Universidad de Valladolid

Escuela de Ingeniería Informática

Trabajo Fin de Grado

Grado en Ingeniería Informática

Mención en Computación

Herramienta basada en técnicas de Deep Learning para el análisis de datos generados por el telescopio ESA Euclid

Autor:

D. David Población Criado

Tutores:

D. Benjamín Sahelices Fernández

D. Fernando Buitrago Alonso

Agradecimientos

Me gustaría dedicar unas palabras de agradecimiento a todas las personas que me han ayudado a lo largo de la realización de este Trabajo Fin de Grado.

En primer lugar, a los profesores del Grado en Ingeniería Informática por haberme proporcionado los conocimientos necesarios para comenzar mi carrera profesional.

A mis tutores, Benjamín y Fernando, por haberme dado la oportunidad de trabajar en este proyecto y dedicarme parte de su tiempo para resolver las dudas que me iban surgiendo.

Al Grupo de Investigación Reconocido "Caracterización de Materiales y Dispositivos Electrónicos" por permitirme instalarme en su laboratorio durante la realización de este trabajo.

A mi familia, en especial a mis padres, por su apoyo, comprensión y paciencia.

Por último, a todas aquellas personas que me han acompañado durante estos últimos años, compañeros y amigos por estar siempre disponibles para lo que fuera necesario, gracias por tanto.

Resumen

El incremento de los volúmenes de datos generados en los últimos años obliga a utilizar nuevos métodos automáticos de minería de datos. La astronomía no es ajena a este aumento, llegando a generar una cantidad de datos nunca vista antes como por ejemplo con los nuevos telescopios que se están desplegando, como el James Webb o Euclid. En este Trabajo Fin de Grado se ha tratado el problema de la detección y caracterización de cuestiones relativas a la morfología de las galaxias, a través de varios conjuntos de datos. Se han utilizado técnicas basadas en *deep learning*, concretamente redes neuronales convolucionales, que han probado ser de gran utilidad en una inmensa variedad de aplicaciones de visión artificial.

El primer conjunto de datos que se ha usado proviene de la colaboración Euclid de la Agencia Espacial Europea (ESA, por sus siglas en inglés). A partir de estos datos se han contestado distintas preguntas relativas a la morfología de las galaxias, por ejemplo si tiene forma de disco, donde se ha conseguido una tasa de acierto y un valor F1 superior al 98 %. Además, se ha analizado el número mínimo de galaxias necesarias para entrenar la red y conseguir el máximo rendimiento, que se encuentra en torno a la mitad del conjunto de datos original con un nivel de confianza del 95 %.

Por último, se ha utilizado un conjunto de datos generado a partir de simulaciones hidrodinámicas para detectar la aparición de estructuras de bajo brillo superficial. En este segundo caso, los resultados expuestos son aún preliminares, consiguiendo en la detección de uno de los tipos de estructuras (*shells*) un RMSE de 0.14.

Abstract

The increase in the volume of data generated in recent years requires the use of new automatic data mining methods. Astronomy is no stranger to this increase, generating an amount of data never seen before, for example with the new telescopes that are being deployed, such as the James Webb or Euclid. In this Bachelor's Thesis we have dealt with the problem of detection and characterization of questions related to the morphology of galaxies, through various data sets. We have used techniques based on deep learning, specifically convolutional neural networks, which have proven to be very useful in a wide variety of computer vision applications.

The first dataset used is from the European Space Agency (ESA) Euclid collaboration. From these data, different questions have been answered regarding the morphology of galaxies, for example whether it is disk-shaped, where an accuracy and F1 score of over 98 % has been achieved. In addition, the minimum number of galaxies needed to train the network and achieve the maximum performance has been analyzed, which is around half of the original data set with a confidence level of 95 %.

Finally, a data set generated from hydrodynamic simulations has been used to detect the appearance of low surface brightness structures. In this second case, the results are still preliminary, achieving an RMSE of 0.14 in the detection of one of the types of structures (*shells*).

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

En los últimos años la astronomía, como muchas otras ciencias, ha experimentado un gran crecimiento en cuanto a la cantidad de datos generados. Este incremento, lejos de pausarse, cada vez está siendo más acentuado con el desarrollo de nuevas tecnologías e instrumentación. Ejemplos de este crecimiento son tanto el reciente telescopio espacial James Webb como, en un futuro cercano, Euclid, que enviarán una cantidad de imágenes del espacio nunca antes vista. Es por ello necesario aplicar métodos automáticos para procesar la ingente cantidad de información obtenida para generar conocimiento de una manera ágil. Esta combinación de conocimientos informáticos y astrofísicos se conoce como "astroinformática".

En el presente Trabajo Fin de Grado (en adelante TFG) se va a investigar sobre técnicas basadas en *deep learning* para la caracterización de galaxias lejanas. La parte más extensa de este proyecto se enmarca dentro de la colaboración Euclid de la Agencia Espacial Europea (ESA), concretamente en el grupo de morfología de galaxias. Para ello se van a utilizar datos del telescopio Hubble que han sido adaptados para simular las lentes de Euclid. De esta forma, aunque no se tengan fotografías tomadas por el nuevo telescopio es posible preparar modelos para que estén listos cuando se empiece a recibir información.

La segunda parte del trabajo muestra unos resultados preliminares sobre el estudio de estructuras de bajo brillo superficial, relacionadas también con la morfología de las galaxias, sobre simulaciones hidrodinámicas.

1.1. Objetivos

El objetivo principal de este TFG es conseguir un modelo de aprendizaje automático para la caracterización de galaxias lejanas por una parte y para la detección de estructuras de bajo brillo superficial por otro. Para conseguir estos objetivos, se han definido los siguientes hitos:

- Investigación y estudio sobre fundamentos del aprendizaje automático y modelos de última generación (*state-of-the-art*).

- Estudio de conceptos básicos de astrofísica necesarios para entender la dimensión del problema, interpretar los datos de partida y los resultados obtenidos.
- Aprendizaje de *frameworks deep learning* y de tratamiento de imágenes astronómicas.
- Detección de ciertas características relativas a la morfología de imágenes astronómicas. Para ello se van a desarrollar múltiples modelos basados en regresión y clasificación.
- Estudio del número de datos necesarios para obtener el máximo rendimiento, en base a ciertas métricas fijadas.
- Detección de estructuras de bajo brillo superficial.

Las cuestiones relativas a los tres últimos hitos son subjetivas: por ejemplo determinar la forma que tiene una galaxia, cuando la imagen de esta no es clara; o detectar si se percibe una barra o no en dicha galaxia. Por tanto, el objetivo es mimetizar el comportamiento humano, tanto los aciertos o los errores que pueda obtener un ser humano al responder las cuestiones que se van a plantear.

1.2. Estructura de la memoria

Este documento sigue la siguiente estructura:

Capítulo 2 - Planificación. Describe la metodología utilizada para desarrollar el proyecto junto al marco temporal. También se realiza una gestión de riesgos y una estimación del presupuesto.

Capítulo 3 - Redes neuronales y *deep learning*. Se explicarán los conocimientos teóricos relativos al aprendizaje automático y redes neuronales, enfatizando las redes y procedimientos que se utilizarán en el trabajo.

