transmitir información. La red neuronal va creando conexiones o modificando las existentes a medida que va aprendiendo.

Siguiendo el mismo razonamiento la neurona artificial posee una serie de componentes como muestra la Figura 3.1:

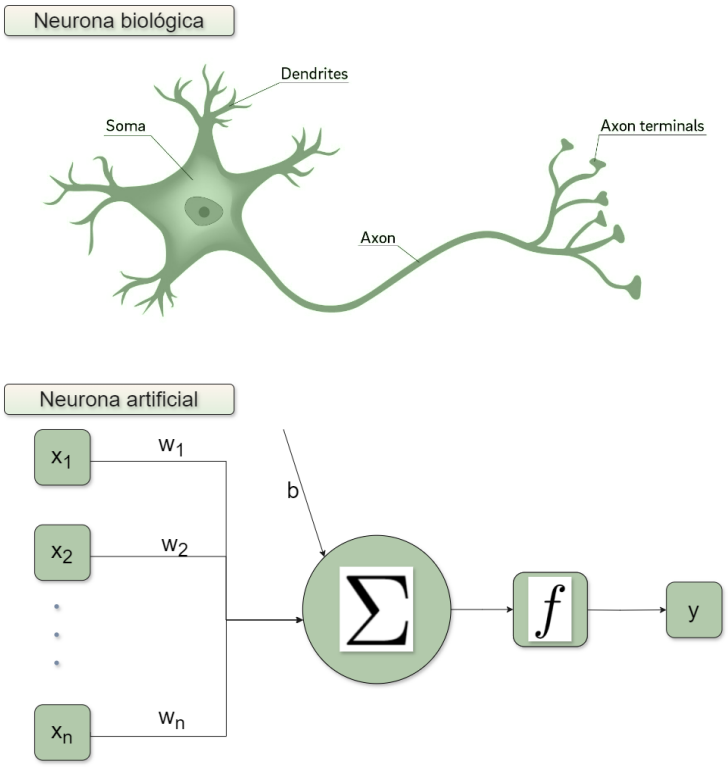


Figura 3.1: Estructura de una neurona [84]

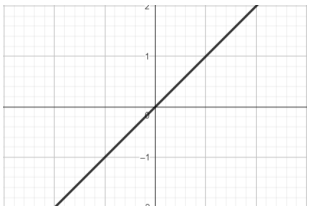
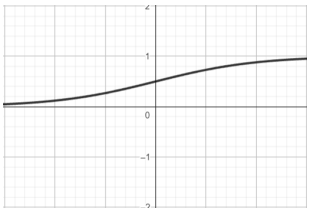
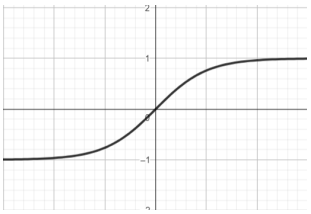
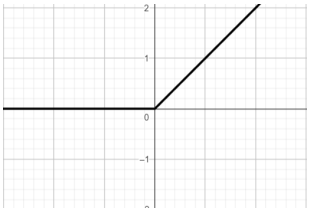
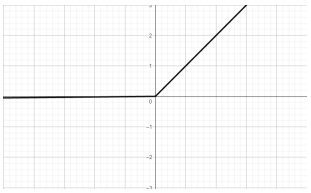
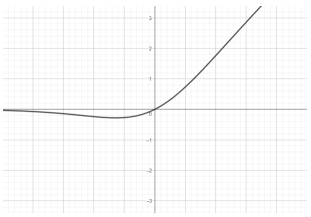
- **Vector de entrada x**: se corresponde con el conjunto de elementos a procesar. Tiene su analogía en las dendritas de una neurona biológica.
- **Vector de pesos w**: se equipara con la sinapsis de las neuronas. Es un vector $\mathbf{w} = \{w_1, ..., w_n\}$. Determina cuánto se pondera cada elemento de entrada en la red.
- **Sesgo** b: es el umbral de acción o activación, representa la facilidad con la que se dispara una neurona.
- **Función de propagación** Σ: es una función lineal que representa una suma del vector de entrada ponderada por el vector de pesos de la red. Se añade al sumatorio un sesgo b.

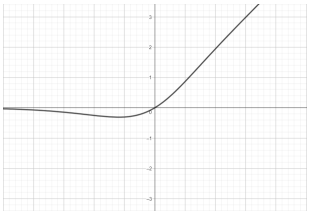
$$\Sigma = \sum_{i=1}^{n} x_i w_i + b$$

- **Función de activación** f: se encarga de transformar el conjunto de salida de una neurona con el fin de limitar la amplitud de los valores resultantes. Generalmente se emplean fun-

ciones continuas monótonas crecientes. Las funciones de activación usuales se encuentran resumidas en la Tabla 3.1.

Tabla 3.1: Funciones de activación

Nombre	Función	Gráfico
Lineal	$f(x) = x$	
Sigmoide	$f(x) = \frac{1}{1+e^{-x}}$	
Tangente hiperbólica	$f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$	
ReLU (Rectified Linear Unit) [61]	$f(x) = \text{máx}(0, x)$	
Leaky-ReLU [52]	$f(x) = \text{máx}(0{,}01x, x)$	
Swish [65]	$f(x) = x\left(\frac{1}{1+e^{-x}}\right)$	

Nombre	Función	Gráfico
Mish [58]	$f(x) = x(tanh(\ln(1 + e^x)))$	

Por lo tanto, una red neuronal se constituye como un conjunto de elementos, denominados neuronas, interconectados que se estructuran en capas. Una red neuronal es un conglomerado de neuronas simples, puesto que trabajar con un modelo de una única neurona queda demasiado limitado.

Una red neuronal posee una capa de entrada, una o varias capas ocultas y una capa de salida. La diferencia para que una red neuronal se considere un modelo de *Deep Learning* reside en el número de capas ocultas que posea. Si la red posee más de una capa oculta o un mayor número de neuronas por capa o neuronas de diverso tipo, entonces el modelo creado será considerado un algoritmo dentro del ámbito del *Deep Learning*. La capa de entrada está formada por tantas neuronas como atributos hay presentes en el conjunto de datos de entrada. La capa de entrada se comunica con una o más capas ocultas. Las capas ocultas son todas aquellas capas existentes, a excepción de la capa de entrada y de salida de la red, encargadas de procesar los datos entrantes y pasarlos a la siguiente capa. Cada una de las neuronas de cada capa posee su propio vector de pesos, y su sesgo. En las capas ocultas es donde se efectúa el procesamiento de la información. Por último, la capa de salida, arroja los resultados de la tarea para la que ha sido diseñada, que generalmente es clasificación o regresión.

El proceso de aprendizaje de la red puede ser supervisado o no supervisado [1]. En ambos casos la red tiene por objetivo ajustar los pesos $\mathbf{w}$ y el sesgo de las neuronas de forma que la red consiga abstraer la distribución de los datos para generar la salida adecuada.

El proceso de entrenamiento, en el caso concreto de aprendizaje supervisado en una red neuronal multicapa sigue una serie de pasos:

- Se introduce el conjunto de datos de entrada en la capa de entrada de la red. La red procesa los datos y produce una salida para cada elemento.
- Se calcula una función de pérdida atendiendo al valor real y al valor obtenido por la red.
- Se emplean algoritmos (en el caso de redes multicapa), como la retropropagación y el descenso del gradiente, mediante los cuales se busca optimizar los parámetros de la red de forma que minimicen el error arrojado [29].

Este proceso constituye una época (*epochs*). En el proceso de entrenamiento se realizan tantas épocas como se considere necesario en función de la naturaleza de los elementos de entrada.

Por otra parte, las redes neuronales pueden ser clasificadas atendiendo a diferentes criterios:

[1] En la mayoría de los casos este proceso de aprendizaje va a ser supervisado. Las redes neuronales no supervisadas son empleadas generalmente para labores de *clustering* o reducción de la dimensionalidad. Las más destacadas son la red de Kohonen o Mapa Autoorganizado y la red neuronal *Autoencoder*.

- **En función del número de capas ocultas**: se distingue entre las redes que poseen una única capa oculta (redes monocapa) o aquellas que poseen múltiples capas ocultas (redes multicapa)
- **En función de la realimentación**: en este grupo se encuentran las redes no recurrentes, aquellas en las que la propagación tiene lugar en un solo sentido, y las redes recurrentes, las cuales poseen realimentación, ya sea con las neuronas de capas anteriores, de la misma capa o de una neurona consigo misma.
- **En función del grado de conexión**: se distingue entre las redes neuronales parcialmente conectadas y las redes neuronales totalmente conectadas (todas las neuronas de una capa se encuentran conectadas con las de la capa posterior y con las de la capa previa en el caso de tratarse de redes recurrentes).

3.1.3. Deep Learning

El *Deep Learning* [51] es un subcampo dentro del *Machine Learning* que posee la capacidad de representar los elementos a tratar como una jerarquía de conceptos más simples, siendo por tanto capaz de trabajar con datos complejos. Además permite trabajar con datos de naturaleza variada, como texto, imagen, audio, entre otros. Las aplicaciones del *Deep Learning* son variadas, destacando las áreas de visión computacional, reconocimiento del habla y procesamiento de lenguaje natural. Ejemplos concretos pueden ser el análisis de sentimientos, traducción de textos o reconstrucción de circuitos cerebrales entre otros.

Los modelos de *Deep Learning*, a diferencia de los existentes en el *Machine Learning*, son capaces de extraer ellos mismos las características de mayor relevancia presentes en el conjunto de datos de entrada. Los modelos que conforman este campo son las redes neuronales multicapa, también llamadas redes neuronales profundas (Figura 3.2), es decir, aquellas redes neuronales que cuentan con varias capas ocultas de procesamiento. Entre los ejemplos de este tipo de redes se encuentran las redes convolucionales (CNN), redes recurrentes (RNN), redes generativas adversarias y *Transformers*. Uno de los motivos de su empleo es debido a que es más ventajoso aumentar la profundidad añadiendo capas que aumentar el número de neuronas en cada capa, de esta forma el número de neuronas necesarias crece de forma lineal con cada capa en vez de que la complejidad aumente de forma exponencial.

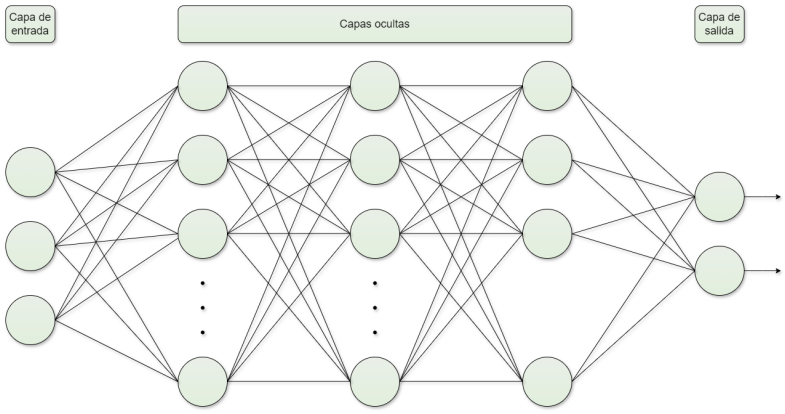


Figura 3.2: Red neuronal profunda

Se pueden distinguir dos etapas principales si se particulariza al caso en el cual se centra este trabajo (redes convolucionales para trabajar con imágenes) y se describirá con mayor detalle en la Subsección 3.1.4. Primeramente se realiza una extracción de características, en la cual se crea a partir del conjunto de datos de entrada una jerarquía de capas con las características más relevantes. Esta fase elabora una estructura estratificada, en la cual progresivamente se obtienen características de menor a mayor complejidad del conjunto de datos, de forma que las características de más alto nivel se conforman en función de los niveles previos. En la segunda fase, se realiza una transformación de características, en la cual a partir de una representación a alto nivel, se consigue obtener la salida correspondiente.

Por ejemplificarlo, partiendo de una red neuronal profunda para la clasificación de imágenes, en las primeras capas ocultas se extraerán detalles de bajo nivel, como pueden ser colores o bordes, mientras que en las capas sucesivas el modelo será capaz de extraer patrones y elementos más complejos, como por ejemplo una cara de una persona. Por último se realizará una transformación adecuada para que la última capa arroje la etiqueta de la clase a la que pertenece la imagen identificada.

La cantidad de elementos en el conjunto de entrada influye notablemente en la capacidad predictiva del modelo. Contar con un gran número de datos permite al modelo ser capaz de generalizar y obtener mejores resultados frente a elementos nunca vistos. Por el contrario, si el conjunto de entrenamiento es pequeño, existe una elevada probabilidad de sobreajuste, además de ser más costoso.

Un inconveniente que presentan las redes neuronales profundas es que no son invariantes a la posición, es decir, ajustará nuevamente todos los pesos y sesgos si como entrada recibe una imagen donde algún objeto se encuentre desplazado con respecto a una imagen anterior.

Por otra parte, las redes neuronales profundas se caracterizan a su vez por el tipo o tipos de neuronas que poseen, destinadas a cumplir una función particular. Entre estos tipos de neuronas se encuentran por ejemplo las neuronas de convolución diseñadas para realizar operaciones de convolución y extraer características locales (Subsección 3.1.4), las neuronas recurrentes que permiten almacenar la secuencia de los datos, las neuronas LSTM (*Long Short-Term Memory*) que almacenan información relevante a largo plazo, las neuronas de atención que permiten a la red que se centre en los datos más relevantes y las neuronas de embedding que convierten datos categóricos en vectores continuos.

3.1.4. Redes convolucionales

Las redes convolucionales se inspiran en la estructura de la parte del cerebro que se encarga de ejecutar las funciones más complejas, el neocórtex. Existen numerosos trabajos que avalan el hecho de que el cerebro humano extraiga la información visual mediante una selección de características de menor a mayor complejidad. En 1980, K. Fukushima introdujo una red neuronal, Neocognitron [26], que trataba de imitar este comportamiento del cerebro humano. Fue ya en 1989 cuando Yann LeCun introdujo la primera red neuronal convolucional [49]. Estas redes introducen nuevos tipos de capas, las de convolución y las de *pooling* que se encargan de la extracción de características del conjunto de datos de entrada y la reducción progresiva de la dimensión de este espacio de características.

La ventaja de las redes convolucionales [35] es que no tienen todas las neuronas de las capas conectadas, es decir, cada neurona no tiene por qué encontrarse conectada con todas las neuronas de la capa tanto anterior como posterior. Una capa que posea esta propiedad se denomina densa

o *fully-connected*. El hecho de no tener capas densas facilita el manejo de grandes volúmenes de datos y datos complejos de entrada, consiguiendo que sea computable en tiempo finito tanto el entrenamiento como la validación de los modelos. Puesto que el estudio que se aborda emplea conjuntos de imágenes como entrada, se particulariza la explicación para este tipo de datos.

A la capa de entrada de una red convolucional le siguen de forma alternada una serie de capas convolucionales y capas de pooling. Posteriormente se colocan una o diversas capas *fully-connected* y generalmente la última capa es una capa *softmax* que tiene por objetivo arrojar la probabilidad de pertenencia a cada clase de cada una de las imágenes introducidas. Se describe a continuación cada uno de los tipos de capa con mayor detalle.

- **Capa de entrada**: conforma la capa inicial de la red, la cual recibe el conjunto de imágenes a tratar. Las dimensiones del conjunto de entrada es una tupla (r, n, n, c) donde r es el número de imágenes que se van a procesar en paralelo, $n \times n$ es la dimensión de la imagen y c es el número de canales de los que está compuesta la imagen, uno si es en escala de grises y tres si es en color.

- **Capa convolucional**: la convolución constituye una operación entre una imagen de entrada y un filtro o *kernel*, generando un mapa de características. Un filtro se encarga de identificar un mismo rasgo a lo largo de una imagen. Una capa de convolución está formada por una o más convoluciones, siendo la salida la concatenación de todas las salidas individuales. Es habitual que el resultado de la convolución posea un tamaño más reducido en comparación con el tamaño del elemento de entrada, puesto que la convolución solo es computable en aquellas celdas donde tanto el elemento como el filtro estén definidos. Si se desea conservar el tamaño se realiza la operación previa de *padding* en la cual se introducen ceros en los extremos de la imagen, para que sea efectiva la convolución en cada una de las celdas del elemento de entrada. Por otra parte, también es posible reducir de forma controlada la dimensión configurando el tamaño del paso o *stride*. En el caso de que el paso de convolución sea p, el tamaño de la imagen de salida de esa capa será

  $$\left\lfloor \frac{n-k}{p} \right\rfloor + 1$$

  siendo n el tamaño del mapa de características y k el tamaño del kernel aplicado.

- **Capa de pooling**: en esta capa se realiza un submuestreo de los mapas de características obtenidos con el objetivo de localizar las características identificadas en las capas convolucionales. Técnicas comunes de esta capa consisten en calcular la media (*average-pooling*) o el valor máximo (*max-pooling*) de una característica en un área concreto de la imagen.

- **Capa *fully-connected***: como su propio nombre indica, esta capa o conjunto de capas tienen sus neuronas totalmente conectadas, tanto con las de la capa anterior como de las de la posterior.

Entre los ejemplos de este tipo de redes se encuentran *AlexNet* [3], *ResNet* [36] entre la que destaca *Resnet50* constituída por 50 capas, *Inception V3* [74] que emplea filtros de distintos tamaños para así obtener características más específicas, *Xception* [15] que optimiza la forma de realizar las convoluciones, *SqueezeNet* [39] que destaca por ocupar poca memoria o *EfficientNet* [74] entre otras.

3.1.5. Técnicas de remuestreo

Este conjunto de técnicas se basa en la alteración del conjunto de datos que se encuentra desbalanceado, de forma que consigan arrojar el conjunto de datos balanceado. Generalmente son técnicas que se aplican a conjuntos de datos numéricos, pero constituyen la base fundamental para posteriormente ser integradas en modelos de balanceo que trabajen con conjuntos de datos de naturaleza más compleja.

A continuación se detallan las técnicas más relevantes dentro de este abanico.

Submuestreo aleatorio

Este método, también denominado *undersampling*, se fundamenta en la eliminación de elementos de la clase mayoritaria del conjunto de datos de entrenamiento. Se selecciona de forma totalmente aleatoria un conjunto de datos pertenecientes a la clase mayoritaria, y posteriormente son eliminados del conjunto de datos original. Un ejemplo de ello se encuentra en la Figura 3.3. El número de instancias eliminadas es variable, pero es común reducir hasta que el número de elementos de la clase mayoritaria iguale al número de elementos de la clase minoritaria.

El principal inconveniente de este método es que la eliminación de datos de forma aleatoria puede suprimir instancias susceptibles a ser relevantes.

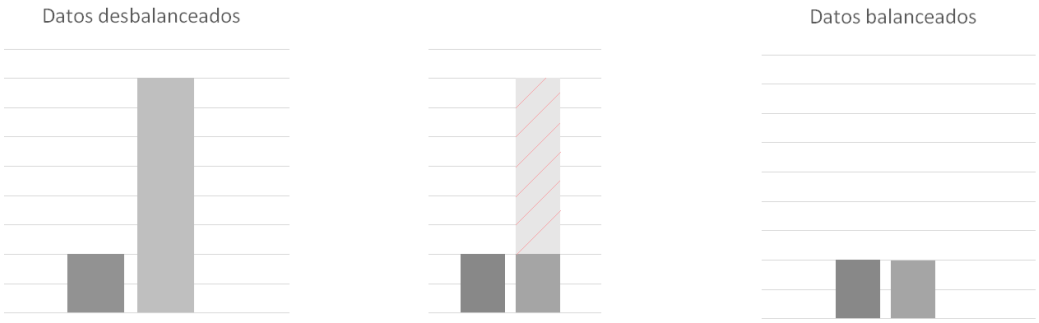


Figura 3.3: Ejemplo undersampling

Condensed Nearest Neighbours, CNN

Como su nombre indica, se sustenta en la técnica de los k-vecinos más próximos para conseguir condensar los datos. Pretende eliminar aquellos que no aportan información útil al conjunto de entrenamiento. De esta forma, se comienza a erradicar la aleatoriedad en la elección de las instancias a eliminar. Esta técnica fue elaborada por Peter E. Hart en 1966 y publicada en [33].

Sea $X = \{x_1, ..., x_n\}$ el conjunto de entrenamiento. El procedimiento seguido consiste, primeramente, en etiquetar todos los elementos del conjunto asignándoles un par (x_i, y_i), $i = 1, ..., n$, donde x_i es la instancia correspondiente e y_i es la clase a la que pertenece el elemento. A partir de ellos se va seleccionando el nuevo conjunto de entrenamiento E, conjunto que, inicialmente, ya posee de forma inamovible los elementos de la clase minoritaria. Para seleccionar los elementos a considerar de la clase mayoritaria, se desarrolla el siguiente proceso iterativo

1. Se escoge de forma aleatoria un primer elemento de la clase mayoritaria que pertenecerá al conjunto E, (x_i, y_i).

2. Se aplica el algoritmo de los k-vecinos mas cercanos, generalmente $k = 1$, y se seleccionan los vecinos del elemento anterior (x_i, y_i), cuyas clases no coincidan con la clase de este elemento.

Este procedimiento continúa escogiendo uno a uno elementos del conjunto X hasta que ocurra una de las dos posibles condiciones de parada

- El nuevo conjunto de datos E esté completo, es decir posea el mismo número de elementos que el conjunto original, y por tanto sea igual a éste.
- En una iteración no se añada ningún elemento al conjunto E, esto implica que en las sucesivas iteraciones tampoco se va a ver modificado E.

Lo descrito se ilustra con el ejemplo de la Figura 3.4. Remarcar que el parámetro k escogido para los k-vecinos es el mismo en todas las iteraciones. La interpretación geométrica de esta técnica consiste en que se reduce el número de elementos de la clase mayoritaria en las regiones donde había gran densidad de instancias de este tipo.

La desventaja de este método radica en la elección aleatoria del primer elemento. Las técnicas posteriores intentar paliar este obstáculo.