Capítulo 4 - Tecnologías utilizadas. Describe las tecnologías empleadas para el desarrollo técnico del proyecto.

Capítulo 5 - Contexto astronómico. Se realizará una contextualización desde el punto de vista astrofísico para comprender el problema que se va a abordar.

Capítulo 6 - Conjunto de datos de Euclid. Apartado en el que se explican las características de los datos de entrada, las operaciones de preprocesado realizadas y las cuestiones que se van a analizar.

Capítulo 7 - Resultados Euclid. Diseño de experimentos para la caracterización de ciertas propiedades de la morfología de galaxias y su evaluación.

Capítulo 8 - Conjunto de datos y resultados generación sintética. Explicación de datos, diseño de experimentos y evaluación de la detección de estructuras de bajo brillo superficial.

Capítulo 9 - Conclusiones y líneas futuras. Para terminar, se realiza una reflexión sobre el trabajo desarrollado y los resultados obtenidos, junto con un análisis de las posibles líneas de trabajo futuras.

Apéndice A - Gráficos de regresión. Incluye gráficos detallados sobre la evolución de la función de pérdida sobre los conjuntos de entrenamiento y validación junto con unas matrices de confusión completas sobre el conjunto de test para la primera ronda de experimentación de los datos de Euclid.

Apéndice B - Gráficos de clasificación. Incluye gráficos detallados sobre la evolución las métricas sobre los conjuntos de entrenamiento y validación para uno de los datos de Euclid. Además, se añaden la evolución de la pérdida sobre entrenamiento/validación y las matrices de confusión sobre test de otro conjunto de Euclid. Ambos corresponden a la segunda fase de experimentación de los datos de Euclid.

Apéndice C - Gráficos de clasificación ponderada Incluye gráficos detallados sobre la evolución de la función de pérdida sobre los conjuntos de entrenamiento y validación junto con unas matrices de confusión completas sobre el conjunto de test para la tercera ronda de experimentación de los datos de Euclid.

Apéndice D - Gráficos de la cuestión 2. Contiene representaciones gráficas de los intervalos de confianza para la segunda cuestión sobre dos de los conjuntos de datos de Euclid.

Capítulo 2

Planificación

2.1. Riesgos

Se ha ido realizando una gestión de riesgos que pueden afectar al trabajo tanto de forma inicial como durante el desarrollo del proyecto. Se considera riesgo tanto a las amenazas, las cuales perjudicarían el correcto desarrollo como a las oportunidades, que ayudarían si se produjeran. Para ello se ha tomado de referencia el Capítulo 11 de [1]. De esta forma, se han considerado las siguientes categorías para agrupar los riesgos (también llamado *Risk Breakdown Structure* (RBS)), mostrado en la Figura 2.1.

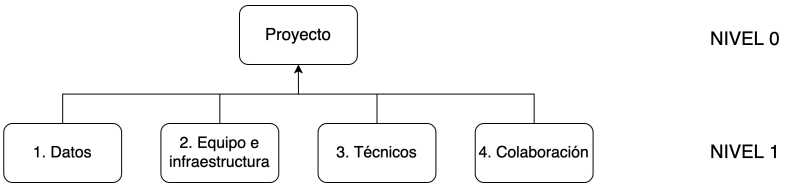


Figura 2.1: *Risk Breakdown Structure* (RBS) de dos niveles

Para cada riesgo, se calcula la probabilidad (P) de que ocurra y el impacto (I) en el proyecto si se da, usando una escala entre 0 y 1. Para que resulte más sencillo asignar dicho valor, se van a usar las siguientes tablas: en vez de asignar una probabilidad o impacto directamente se especifica en términos de 5 niveles diferentes: {muy bajo, bajo, medio, alto, muy alto}. De esta forma ganamos comprensión y legibilidad.

Dados P e I es posible calcular la importancia (Im) de cada riesgo usando la siguiente fórmula:

$$Im = P \cdot I \tag{2.1}$$

Con ello conseguimos tener un orden de los riesgos para administrarlos correctamente, pensando una estrategia de antemano para evitarlos y/o paliar sus efectos en el caso de las amenazas o aprovecharlos en el caso de las oportunidades. Las diferentes estrategias que se han usado vienen enumeradas en el Capítulo 11 de [1].

Probabilidad	Valor
Muy alto	0.90
Alto	0.70
Medio	0.50
Bajo	0.30
Muy bajo	0.10

Tabla 2.1: Valores para las probabilidades

Impacto	Valor
Muy alto	0.80
Alto	0.40
Medio	0.20
Bajo	0.10
Muy bajo	0.05

Tabla 2.2: Valores para el impacto

Tabla 2.3: Correspondencias para probabilidad e impacto. Valores extraídos de [1]

Riesgo 1.1	**Conjunto de datos con poca calidad**
Descripción	Es posible que los datos tengan poca calidad, esto es que tengan un gran número de datos ausentes (incompletos), que sean incorrectos, que estén sesgados, ... Esto provocaría seguramente que la red aprendiera mal o directamente no consiguiera aprender dado que son los datos con los que se entrena
Tipo	Amenaza
Probabilidad	Bajo
Impacto	Muy alto
Importancia	0.24
Estrategia	Escalar. En principio no debería ocurrir dado que provienen de la colaboración Euclid

Tabla 2.4: Riesgo 1.1: "Conjunto de datos con poca calidad"

Riesgo 1.2	**Dificultad para comprender los datos**
Descripción	Como se detallará en el Capítulo 6, el proyecto consta de varias tareas cuyos datos están reflejados en el mismo archivo CSV, por lo que tienen una elevada dimensión. Esto puede provocar una malinterpretación o confusiones a la hora de comprenderlos
Tipo	Amenaza
Probabilidad	Bajo
Impacto	Bajo
Importancia	0.03
Estrategia	Aceptar. Si fuera necesario, se podría consultar a miembros de la colaboración sobre la interpretación de los datos

Tabla 2.5: Riesgo 1.2: "Dificultad para comprender los datos"

Riesgo 1.3	**Tratamiento inicial de los datos**
Descripción	Los datos se obtienen en un formato desconocido en un primer momento (FITS), que puede provocar retrasos al tratarlos por no ser familiares
Tipo	Amenaza
Probabilidad	Bajo
Impacto	Bajo
Importancia	0.03
Estrategia	Aceptar. Si se produjera, es posible acceder a código fuente en internet al ser ampliamente utilizado en astronomía

Tabla 2.6: Riesgo 1.3: "Tratamiento inicial de los datos"

Riesgo 2.1	**Falta de comprensión de los conceptos astronómicos**
Descripción	Al necesitar cierto conocimiento básico de astrofísica, es posible que la comprensión de los conceptos se demore y afecte a la velocidad con la que se desarrolla el proyecto
Tipo	Amenaza
Probabilidad	Medio
Impacto	Bajo
Importancia	0.05
Estrategia	Mitigar. Al tener como tutor a Fernando Buitrago (astrofísico) es posible acceder a él para preguntarle las cuestiones que puedan surgir

Tabla 2.7: Riesgo 2.1: "Falta de comprensión de los conceptos astronómicos"

Riesgo 2.2	**Enfermedad**
Descripción	Es posible que el proyecto se retrase si se contrae alguna enfermedad, más aún en la etapa en la que se realizó de pandemia provocada por el virus SARS-CoV-2. En principio esta enfermedad no debería afectar demasiado, tres días como mucho considerando la duración en pacientes jóvenes
Tipo	Amenaza
Probabilidad	Alto
Impacto	Bajo
Importancia	0.07
Estrategia	Aceptar. Seguir las recomendaciones de las autoridades sanitarias

Tabla 2.8: Riesgo 2.2: "Enfermedad"

Riesgo 2.3	**Imposibilidad de usar los recursos hardware**
Descripción	Las máquinas en las que se realiza la computación son compartidas por varios usuarios, teniendo capacidades limitadas. Por ello, es posible que en algún momento no haya posibilidad de cómputo
Tipo	Amenaza
Probabilidad	Medio
Impacto	Alto
Importancia	0.2
Estrategia	Aceptar. Se coordina con los demás usuarios el uso de las máquinas para evitar confrontaciones

Tabla 2.9: Riesgo 2.3: "Imposibilidad de usar los recursos hardware"

Riesgo 3.1	**Pérdida del código fuente**
Descripción	La programación se ha realizado en un único dispositivo por una sola persona, por lo que cabe el riesgo de perder partes del código escrito por fallos en el ordenador o errores humanos
Tipo	Amenaza
Probabilidad	Muy bajo
Impacto	Muy alto
Importancia	0.08
Estrategia	Mitigar. Se utiliza un sistema de control de versiones especializado en aprendizaje automático: Comet.ml. De esta forma si se produce un fallo es posible recuperar el código fuente

Tabla 2.10: Riesgo 3.1: "Pérdida del código fuente"

Riesgo 3.2	**Pérdida de los resultados de los experimentos**
Descripción	De igual forma que con el código fuente, es posible que se pierdan los resultados de los experimentos al ser almacenados solamente en un dispositivo. En este caso es un error incluso más probable dado que muchas veces se automatiza la creación y eliminación de archivos, un fallo en la programación puede provocar perder horas de experimentación
Tipo	Amenaza
Probabilidad	Bajo
Impacto	Alto
Importancia	0.12

Estrategia	Mitigar. La plataforma Comet.ml permite guardar toda la información relativa a los experimentos, de tal forma que es posible reproducirlos en las mismas condiciones en las que se hicieron

Tabla 2.11: Riesgo 3.2: "Pérdida de los resultados de los experimentos"

Riesgo 3.3	**Falta de conocimientos del framework**
Descripción	El proyecto utiliza PyTorch y fast.ai, los cuales son entornos en los que nunca se han trabajado y se tiene que aprender desde cero. El tiempo requerido para entender ambos *frameworks* puede ser superior al estimado
Tipo	Amenaza
Probabilidad	Bajo
Impacto	Medio
Importancia	0.06
Estrategia	Mitigar. Se aprenderá el funcionamiento de las librerías antes de comenzar con el trabajo

Tabla 2.12: Riesgo 3.3: "Falta de conocimientos del framework"

Riesgo 3.4	**Limitaciones del framework**
Descripción	Principalmente se va a usar fast.ai, el cual es un framework de medio y alto nivel. Por ello, es posible que no ofrezca la flexibilidad querida en ciertos momentos, teniendo que cambiar parte del bajo nivel o directamente utilizar PyTorch
Tipo	Amenaza
Probabilidad	Medio
Impacto	Medio
Importancia	0.1
Estrategia	Mitigar. Se dedicará parte del aprendizaje anterior al proyecto para entender las cuestiones de más bajo nivel de fast.ai así como PyTorch

Tabla 2.13: Riesgo 3.4: "Limitaciones del framework"

Riesgo 3.5	**Modelos que ajustan y generalizan correctamente**
Descripción	Si los modelos se comportan bien en primer lugar es posible que etapas de la experimentación se vean reducidas
Tipo	Oportunidad
Probabilidad	Medio
Impacto	Alto

Importancia	0.2
Estrategia	Aprovechar. Es posible que la experimentación se reduzca si se produce este suceso

Tabla 2.14: Riesgo 3.5: "Modelos que ajustan y generalizan correctamente"

Riesgo 4.1	**Barrera lingüística**
Descripción	La colaboración involucra personas de distintos países y lenguas, por lo que en las reuniones puede haber algún problema de comunicación
Tipo	Amenaza
Probabilidad	Alta
Impacto	Bajo
Importancia	0.07
Estrategia	Evitar. Se usará el inglés para la comunicación entre las partes, conocido por todos los miembros

Tabla 2.15: Riesgo 4.1: "Barrera lingüística"

Riesgo 4.2	**Cambio de objetivos del proyecto**
Descripción	Los objetivos del trabajo pueden sufrir modificaciones por la colaboración, como el posible cambio de rumbo hacia otros paradigmas, por ejemplo
Tipo	Amenaza
Probabilidad	Medio
Impacto	Alta
Importancia	0.2
Estrategia	Aceptar. Aunque el proyecto cambiara de objetivos es posible seguir desarrollando modelos y obtener resultados que puedan aportar a este u otros trabajos futuros

Tabla 2.16: Riesgo 4.2: "Cambio de objetivos del proyecto"

Riesgo 4.3	**Falta de comunicación con las partes**
Descripción	Es posible que durante el desarrollo no se consiga mantener un diálogo abierto con otros miembros de la colaboración o con los propios tutores
Tipo	Amenaza
Probabilidad	Bajo
Impacto	Alto
Importancia	0.12

Estrategia	Evitar. Existe un espacio en Slack para los miembros de la colaboración que forman parte del grupo de trabajo de morfología de galaxias, por lo que es posible comunicarse de forma abierta con otros miembros. Además, al estar físicamente en la Escuela, es posible la comunicación directa con los tutores de este trabajo

Tabla 2.17: Riesgo 4.3: "Falta de comunicación con las partes"

En la Figura 2.2 se ha representado las importancias de los riesgos según los valores de la probabilidad y del impacto.

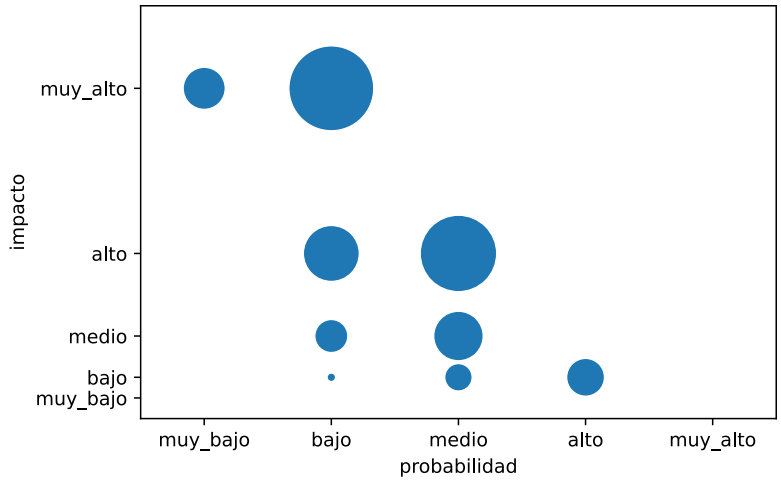


Figura 2.2: Gráfico de burbujas de los riesgos. El tamaño de la burbuja indica la importancia del riesgo

2.2. Planificación

El primer paso es realizar la descomposición de las actividades necesarias para llevar a cabo el proyecto. Esta descomposición se muestra en la Figura 2.3, donde las flechas indican actividades que dependen de otras y no se pueden realizar hasta que no haya terminado la anterior.