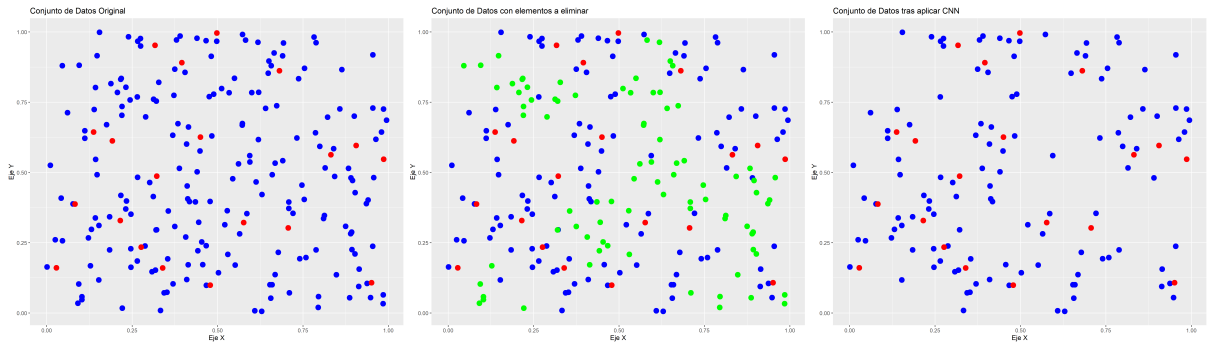


Figura 3.4: Ejemplo CNN

Tomek Links

Este método pretende subsanar la carencia fundamental del método anterior *Condensed Nearest Neighbours*. Es por esto que minimiza la aleatoriedad del proceso de elección de la primera instancia del proceso. Se basa en la distancia mínima entre dos elementos pertenecientes a clases diferentes. Un *Tomek link* se forma cuando dos elementos pertenecientes cada uno a una clase distinta, poseen una relación directa, sin la existencia de elementos en el medio. De forma que, las instancias que poseen estas características serán descartadas del conjunto resultante. Esto permite una limpieza de elementos pertenecientes a las regiones de contacto entre clases.

Sean A y B dos conjuntos, cada uno de los cuales contiene elementos de una única clase. Se define E como el conjunto que va a albergar los datos seleccionados del conjunto original, que ya cuenta inicialmente con los elementos de la clase minoritaria (Figura 3.5). La descripción del procedimiento iterativo es la que sigue

1. Se escoge un elemento del primer conjunto $a \in A$ y otro elemento del segundo conjunto $b \in B$. Se calcula la distancia entre ambos $d(a, b)$. La elección de la distancia empleada depende de la geometría del conjunto, pero generalmente se hace uso de la distancia euclídea.

2. Si para ningún elemento x del conjunto original ocurre

$$d(a,x) < d(a,b) \quad \text{o} \quad d(x,b) < d(a,b)$$

entonces el elemento de la clase mayoritaria será descartado y no formará parte del conjunto final E.

Generalmente el número de elementos eliminados es sustancialmente menor. Por otra parte, este procedimiento consigue efectuar una eliminación de elementos en las regiones de contacto entre las clases. Mientras que con el método *Condensed Nearest Neighbours* lo que se conseguía era despejar las regiones donde predominaba la clase mayoritaria, mediante la eliminación de elementos de esa región, manteniendo los elementos próximos a la frontera de contacto entre las clases.

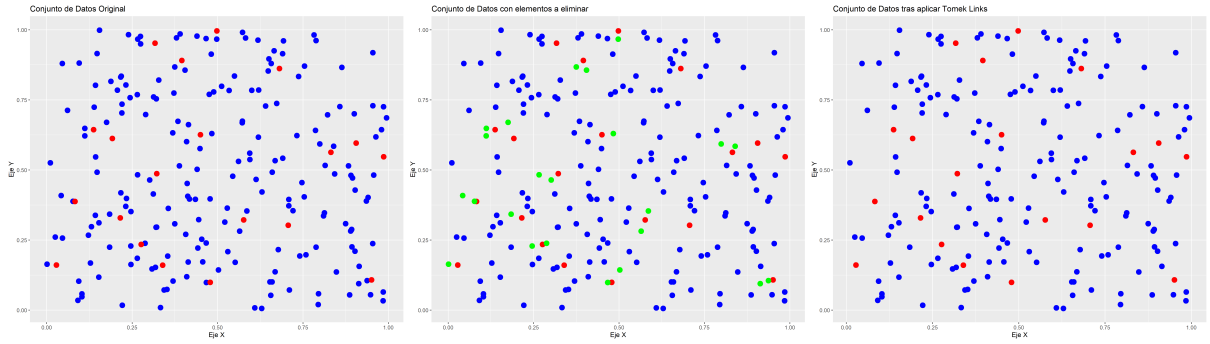


Figura 3.5: Ejemplo Tomek Links

Edited Nearest Neighbours, ENN

En 1972, Wilson en [85] modifica los métodos anteriores para arrojar una solución que elimina de forma más directa las instancias del conjunto de datos no deseadas. De forma resumida, se centra en eliminar los elementos del conjunto original cuya clase difiere de la clase mayoritaria de sus k-vecinos más cercanos.

El método actúa como se detalla

1. Se escoge un elemento (x_i, y_i) del conjunto de datos de entrenamiento y perteneciente, a su vez, a la clase mayoritaria.

2. Se realiza el cálculo de sus k-vecinos más cercanos. Generalmente esta técnica suele emplear $k = 3$.

3. Se observa la clase predominante en esos k vecinos tomados, y si esa clase no coincide con la clase del elemento (x_i, y_i), éste es eliminado.

El procedimiento se realiza para cada elemento del conjunto de datos. Un ejemplo se puede visualizar en la Figura 3.6. Existe una variante iterativa denominada *Repeated Edited Nearest Neighbours*.

Wilson en [85] realiza un análisis del rendimiento que arroja el método en función del número de vecinos escogido, k. Concluye que escoger $k = 3$ posee mejor rendimiento que escoger $k = 1$.

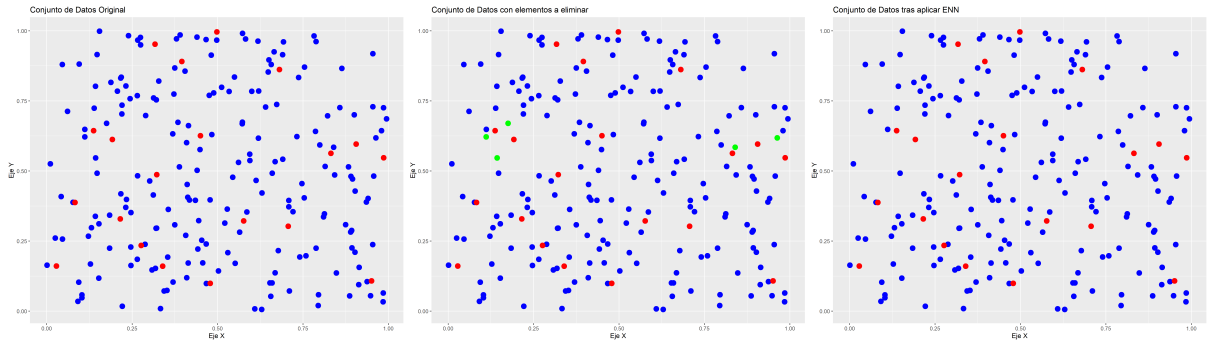


Figura 3.6: Ejemplo ENN

Sobremuestreo aleatorio

El conjunto de datos original se ve alterado añadiendo nuevamente de forma aleatoria instancias pertenecientes a la clase minoritaria. En particular, lo habitual es añadir instancias hasta que ambas clases posean el mismo número de elementos o lleguen a una cota deseada (Figura 3.7). Además, el coste computacional de realización del método es reducido. Esta técnica se conoce también con el nombre de *oversampling*. El perjuicio fundamental de este método es que produce sobreajuste en las posteriores clasificaciones.

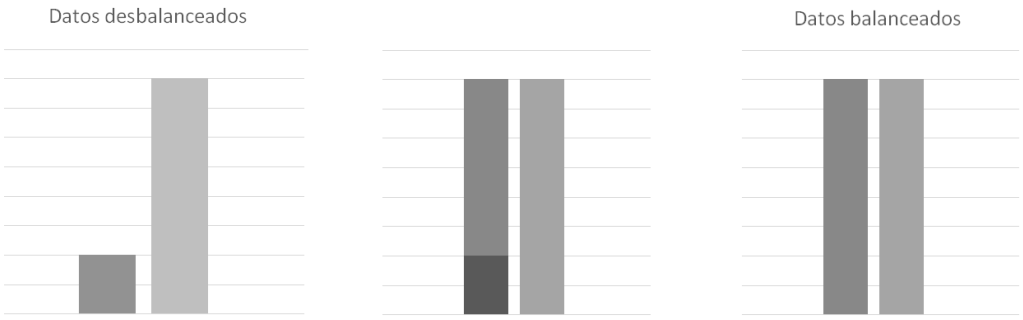


Figura 3.7: Ejemplo oversampling

Synthetic Minority Over-sampling Technique, SMOTE

La técnica SMOTE, *Synthetic Minority Over-sampling Technique* [14], es un método que permite obtener clases balanceadas mediante el empleo de un sobremuestreo (*oversampling*). Se fundamenta en la creación de instancias sintéticas de la clase minoritaria. Comúnmente, va seguido de un *undersampling* de la clase mayoritaria.

Esta técnica surge tras constatar que la predicción de la clase minoritaria no mejora cuando se realiza un sobremuestreo aleatorio de esta clase. Está basada en la generación de instancias sintéticas dentro de un espacio de características en vez de dentro del conjunto de datos original. Para crear un elemento sintético, es necesario elegir una instancia de la clase minoritaria. A continuación, se escoge un punto intermedio sobre la línea que une el elemento elegido con uno de sus k-vecinos elegido aleatoriamente. Un ejemplo se puede encontrar en la Figura 3.8.

Por precisar, este procedimiento se describe como sigue:

1. Se escoge un elemento aleatorio x perteneciente a la clase minoritaria.
2. Se calculan sus k vecinos más cercanos, con k dependiendo del problema a tratar, y se escoge de forma aleatoria uno de esos vecinos y.
3. Entre el vecino escogido y y el elemento inicial x se calcula su distancia $d(x, y)$. Generalmente se emplea la distancia euclídea, pero depende de la geometría del problema.
4. Se escoge de forma aleatoria un número perteneciente al intervalo $a \in [0, 1]$ y se multiplica por la distancia anteriormente calculada.
5. El punto sintético s se conforma trasladando el elemento inicial, es decir, sumando el valor anteriormente obtenido al elemento x.

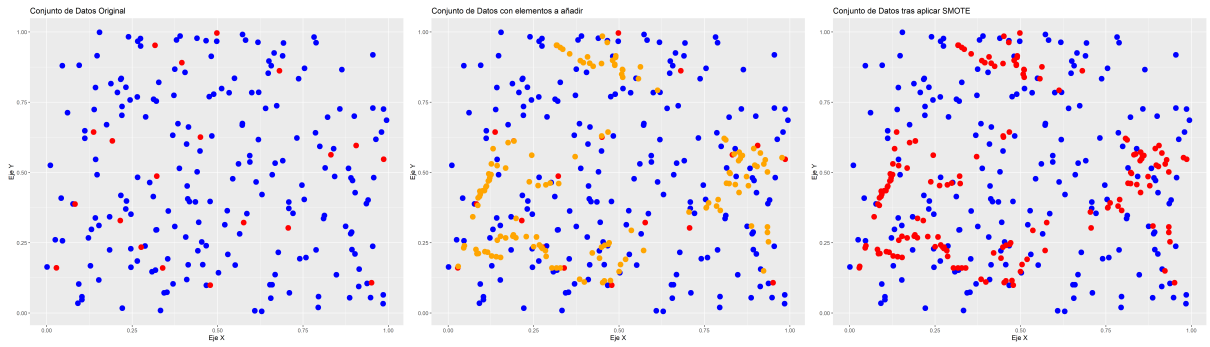


Figura 3.8: Ejemplo SMOTE

La consecuencia de aplicar esta técnica, es que se generan regiones más amplias y menos específicas, en contraposición con las regiones de menor tamaño y más específicas de las que se partía.

La técnica SMOTE se puede combinar con otras técnicas de *undersampling* (Figura 3.9). El hecho de aplicar una técnica para eliminar instancias de la clase mayoritaria, para posteriormente aumentar el número de elementos de la clase minoritaria gracias a SMOTE, arroja generalmente mejores resultados que los obtenidos aplicando cada técnica por separado. De esta forma es frecuente emplear *SMOTE+Tomek Links*, *SMOTE+ENN*.

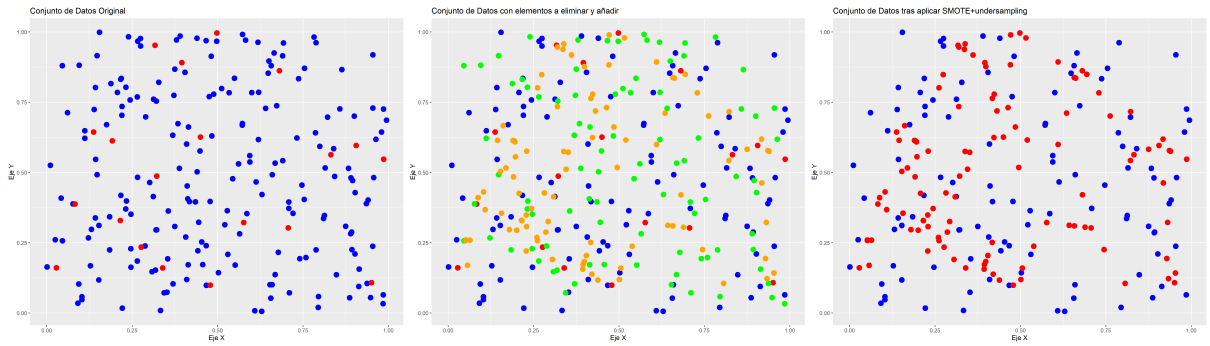


Figura 3.9: Ejemplo SMOTE combinado con undersampling

Adaptative Syntetic Sampling, ADASYN

Esta técnica de reducción del sesgo como consecuencia del desbalanceo de clases fue introducida por varios autores en 2008 en [34]. Se fundamenta en la técnica SMOTE. Trata de añadir variabilidad en los datos generados.

El procedimiento a seguir se describe a continuación:

1. Se calcula el grado de desbalanceo, expresado como el cociente entre el número de elementos de la clase minoritaria y el número de elementos de la clase mayoritaria.

$$d = \frac{|X|}{|Y|}$$

   denotando por X el conjunto de elementos de la clase minoritaria y por Y al de la mayoritaria.

2. Para que este algoritmo se lleve a cabo, es condición indispensable que d sea menor que una tolerancia establecida, $d < tol$. De no ser así, las sucesivas etapas no podrían ser efectuadas.

3. Se calculan el número de elementos sintéticos que es necesario generar

$$G = (|X| - |Y|) \times \beta$$

   siendo $\beta \in [0, 1]$ el nivel de balanceo de datos deseado. Un valor $\beta = 1$ significa que el conjunto de datos que se cree estará completamente balanceado.

4. Para cada elemento perteneciente a la clase minoritaria $\mathbf{x}_i \in X$, se calculan sus k-vecinos más próximos.

5. Se calcula el ratio

$$r_i = \frac{\Delta_i}{k} \quad \forall i$$

   donde Δ_i es el número de elementos encontrados en los k vecinos más cercanos y que pertenecen a la clase mayoritaria. Es por tanto inmediato comprobar que $r_i \in [0, 1]$.

6. Se normaliza el valor del ratio r_i

$$\hat{r_i} = \frac{r_i}{\sum_{j=i}^{|X|} r_j}$$

7. Se calcula el número de elementos que van a ser generados para cada una de las instancias de la clase minoritaria

$$g_i = \hat{r_i} \times G$$

8. Para generar los g_i elementos de cada instancia $\mathbf{x_i}$ se realiza un proceso iterativo. Para cada $\mathbf{x_i}$ y en cada iteración, se escoge un elemento aleatorio de la clase minoritaria perteneciente a estos k-vecinos más cercanos $\mathbf{a_{ij}}$. Se escoge un número aleatorio $\lambda \in [0, 1]$. Luego el elemento generado es

$$\mathbf{s_i} = \mathbf{x_i} + (\mathbf{a_{ij}} - \mathbf{x_i}) \times \lambda$$

Un ejemplo del proceso descrito se muestra en la Figura 3.10.

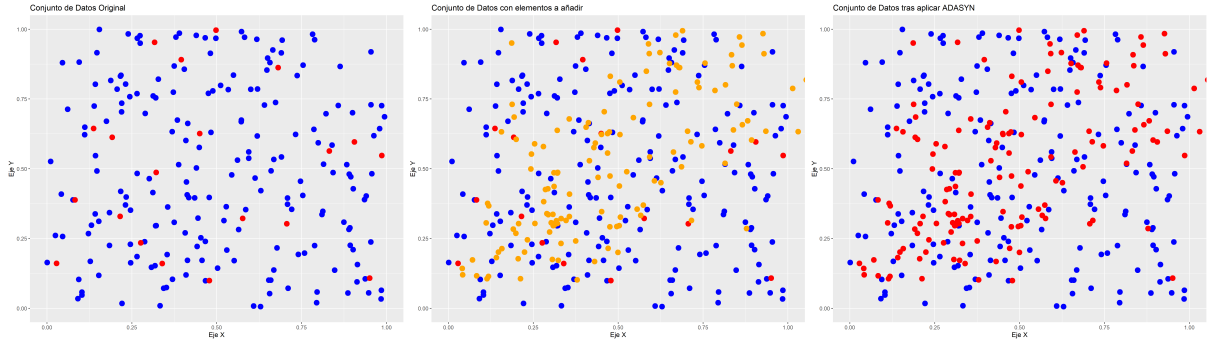


Figura 3.10: Ejemplo ADASYN

Otras técnicas

Las técnicas basadas en remuestreos son abundantes, se han detallado las más relevantes. Por enumerar, entre las técnicas fundamentadas en la estrategia de los vecinos más cercanos y que no han sido mencionadas, se encuentran *All k-Nearest Neighbors* (AKNN) [77], *Neighbour Cleaning Rule* (NCL) [47] reduciendo la presencia de outliers, *Repeated Edited Nearest Neighbour* (RENN) [77].

También destacar técnicas posteriores a SMOTE que introducen ciertas mejoras en cuanto a la elección de los vecinos más cercanos: *Radial-Based Undersampling* [46], bebiendo a su vez de las ideas del método *Radial-Based Oversampling* (RBO), *Adaptative Mahalanobis Distance-based Over-sampling* (AMDO) [88] que introduce una descomposición de valores singulares para los distintos tipos de instancias. Éste último está basado en MDO (*Mahalanobis Distance-based Over-sampling*) que emplea el cálculo de la covarianza para generar las instancias de las clases minoritarias.

3.1.6. Técnicas basadas en la función de coste o pérdida

Estas técnicas o métodos sensibles al coste tienen como base el coste o error asociado de clasificar incorrectamente una instancia del conjunto de datos. Una función de pérdida se construye sumando los costes individuales o simplemente los errores obtenidos al clasificar cada elemento del conjunto. Es necesario contar con una buena aproximación de los costes referentes a cada error de clasificación.

Existen numerosas variantes a la hora de implementarlo en los algoritmos de *Machine Learning*. Por la naturaleza de este trabajo, se limita esta descripción a las funciones de pérdida dentro del ámbito de las redes neuronales. En estas, las funciones de coste se incorporan directamente en los algoritmos, consistiendo el aprendizaje durante el entrenamiento en minimizar estas funciones.

La función de pérdida toma la forma

$$E(\theta) = \sum_{i=1}^{N} f(\mathbf{y}^{(i)}, \mathbf{p}_\theta^{(i)})$$

donde N es el número total de elementos del conjunto de datos a clasificar, $\mathbf{y}^{(i)}$ constituye el vector normalizado de valores reales esperados para cada uno de los elementos, mientras que $\mathbf{p}_\theta^{(i)}$ conforman los valores predichos por el modelo para ese elemento, parametrizados por los pesos y sesgos de la red. Para simplificar la notación se considerará en adelante $\mathbf{p}^{(i)} = \mathbf{p}_\theta^{(i)}$. Es de observar que, al tratarse de probabilidades, la suma de las posiciones del vector $\mathbf{y}^{(i)}$ debe ser igual a uno

$$\sum_{i=1}^{n} \mathbf{y}_i^{(i)} = 1$$

siendo n el número de neuronas en la capa de salida, coincidente con el número de clases existentes en el conjunto de datos a analizar.

De esta forma, el aprendizaje del modelo tiene como objetivo la búsqueda del parámetro θ óptimo que arroja el menor valor posible de la función de error $E(\theta)$

$$\theta = arg\min_{\theta'} E(\theta')$$

Este tipo de técnicas pueden mejorar significativamente la clasificación de un conjunto de datos desbalanceado al poner el foco en la errónea clasificación de las instancias de las clases minoritarias.

Por ejemplificarlo, existen métodos que asignan diferentes valores de coste en función de la clase a la que pertenezca el elemento clasificado erróneamente, penalizando notablemente un dato mal clasificado perteneciente a una clase minoritaria frente a uno incorrecto de la clase mayoritaria.

A continuación se enumeran brevemente las funciones de pérdida más comunes para este propósito.

Error cuadrático medio, MSE

Minimiza el error cuadrático de la diferencia entre el valor esperado y el predicho.

$$E(\theta) = \sum_{i=1}^{N} f(\mathbf{y}^{(i)}, \mathbf{p}_\theta^{(i)}) = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{n} \left(\frac{(y_j^{(i)} - p_j^{(i)})^2}{2} \right)$$

entendiendo la notación $p_j^{(i)}$ como la probabilidad de que el elemento i pertenezca a la clase j.

Mean False Error, MFE

Esta función [80] tiene en consideración los falsos positivos y falsos negativos que se recogen en la matriz de confusión. Se denomina, por simplicidad, clase positiva a la clase o conjunto de clases las cuales poseen los elementos que se pretende clasificar correctamente. Generalmente se corresponde con la clase o clases minoritarias. En contraposición, la clase negativa lo constituye el conjunto de clases de las que se posee menor interés en clasificar correctamente. En función de la clasificación que arroje el modelo, se pueden observar cuatro escenarios posibles:

- **Verdadero positivo (VP)**: aquellos casos en los cuales el modelo predijo correctamente una instancia perteneciente a la clase positiva.