En la Figura 2.4 se muestra la distribución temporal de cada una de las actividades realizadas, las cuales se han ido reformulando a lo largo del proyecto: al realizar la experimentación, según los resultados de la fase anterior se decide la siguiente fase. Por ejemplo la necesidad de añadir pesos (clasificación ponderada) aparece al interpretar los resultados de las anteriores fases.

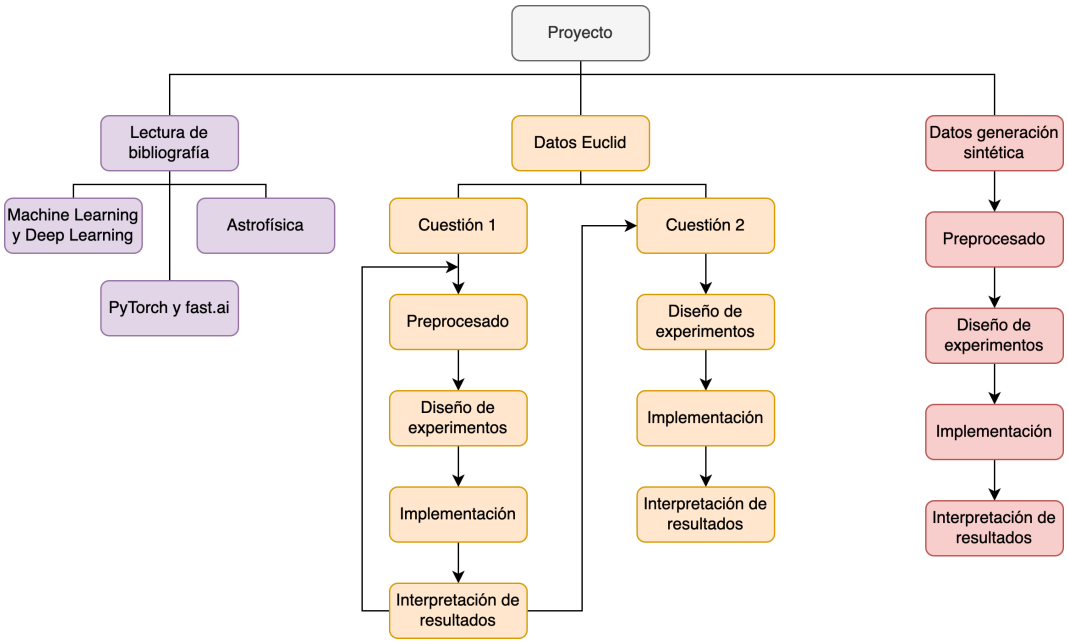


Figura 2.3: Descomposición del proyecto en actividades

2.3. Metodología

Se ha elegido una metodología ágil para desarrollar el proyecto dado el marco temporal limitado y la necesidad de obtener prototipos rápidamente. Esta necesidad surge principalmente de adaptarnos a la evolución de otros modelos por parte de miembros de la colaboración Euclid.

Los prototipos se han realizado de forma incremental, esto es, se parte de un modelo base de *deep learning* y se va mejorando paulatinamente, obteniendo de esta forma prototipos más sencillos en las primeras fases los cuales van evolucionando con el tiempo. Incluso en las primeras fases se consigue un modelo aunque básico, funcional. Las iteraciones se pueden resumir en el bucle de la Figura 2.5.

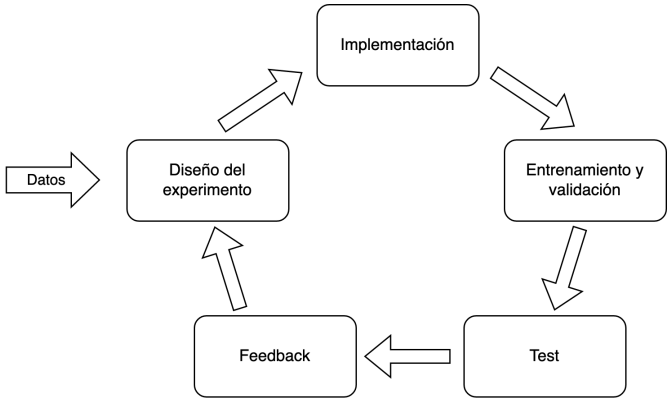


Figura 2.5: Metodología incremental

En la etapa inicial, marcada como “datos”, están incluidos el preprocesado de los datos y su

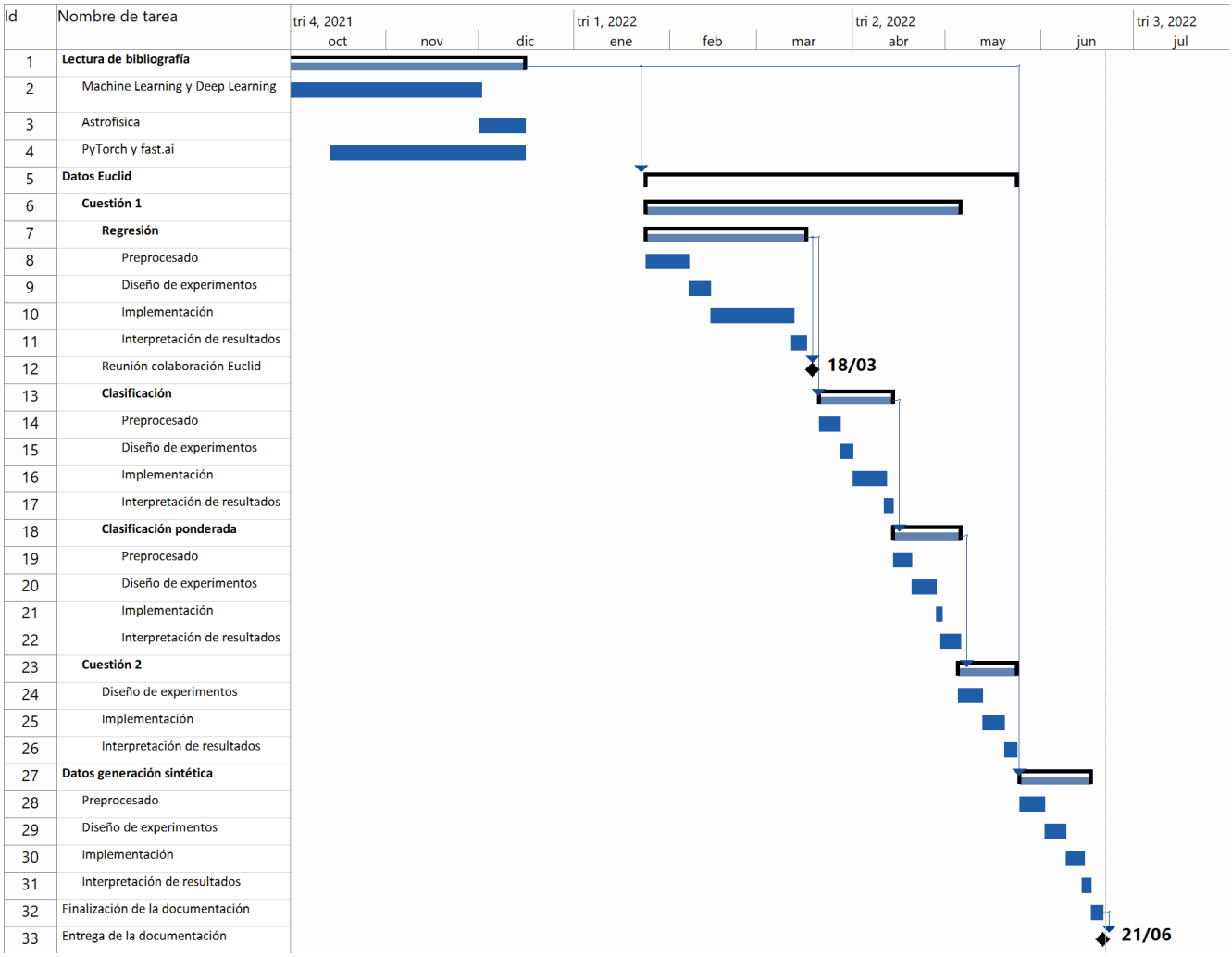


Figura 2.4: Diagrama de Gantt con la distribución de las tareas

preparación previa al diseño de experimentos. Cada iteración consiste básicamente en la prueba de una batería de experimentos. En cada una de las iteraciones importa más la mejora conseguida con respecto a la iteración anterior que en términos absolutos, dado que dependiendo de la naturaleza del problema es posible que no sea fácilmente atajable computacionalmente.

Por otra parte, las dos cuestiones que se van a tratar (Q1 y Q2, ver Capítulo 6) siguen una metodología en cascada: hasta que no se termine la primera no se comienza la segunda ya que son incompatibles en el tiempo, al necesitar información de la primera. Igual ocurre con los dos conjuntos de datos probados, Euclid y generación sintética.

2.4. Presupuesto

El presupuesto que se va a desarrollar en esta sección es una estimación del coste que tendría la realización del proyecto en el mercado laboral (no académico). El tiempo considerado en este TFG es inferior a lo que costaría poner el sistema en producción, ya que no se han contado las etapas

de *testing* ni de despliegue. Por tanto, se estima que un proyecto completo de estas características en un ámbito empresarial tardaría unos diez meses a tiempo completo.

Con respecto al equipo de desarrollo, solamente se considera el sueldo de un investigador contratado. Al requerir conocimientos de metodología de inteligencia artificial, modelos de redes neuronales y ciertos conceptos de astrofísica, el sueldo mensual que se va a considerar es de 2000€ netos mensuales, a los que habría que sumar las cuotas de la Seguridad Social y el Impuesto sobre la Renta de las Personas Físicas (IRPF).

Los recursos utilizados para el desarrollo del proyecto incluyen un equipo personal y un servidor, que es el que realiza la mayor parte del cómputo. El ordenador personal es un portátil MacBook Air de 13 pulgadas (M1, 2020) valorado en 1399€. El servidor utilizado para hacer los cálculos es una CPU Intel Core i5-11600K @ 3.90, con 64GB de memoria RAM y dos GPU NVIDIA GeForce RTX 3060 12GB GDDR6, lo cual supone un valor aproximado de 2900€.

Según la Agencia Tributaria los "equipos para tratamiento de la información y sistemas y programas informáticos" tienen un coeficiente de amortización lineal máximo del 26 % anual [2]. Como la previsión del trabajo es de unos diez meses, la amortización en este período es de 303.12€ y 628.33€, respectivamente.

La mayor parte del *software* empleado es libre y gratuito, excepto Microsoft Project, utilizado para la planificación y seguimiento; y Comet.ml, como sistema de control de versiones. En este TFG no se ha necesitado la compra de una licencia ya que en ambos se dispone de licencias de estudiante. No obstante, las licencias de pago de estos productos son de 25.30€/mes (*Project Plan 3*) y de 170.55€/mes (*Teams*) respectivamente.

Si aplicamos un margen de contingencia del 10 %, el presupuesto por mes quedaría en:

Salario neto	2000
Hardware	93.145
Software	195.85
Subtotal	1157.08
Margen de contingencia (10 %)	228.899
Total	**2517.894**

Tabla 2.18: Presupuesto simulado para el proyecto

Como la duración total estimada es de diez meses, el presupuesto total queda en $2517.894 \cdot 10 =$ **25178.94 €**.

Capítulo 3

Redes neuronales y deep learning

El aprendizaje automático es la rama de las ciencias de la computación encargada de estudiar diferentes algoritmos que hagan que una máquina aprenda. En términos formales, se dice que un sistema (o agente) computacional aprende si después de una experiencia su rendimiento mejora para una serie de tareas (un problema determinado) [3].

Según los datos disponibles (experiencia) para que un agente aprenda sobre un problema concreto, podemos hacer la siguiente clasificación de los métodos de aprendizaje [4]:

- **Aprendizaje supervisado**: dadas unas etiquetas de los datos, el agente es capaz de crear una correspondencia (función) entre los datos de entrada (variables explicativas) y la etiqueta de cada variable (valor a predecir). Es posible que en el conjunto de datos no tengamos todos los datos etiquetados y queramos hacer inferencias sobre todos los datos, muchos de ellos sin etiquetar. O podemos encontrarnos la situación de que las etiquetas de los datos sean incorrectas (o subjetivas), por lo que estaríamos ante un **aprendizaje semi-supervisado**. Ejemplos de algoritmos supervisados son *Support Vector Machines* (SVM), árboles de decisión, Naive Bayes o redes neuronales convolucionales.
- **Aprendizaje no supervisado**: el agente aprende patrones sin tener que proporcionar ningún tipo de etiqueta sobre las variables explicativas. La forma más común es el clustering, el cual consiste en ir agrupando los datos dependiendo de una medida de distancia o similaridad, por ejemplo. Algoritmos comunes de este tipo de aprendizaje son k-medias, DBSCAN o mapas autorganizados (SOM).
- **Aprendizaje por refuerzo**: el agente aprende según recompensas o castigos: va tomando decisiones tratando de maximizar una función de recompensa, la cual debe ser definida cuidadosamente. Los algoritmos genéticos son un ejemplo de este paradigma.

La característica más interesante de que un sistema aprenda es que tenga capacidad de generalizar, es decir, conseguir que aporte un conocimiento a partir de datos con los que nunca ha tenido relación. Por ejemplo, en un problema de clasificación supervisado, conseguir que clasifique correctamente instancias nuevas usando el conocimiento de experiencias anteriores.

3.1. Conceptos de las redes neuronales

En primer lugar, se va a hacer una breve introducción a las redes neuronales, hasta llegar al tipo de redes que se usarán en el proyecto, las redes neuronales convolucionales. Estos conceptos han sido extraídos y adaptados de [5] y [6].