- **Verdadero negativo (VN)**: aquellos casos en los cuales el modelo predijo correctamente una instancia perteneciente a la clase negativa.
- **Falso positivo (FP)**: aquellos casos en los cuales el modelo predijo una instancia como si perteneciera a clase positiva, cuando en realidad ésta pertenecía a la clase negativa. Se conoce también con el nombre de error de tipo I.
- **Falso negativo (FN)**: aquellos casos en los cuales el modelo predijo una instancia como si perteneciera a clase negativa, cuando en realidad ésta pertenecía a la clase positiva. Se conoce también con el nombre de error de tipo II.

La matriz de confusión aúna estos cuatro conceptos bajo una misma estructura

VP	FP
FN	VN

De esta forma se consigue que el modelo sea más sensible a los errores pertenecientes a la clase minoritaria. La función de error es la suma de la función que calcula el error de falsos positivos, FPE, y la función de error de los falsos negativos, FNE.

$$FPE = \frac{1}{A}\sum_{i=1}^{A}\sum_{j=1}^{n}\left(\frac{(y_j^{(i)} - p_j^{(i)})^2}{2}\right)$$

$$FNE = \frac{1}{B}\sum_{i=1}^{B}\sum_{j=1}^{n}\left(\frac{(y_j^{(i)} - p_j^{(i)})^2}{2}\right)$$

$$E(\theta) = FPE + FNE$$

siendo A y B el número de elementos en la clase negativa y positiva respectivamente.

Mean Squared False Error, MSFE

Aumenta la precisión con respecto a la función de pérdida anterior. MSFE se fundamenta en el hecho de que el error cuantificando los falsos negativos contribuya en mayor medida que el FPE. Se formula como el cuadrado de la suma de ambas cantidades tratadas.

$$E(\theta) = (FPE + FNE)^2$$

Cross Entropy Loss

La función de entropía cruzada [91] supone una mejora significativa con respecto a las métricas anteriores. Consigue arrojar ajustes valiosos para el descenso del gradiente, algoritmo más común para optimizar los pesos y sesgos de un modelo de clasificación.

$$L_{CE} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{n} y_j^{(i)} \log(p_j^{(i)})$$

Bilinear Loss y Log Bilinear Loss

En estas funciones de pérdida se tiene en consideración la matriz de coste y, en consecuencia, consiguen un incremento de la precisión. Esto es debido a considerar la cuantificación de los distintos errores de clasificación de los elementos del conjunto de datos.

Se define la matriz de coste M como aquella en la que cada elemento $m_{i,j}$ se corresponde con el costo de asignar a un elemento la clase j cuando verdaderamente pertenece a la clase i. De esta forma, los costes no negativos se asignan a las clasificaciones incorrectas.

La función de pérdida *Bilinear Loss* se define como

$$L_B = \mathbf{y}^T M \mathbf{p}$$

La función de pérdida bilineal es conveniente cuando se pretende encontrar dónde se concentran los errores de clasificación.

De forma análoga la función *log-Bilinear loss* [67] toma la forma

$$L_{LB} = -\mathbf{y}^T M \log(1 - \mathbf{p})$$

Estas funciones se combinan linealmente a su vez con la función de pérdida basada en la entropía cruzada, beneficiándose de las ventajas que proporcionan ambas.

$$L_{CE+B} = (1 - \alpha) L_{CE} + \alpha \mathbf{y}^T M \mathbf{p} \qquad 0 < \alpha < 1$$

$$L_{CE+LB} = (1 - \alpha) L_{CE} - \alpha \mathbf{y}^T M \log(1 - \mathbf{p}) \qquad 0 < \alpha < 1$$

3.1.7. Técnicas basadas en redes GAN

Las redes generativas adversarias (GAN) fueron introducidas en 2014 en [28] con el objetivo de generar nuevas instancias que sigan la misma distribución que el conjunto de datos original. Están formadas por dos redes neuronales, una red generadora G y otra red discriminadora D como muestra la Figura 3.11.

La red generadora G tiene como función generar las nuevas instancias respetando la distribución de los elementos del conjunto de entrenamiento. Por su parte, la red discriminadora D discierne si las instancias de entrada pertenecen al conjunto original, es decir, son reales o por el contrario han sido generadas artificialmente. Este discriminador penaliza al generador si produce muestras inverosímiles. El objetivo del entrenamiento de una red GAN consiste en conseguir que el generador maximice sus posibilidades de no ser detectado, mientras que el discriminador afine su método de detección entre las instancias reales y falsas. Esta competencia entre ambas redes tiene como resultado que el generador cada vez arroje instancias más realistas, semejantes a las ya existentes en el conjunto de datos original.

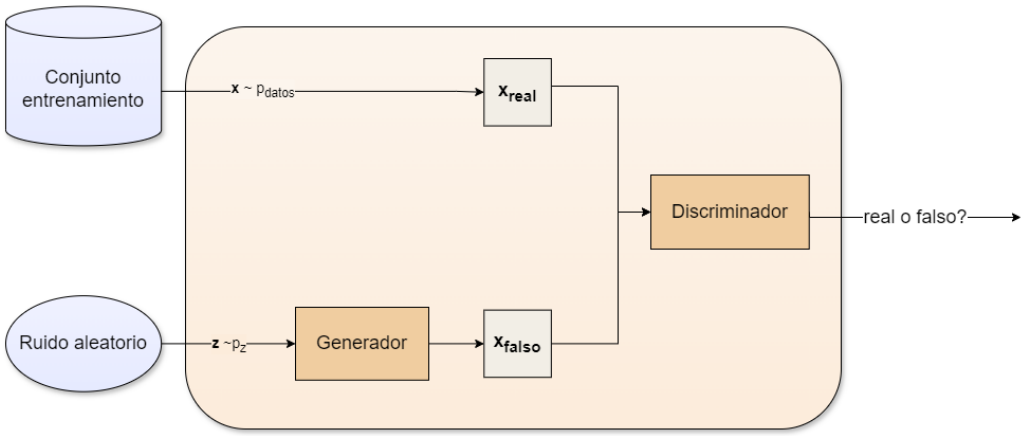


Figura 3.11: Esquema GAN

Las redes generativas adversarias implementan a su vez modelos para subsanar el desbalanceo de datos, basándose en el mismo problema de que las instancias de las clases minoritarias no son suficientes para poder entrenar correctamente una red GAN.

Por ser el propósito de este trabajo, se va a profundizar en técnicas que tomen como tipo de datos imágenes. Este conjunto de métodos logran un aumento de la precisión y de la calidad de las imágenes, consiguiendo a su vez gran variabilidad en las imágenes generadas de forma artificial.

En contraposición, existen limitaciones, especialmente debido a que, con recursos computacionales limitados, se generan modelos inconsistentes. Otra desventaja es que sigue siendo un campo abierto de investigación el encontrar una métrica precisa para evaluar la calidad de las instancias generadas.

Generative Adversarial Minority Oversampling, GAMO

GAMO [59] constituye una técnica de *oversampling* en el ámbito del *Deep Learning*. Posee tres fases diferenciadas: un generador convexo, un clasificador multiclase y un discriminador, como se muestra en la Figura 3.12.

El generador convexo tiene como función producir las instancias pertenecientes a las clases minoritarias como combinaciones convexas de las muestras ya existentes. Por la forma de proceder, las instancias generadas se sitúan próximas a la frontera de la clase a la que pertenecen. El propósito de esto es conseguir que el clasificador discierna correctamente los límites de las clases, siendo así más robusto frente a clases desbalanceadas.

El discriminador, por su parte, cumple la función de asegurar que las instancias generadas sigan la distribución prevista por las clases minoritarias. Cuando mayor relevancia toma es en el momento en el cual una clase minoritaria no constituye un conjunto convexo.

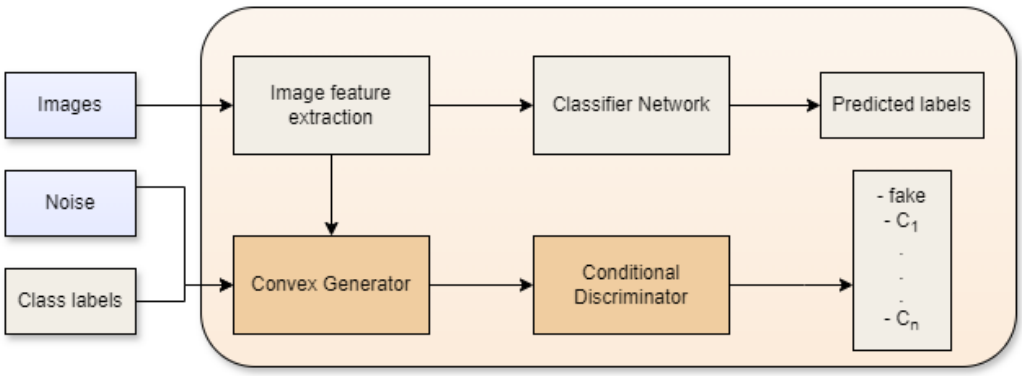


Figura 3.12: Esquema GAMO

Balancing Generative Adversarial Networks, BAGAN

BAGAN [53] se fundamenta en el concepto de crear instancias de la clase minoritaria (*over-sampling*), haciendo uso de una combinación eficiente entre el codificador (*encoder*) y el decodificador (*decoder*). Parte de tres premisas, las imágenes generadas deben pertenecer a las clases deseadas, además no debe generar imágenes repetitivas entre ellas ni idénticas a las ya existentes en el conjunto de datos original. BAGAN introduce dos novedades fundamentales con respecto a las técnicas precedentes:

- Hace uso de una estructura de autoencoder que permite inicializar el entrenamiento de la red, proporcionando a ésta información acerca de la distribución de las clases existentes. De esta forma se proporciona estabilidad, evitando que la red GAN colapse.
- El discriminador combina en una única salida la pérdida que obtiene al discernir entre la categoría real/falsa ($\{real, fake\}$) y la pérdida producida al clasificar la imagen en una de las clases existentes $\{c_1, ..., c_n\}$. De esta forma el discriminador únicamente debe clasificar la imagen en una categoría, incluyendo la categoría falsa como una más $\{c_1, ..., c_n, fake\}$.Esto asegura que se realice un entrenamiento equilibrado entre las clases.

El artículo original emplea el cálculo de una métrica de similaridad, como por ejemplo SSIM (*Structural Similarity*) [81], para cuantificar la proximidad entre dos imágenes y por tanto determinar la consecución o no de los objetivos propuestos. SSIM toma el valor uno cuando ambas imágenes son idénticas y decrece hasta cero dependiendo del grado de diversidad que exista entre ellas.

BAGAN se desarrolla en tres fases: un entrenamiento del autoencoder, una inicialización de GAN, y un entrenamiento de esta red adversaria generativa. La primera parte consiste en entrenar el autoencoder para que obtenga información acerca de la distribución de las clases y así poder inicializar la GAN cerca de una buena solución, evitando que ésta colapse. Esto se muestra en la Figura 3.13. Se trata de un aprendizaje no supervisado puesto que no posee información sobre la distribución inicial de clases. Se entrena empleando todas las imágenes del conjunto de datos, independientemente de la clase a la que puedan pertenecer. El codificador reduce una imagen y la traslada al espacio latente para, posteriormente, el decodificador a partir de un vector del espacio latente, reconstruye la imagen inicial. Se emplea además como función de pérdida la norma l_2 entre las imágenes originales y las reconstruidas.

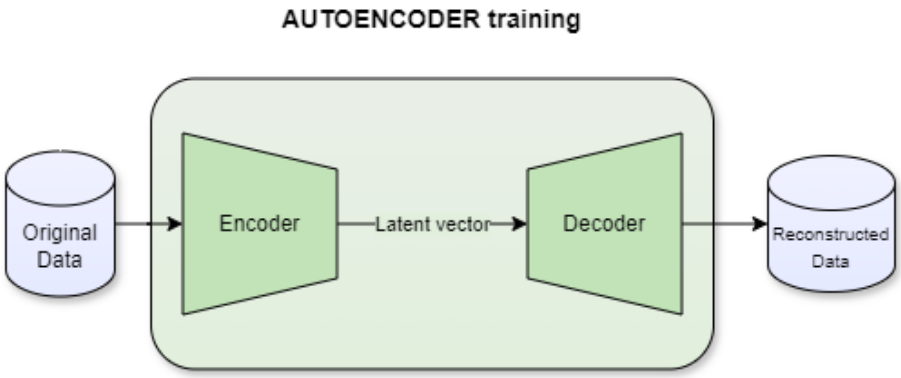


Figura 3.13: Esquema BAGAN. Entrenamiento del autoencoder

Posteriormente, el conocimiento del entrenamiento es transferido a la red GAN, consiguiendo inicializar los pesos acorde al propósito e inferir en el espacio latente la distribución que posee cada una de las clases. Tanto el generador G, como el discriminador D, poseen conciencia explícita de cada una de las clases. Se inicializa de forma que el generador recibe los pesos ajustados del decodificador del autoencoder, mientras que las primeras capas del discriminador se inicializan con los pesos del codificador. Además, la última capa del discriminador es una capa densa con función de activación Softmax. Lo descrito es posible debido a que el decodificador del autoencoder posee la misma topología que el generador, así como el codificador del autoencoder y las primeras capas del discriminador cuentan con idéntica estructura. Partiendo de la hipótesis de que las clases siguen una distribución normal, el generador tiene por función obtener la distribución $N_c = N(\mu_c, \Sigma_c)$ original de las clases, mediante el cálculo de la media μ_c y la matriz de covarianzas Σ_c de cada una de las clases.

El motivo de la inicialización del discriminador es incluir características significativas útiles en el momento de clasificar las imágenes. Por su parte, la inicialización del generador consigue equiparar los elementos de entrada en el generador con los elementos pertenecientes al espacio latente del autoencoder para partir de una inicialización más precisa en vistas a un correcto entrenamiento de la red, evitando que colapse.

La última fase consiste en el entrenamiento propio de la red. Los conjuntos de datos se envían al generador y al discriminador ajustando sus pesos con el fin de reducir la función de pérdida. En esta fase, el generador G se encarga de generar imágenes de las diferentes clases. El discriminador, por su parte, trata de discernir la clase de cada una de las imágenes generadas, incluyendo la clase "falsaçomo una más, indicando que esa imagen ha sido generada artificialmente. Además en cada lote de entrenamiento el número de elementos de imágenes generadas artificialmente es similar al del resto de clases. De esta forma los gradientes de propagan de la misma forma para cada una de las clases, no ignorando así los elementos pertenecientes a la clase minoritaria.

Estas dos últimas fases quedan recogidas en la Figura 3.14.

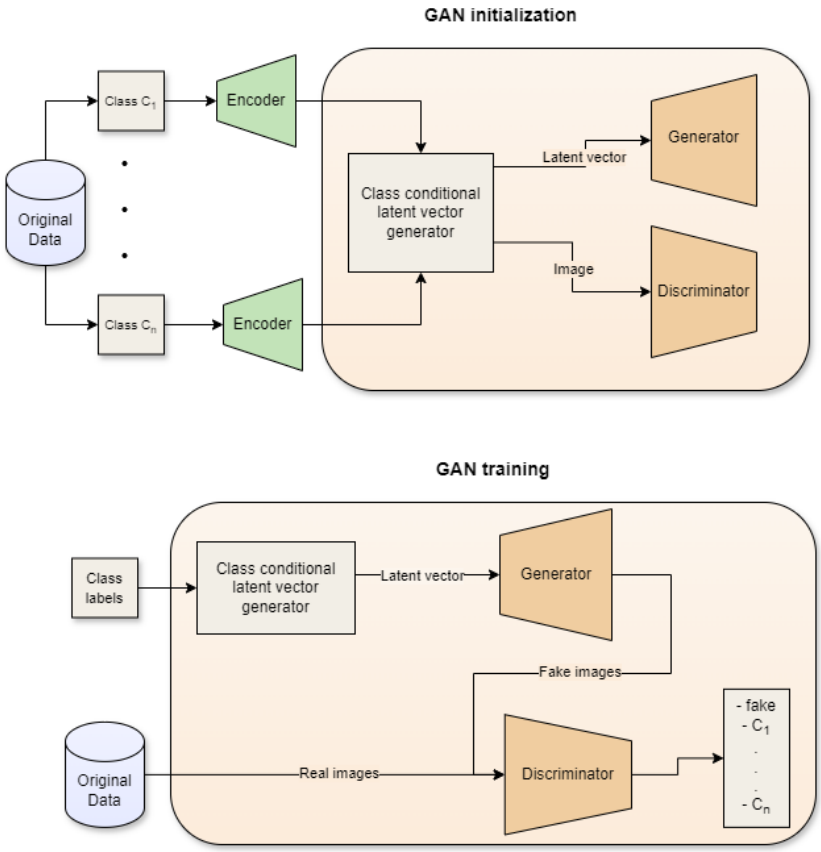


Figura 3.14: Esquema BAGAN. Inicialización y entrenamiento

Por otra parte, BAGAN cuenta con una serie de limitaciones, la principal es que frente a conjuntos de datos en los cuales existe poca distinción entre las clases este método resulta inestable.

Enhanced BAGAN with gradient penalty, BAGAN-GP

Como su propio nombre indica, BAGAN-GP [37] nace de la necesidad de solventar las desventajas que posee BAGAN. La arquitectura empleada coincide con la de este método, a excepción de dos mejoras:

- Propone el empleo de un modelo de *embedding* dentro del autoencoder, que ayude a éste a conseguir conciencia sobre la distribución de cada una de las clases. Además el entrenamiento del autoencoder es supervisado, teniendo conocimiento de la clase a la que pertenece cada imagen. Con esta estructura adicional ya no es necesario presuponer que los vectores del espacio latente siguen distribuciones normales como ocurría con el modelo original BAGAN. De esta forma se evitan solapamientos entre las distribuciones, de forma que no se generan muestras como resultado de la combinación de dos clases distintas.

- Mejora la función de pérdida, añadiendo una penalización del gradiente propio de otras arquitecturas. Esta necesidad surge de que cuando se trabaja con conjuntos desbalancea-

dos, los gradientes tienden a la predicción de las clases mayoritarias, sin apenas tener en consideración las minoritarias.

En primer lugar sustituye la función de pérdida original de BAGAN (divergencia de Kullback-Leibler Jensen-Shannon) por la distancia de Wasserstein. Mejora el rendimiento y la estabilidad del entrenamiento al tratarse de una función continua y diferenciable, de forma que los gradientes son siempre significativos. El uso de esta distancia como función de pérdida no es novedosa, BAGAN-GP se inspira en la técnica WGAN [4] que la introdujo por primera vez. La distancia de Wasserstein o también denominada EMD (Earth Mover Distance) mide el costo de transformar una distribución de probabilidad en otra.

A esta función de pérdida del discriminador, se le añade un término penalizador. Esto ya aparecía evidenciado y empleado en los métodos WGAN-GP [32] y DRAGAN [31]. Este término está basado en la equivalencia entre la norma de los gradientes y las funciones 1-Lipschitz continuas $||\nabla D(x)||_2 \leq 1$.

Por otra parte, es necesario aplicar una GAN condicional para generar las imágenes pertenecientes a las clases minoritarias. BAGAN-GP en su implementación replica las arquitecturas existentes en ACGAN [62] y cWGAN-GP [90].

Todo esto permite que BAGAN-GP converja de forma más rápida y estable. Además, las imágenes generadas poseen mayor calidad. Al igual que BAGAN posee tres fases: una inicialización y entrenamiento del autoencoder (Figura 3.15), una inicialización de la red BAGAN-GP y un entrenamiento de la propia red (Figura 3.16). Esta arquitectura se encuentra especificada en las figuras citados. Además se muestra de forma detallada en la Figura 3.17 la estructura del discriminador.

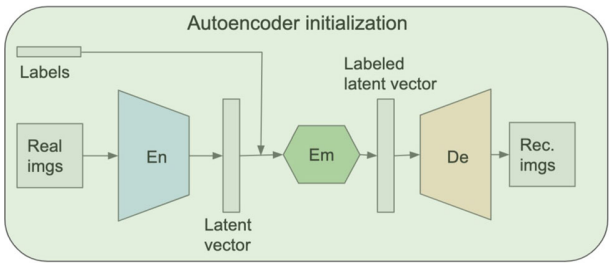


Figura 3.15: Estructura del autoencoder BAGAN-GP. Imagen obtenida de [37]

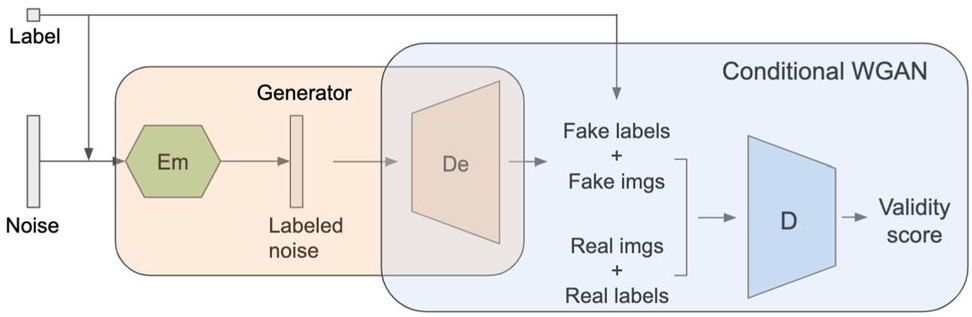


Figura 3.16: Estructura red generativa adversaria BAGAN-GP. Imagen obtenida de [37]

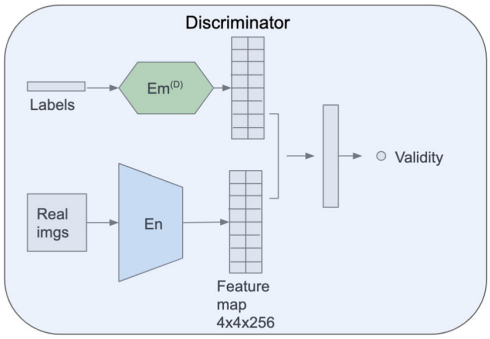


Figura 3.17: Discriminador BAGAN-GP. Imagen obtenida de [37]

3.1.8. DeepSMOTE

En 2022, D.Dablain, B. Krawczyk y N. Chawla publicaron el algoritmo DeepSMOTE [19]. Consiguieron de esta forma aunar el método SMOTE, algoritmo de *resampling*, con el *Deep Learning*. Solventan así el principal problema de SMOTE: su coste computacional puesto que requiere grandes cantidades de memoria para cargar el conjunto de datos completo. DeepSMOTE constituye una solución efectiva para el problema de clasificación en clases desbalanceadas que se está tratando.