3.1.1. Perceptrón simple

Dado que el aprendizaje siempre se ha relacionado con los seres vivos, especialmente con los animales (y humanos), se ha intentado mimetizar ese comportamiento para que las máquinas aprendan del mismo modo. Por ello, en 1943 McCulloch y Pitts diseñaron un modelo matemático de neurona [7], mejorado y desarrollado más profundamente por otros autores en las décadas de 1950 y 1960 y conocido como perceptrón.

Un perceptrón es la unidad básica de cómputo que toma una serie de entradas, $\{x_1, x_2, ...\}$ y produce una única salida binaria ($y = \{0, 1\}$).

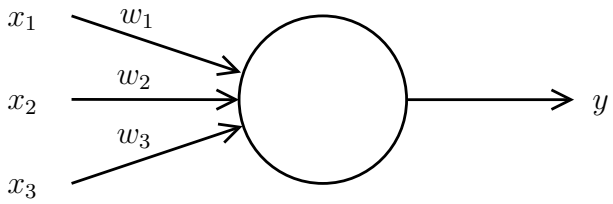


Figura 3.1: Perceptrón simple

Para generar dicha salida, se introducen una serie de pesos para cada una de las entradas, $\{w_1, w_2, ...\}$, realizando una suma ponderada de las entradas. Si dicha suma es mayor que un límite marcado, la neurona se activa (en caso contrario no lo hace):

$$y = \begin{cases} 0 & \text{si } \sum_j w_j x_j \leq \text{umbral} \\ 1 & \text{si } \sum_j w_j x_j > \text{umbral} \end{cases}$$

Para obtener dicho límite, se suele usar una función matemática llamada de activación (F), que es la que "decide" si cada neurona está o no activada, dado que los valores de esta suma son reales y necesitamos un método para que sean binarios. Además, se suelen preferir funciones que sean diferenciables para facilitar los cálculos, y acotadas, aunque no son condiciones estrictamente necesarias, pero sí en la mayor parte de los casos recomendables. Las más utilizadas son la sigmoide, la tangente hiperbólica y el rectificador lineal (ReLU).

Además de los pesos vistos, se suele añadir un término llamado *bias*: es una constante que se añade a mayores en la suma para poder desplazar la función de activación. Para ilustrarlo, consideremos una neurona con una sola entrada, por lo que la salida será $y = w_1 x_1$, similar a un modelo lineal que siempre pasa por el origen de coordenadas. Añadiendo el *bias*, tenemos que

$y = w_1 x_1 + b$, lo que nos da más flexibilidad al modelo, al no tener que pasar por el origen (llamado comúnmente *intercept* en una regresión lineal).

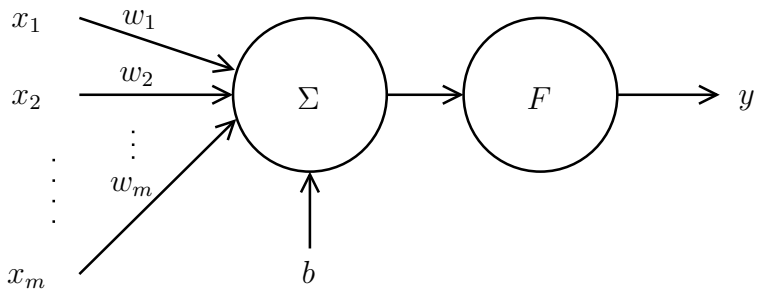


Figura 3.2: Perceptrón simple detallado

Por tanto, la salida de cada neurona queda definida por la siguiente función matemática:

$$y = F(\sum_{j=1}^{m} w_j \cdot x_j + b) \tag{3.1}$$

3.1.2. Función de activación

Como se ha introducido anteriormente, es posible usar muchas funciones de activación, que son las encargada de definir la salida de la neurona transformando la suma ponderada de los pesos multiplicados por las variables de entrada. Generalmente, se buscan funciones diferenciables dado que es necesario para diversos algoritmos de retroprapagación del error como el descenso del gradiente (ver Sección 3.1.6).

Sigmoide

La sigmoide (Figura 3.3) es una función de activación no lineal cuya salida se encuentra acotada en el rango $(0, 1)$. Es especialmente usada cuando queremos calcular las probabilidades de la salida.

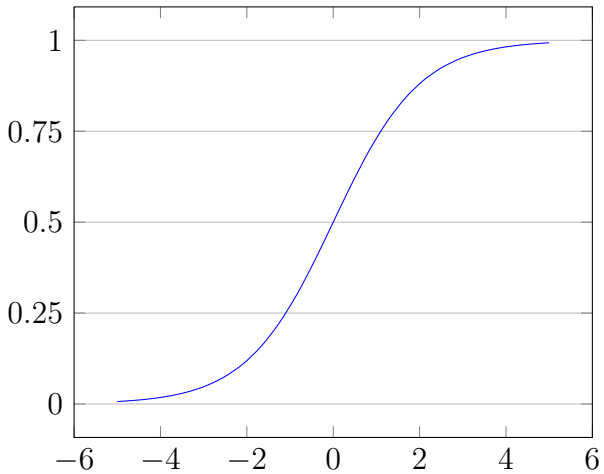


Figura 3.3: Sigmoide: $f(z) = \frac{1}{1+e^{-z}}$

Además, es una función monótona y diferenciable, cuya derivada (Ecuación 3.2) se puede calcular de forma muy sencilla con la propia función, lo cual es muy útil dado que el coste computacional de calcular la derivada es muy bajo.

$$f'(x) = f(x)(1 - f(x)) \tag{3.2}$$

Sin embargo, tiene varias desventajas: problema de desvanecimiento del gradiente (ver Sección 3.1.7) y que la salida no está centrada en 0 (lo está en 0.5).

Tangente hiperbólica

La diferencia principal de la tangente hiperbólica (Figura 3.4) con la sigmoide es que el rango de salida es $(-1, 1)$. Aunque no es directamente interpretable como probabilidad, esto permite que los valores negativos tengan salidas próximas a -1 (también negativo) y los cercanos a cero queden como tal. Esto es especialmente útil para problemas de clasificación binaria.

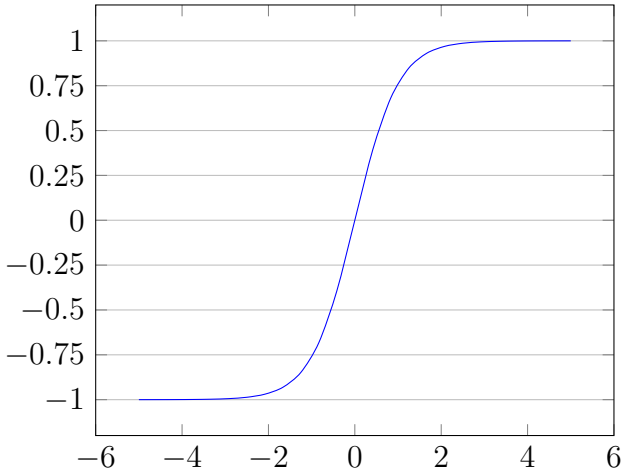


Figura 3.4: Tangente hiperbólica: $f(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$

Igual que la anterior, es monótona y diferenciable, cuya derivada es también fácil de calcular a partir de la función original (Ecuación 3.3) pero mantiene el problema del desvanecimiento del gradiente.