Con este método novedoso quedan satisfechos fundamentalmente tres criterios:

- Es un modelo que trabaja de extremo a extremo.
- Estudiando la distribución de los datos, es capaz de reducir el espacio de características. Esto se emplea en la labor de sobremuestreo.
- Consigue generar una salida clara y con alta calidad para ser inspeccionada sin necesidad de realizar complejas manipulaciones para ese propósito. Las instancias generadas poseen la misma calidad que los elementos del conjunto de datos original.

A su vez, DeepSMOTE introduce una serie de ventajas y novedades.

- Proporciona una arquitectura robusta y consistente. Permite realizar el balanceo del conjunto de datos haciendo frente a conjuntos de datos extremadamente desbalanceados. Además, admite como entrada datos complejos, tales como imágenes.
- No requiere el uso de un discriminador durante el entrenamiento del modelo. Esto supone un avance notable con respecto a métodos basados en redes generativas adversarias, como BAGAN, BAGAN-GP o GAMO. DeepSMOTE emplea una función de pérdida que asegura un correcto entrenamiento del generador.

DeepSMOTE emplea una estructura basada en DCGAN pero con ligeras modificaciones. Una red adversaria generativa convolucional profunda (DCGAN) [63], está fundamentada en el uso de un generador y un discriminador. DeepSMOTE toma como partida que la estructura generador-discriminador de DCGAN se asemeja a un codificador - decodificador donde el discriminador codifica la entrada y el generador funciona como un decodificador arrojando la salida.

El procedimiento y composición exhaustiva de cada una de las estructuras se describe a continuación. El codificador está conformado por una red convolucional con cuatro capas, cada una con función de activación Leaky ReLU [52]. La función de activación de la última capa es lineal. Por su parte, el decodificador está compuesto por una red neuronal con cuatro capas de convolución traspuesta y funciones de activación de tipo ReLU [61] excepto en la última capa, que emplea la función tangente hiperbólica.

Por otro lado, se emplea ADAM [45] para la optimización de parámetros de ambas redes con una tasa de aprendizaje de 0.0002. Se emplea *batch normalization* [2] entre cada capa de cada una de las redes, excepto la capa de salida del decodificador y la capa de entrada del codificador. De esta forma se consigue normalizar, media cero y varianza uno, cada uno de los elementos de entrada en las capas. Esto incrementa la estabilidad y robustez del modelo.

Entrenamiento del modelo

Se podría dividir el entrenamiento en dos fases, cada una de las cuales arroja una función de pérdida que posteriormente se combina para obtener el error de cada iteración. La primera fase tiene como entrada el conjunto con elementos pertenecientes a todas las clases, mientras que la segunda fase divide al conjunto en clases antes de trabajar con él.

El entrenamiento se realiza por lotes. De esta forma, se extraen aleatoriamente del conjunto de entrenamiento tantas imágenes como sea necesario para completar un *batch* o lote. Cada lote es procesado por una estructura codificador-decodificador, arrojando la función de pérdida ($loss_1$) basada en el error cuadrático medio. Es necesario mencionar que tanto codificador como decodificador se entrenan con elementos pertenecientes a todas las clases existentes. Luego el modelo es capaz de reconstruir tanto las clases minoritarias como las mayoritarias. El papel de las clases mayoritarias consiste en ayudar a reconstruir patrones comunes presentes en los datos, puesto que parte de la premisa de que las clases del conjunto de datos comparten características similares.

Para enriquecer esta función de pérdida, se añade un término penalizador. El proceso para conseguirlo comienza escogiendo un número determinado de instancias pertenecientes todas a una única clase. La elección de esta clase se realiza de forma aleatoria, mientras que el número de instancias escogidas en esta fase coincide con el número de imágenes empleadas en la fase anterior, es decir, el tamaño del lote. El codificador reduce este subconjunto a un espacio de características de dimensión menor. Tras pasar por el decodificador, estos elementos no son reconstruidos en el mismo orden que poseían inicialmente. Se aplica previamente una permutación dentro del subconjunto. El propósito de cambiar el orden es introducir cierta varianza en los datos, de forma que facilite la generación de instancias posterior. Se calcula la función de pérdida ($loss_2$) de esta fase, también basada en el error cuadrático medio.

La función de pérdida de la fase de entrenamiento del modelo, se conforma entonces sumando ambas funciones de pérdida calculadas.

$$loss = loss_1 + loss_2$$

La segunda función de pérdida $loss_2$ es empleada como factor de penalización implícita en función de pérdida del modelo.

[2] *Batch normalization* consiste realizar una normalización de los elementos relativos a cada *batch* a medida que van pasando por cada capa de la red y antes de que pasen por la función de activación. Realizar esta técnica favorece una mejor convergencia del entrenamiento del algoritmo.

Por enfatizar lo expuesto, la novedad de DeepSMOTE radica principalmente en la existencia de esta función de pérdida que posee el término penalizador. Además, no es necesario el uso de un discriminador, puesto que la introducción de la técnica de permutar las imágenes añade la suficiente variabilidad para que el generador sea entrenado de forma robusta.

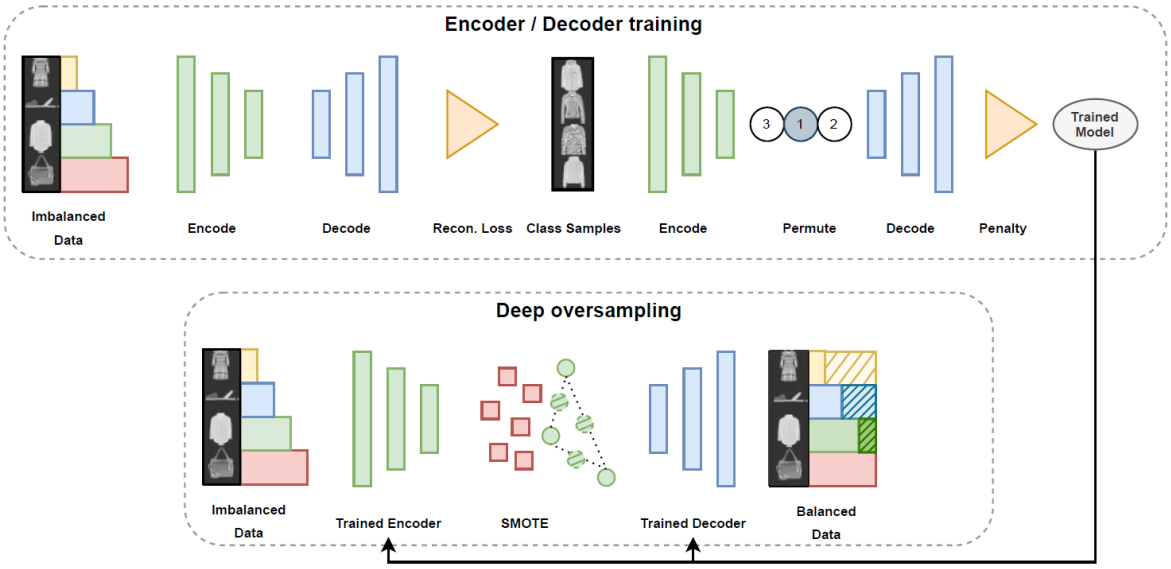


Figura 3.18: Esquema DeepSMOTE

Generación de instancias

Una vez finalizado el entrenamiento, la generación de instancias artificiales sigue la estructura codificador-decodificador. El codificador reduce el espacio de características, posteriormente se aplica SMOTE. El decodificador se encarga de reconstruir una imagen (en el mismo espacio de características que las imágenes originales) a partir del vector generado mediante SMOTE.

Para generar imágenes mediante SMOTE, se escoge en cada iteración una clase minoritaria y sus clases vecinas. Se escoge un elemento de la clase minoritaria y se calcula la distancia a uno de sus vecinos seleccionado aleatoriamente. Esta distancia se multiplica por un factor $(0 < \alpha < 1)$ y se añade a la imagen escogida para generar una instancia sintética.

Existe una diferencia principal entre la fase de entrenamiento y la de generación de instancias de DeepSMOTE. En la fase de entrenamiento, el procedimiento de SMOTE es sustituido por una permutación de los datos, cumpliendo de igual manera con el objetivo de esta técnica, introducir variación en los datos. La justificación de por qué puede realizarse radica en que SMOTE no necesita un entrenamiento previo al tratarse de un método no paramétrico.

La Figura 3.18 extraída de [19], resume el proceso descrito. Se adjunta además, para mayor claridad, un Pseudocódigo 1 de las fases de entrenamiento y generación de las imágenes con DeepSMOTE.

Algoritmo 1: DeepSMOTE

Data: conjunto de lotes $B = \{b_1, ..., b_n\}$
Parámetros del modelo: $\theta = \{\theta_1, ..., \theta_k\}$
Tasa de aprendizaje: α
Result: conjunto balanceado

Entrenamiento
for *e en epochs* **do**
  for *b en B* **do**
    $E_b \longleftarrow$ codificar(b)
    $D_b \longleftarrow$ decodificar(E_b)
    $R_L = \frac{1}{m}\sum_{i=1}^{m}(D_{b_i} - b_i)^2$
    $C_D \longleftarrow$ muestra(clase$_j$)
    $E_S \longleftarrow$ codificar(C_D)
    $P_E \longleftarrow$ permutación(E_S)
    $D_P \longleftarrow$ decodificar(P_E)
    $P_L = \frac{1}{m}\sum_{i=1}^{m}(D_{P_i} - CD_i)^2$
    $T_L = R_L + P_L$
    $\theta = \theta - \alpha\frac{\partial T_L}{\partial \theta}$

Generación instancias
$M :=$ {clases minoritarias}
for *i en* $|M|$ **do**
  $C \longleftarrow$ seleccionar(clase$_j$)
  $E \longleftarrow$ codificar(C)
  $G \longleftarrow$ SMOTE(E)
  $S \longleftarrow$ decodificar(G)

3.1.9. Elección de las técnicas de trabajo

Una vez detalladas todas estas técnicas de balanceo de datos, es necesario escoger con cuales se va a trabajar. Como mínimo se escogen dos técnicas para poder realizar una comparación efectiva. Puesto que se pretende trabajar con una técnica de generación de imágenes, la elección se reduce a elegir una técnica basada en redes GAN y por otra parte DeepSMOTE. De entre las técnicas basadas en redes GAN se escoge BAGAN-GP. Es la más novedosa y pretende paliar los inconvenientes de la técnica BAGAN publicada con anterioridad. De igual forma, DeepSMOTE es una elección acertada al ser también una técnica reciente, además de innovadora. Es interesante constatar el rendimiento de una técnica que no emplea un discriminador, si no una función de pérdida mejorada. Por todo lo relatado, se va a trabajar con las técnicas DeepSMOTE y BAGAN-GP.

Parte II

Desarrollo de la propuesta y resultados

Capítulo 4

Desarrollo de la propuesta y experimentación

Se presenta detalladamente en este capítulo las dos propuestas escogidas de balanceo de datos: DeepSMOTE y BAGAN, sobre las cuales se va a realizar un análisis y ajuste haciendo uso de distintos conjuntos de datos y combinaciones de hiperparámetros para obtener un correcto entrenamiento de los modelos. Se presentarán primeramente conceptos de carácter técnico y descriptivo necesarios para el desarrollo de la propuesta, para posteriormente mostrar la arquitectura de cada uno de los modelos y su proceso de entrenamiento y generación de imágenes.

4.1. Diseño experimental

4.1.1. Conjuntos de datos

Esta sección recoge los distintos conjuntos de datos (*datasets*) empleados para el entrenamiento, generación de imágenes y posterior clasificación. Han sido tres los conjuntos de datos escogidos: MNIST, FashionMNIST y CIFAR10.

MNIST

MNIST (*Modified National Institute of Standards and Technology database*) [50] es el conjunto de datos por excelencia para labores de visión computacional. Consta de diez clases, cada una de las cuales se corresponde con los dígitos manuscritos desde el cero hasta el nueve y representadas en la Figura 4.1 . Las imágenes poseen un tamaño de 28×28 píxeles, con un solo canal de color, es decir, se encuentran en escala de grises. El conjunto de entrenamiento consta de 60000 imágenes mientras que el conjunto de test consta de 10000 imágenes.

Inicialmente estas imágenes fueron obtenidas del conjunto de datos que recogió el NIST (*National Institute of Standards and Technology*) empleando dígitos manuscritos de los empleados de la oficina del censo de Estados Unidos y de estudiantes de secundaria. Poseían tamaño 20×20, luego para obtener el conjunto MNIST se normalizaron y suavizaron las imágenes.

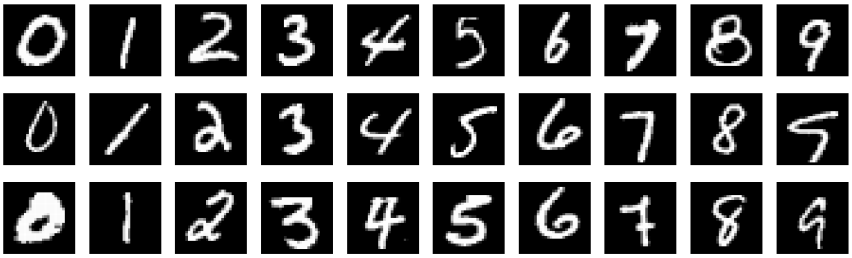


Figura 4.1: MNIST [50]

FashionMNIST

Este conjunto de datos [87] nace en 2017 y está formado por imágenes de prendas de vestir proporcionadas por Zalando. Nuevamente contiene diez categorías:

0. Camiseta
1. Pantalones
2. Jersey
3. Vestido
4. Abrigo
5. Sandalias
6. Camisa
7. Zapatos
8. Bolso
9. Botas

Las imágenes poseen un tamaño de 28×28 píxeles en escala de grises. El conjunto de entrenamiento consta de 60000 imágenes y el conjunto de test de 10000. Una muestra de ellas se encuentra representado en la Figura 4.2.

Figura 4.2: FashionMNIST [87]

CIFAR10

CIFAR10 (*Canadian Institute For Advanced Research*) [48] está formado por un conjunto de 60000 imágenes, 50000 destinadas al conjunto de entrenamiento y 10000 destinadas al test. Poseen tamaño 32×32 píxeles contando con 3 canales de color (RGB). Las imágenes se encuentran agrupadas en diez clases y representadas en la Figura 4.3

0. Avión
1. Automóvil
2. Pájaro
3. Gato
4. Ciervo
5. Perro
6. Rana
7. Caballo
8. Oveja
9. Camión

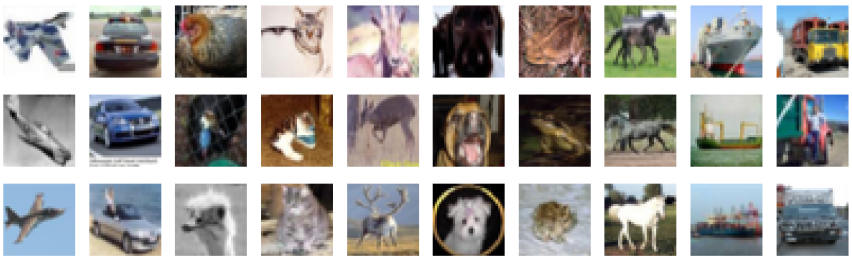


Figura 4.3: CIFAR10 [48]

4.1.2. Hiperparámetros

Los modelos realizados poseen una serie de parámetros externas que el usuario ajusta con el propósito de controlar el proceso de aprendizaje. De esta forma es posible tener cierto control sobre la calidad y velocidad de convergencia.

- **Número de épocas o *epochs***: una época es la ejecución de un algoritmo con todo el conjunto de datos de entrenamiento una única vez, un único ciclo. De esta forma se ajusta el número de épocas, que es un número entero, para repetir el proceso y favorecer la convergencia del algoritmo. Si el número de épocas es reducido, el modelo no será capaz de aprender la distribución de los elementos y producirá subajuste, pero en el caso contrario puede llegar a producir sobreajuste. Cuando una época es lo suficientemente grande, se divide en un conjunto de lotes.
- **Tamaño de lote o *batch size***: representa el número de elementos empleado en una iteración de la red. Influye directamente en la eficiencia y *accuracy* del proceso de entrenamiento. Si se escogen tamaños de lote grandes, el entrenamiento se realiza de forma más rápida pero consiguiendo una baja *accuracy* e incluso un sobreajuste. El caso contrario, en el que se elige un tamaño pequeño, permite obtener mayor *accuracy* pero se convierte en un proceso computacionalmente más costoso y menos susceptible a variaciones en el conjunto de datos.
- **Tasa de aprendizaje o *learning rate***: determina la magnitud de actualización de los pesos y sesgos de la red. Posee influencia directa sobre la función de pérdida controlando así el cambio que se produce en ella. Si toma un valor pequeño significa que la convergencia del entrenamiento va a ser lenta, por el contrario, si el valor es alto puede conducir a un modelo que ha aprendido demasiado rápido y que por tanto sea inestable.

- **Función de pérdida**: es una función que cuantifica la diferencia entre los valores reales y los valores predichos por el modelo.

- **Optimizador**: tiene por objetivo optimizar los pesos y sesgos de la red empleando un procedimiento denominado *backpropagation*. Tiene en cuenta el valor del *learning rate*. Dos optimizadores comunes son SGD (*Stochastic Gradient Descent*) [30] y Adam (*Adaptative Moment Estimation*) [45].

- **Número de capas de la red**: como su propio nombre indica la profundidad de capas de un modelo.

- **Dimensión del espacio latente**: dimensión que va a tomar el espacio latente subyacente.

4.2. Propuesta

Este trabajo va a realizar una comparativa entre dos técnicas de balanceo de datos: DeepSMOTE y BAGAN-GP. Estas técnicas se encargan de conseguir, a partir de un conjunto de imágenes cuyas clases se encuentran desbalanceadas, un conjunto de datos aumentado tras generar imágenes sintéticas pertenecientes a las clases minoritarias. Se detallarán por lo tanto las fases de entrenamiento y generación de imágenes.

Una vez generadas las imágenes y balanceado el conjunto de datos, para la evaluación de su rendimiento se hace uso de distintas redes convolucionales. Clasificando cada una de las imágenes y calculando las métricas pertinentes se consiguen obtener conclusiones valiosas. Para cada técnica y conjunto de datos se entrenan dos redes convolucionales: una con el conjunto de datos inicial desbalanceado y otra con el conjunto de datos aumentado. De esta manera se cuantifica de forma precisa el porcentaje de mejora de cada uno de los modelos. La Figura 4.4 describe de forma concisa lo relatado.

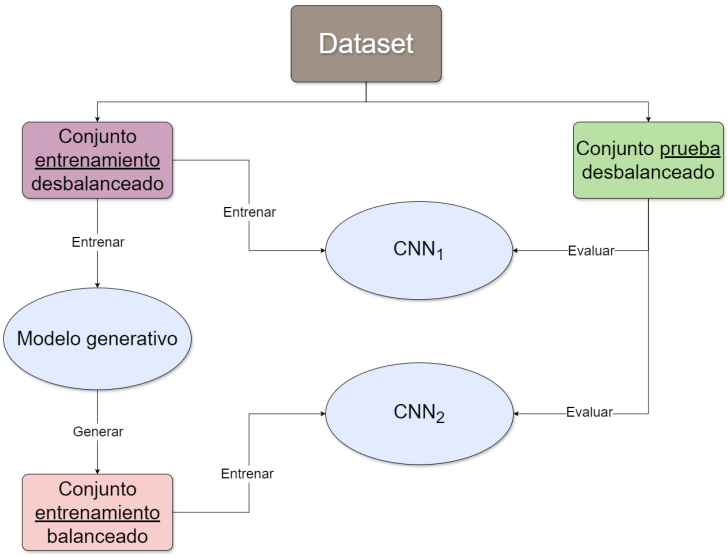


Figura 4.4: Flujo de trabajo

Implementación DeepSMOTE

Tras realizar una descripción teórica de este método, se procede a su compilación y experimentación. La base desde la cual partir es el código elaborado por sus creadores Dablain, Krawczyk y Chawla, alojado en el repositorio de GitHub [20].

Este repositorio contiene el código en *Python* de entrenamiento del modelo y el correspondiente para la generación de imágenes, ambos particularizados al caso del conjunto de datos MNIST. Proporcionan además dos ficheros de texto *.txt* que se corresponden con el conjunto de imágenes desbalanceado y sus correspondientes etiquetas de clase. Para este conjunto de datos se incluyen en una carpeta de *Google Drive* dos archivos *.pth* correspondientes al entrenamiento del codificador y decodificador del modelo.

Implementación BAGAN-GP

El código que se emplea como base para experimentación ha sido obtenido de un repositorio común de *GitHub* [24]. En este código los autores trabajan con tres conjuntos de datos: Fashion-MNIST, CIFAR10 y un conjunto que posee imágenes de células y sobre el cual ellos han basado su trabajo. El código emplea *Python* como lenguaje y la librería *Tensorflow* para conseguir trabajar con los conjuntos de datos y los algoritmos de *DeepLearning*. El repositorio cuenta con el propio archivo de entrenamiento y generación de imágenes de BAGAN-GP, otro archivo calcula la métrica *FID score* [89] que muestra el nivel de diversidad de los datos generados, otros se encargan de mostrar la distribución de los vectores del espacio latente.

BAGAN-GP en el procesamiento previo de los datos añade una novedad, además de desbalancear los conjuntos, aumenta la dimensión de las imágenes para que todas posean un tamaño de 64x64 píxeles. De esta forma se avanza en la dirección de que estas técnicas sean capaces de cada vez generar imágenes de más alta resolución y calidad.

4.3. Entrenamiento y generación de imágenes

Para obtener resultados directamente comparables es necesario que se entrene cada modelo generativo con los mismos conjuntos de datos. De esta forma se detalla a continuación la tasa de desbalanceo y el número de elementos por clase de cada uno de los conjuntos de datos empleados.

El conjunto de entrenamiento MNIST cuenta con 60000 imágenes. Para la labor desarrollada se realiza un filtrado aleatorio y desbalanceo de forma que se reduce este conjunto a 9000 imágenes. El número de elementos para cada clase se corresponde con los mostrados en la Tabla 4.1

Clase	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Número elementos	4000	2000	1000	750	500	350	200	100	60	40

Tabla 4.1: Número de elementos MNIST y FashionMNIST

La tasa de desbalanceo de las clases en el caso de FashionMNIST es el mismo que para MNIST, mientras que para CIFAR10 se ha empleado la configuración de clases que se muestra en la Tabla 4.2.

Clase	0	1	2	3	4	5	6	7	8	9
Número elementos	4500	2000	1000	800	600	500	400	250	150	80

Tabla 4.2: Número de elementos CIFAR10

Por otra parte, para conseguir ejecutar el código, se ha hecho necesario emplear en un primer momento la plataforma *Google Colab* para aumentar la capacidad de cómputo con una GPU de mayor potencia. El ajuste de hiperparámetros se ha realizado haciendo uso de la técnica *GridSearch* [44]. Puesto que ésta se encarga de ejecutar numerosas veces el entrenamiento con distintas combinaciones de hiperparámetros, la GPU que *Google Colab* presta de forma gratuita no es suficiente para el modelo DeepSMOTE. Este es el motivo por el cual se ha hecho uso de una GPU NVIDIA A40 con 48GB de RAM. Para el caso de BAGAN-GP, al poseer entrenamientos más ligeros, este proceso se ha podido llevar a cabo mediante la GPU NVIDIA T4 de *Google Colab*.

4.3.1. Ajuste de hiperparámetros

El ajuste de hiperparámetros o *GridSearch* es una técnica de *Machine Learning* que trata de encontrar la mejor combinación de hiperparámetros para un modelo dado. Este método funciona definiendo una malla con todos los posibles valores que se desea probar para cada hiperparámetro. A partir de esta malla se realiza el entrenamiento y evaluación del modelo para cada una de las combinaciones existentes, registrando sus funciones de pérdida o métricas de validación. De esta forma se escoge aquel modelo con el cual se obtenga el mejor rendimiento.

En este trabajo ha sido necesario ajustar los hiperparámetros para cada uno de los modelos generativos y cada uno de los conjuntos de datos empleados. Se ha hecho uso de la herramienta *Weights and Biases* integrada en *Python* mediante la librería *wandb*. Esta plataforma proporciona un seguimiento y registro de los entrenamientos realizados contando con herramientas de visualización que ayudan a la toma de decisiones.

DeepSMOTE

En este modelo, los hiperparámetros que se desean ajustar son el tamaño del lote ($batch\_size$), el número de épocas ($epochs$), la dimensión del espacio latente (dim_h), el *learning rate* (lr) y el tamaño de las capas ocultas (n_z). La malla inicial definida para el conjunto de datos MNIST se describe en la Figura 4.5, de igual forma se puede constatar la combinación probada para el cojunto de datos FashionMNIST y CIFAR10 en las figuras 4.6 y 4.7 respectivamente.

```
mallaMNIST = {
    'lr': [0.0002, 0.00005],
    'epochs': [100],
    'batch_size': [64, 128],
    'dim_h': [64, 128],
    'n_z': [300, 600],
}
```

Figura 4.5: Malla conjunto de datos MNIST método DeepSMOTE

```
mallaFashionMNIST = {
    'lr': [0.0002, 0.00005],
    'epochs': [150],
    'batch_size': [32, 64, 128],
    'dim_h': [32, 64, 128],
    'n_z': [300, 600],
}
```

Figura 4.6: Malla conjunto de datos FashionMNIST método DeepSMOTE

```
mallaCIFAR10 = {
    'lr': [0.0002, 0.00005],
    'epochs': [300],
    'batch_size': [32, 64, 128],
    'dim_h': [32, 64, 128],
    'n_z': [600, 1200],
}
```

Figura 4.7: Malla conjunto de datos CIFAR10 método DeepSMOTE

La herramienta *Weights and Biases* registra los resultados de la función de pérdida para cada una de las épocas, proporcionando gráficos de este proceso Figura 4.8, Figura 4.9, Figura 4.10.

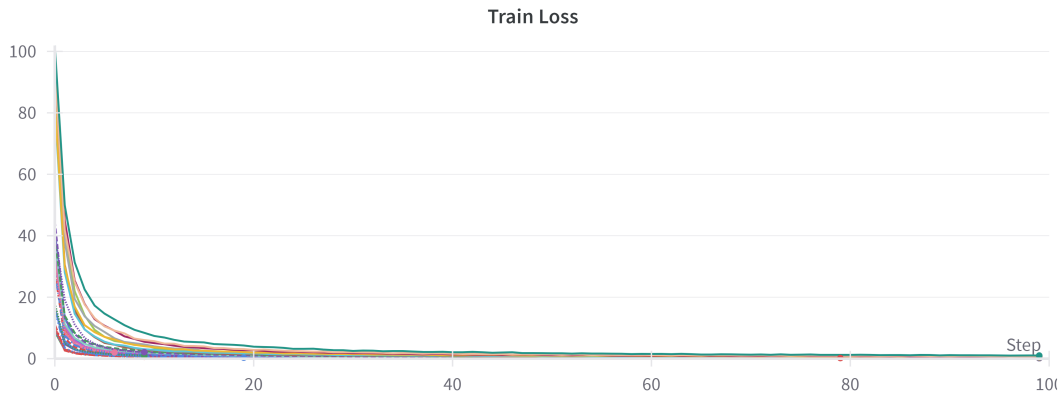


Figura 4.8: Evolución error con distintas combinaciones de hiperparámetros para MNIST usando DeepSMOTE

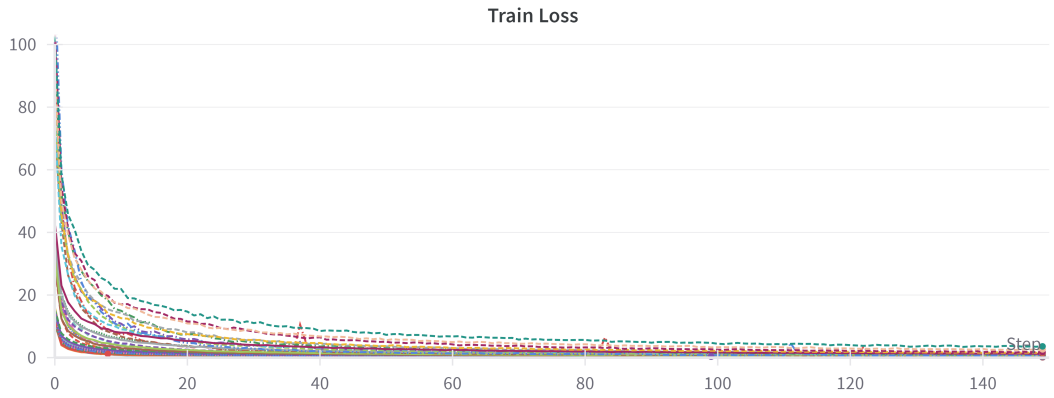


Figura 4.9: Evolución error con distintas combinaciones de hiperparámetros para FashionMNIST usando DeepSMOTE

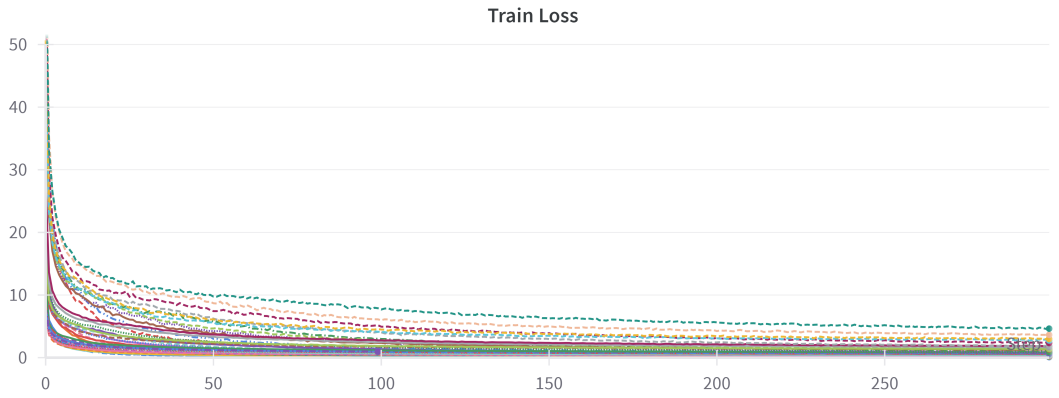


Figura 4.10: Evolución error con distintas combinaciones de hiperparámetros para CIFAR10 usando DeepSMOTE

Tras realizar el proceso de entrenamiento con cada una de las combinaciones, se escoge aquella que arroje menor valor de la función de pérdida. El empleo de la técnica *GreedSearch* ha permitido experimentar una mejora notable en el entrenamiento del modelo, siendo el valor de función de pérdida el parámetro que lo avala como se puede constatar en la Figura 4.3.

Conjunto de datos	**MNIST**	**FashionMNIST**	**CIFAR10**
Loss sin hacer ajuste de hiperparámetros	0.15	0.20	0.94
Loss tras hacer ajuste de hiperparámetros	0.02443	0.04555	0.0873

Tabla 4.3: Cuantificación mejora entrenamiento DeepSMOTE

BAGAN-GP

Para el modelo generativo BAGAN-GP los hiperparámetros que se desea ajustar son el *learning rate* (lr), el tamaño del lote ($batch\_size$), la dimensión del espacio latente ($latent\_dim$) y el número de épocas ($epochs$). Las combinaciones de hiperparámetros probadas para el conjunto de datos MNIST se encuentran definidas en la Figura 4.11, de igual manera se muestra la malla para FashionMNIST (Figura 4.12) y CIFAR10 (Figura 4.13).

```
mallaCIFAR10 = {
    'lr': [0.001, 0.0002, 0.00005],
    'epochs': [25],
    'batch_size': [32, 64, 128],
    'latent_dim': [64, 128, 256],
}
```

Figura 4.11: Malla conjunto de datos MNIST método BAGAN-GP

```
mallaCIFAR10 = {
    'lr': [0.0002, 0.00005],
    'epochs': [20],
    'batch_size': [32, 64],
    'latent_dim': [128, 256],
}
```

Figura 4.12: Malla conjunto de datos FashionMNIST método BAGAN-GP

```
mallaCIFAR10 = {
    'lr': [0.0002, 0.00005],
    'epochs': [20],
    'batch_size': [32, 64],
    'latent_dim': [128, 256],
}
```

Figura 4.13: Malla conjunto de datos CIFAR10 método BAGAN-GP

En los siguientes gráficos Figura 4.14, Figura 4.15, Figura 4.16 se muestra la evolución de la función de pérdida a lo largo de las distintas épocas para cada una de las combinaciones.

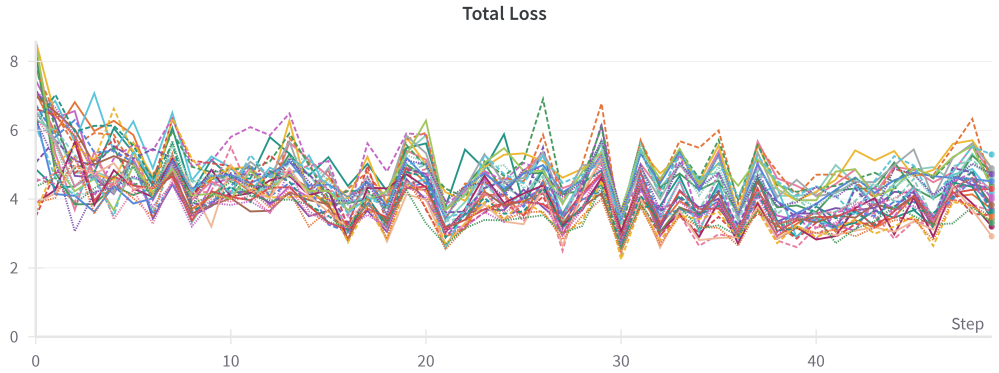


Figura 4.14: Evolución error con distintas combinaciones de hiperparámetros para MNIST usando BAGAN-GP

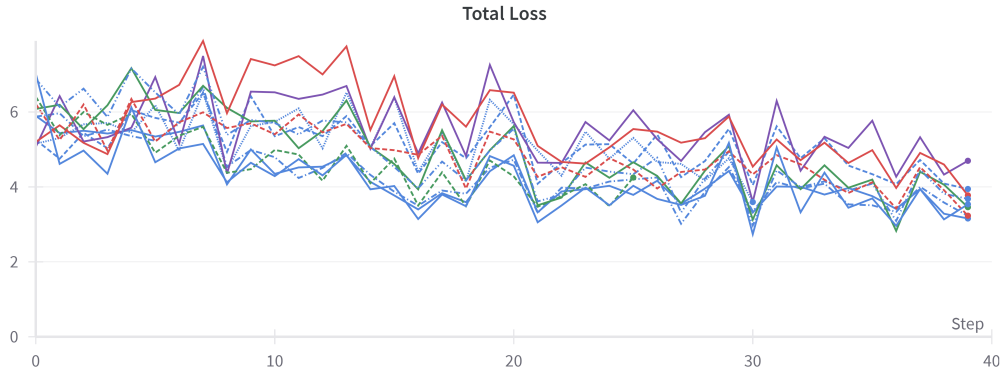


Figura 4.15: Evolución error con distintas combinaciones de hiperparámetros para FashionMNIST usando BAGAN-GP

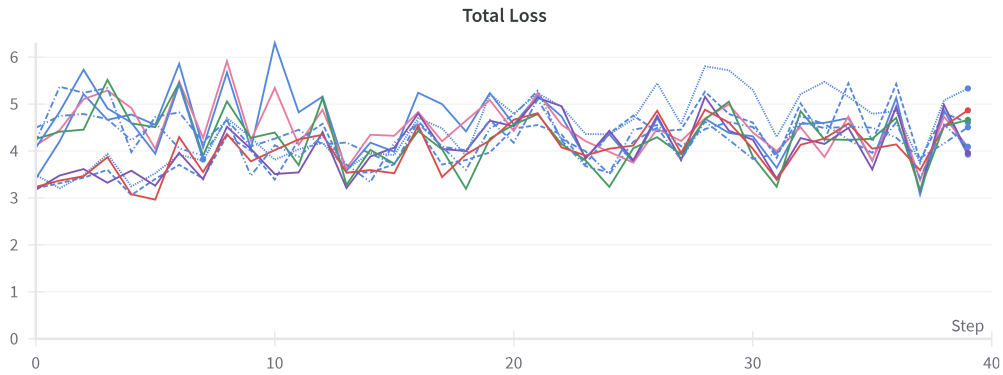


Figura 4.16: Evolución error con distintas combinaciones de hiperparámetros para CIFAR10 usando BAGAN-GP

Se observa un entrenamiento más inestable en comparación con la técnica DeepSMOTE. Pero tras realizar alguna combinación de hiperparámetros más y un análisis de cada uno de los resultados obtenidos se escoge la combinación de hiperparámetros precisa para cada uno de los conjuntos de datos. Este proceso ha conseguido mejorar las métricas de entrenamiento considerablemente como se puede observar en la tabla comparativa Tabla 4.4.

Conjunto de datos	**MNIST**	**FashionMNIST**	**CIFAR10**
Loss sin hacer ajuste de hiperparámetros	3.2641	4.454	3.9265
Loss tras hacer ajuste de hiperparámetros	2.7134	3.8581	3.8653

Tabla 4.4: Cuantificación mejora entrenamiento DeepSMOTE

4.3.2. Entrenamiento y generación de imágenes con DeepSMOTE

Tras el correcto ajuste de hiperparámetros, se realiza para MNIST un entrenamiento con 300 épocas, 64 imágenes en cada lote, 128 capas ocultas, un espacio latente de dimensión 600 y el optimizador Adam cuenta con una tasa de aprendizaje de 0.0002.

Para entrenar FashionMNIST la configuración de hiperparámetros escogida ha sido emplear 300 épocas, 32 imágenes en cada lote, 128 capas ocultas, un espacio latente de dimensión 600 y una tasa de aprendizaje de 0.0002. Para el entrenamiento del conjunto de datos CIFAR10 se ha empleado 300 épocas, 32 imágenes en cada lote, un espacio latente de dimensión 1200, 256 capas ocultas y una tasa de aprendizaje de 0.0002.

Se pueden observar en la Tabla 4.5 los tiempos de entrenamiento empleados con la configuración de hiperparámetros mencionada y en las figuras 4.17, 4.18 y 4.19 la reducción del valor de la función de pérdida a medida que avanza el entrenamiento.

Conjunto de datos	**MNIST**	**FashionMNIST**	**CIFAR10**
Épocas	300	300	300
Tamaño de lote	64	32	32
Tasa Aprendizaje	0.0002	0.0002	0.0002
Dimensión espacio latente	600	600	1200
Tamaño capas ocultas	128	128	256
Tiempo (min)	27	45	116
Loss	0.02443	0.04555	0.0873

Tabla 4.5: Resumen entrenamiento DeepSMOTE

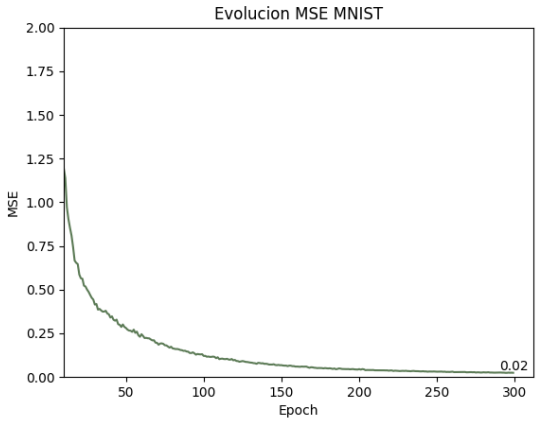


Figura 4.17: Función de pérdida en el entrenamiento de DeepSMOTE con MNIST

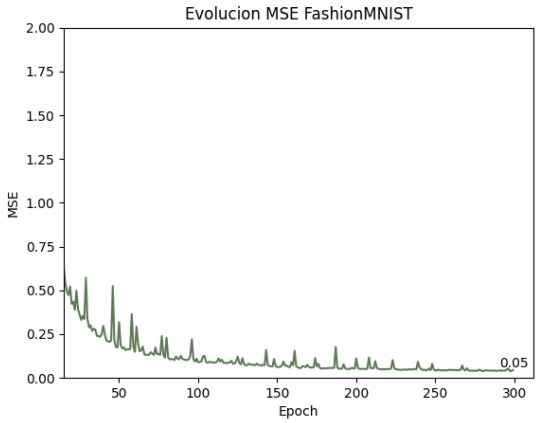


Figura 4.18: Función de pérdida en el entrenamiento de DeepSMOTE con FashionMNIST

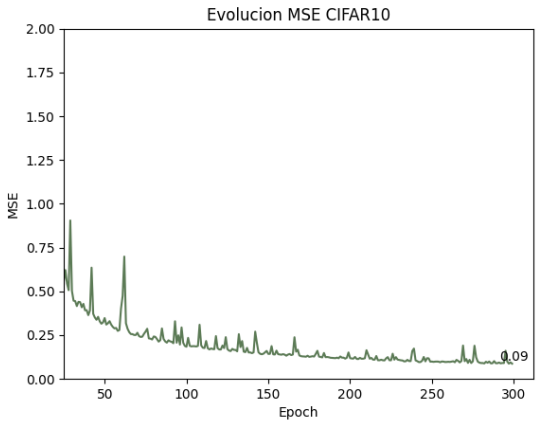


Figura 4.19: Función de pérdida en el entrenamiento de DeepSMOTE con CIFAR10

Se observa que todos los entrenamientos, independientemente del conjunto de datos, experimentan una mayor velocidad de disminución del error en las épocas iniciales. El modelo aprende rápidamente y cada vez genera imágenes más próximas a las existentes en el conjunto de datos original. Alrededor de la época 100 el error se comienza a estabilizar, encontrándose próximo a una etapa de convergencia. Por otro lado, se evidencian oscilaciones propias del optimizador a lo largo del entrenamiento, cada vez de menor longitud a medida que éste avanza. Como consecuencia de todo esto, el error obtenido tras 300 épocas indica que el modelo ha sido capaz de aprender correctamente la distribución de cada una de las clases de forma precisa, siendo capaz de generar instancias de cada una de ellas.

Por otro lado, en la fase de generación de imágenes, el número de imágenes generadas de cada clase depende del mayor número de elementos de una de sus clases. Por lo tanto, se generan tantas imágenes como sean necesarias para igualar esta cifra y balancear el conjunto. De esta forma los conjuntos de datos aumentados MNIST y Fashion MNIST poseen 40000 imágenes (4000 imágenes en cada una de las diez clases) y CIFAR10 posee 45000 imágenes (4500 imágenes en cada clase).

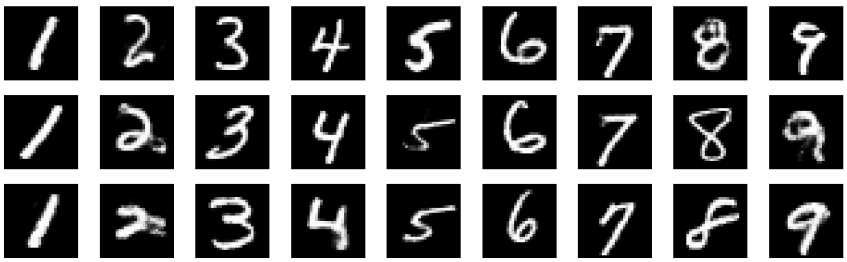


Figura 4.20: Imágenes generadas MNIST

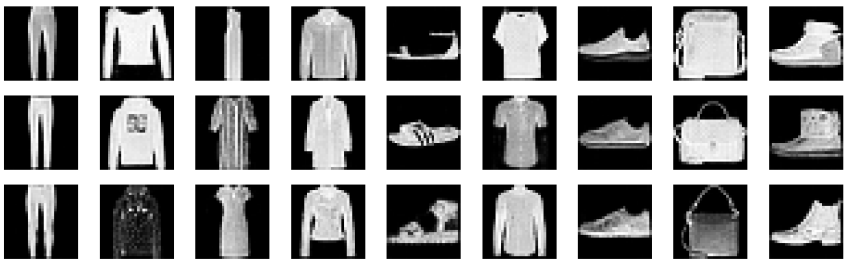


Figura 4.21: Imágenes generadas FashionMNIST

4.3.3. Entrenamiento y generación de imágenes con BAGAN-GP

En la fase de entrenamiento, tanto generador como el decodificador del autoencoder emplean *batch normalization* para normalizar el resultado de una capa antes de pasar a la siguiente. Por otro lado, la función de activación del generador es *tanh*, mientras que el discriminador tiene

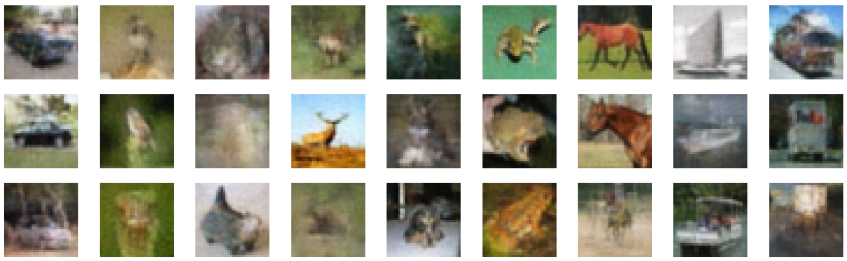


Figura 4.22: Imágenes generadas CIFAR10

una función e activación lineal. Se realizan 20 épocas para el conjunto de datos MNIST, 16 para FashionMNIST y 30 para CIFAR10. Se emplea además un tamaño de lote de 32, el espacio latente posee dimensión 256 para los conjuntos de datos MNIST y CIFAR10 y 128 para FashionMNIST, el optimizador Adam utiliza una tasa de aprendizaje de 0.0002 en todos los entrenamientos.