$$f'(x) = 1 - f(x)^2 \tag{3.3}$$

Rectificador lineal

El rectificador lineal (ReLU) (Figura 3.5) es la función de activación más usada en este momento, sobretodo en modelos convolucionales o de *deep learning*. En este caso es una función no acotada cuyo rango es $[0, \infty)$. Tanto la propia función como su derivada (Ecuación 3.4) son monótonas.

$$f'(x) = \begin{cases} 0 & \text{si } x < 0 \\ 1 & \text{si } x \geq 0 \end{cases} \tag{3.4}$$

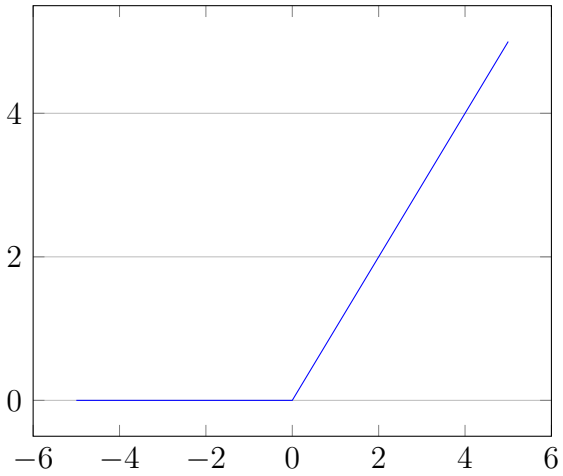


Figura 3.5: Rectificador lineal: $f(z) = max(0, z)$

El problema principal de esta función es que todos los valores negativos se corresponden directamente con 0, lo que perjudica la capacidad del modelo de aprender de los datos. En la literatura se han propuesto diferentes soluciones como puede ser la *Leaky ReLU*, entre otras.

3.1.3. Perceptrón multicapa

Visto el perceptrón simple, que es la unidad básica de cómputo en las redes neuronales (de igual manera que en los humanos), el siguiente paso es tener una serie de perceptrones que trabajen conjuntamente para tener una capacidad de aprendizaje mayor. La agrupación de las neuronas se realiza por capas de diferentes niveles, cada una de ellas tiene una serie de perceptrones que reciben la entrada de la capa inmediatamente anterior y comunican su salida a la siguiente capa.

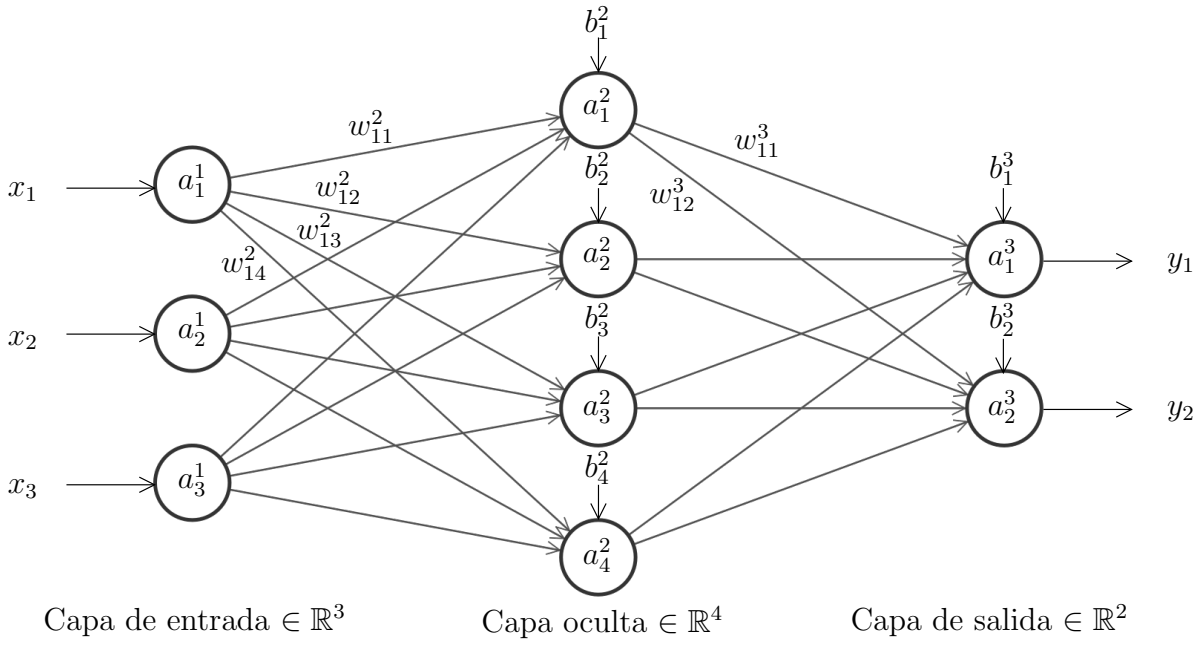


Figura 3.6: Ejemplo de perceptrón multicapa

Como se ve en la Figura 3.6, se pueden distinguir 3 tipos de capas: la capa de entrada, cuyo número de neuronas coincide con el número de variables explicativas del problema que queremos tratar (cada neurona solamente tiene una entrada) y no realizan ningún tipo de procesamiento, únicamente reciben las señales; las capas ocultas, que pueden ser cero, una o más, son las que realizan el procesamiento de los patrones recibidos y por último la capa de salida, que actúa solo como salida de la red.

3.1.4. Método de entrenamiento

Sean (x_{ij}, y_i), $i = 1, .., n$, $j = 1, ..., m$ los datos de entrada, tales que x_{ij} son el conjunto de variables explicativas donde i representa la instancia y j cada una de las características de la observación. Al ser aprendizaje supervisado, y_i es la etiqueta o valor a predecir del individuo i-ésimo.

El entrenamiento de una red neuronal multicapa sigue principalmente los siguientes pasos:

1. **Propagación de la entrada**: se alimenta la red con las variables explicativas (por la capa de entrada) y se van calculando las activaciones de las siguientes capas, donde la salida de la capa anterior sirve como entrada de la siguiente.

2. **Cálculo de la función de pérdida**: después de que la fase anterior termine por completo obtenemos en la capa de salida unos valores finales, a los que denotamos como $\hat{y}_i$. Dado que tenemos los valores reales de cada instancia, es posible calcular un coste o pérdida según una función concreta, obteniendo una medida del error E.

3. **Propagación hacia atrás del error**: dado E es posible ir obteniendo errores para cada una de las capas y así actualizar los pesos, lo que hace que la red aprenda realmente.

Estos pasos no se realizan a la vez con todo el conjunto de entrada, sino que se divide en particiones de igual tamaño, llamado tamaño del *batch* (*batch size*). Esto permite realizar la etapa de aprendizaje más de una vez por conjunto de datos (una por cada partición) dado que si no los pesos solo se actualizarían una vez y daría unos resultados muy pobres. Además, para actualizar los pesos más veces no se entrena solamente una vez con todo el conjunto de datos, sino que se hace repetidas veces. Cada vez que la red ha podido aprender con todo el conjunto de datos se llama época.