Conjunto de datos	**MNIST**	**FashionMNIST**	**CIFAR10**
Épocas	20	16	30
Tamaño de lote	32	32	32
Tasa Aprendizaje	0.0002	0.0002	0.0002
Dimensión espacio latente	256	128	256
Tiempo (min)	32	33	36
Loss	2.7134	3.8581	3.8653

Tabla 4.6: Resumen entrenamiento BAGAN-GP

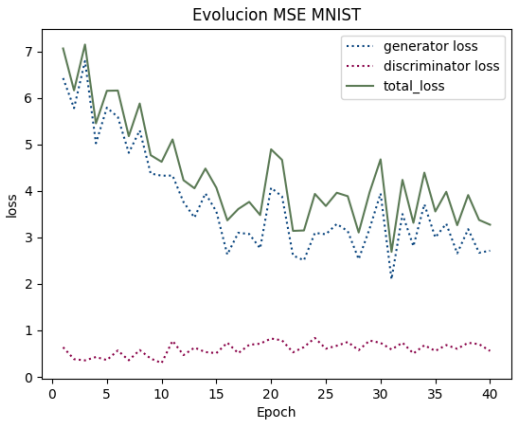


Figura 4.23: Función de pérdida en el entrenamiento de BAGAN-GP con MNIST

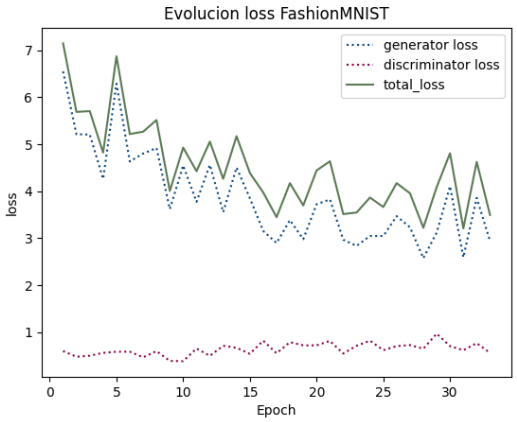


Figura 4.24: Función de pérdida en el entrenamiento de BAGAN-GP con FashionMNIST

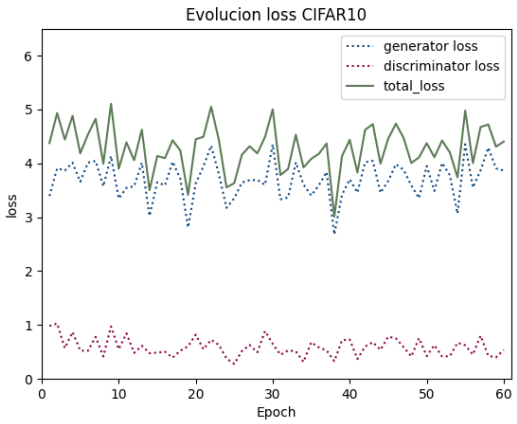


Figura 4.25: Función de pérdida en el entrenamiento de BAGAN-GP con CIFAR10

Se observa en estas figuras que, por lo general, el generador reduce su valor de la función de pérdida a medida que avanza el entrenamiento. El valor de la pérdida del discriminador es menor, manteniéndose casi constante y aumentando ligeramente de valor en las últimas épocas. Esto es debido a que cada vez las imágenes generadas son cada vez más próximas a las existentes en el conjunto de datos original y por tanto el discriminador cada vez posee más dificultar para discernir entre las originales y las generadas. Por otra parte, las oscilaciones que se reflejan en los valores de la función de pérdida tanto para el generador como para el discriminador son frecuentes al realizar un entrenamiento con redes GAN. Finalmente, la conclusión que se puede obtener del entrenamiento de BAGAN-GP es que, observando la tendencia de la función de pérdida para los conjuntos de datos MNIST y FashionMNIST, el modelo mejora a medida que transcurren las épocas. No sucede lo mismo para el caso de CIFAR10 donde se observan valores constantes a lo largo del entrenamiento, teniendo por lo tanto mayor dificultad para llevar a cabo una correcta generación de imágenes.

En la fase de generación de imágenes, esta técnica genera de cada clase las imágenes que sean necesarias para igualar el número de elementos de cada clase. De esta forma las imágenes

generadas junto con las originales, constituyen un conjunto de datos balanceado. Los conjuntos aumentados poseen cada uno 40000 imágenes en el caso de MNIST y FashionMNIST y 45000 en el caso de CIFAR10.

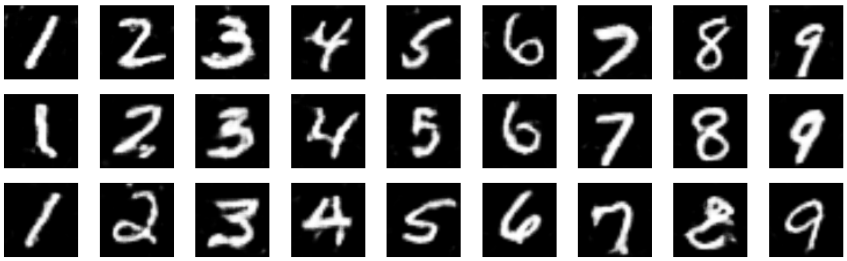


Figura 4.26: Imágenes generadas MNIST

Figura 4.27: Imágenes generadas FashionMNIST

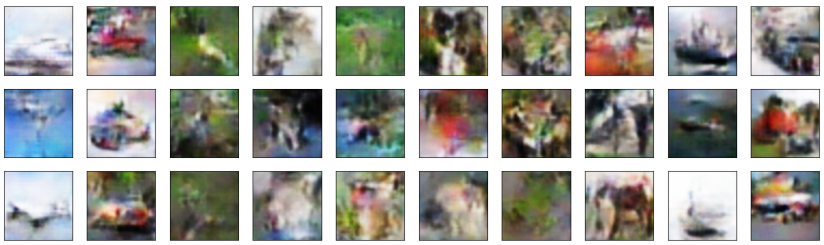


Figura 4.28: Imágenes generadas CIFAR10

Comparando estas imágenes con las generadas por la técnica DeepSMOTE, se observa que BAGAN-GP genera imágenes menos realistas. Sobre todo en el caso del conjunto de datos CIFAR10, donde es difícil conseguir reconocer lo que representa cada imagen. Generar imágenes menos realistas no implica obligatoriamente tener mal desempeño en las labores de clasificación. Puede ocurrir que la extracción de características que realiza la red convolucional sea satisfactoria y por lo tanto consiga buenos valores en las métricas calculadas, consiguiendo dar solución al

problema de clasificación con clases desbalanceadas. En el siguiente capítulo se analiza en detalle la clasificación de los conjuntos balanceados con cada técnica.

Capítulo 5

Evaluación

Para evaluar las capacidades generativas de cada una de las técnicas, el conjunto de datos balanceado y el conjunto de datos no balanceado han sido clasificado con distintas redes convolucionales. Estas redes implementan en la validación un conjunto de métricas para cuantificar el desempeño de DeepSMOTE y BAGAN-GP. Las métricas escogidas han sido la exactitud o *Accuracy* y la métrica *Macro-averaged* F1. Este capítulo muestra una introducción a estas métricas para posteriormente detallar el proceso de evaluación llevado a cabo con las redes convoluciones y mostrar los resultados obtenidos.

5.1. Métricas

Accuracy

Para un problema de clasificación multiclase, representa la tasa de clasificaciones correctas en relación con el total de clasificaciones realizado. Se expresa como

$$Accuracy = \frac{\text{Número de predicciones correctas}}{\text{Número total de predicciones}}$$

Posee el inconveniente de no arrojar suficientes evidencias a la hora de manejar un conjunto de datos desbalanceado puesto que contabiliza los aciertos sin tener en cuenta los elementos mal clasificados. En este caso no supone un problema, puesto que las clases se encuentran balanceadas en el momento de realizar el proceso de clasificación.

Macro-averaged F1

La sensibilidad o *Recall* cuantifica el número de elementos pertenecientes a la clase positiva que fueron clasificados correctamente en relación con el número de elementos de la clase positiva. La situación idónea se presenta cuando el valor es igual a uno.

$$Recall = \frac{\text{VP}}{\text{VP+FN}}$$

La precisión o *Precision* representa el cociente entre los elementos pertenecientes a la clase positiva que fueron clasificados correctamente y el número total de instancias clasificadas como pertenecientes a la clase positiva (tanto verdaderos positivos como falsos positivos).

$$Precision = \frac{\text{VP}}{\text{VP+FP}}$$

La métrica *F1-Score* combina las métricas anteriores, *Precision* y *Recall*:

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

Representa la media armónica entre estos dos valores. Los valores se encuentran comprendidos en el intervalo $[0, 1]$, correspondiéndose un valor de uno con un ajuste de la *Precision* y *Recall* perfecto.

La métrica empleada es la variante denominada *Macro-averaged F1*, consiste en calcular la métrica *F1* para cada una de las clases existentes en el conjunto de datos y posteriormente realizar la media de estos valores.

$$Macro\text{-}averaged\ F1 = \frac{F1_{clase1} + ... + F1_{claseN}}{N}$$

5.2. Proceso de evaluación

Como se mencionó en el capítulo previo, se entrenan dos redes para cada modelo y cada conjunto de datos. La red convolucional "desbalanceada" emplea como conjunto de entrenamiento el conjunto de datos original desbalanceado, mientras que la red convolucional "balanceada" emplea el conjunto de datos aumentado obtenido con las técnicas anteriores. Por su parte, el conjunto de test tiene las mismas tasas de desbalanceado para cada tipo de conjunto de datos, contando con 2069 imágenes en total.

Implementación redes convolucionales

Se han empleado distintas arquitecturas, en concreto *Resnet50*, *Resnet34*, *Resnet18*, *Squeezenet 1.1* y *Efficientnet*, comparando su desempeño con cada conjunto de datos. No ha existido restricciones de computación puesto que nuevamente se ha hecho uso de la GPU A40.

La construcción de estas redes se ha realizado empleando como lenguaje de programación *Python* y como librerías vertebradoras *Pytorch* y *Fastai* [83]. *Fastai* es una librería construida sobre *Pytorch* que proporciona numerosas funciones de alto nivel referentes a la preparación de los conjuntos de datos, entrenamientos y evaluaciones de modelos, reduciendo por tanto la dificultad de manejo para labores de *Machine Learning*.

Los hiperparámetros de las redes convolucionales son semejantes en todas ellas, empleando un tamaño de lote de 8 y la entropía cruzada como función de pérdida. El número de épocas es variable puesto que está implementado *Early Stopping*. Esta técnica detiene el proceso de entrenamiento cuando el conjunto de validación deja de mejorar, de forma que evita el sobreajuste. Por otra parte, el conjunto de datos inicial se divide de forma que el 80 % es empleado para el entrenamiento del modelo y el 20 % restante para validación. El modelo arroja cada una de las métricas (*Accuracy* y *Macro-averaged* F1) calculadas durante el entrenamiento, así como una vez entrenado la evaluación con el conjunto de test.

5.3. Experimentación y resultados

En esta sección se muestran los resultados de las métricas *Accuracy* y *Macro-Averaged F1* calculadas para cada modelo y cada uno de los conjuntos de datos. Se pueden constatar en verde las métricas cuyo entrenamiento se ha realizado con el conjunto de datos balanceado, como resultado de aplicar cada uno de los modelos generativos, y en naranja, el conjunto de datos original desbalanceado. De esta forma se puede evidenciar realmente si los modelos generativos DeepSMOTE y BAGAN-GP son capaces de mejorar las labores de clasificación de imágenes.

Entrenamiento y validación de las redes convolucionales

Los gráficos siguientes hacen referencia a las métricas obtenidas con el conjunto de validación durante el entrenamiento de las distintas redes convolucionales.

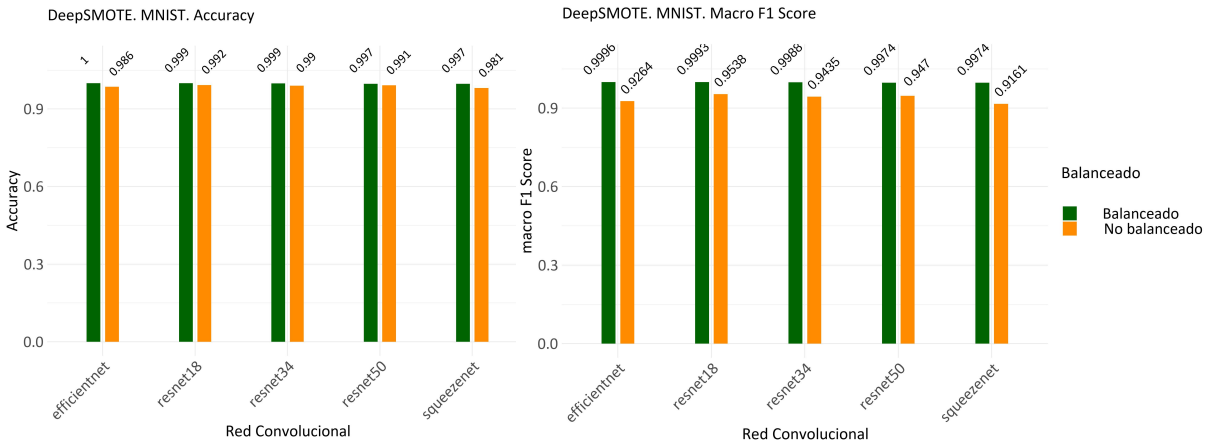


Figura 5.1: Comparación de métricas de validación para distintas redes convolucionales con el *dataset* MNIST.

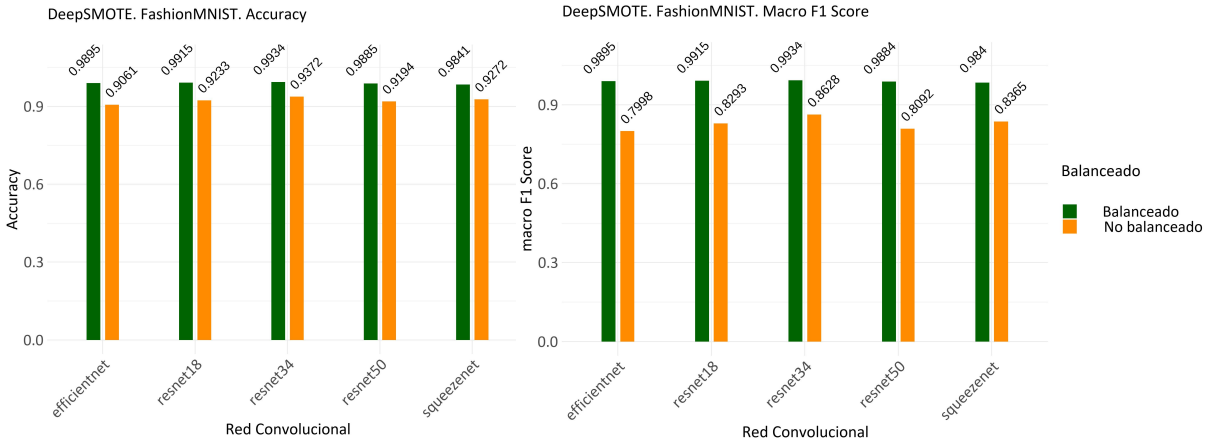


Figura 5.2: Comparación de métricas de validación para distintas redes convolucionales con el *dataset* FashionMNIST.

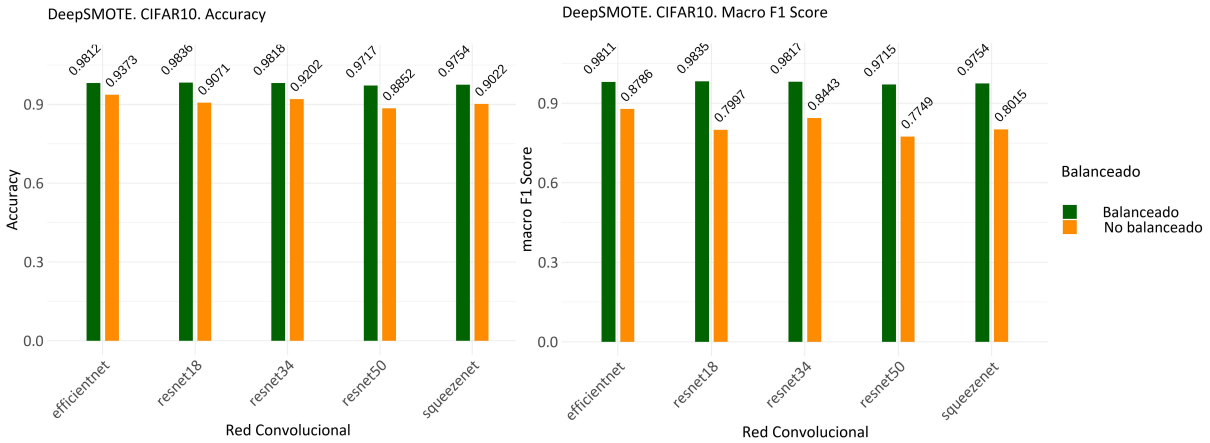


Figura 5.3: Comparación de métricas de validación para distintas redes convolucionales con el *dataset* CIFAR10.

Se observa que la aplicación de la técnica DeepSMOTE consigue una notable mejora durante el entrenamiento. Esto se evidencia en las figuras 5.1, 5.2 y 5.3. Las redes convolucionales para el conjunto de datos MNIST balanceado arrojan una métrica *Macro-Averaged F1* del 99 % en comparación con su versión no balanceada que varía entre 91 % – 95 %. Para el caso del conjunto de datos FashionMNIST el conjunto balanceado al clasificarlo muestra un valor entre 98 % – 99 % para la métrica *Macro-Averaged F1*, mientras que los valores de la misma para el conjunto no balanceado oscilan entre 79 % – 86 %. En el caso del conjunto de datos CIFAR10 la diferencia es más evidente, el conjunto de datos balanceado muestra un valor de *Macro-Averaged F1* entre 97 – 98 % y el no balanceado entre 77 – 87 %.

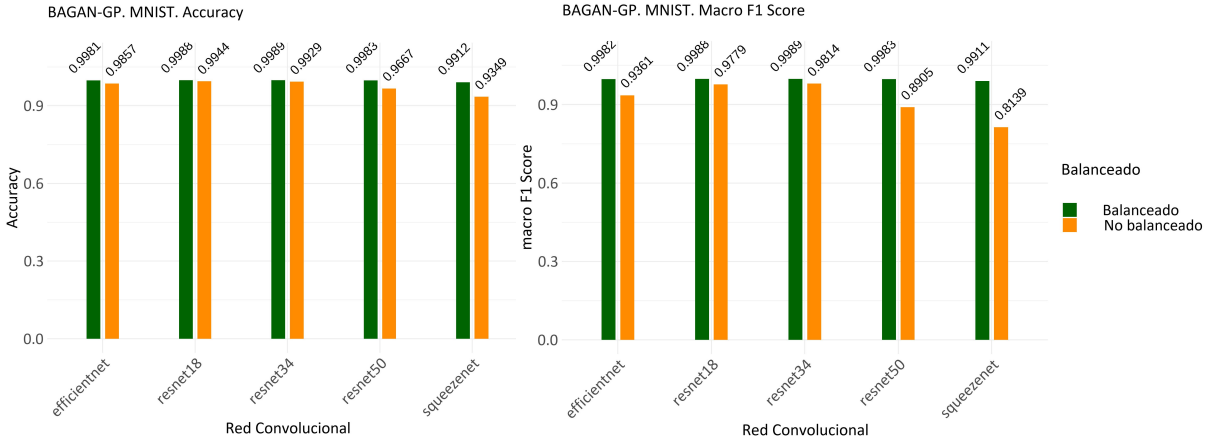


Figura 5.4: Comparación de métricas de validación para distintas redes convolucionales con el *dataset* MNIST.

En las figuras 5.4, 5.5 y 5.6 se puede constatar la mejora que experimentan las métricas de validación al haber entrenado la red empleando el conjunto de datos balanceado (como resultado de aplicar BAGAN-GP) con respecto al no balanceado. Haciendo referencia a la métrica *Macro-Averaged F1* en el caso del conjunto de datos MNIST, con el conjunto de datos balanceado se

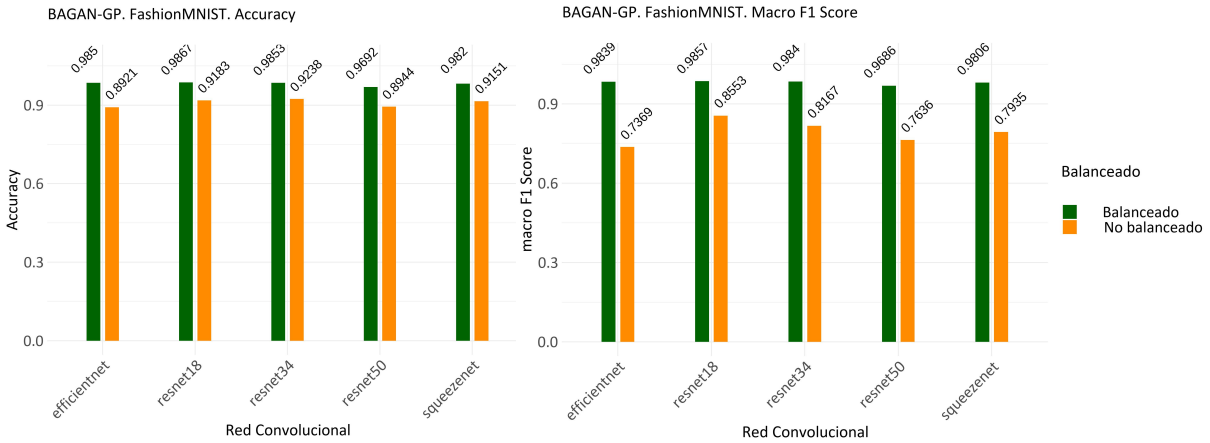


Figura 5.5: Comparación de métricas de validación para distintas redes convolucionales con el *dataset* FashionMNIST.

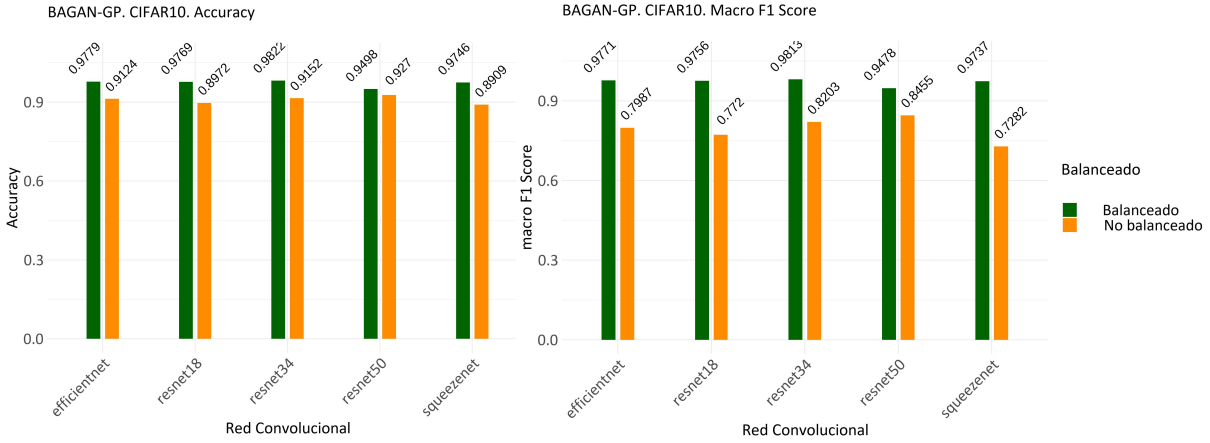


Figura 5.6: Comparación de métricas de validación para distintas redes convolucionales con el *dataset* CIFAR10.

obtienen valores del 99 % mientras que en el caso del no balanceado oscilan entre 81 %−97 %. Para las redes convolucionales con el conjunto de datos FashionMNIST balanceado se obtienen valores entre 96 % − 98 % y en el caso del conjunto de datos no balanceado los valores se encuentran entre 73 % − 85 %. Por último, en el caso del conjunto de datos balanceado CIFAR10 el valor de la métrica *Macro-Averaged F1* varía entre 94 % − 98 %, mientras que para su versión desbalanceada toma el valor 72 % − 84 %.

Evaluación con el conjunto de test

Evaluar el desempeño de cada una de las distintas redes convolucionales con los respectivos conjuntos de test permite obtener una evidencia sólida sobre si el balanceo de datos resultante de aplicar un modelo generativo concreto mejora las labores de clasificación. Al igual que con el conjunto de validación, las métricas calculadas son *Accuracy* y *Macro-Averaged F1*. Esta última métrica posee mayor importancia puesto que es capaz de reflejar la incorrecta clasificación de los

elementos de la clase minoritaria cuando el conjunto de datos se encuentra desbalanceado.

Las figuras 5.7, 5.8 y 5.9 muestran el valor de las métricas de test para el conjunto de test cuando los modelos han sido entrenados con los conjuntos de datos balanceados empleando la técnica DeepSMOTE (en verde) y los conjuntos de datos desbalanceados originales (en naranja). El *accuracy* es del 99 % para la mayoría de casos, tanto para el conjunto balanceado como el no balanceado, alcanzando la red *EfficientNet* la totalidad de clasificaciones correctas con el conjunto balanceado. Atendiendo por otra parte a la métrica *Macro-Averaged F1* se puede constatar una sutil mejora del conjunto de datos balanceado con respecto al desbalanceado, tomando en el primer caso valores entre 96 % − 98 % y en el segundo 95 % − 97 %. La red que mejor combinación de las dos métricas arroja es *ResNet34*.

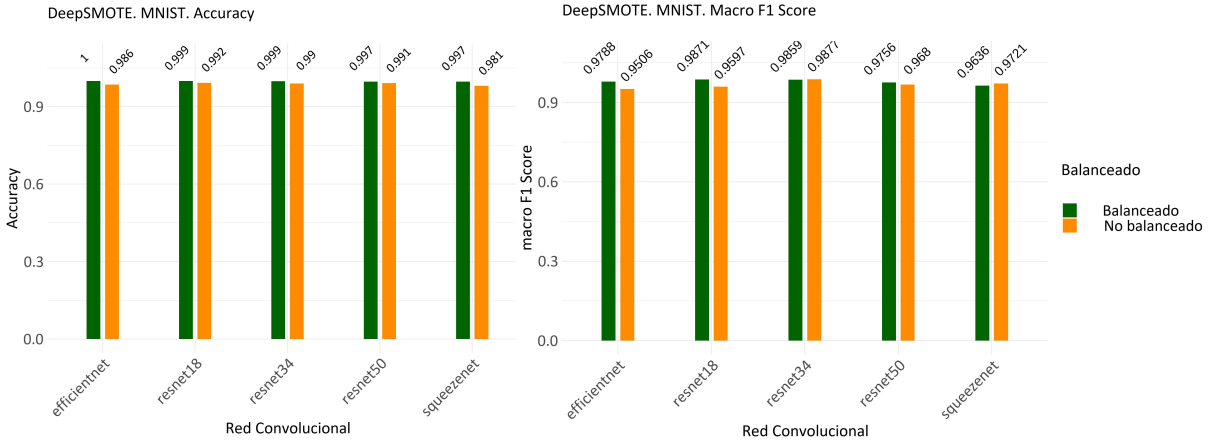


Figura 5.7: Comparación de métricas de test para distintas redes convolucionales con el *dataset* MNIST.

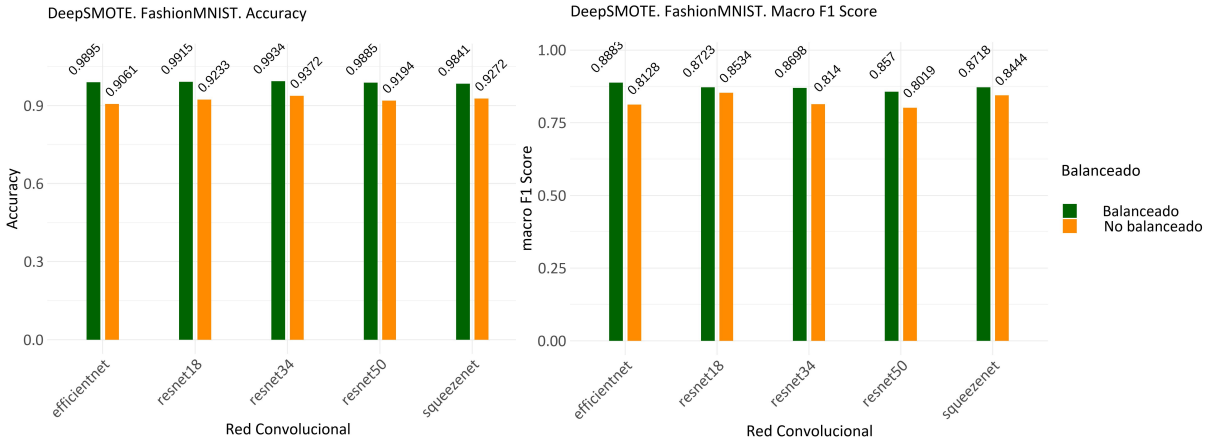


Figura 5.8: Comparación de métricas de test para distintas redes convolucionales con el *dataset* FashionMNIST.

Para el conjunto de datos FashionMNIST (Figura 5.2) se observan diferencias notables en las métricas, constatando una notable mejora en favor del entrenamiento con el conjunto de datos balanceado. El *Accuracy* en el caso del conjunto de datos balanceado oscila entre el 98 % y el

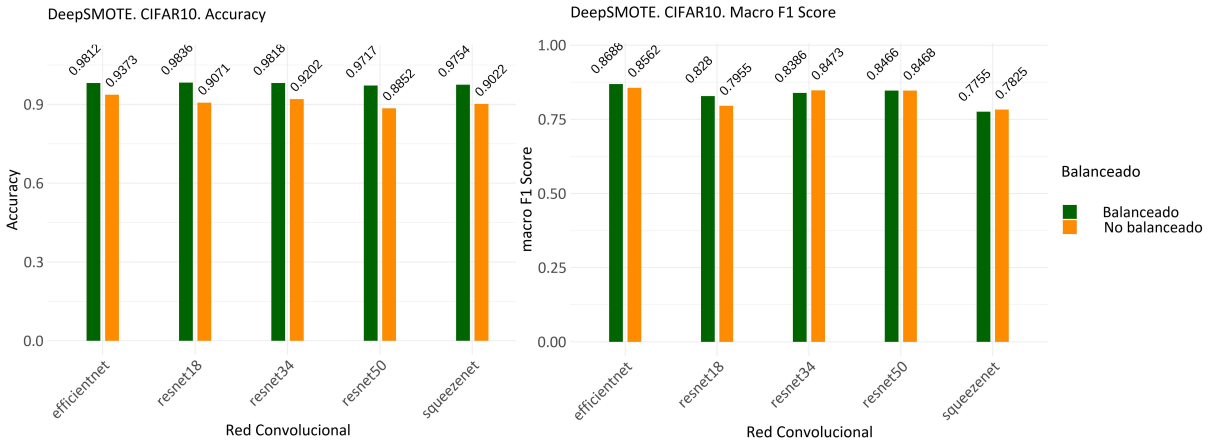


Figura 5.9: Comparación de métricas de test para distintas redes convolucionales con el *dataset* CIFAR10.

99 % mientras que para el conjunto desbalanceado el valor está comprendido entre el 90 % − 93 %. Atendiendo a esta métrica cualquiera de las redes usando el conjunto de datos balanceado arroja resultados totalmente aceptables. Teniendo en consideración la métrica *Macro-Averaged F1* se observa que en el caso en el cual las redes han sido entrenadas con el conjunto de datos balanceado los valores de esta oscilan entre 86 % − 88 %, mientras que si han sido entrenadas con el conjunto desbalanceado se obtiene 80 % − 85 %. Si hubiera que destacar un modelo convolucional que arroje los mejores resultados sería la red *EfficientNet* entrenada con el conjunto de datos balanceado.

En el caso de CIFAR10 (Figura 5.3), la métrica *Accuracy* de los modelos entrenados con el conjunto balanceado presentan una ventaja con respecto a los no balanceados. El valor de esta métrica en el caso de los conjuntos balanceados oscila entre 97 % − 98 % mientras que en el caso de los conjuntos no balanceados 88 % − 92 %. Observando ahora la métrica *Macro-Averaged F1*, los resultados obtenidos para alguna de las redes convolucionales comparando si el conjunto se encuentra o no balanceado son más similares. La red que mejor es capaz de clasificar las imágenes es *EfficientNet* empleando en el entrenamiento el conjunto de datos balanceado y alcanzando un valor de esta última métrica de 86 %.

Por otra parte, las figuras 5.10, 5.11 y 5.12 muestran las métricas resultantes de clasificar el conjunto de test. Se exponen tanto en el caso en el cual las redes convolucionales han sido entrenadas con un conjunto de datos desbalanceado como cuando han sido entrenadas con un conjunto de datos balanceado. Este conjunto balanceado es el resultante de emplear la técnica de balanceo BAGAN-GP.

Empleando el conjunto de datos MNIST (Figura 5.10) se observa una clasificación satisfactoria en cada uno de los casos. Es cierto que no se observa una clara dominancia cuando el entrenamiento ha sido realizado con el conjunto de datos balanceado en contraste con haberlo realizado con el no balanceado. La métrica *Accuracy* para el conjunto de datos balanceado es de 99 % mientras que para el caso no balanceado oscila entre 93 % − 99 %. Atendiendo a la métrica *Macro-Average F1* se obtiene un valor máximo de 98 % tanto con la red *Squeezenet* tomando el conjunto de datos balanceado como con la red *ResNet34* tomando el conjunto de datos desbalanceado.

En el caso de trabajar con el conjunto de datos FashionMNIST (Figura 5.11), se puede

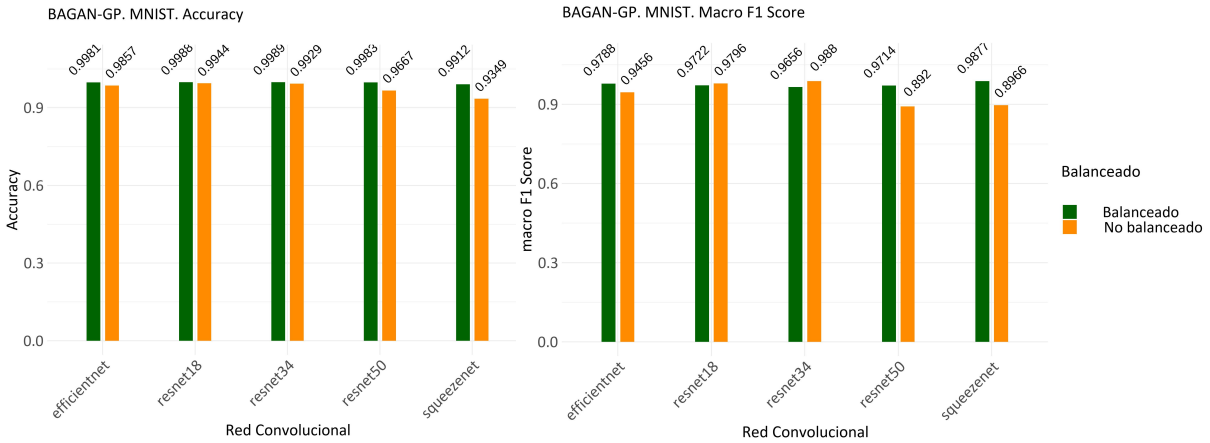


Figura 5.10: Comparación de métricas de test para distintas redes convolucionales con el *dataset* MNIST.

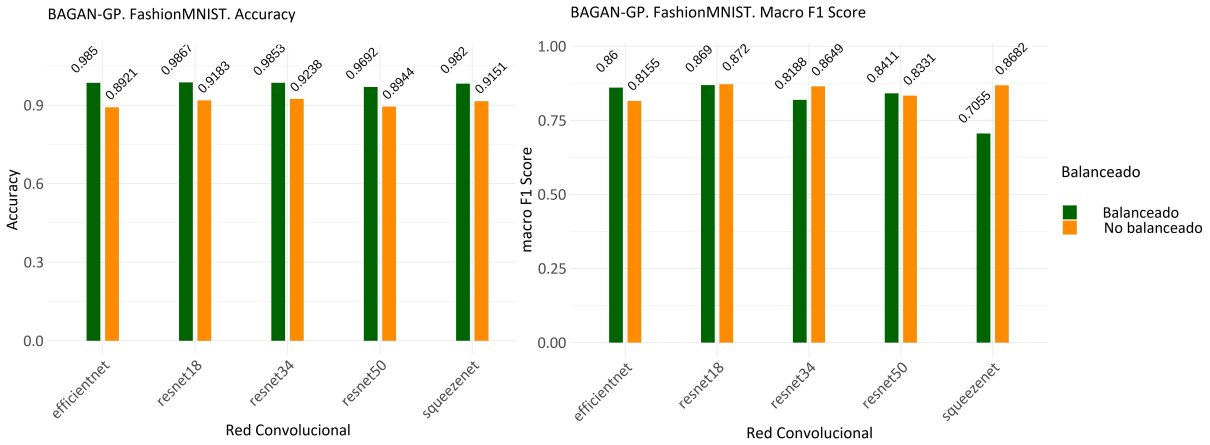


Figura 5.11: Comparación de métricas de test para distintas redes convolucionales con el *dataset* FashionMNIST.

observar nuevamente cómo no se puede obtener una única conclusión a favor o en contra del modelo generativo BAGAN-GP, puesto que ésta depende de la red convolucional con la que se clasifique. La red que proporciona mejores resultados globales es *EfficientNet* cuando ha sido entrenada con el conjunto de datos balanceado. Ésta arroja un 98 % de *Accuracy* y 86 % de *Macro-Average F1*, siendo un correcto desempeño para la labor de clasificación que se desea llevar a cabo.

Por último, la generación de imágenes que realiza BAGAN-GP para el conjunto de datos CIFAR10 (Figura 5.12) no siempre mejora la clasificación, si no que para la mayoría de redes empeora. *EfficientNet* es nuevamente la red que mejor capacidad de clasificar las imágenes tiene, cuando ha sido entrenada con el conjunto de datos balanceado, proporcionando un 98 % de *Accuracy* y un 86 % de *Macro-Averaged F1*.

Tras la comparativa individual de cada método como técnica de balanceo de datos, se realiza un análisis de las métricas obtenidas para conjunto de datos balanceado con cada una de las

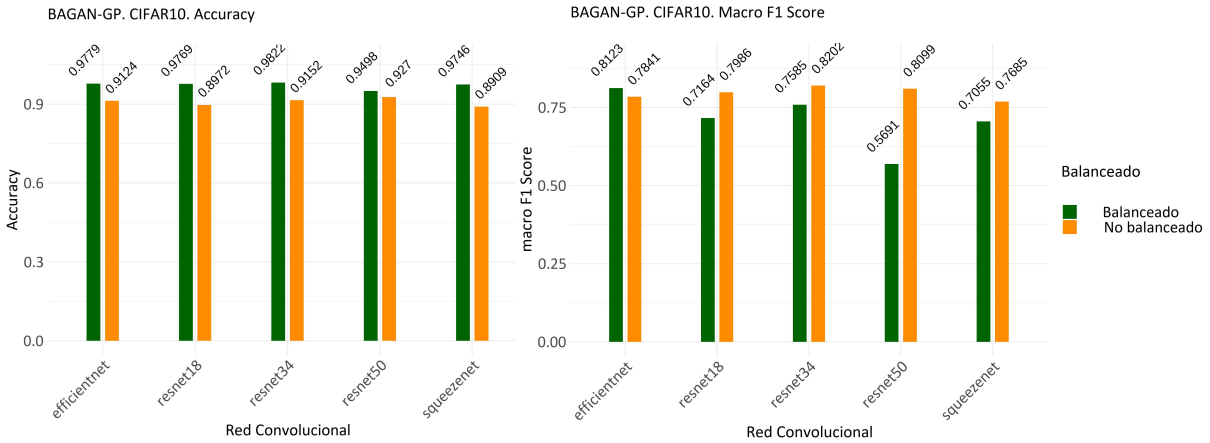


Figura 5.12: Comparación de métricas de test para distintas redes convolucionales con el *dataset* CIFAR10.

técnicas. Para el conjunto de datos MNIST, ambas técnicas presentan un valor similar de *Accuracy*. Atendiendo al valor de la métrica *Macro-Averaged F1* DeepSMOTE presenta resultados un 1 % o 2 % mejor en función de la red con la que se ha clasificado. Para el conjunto de datos FashionMNIST, DeepSMOTE presenta en todos los casos valores más altos que BAGAN-GP para cada una de las dos métricas evaluadas. La diferencia en la *Macro-Averaged F1* oscila entre $1\,\% - 5\,\%$ a excepción de la red *Squeezenet* donde se observa una diferencia entre la clasificación de cada conjunto generado con cada uno de los métodos del 17 %, siempre en favor de DeepSMOTE. Para el caso del dataset CIFAR10 nuevamente arroja mejores resultados DeepSMOTE, evidenciando las diferencias más notables en la métrica *Macro-Averaged F1* $(4\,\% - 28\,\%)$. Es por tanto lícito concluir que para los conjuntos de datos MNIST, FashionMNIST y CIFAR10 el conjunto balanceado generado con DeepSMOTE ofrece mejores resultados que BAGAN-GP.

Tras lo expuesto, se observa cómo el modelo generativo DeepSMOTE permite mejorar notablemente las métricas resultantes en las labores de clasificación de imágenes. Luego es válido emplearlo cuando se presenta un conjunto de datos con clases desbalanceadas, para así conseguir un correcto entrenamiento y posterior clasificación de todas las clases involucradas.

Por otro lado el modelo generativo BAGAN-GP posee luces y sombras. Las métricas más altas tras la clasificación las arrojan modelos entrenados con conjuntos balanceados, pero también es cierto que para otras redes empeora notablemente. Además, como se observaba anteriormente, las métricas que arrojaba el conjunto de validación eran más que aceptables, pero sin embargo no es capaz de generalizar para imágenes reales. La redes han aprendido los patrones de las imágenes generadas. Observando éstas, por ejemplo en el caso del conjunto de datos CIFAR10, el modelo poseía mayor dificultad para reconstruir cada una de las imágenes. Esto no desmerece la labor que es capaz de hacer este modelo de balanceo, pero sí evidencia que la clasificación con conjuntos de datos desbalanceados es un área en constante avance.

Capítulo 6

Conclusiones y trabajo futuro

6.1. Conclusiones

Con una visión crítica se realiza un balance global del TFG. Se valora el cumplimiento o no de los diferentes objetivos, junto con la discusión sobre si la metodología de trabajo escogida ha sido la adecuada. Además se incluye la perspectiva personal acerca de la realización de este proyecto.

6.1.1. Perspectiva del proyecto

Este trabajo abarcaba en un inicio una serie de objetivos, los cuales han sido satisfechos en su totalidad:

- Se ha investigado y adquirido conocimiento sobre las distintas técnicas existentes de balanceo de datos (**OBJ-01**), es decir, tanto las técnicas de muestreo como las basadas en la función de coste o pérdida.

- Se ha incidido en el estudio y manejo de técnicas de balanceo de datos en el ámbito del *Deep Learning* (**OBJ-02**), investigando modelos generativos que emplean redes GAN como BAGAN-GP y otros modelos como DeepSMOTE.

- Se ha trabajado con los modelos generativos BAGAN-GP y DeepSMOTE (**OBJ-03**) y distintos conjuntos de datos MNIST, FashionMNIST y CIFAR10, consiguiendo un conocimiento preciso y exhaustivo. Se ha intentado, a su vez, mejorar el entrenamiento de cada modelo para cada conjunto de datos de forma particular realizando un correcto ajuste de los hiperparámetros.

- Se ha investigado y adquirido conocimiento sobre la lógica y el manejo de las redes neuronales convolucionales (**OBJ-04**) particularizado a la labor de conseguir clasificar los conjuntos de imágenes con los que se ha trabajado.

Este trabajo de investigación ha comprobado la consistencia y desempeño de cada uno de los dos modelos generativos, luego supone una evidencia más para su uso futuro. Ambos constituyen una solución al problema de desbalanceo de clases cuando se trabaja con conjuntos de imágenes.

La metodología ASAP ha resultado completamente adecuada. Seguir una metodología ágil permite mantener un ritmo de realización constante y con continua retroalimentación por parte de los tutores y de la comunidad.

6.1.2. Perspectiva personal

Este TFG me ha permitido ser consciente de la problemática real y muy común del desbalanceo de datos, siendo capaz de dar distintas soluciones para mejorar las labores de clasificación. Además me ha abierto las puertas al ámbito del *Machine Learning* y del *Deep Learning*, puesto que he podido ampliar en gran medida los reducidos conocimientos que poseía al comienzo. Además he podido trabajar, entender y familiarizarme con las librerías *Tensorflow*, *Pytorch* y *Fastapi* de *Python* siendo un excelente punto de partida de cara a seguir trabajando en el ámbito de la ciencia de datos.

Por otra parte, la metodología empleada me ha facilitado el continuo y constante progreso. Subdividir el trabajo en *sprints* de menor longitud favorece a la consecución de cada uno de los objetivos específicos y permite poseer un mejor control sobre el trabajo realizado. Contar, a su vez, con el feedback de la comunidad me ha permitido tanto dar solución a los problemas que me iban surgiendo como plantearme nuevas preguntas a las que ir dando respuesta.

Además este trabajo también me ha permitido adquirir objetivos generales aplicables a futuros proyectos profesionales. He aprendido la forma en la cual se aborda un problema y la búsqueda de soluciones, la exposición oral de resultados, he sido capaz de trabajar con artículos científicos de rigor y enfrentarme a la redacción de una memoria expresando los conceptos de forma concisa y clara.

6.2. Trabajo futuro

De cara seguir investigando en esta línea, el trabajo futuro consistiría en seguir aprendiendo de las estructuras que proporcionan estas técnicas y cómo se combinan para añadir ventajas adicionales. Algunas propuestas a investigar podrían ser:

- Realizar ensayos con otros conjunto de datos distintos y de mayor dimensión para evaluar el rendimiento y limitaciones en caso de existir.
- Realizar ensayos con distintas proporciones de balanceo. Sería interesante valorar si se obtiene una mejor clasificación, si el conjunto de datos se aumenta pero no hasta que todas las clases posean el mismo número de elementos.
- Conseguir una técnica de desbalanceo que sea más ligera, es decir, requiera menor capacidad computacional pero arroje resultados robustos y de calidad. Esto permitiría acercar estas técnicas a ordenadores convencionales los cuales no posean GPUs de gran potencia.
- Seguir aprendiendo sobre funciones de pérdida mejoradas y evaluar cuáles se comportan mejor durante el entrenamiento del modelo generativo.

Parte III

Apéndices

Apéndice A

Manual de Instalación

El contenido se muestra en cuadernos de *Jupyter Notebook* de extensión `.ipynb`. Para poder ejecutarlos hay que realizar unas comprobaciones o intalaciones previas. Se muestra el proceso para ejecutarlo en un sistema operativo Windows:

1. Crear y activar un entorno virtual para gestionar las dependencias de forma independiente.
   a) Instalar `virtualenv` mediante una terminal.
   `pip install virtualenv`
   b) Moverse con el comando `cd` hasta el directorio donde se desee crear el entorno.
   c) Crear un entorno.
   `virtualenv nombre_del_entorno`
   d) Activar el entorno creado.
   `nombre_del_entorno\Scripts\activate`
2. Descargar e instalar la versión 3.10.12 de *Python* desde su página oficial [22].
3. Instalar mediante linea de comandos *Jupyter Notebook*.
   `pip install jupyter`
4. Instalar las librerias en las versiones necesarias.
   a) `torch=2.2.1`
   b) `fastai==2.7.14`
   c) `tensorflow=2.15.0`
   d) `pandas==2.0.3`
   e) `matplotlib==3.7.1`
   f) `wandb=0.16.6`
   g) `sklearn-pandas==2.2.0`
5. Inicializar *Jupyter Notebook*.
   `jupyter notebook`

Al finalizar para desconectar el entorno se emplea el comando `deactivate`.

Apéndice B

Contenido adjunto

El proyecto consta de seis cuadernos de *Jupyter Notebook*:

- `inicializacion.ipynb`: se encarga de crear la estructura de directorios adecuada para albergar los *datasets*, modelos y métricas generados. Además crea los conjuntos de datos desbalanceados necesarios para los posteriores entrenamientos.
- `DeepSMOTE_train.ipynb`: es el entrenamiento del modelo DeepSMOTE para cada uno de los conjuntos de datos establecidos.
- `DeepSMOTE_generate.ipynb`: crea los conjuntos aumentados y balanceados para DeepSMOTE a partir del entrenamiento llevado a cabo.
- `enhanced_bagan_gp.ipynb`: posee tanto el entrenamiento como la generación de imágenes para el modelo BAGAN-GP. Arroja por tanto el conjunto aumentado y balanceado.
- `transformacion_dataset.ipynb`: es un cuaderno auxiliar que transforma la manera de guardar los conjuntos de datos. Cada conjunto de imágenes, que está guardado como un tensor, pasa a guardarse en una estructura de directorios, de forma que se creen carpetas para cada una de las clases y en ellas se guarden las imágenes individuales en formato *.jpeg*.
- `convolucionales.ipynb`: contiene el entrenamiento de las distintas redes convolucionales (*Resnet50*, *Resnet18*, *Resnet34*, *SqueezeNet*, *EfficientNet*) entrenadas para la evaluación de los modelos. Se encarga a su vez de crear otra estructura de directorios para almacenar sus resultados.

El directorio de carpetas creado por el cuaderno `inicializacion.ipynb` es el siguiente:

```
generative_models/
├── models/
│   └── metrics/
└── datasets/
    ├── train_sets/
    ├── test_sets/
    └── augmented_sets/
        └── imgGeneratedBAGAN-GP/
```

En la carpeta `generative_models` se almacenan los entrenamientos de cada uno de los modelos para cada conjunto de datos. Además, en la carpeta `metrics` quedan registradas las métricas de estos entrenamientos. La carpeta `datasets` almacena los conjuntos de entrenamiento desbalanceados, los conjuntos de test y los conjuntos que tras aplicar un modelo han sido balanceados.

Por otra parte, el sistema de directorios creado por `convolucionales.ipynb` es

```
resnet/
├── results_resnet/
│   ├── DeepSMOTE/
│   │   ├── unbalanced/
│   │   └── balanced/
│   └── BAGAN-GP/
│       ├── unbalanced/
│       └── balanced/
├── models_resnet/
│   ├── DeepSMOTE/
│   │   ├── unbalanced/
│   │   └── balanced/
│   └── BAGAN-GP/
│       ├── unbalanced/
│       └── balanced/
└── datasets/
    ├── train_unbalance_sets/
    ├── test_sets/
    └── augmented_sets/
```

Bibliografía

[1] *Accuracy, precision, and recall in multi-class classification*. Evidently AI. URL: https://www.evidentlyai.com/classification-metrics.

[2] Ashraf Ali, Hasanen Abdullah y Mohammad Fadhil. «Voice recognition system using machine learning techniques». En: *Materials Today: Proceedings* (abr. de 2021).

[3] Md Zahangir Alom et al. «The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches». En: *arXiv* (2018).

[4] Martin Arjovsky, Soumith Chintala y Léon Bottou. «Wasserstein GAN». En: *arXiv:1701.07875* (2017).

[5] Shahin Atakishiyev et al. «Explainable Artificial Intelligence for Autonomous Driving: A Comprehensive Overview and Field Guide for Future Research Directions». En: *CoRR* abs/2112.11561 (2021).

[6] Dor Bank, Noam Koenigstein y Raja Giryes. «Autoencoders». En: *arXiv* (mar. de 2021).

[7] H.G. Barrow y J.M. Tenenbaum. «Computational vision». En: *Proceedings of the IEEE* 69.5 (1981), págs. 572-595.

[8] *Bases y tipos de cotización 2024*. Seguridad Social. URL: https://www.seg-social.es/wps/portal/wss/internet/Trabajadores/CotizacionRecaudacionTrabajadores.

[9] Isaac Pérez Borrero y Manuel E. Gegúndez Arias. *Deep Learning. Fundamentos, teoría y aplicación*. http://www.deeplearningbook.org. uhu.es Publicaciones, 2021.

[10] Anibal Bregón Bregón. *Apuntes tema 4 Sistemas Inteligentes*. 2023-2024.

[11] Anibal Bregón Bregón. *Apuntes tema 7 Sistemas Inteligentes*. 2023-2024.

[12] Anibal Bregón Bregón. *Apuntes tema 8 Sistemas Inteligentes*. 2023-2024.

[13] L. Breiman. «Random Forests». En: *Machine Learning* 45 (oct. de 2001), págs. 5-32.

[14] Nitesh V. Chawla, Kevin W. Bowyer y W. Philip Kegelmeyer Lawrence O. Hall. «SMOTE: Synthetic Minority Over-sampling Technique». En: *Journal of Artificial Inteligence Research* 16 (2002), págs. 321-357.

[15] François Chollet. «Xception: Deep Learning with Depthwise Separable Convolutions». En: *arXiv* (2017).

[16] David Colton y Markus Hofmann. «Sampling Techniques to Overcome Class Imbalance in a Cyberbullying Context». En: *Journal of Computer-Assisted Linguistic Research* 3 (jul. de 2019), pág. 21.

[17] Corinna Cortes y Vladimirs Vapnik. «Support-vector networks». En: *Machine Learning* 20 (1995), págs. 273-297.

[18] *Crear Red Neuronal desde las matemáticas*. European Valley. Julio 2020. URL: https://www.europeanvalley.es/noticias/crear-red-neuronal-desde-las-matematicas/.

[19] Damien Dablain, Bartosz Krawczyk y Nitesh V. Chawla. «DeepSMOTE: Fusing Deep Learning and SMOTE for Imbalanced Data». En: *IEEE Transactions on Neural Networks and Learning Systems* 34 (2022), págs. 6390-6404.

[20] *DeepSMOTE*. GitHub. URL: https://github.com/dd1github/DeepSMOTE.

[21] Matias Di Martino et al. «Improving electric fraud detection using class imbalance strategies». En: *Proceedings of the 1st International Conference on Pattern Recognition Applications and Methods* 2 (ene. de 2012), págs. 135-141.

[22] *Download Python*. Python. URL: https://www.python.org/downloads/.

[23] *El dataset MNIST*. InteractiveChaos. URL: https://interactivechaos.com/es/manual/tutorial-de-deep-learning/el-dataset-mnist.

[24] *Enhanced BAGAN-GP*. GitHub. URL: https://github.com/GH920/improved-bagan-gp.

[25] *Fashion MNIST Dataset*. DeepLake. URL: https://datasets.activeloop.ai/docs/ml/datasets/fashion-mnist-dataset/.

[26] K. Fukushima. «Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position». En: *Biological Cybernetics* 36.4 (1980), págs. 193-202.

[27] Nancy Beatriz Ganz, Alicia Esther Ares y Horacio Daniel Kuna. «Predicción de fracasos en implantes dentales mediante la integración de múltiples clasificadores». En: *Universidad Nacional de Misiones. Facultad de Ciencias Exactas, Químicas y Naturales. Centro de Investigación y Desarrollo Tecnológico; Revista de Ciencia y Tecnología* 34 (2020), págs. 13-23.

[28] I. Goodfellow et al. «Generative adversarial nets». En: *Communications of the ACM* 63 (2014), págs. 139 -144.

[29] Ian Goodfellow, Yoshua Bengio y Aaron Courville. *Deep Learning*. MIT Press, 2016.

[30] Robert Mansel Gower et al. «SGD: General Analysis and Improved Rates». En: 97 (2019), págs. 5200-5209.

[31] Leon O. Guertler, Andri Ashfahani y Anh Tuan Luu. «How to train your draGAN: A task oriented solution to imbalanced classification». En: *arXiv:2211.10065* (2022).

[32] Ishaan Gulrajani et al. «Improved Training of Wasserstein GANs». En: *arXiv:1704.00028* (2017).

[33] Peter E. Hart. «The Condensed Nearest Neighbor Rule». En: *IEEE Transactions on information theory* (1968).

[34] Haibo He et al. «ADASYN: Adaptive Synthetic Sampling Approach for Imbalanced Learning». En: *IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)* (2008), págs. 1322-1328.

[35] Kaiming He et al. «Deep Residual Learning for Image Recognition». En: *arXiv:1512.03385v1* (2015).

[36] Kaiming He et al. «Deep Residual Learning for Image Recognition». En: (2015).

[37] Gaofeng Huang y Amir H. Jafari. «Enhanced Balancing GAN: Minority-class Image Generation». En: *arXiv:2011.00189* (2020).

[38] Pariona Huarhuachi y Jefferson Clauss. «Clasificación de fuga de clientes en una entidad financiera utilizando el algoritmo Smote para datos desbalanceados en una regresión logística». En: *Universidad Nacional Agraria La Molina (UNALM)* (2019).

[39] Forrest N. Iandola et al. «SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size». En: *arXiv* (2016).

[40] IBM. *¿Qué es el aprendizaje no supervisado?* URL: https://www.ibm.com/es-es/topics/unsupervised-learning.

[41] IBM. *¿Qué es el aprendizaje supervisado?* URL: https://www.ibm.com/es-es/topics/supervised-learning.

[42] P.M. Institute. *Guide to the Project Management Body of Knowledge (PMBOK Guide). Sixth Edition*. Project Management Institute, 2017. ISBN: 9781628254518.

[43] Xin Jin y Jiawei Han. «K-Means Clustering». En: *Encyclopedia of Machine Learning*. Ed. por Claude Sammut y Geoffrey I. Webb. Springer US, 2010, págs. 563-564. ISBN: 978-0-387-30164-8.

[44] Jinhyon Kim, R. G. Krutchkoff y George R. Terrell. «Iterated grid search algorithm on unimodal criteria». En: (1997).

[45] Diederik P. Kingma y Jimmy Lei Ba. «Adam: A Method for Stochastic Optimization». En: *arXiv:1412.6980* (2014).

[46] Michał Koziarski. «Radial-Based Undersampling for imbalanced data classification». En: *Pattern Recognition* 102 (2020).

[47] J. Laurikkala. «Improving identification of difficult small classes by balancing class distribution». En: *Conference on Artificial Intelligence in Medicine in Eu- rope, Springer* (2001), págs. 63-66.

[48] *Learning Multiple Layers of Features from Tiny Images*. Krizhevsky, Alex. 2009. URL: https://www.cs.toronto.edu/~kriz/cifar.html.

[49] Y. Lecun et al. «Gradient-based learning applied to document recognition». En: *Proceedings of the IEEE* 86.11 (1998), págs. 2278-2324.

[50] Y. Lecun et al. «Gradient-based learning applied to document recognition». En: *Proceedings of the IEEE* 86.11 (1998), págs. 2278-2324.

[51] Yann Lecun, Yoshua Bengio y Geoffrey Hinton. «Deep learning». En: *HAL Open Science* 521.7553 (2023), págs. 436-444.

[52] Andrew L. Maa, Awni Y. Hannun y Andrew Y. Ng. «Rectifier Nonlinearities Improve Neural Network Acoustic Models». En: *Proceedings of the 30th International Conference on Machine Learning* 28 (2013).

[53] Giovanni Mariani et al. «BAGAN: Data Augmentation with Balancing GAN». En: *IBM Research* (2018).

[54] Miguel A. Martínez-Prieto et al. «Una metodología basada en prácticas ágiles para la realización de Trabajos Fin de Grado». En: *Actas de las XXVIII JENUI 8* (2023).

[55] José Manuel Martínez Raya. «Modelos de clasificación para incidencias en entornos industriales con datos no balanceados». En: *Universitat Oberta de Catalunya (UOC)* (2019).

[56] Warren S. McCulloch y Walter Pitts. «A logical calculus of the ideas immanent in nervous activity». En: *Springer* ().

[57] Luis Javier Mena Camaré. «Aprendizaje Automático a partir de Conjuntos de Datos No Balanceados y su Aplicación en el Diagnóstico y Pronóstico Médico». En: *INAOE* (2008).

[58] Diganta Misra. «Mish: A Self Regularized Non-Monotonic Activation Function». En: *arXiv:1908.08681v3* (2020).

[59] Sankha Subhra Mullick, Shounak Datta y Swagatam Das. «Generative Adversarial Minority Oversampling». En: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)* (2019), págs. 1695-1704.

[60] Prakash M Nadkarni, Lucila Ohno-Machado y Wendy W Chapman. «Natural language processing: an introduction». En: *Journal of the American Medical Informatics Association* 18.5 (sep. de 2011), págs. 544-551.

[61] Vinod Naira y Geoffrey E. Hinton. «Rectified Linear Units Improve Restricted Boltzmann Machines». En: *ICML'10: Proceedings of the 27th International Conference on International Conference on Machine Learning* (2010), págs. 807-814.

[62] Augustus Odena, Christopher Olah y Jonathon Shlens. «Conditional Image Synthesis With Auxiliary Classifier GANs». En: *arXiv:1610.09585* (2017).

[63] Alec Radford, Luke Metz y Soumith Chintala. «Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks». En: *arXiv:1511.06434* (2016).

[64] Manav Raj y Robert Seamans. «Primer on artificial intelligence and robotics». En: *Journal of Organization Design* 8.11 (2019).

[65] Prajit Ramachandran, Barret Zoph y Quoc V. Le. «Swish: A self-gated activation function». En: *arXiv:1710.05941v1* (2017).

[66] Laukik Raut, Rajat Wakode y Pravin Talmale. «Overview on Kanban Methodology and its Implementation». En: *International Journal for Scientific Research Development* 03 (jul. de 2015), págs. 2518-2521.

[67] Yehezkel S. Reshe, Amit Mandelbom y Daphna Weinshall. «Controlling Imbalanced Error in Deep Learning with the Log Bilinear Loss». En: *Proceedings of Machine Learning Research* 74 (2017), págs. 141-151.

[68] Douglas Reynolds. «Gaussian Mixture Models». En: *Bulletin of Matematical Biophysics* 5 (1943).

[69] Stuart Russell y Peter Norvig. *Artificial Intelligence. A modern Approach*. Prentice Hall, 1995.

[70] Shai Shalev-Shwartz y Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014.

[71] G. W. Stewart. «On the Early History of the Singular Value Decomposition». En: *SIAM Review* 35.4 (1993), págs. 551-566.

[72] Jingwen Sun, Weixing Du y Niancai Shi. «A Survey of kNN Algorithm». En: *Information Engineering and Applied Computing* 1 (mayo de 2018).

[73] Jaime Durán Suárez. «Trabajo de Fin de Grado: Redes Neuronales Convolucionales en R». En: *Universidad de Sevilla* (2017).

[74] Christian Szegedy et al. «Rethinking the Inception Architecture for Computer Vision». En: *arXiv* (2015).

[75] *The 2020 Scrum Guide*. ScrumGuides. 2020. URL: https://scrumguides.org/scrum-guide.html.

[76] *The CIFAR-10 dataset*. Toronto. URL: https://www.cs.toronto.edu/~kriz/cifar.html.

[77] Ivan Tomek. «An experiment with the edited nearest-neighbor rule». En: *IEEE Transactions on Systems, Man, and Cybernetics* 6 (1976), págs. 448-452.

[78] Jordi de la Torre. «Redes Generativas Adversarias (GAN) Fundamentos Teóricos y Aplicaciones». En: *arXiv. Computer Science. Artificial Intelligence* (2023).

[79] Tsung-Yi et al. «Focal Loss for Dense Object Detection». En: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42 (2018), págs. 318-327.

[80] Shoujin Wang et al. «Training Deep Neural Networks on Imbalanced Data Sets». En: *2016 International Joint Conference on Neural Networks (IJCNN)* (2016), págs. 4368-4374.

[81] Zhou Wang et al. «Image Quality Assessment: From Error Visibility to Structural Similarity». En: *IEEE Transactions on Image Processing* 13 (2004), págs. 600 -612.

[82] Christopher J.C.H. Watkins. «Q-Learning». En: *Machine Learning* 8 (1992), págs. 279-292.

[83] *Welcome to fastai*. Fastai. URL: https://www.fast.ai.

[84] *Why are neuron axons long and spindly? Study shows they're optimizing signaling efficiency*. MedicalXPress. URL: https://medicalxpress.com/news/2018-07-neuron-axons-spindly-theyre-optimizing.html.

[85] Dennis L. Wilson. «Asymptotic Properties of Nearest Neighbor Rules Using Edited Data». En: *IEEE Transactions on systems, man and cybernetics* SMC-2. Issue 3 (1972).

[86] Svante Wold, Kim Esbensen y Paul Geladi. «Principal component analysis». En: *Chemometrics and Intelligent Laboratory Systems* 2.1 (1987), págs. 37-52.

[87] Han Xiao, Kashif Rasul y Roland Vollgraf. «Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms». En: *arXiv* (2017).

[88] Xuebing Yang et al. «AMDO: an Over-Sampling Technique for Multi-Class Imbalanced Problems». En: *IEEE Transactions on knowledge and data engineering* 30 (2017), págs. 1672-1685.

[89] Yu Yu, Weibin Zhang y Yun Deng. «Frechet Inception Distance (FID) for Evaluating GANs». En: *Research Gate* (2021).

[90] Aiming Zhang et al. «EEG data augmentation for emotion recognition with a multiple generator conditional Wasserstein GAN». En: *Complex & Intelligent Systems Springer* (2021).

[91] Zhilu Zhang y Mert R. Sabuncu. «Generalized Cross Entropy Loss for Training Deep Neural Networks with Noisy Labels». En: *32nd Conference on Neural Information Processing Systems* (2018).