Por tanto, el número total de iteraciones del bucle de entrenamiento viene dado por:

$$n_{\text{iter}} = \frac{n}{\text{batch size}} \cdot n_{\text{épocas}} \tag{3.5}$$

Tanto el *batch size* como el número de épocas son hiperparámetros del modelo, valores de las configuraciones para controlar el proceso de aprendizaje.

3.1.5. Propagación de la entrada

El primer paso es proveer de datos a la red mediante la capa de entrada e ir propagando la salida de cada capa k a la inmediatamente posterior $k+1$ (recordar esquema de la Figura 3.6). La notación que se sigue es la siguiente:

- Activación de la neurona i en la capa k: a_i^k
- Pesos para la salida desde la neurona i de la capa $k-1$ hasta la j de la capa k: w_{ij}^k
- Bias de la neurona i de la capa k: b_i^k

Las activaciones de las neuronas de entrada quedan definidas por:

$$a_i^k = x_i$$

Las de la(s) capa(s) oculta(s):

$$a_j^k = f(\sum_{i=1}^{n_{k-1}} w_{ij}^k a_i^{k-1} + b_j^k) \ , \quad \text{donde } n_l \text{ es el número de neuronas en la capa } l \tag{3.6}$$

La activación de la capa de salida se calcula de igual forma que las ocultas y es la salida de la red $y_j = a_j^K$, donde K es la última capa. Después de calcular la salida, hay que actualizar los pesos dado que es la forma de aprender de la red. Para ello se usa el algoritmo de retropropagación o propagación hacia atrás.

En la primera iteración, los pesos w deben de estar inicializados para obtener alguna salida. Hay dos formas generalmente de realizar este inicio:

- Inicialización aleatoria siguiendo una distribución de probabilidad como puede ser $Uniforme$, $Normal$, ...
- Inicialización a partir de una red preentrenada con otros datos anteriormente, usando el estado final de los pesos. Esta técnica se conoce como *transfer learning*.

3.1.6. Algoritmo de propagación hacia atrás

El algoritmo de aprendizaje de las redes neuronales, mediante el cual se van actualizado los pesos de cada una de las capas de la red, se llama propagación hacia atrás (también conocido por su nombre en inglés *backpropagation*). Es un algoritmo supervisado, la idea fundamental es tener una forma de calcular el error de la salida e ir propagando hacia atrás dicho error. Por tanto, es un paso que se hace después de haber conseguido las salidas de la red.

Sea E una medida del error, en la mayoría de los problemas definimos dicha función como:

$$E = \frac{1}{n} \sum_{l=1}^{n} e(l) \tag{3.7}$$

donde n es el número de muestras de entrada y $e(l)$ es el error cometido por cada una de ellas según una función de error en la capa de salida. Si tenemos n_K neuronas en la capa de salida:

$$e(l) = \frac{1}{n_K} \sum_{i=1}^{n_K} e_i(l) \tag{3.8}$$

Las funciones de error más comunes son, dependiendo al problema al que nos enfrentemos (cómo sea la variable respuesta)

- Regresión:
  - Error cuadrático medio (MSE): $e(l) = (y(l) - \hat{y}(l))^2$
  - Error medio absoluto (MAE): $e(l) = |y(l) - \hat{y}(l)|$
- Clasificación: Entropía cruzada (*Cross entropy*): $e(l) = -(y(l) \cdot \log \hat{y}(l) + (1 - y(l)) \log(1 - \hat{y}(l))$

El objetivo es minimizar E, por lo que se convierte en un problema de optimización, que se puede abordar con varios métodos; los más conocidos son el descenso del gradiente, el descenso estocástico del gradiente, RMSProp y Adam.

Descenso del gradiente

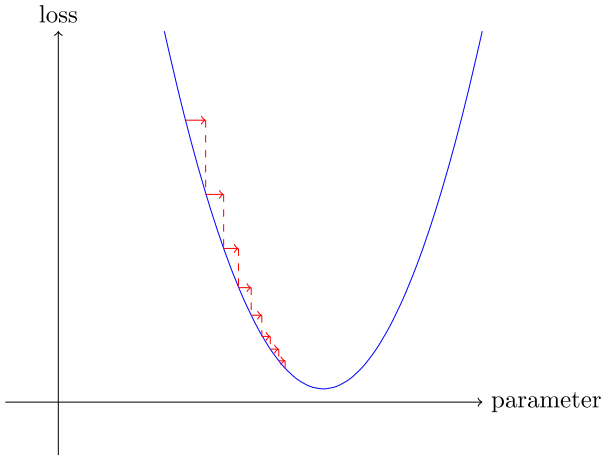
La forma de alimentar la red consiste en ir pasando de una a una las instancias a la red, calculando la salida y realizando la retropropagación del error, todo ello antes de pasar una nueva instancia. Como se ha visto en la sección anterior, ya tenemos la salida de la red y calculada una función de error para la salida de toda la red sobre uno de los ejemplos (e_i). Ahora, para actualizar los pesos de la capa inmediatamente anterior a la última, $K - 1$, antes de pasar una nueva instancia (l):

$$w_{ij}^{K-1}(l) = w_{ij}^{K-1}(l-1) - \alpha \frac{\partial e_j(l)}{\partial w_{ij}^{K-1}} \tag{3.9}$$

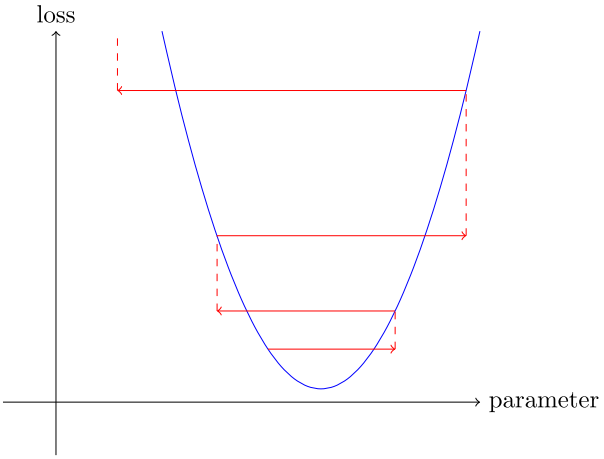
Como se aprecia en la Ecuación 3.9, los pesos calculados antes de la instancia l-ésima se calculan de forma ponderada con los pesos de la instancia anterior $l - 1$ y el error de la observación l. El peso de cada una de ellas viene dado por $\alpha \in [0, 1]$, hiperparámetro muy importante en las redes neuronales denominado **tasa de aprendizaje**. Una tasa de aprendizaje muy alta tiene poco en cuenta los pesos anteriores. En cambio, si es muy baja apenas cambiarán los pesos. Ambos casos no son deseables, dado que en un caso no llegaremos al óptimo porque la red aprende muy lento y en el segundo caso tampoco conseguiríamos llegar al diverger, como se puede ver en la Figura 3.7

Para calcular la actualización de los pesos, es necesario calcular la derivada del error $e(l)$. Si tomamos como medida del error el MSE, se calcularía de la siguiente forma:

$$\frac{\partial e_j(l)}{\partial w_{ij}^{K-1}} = -(y_j(l) - \hat{y}_j(l)) \frac{\partial y_j(l)}{\partial w_{ij}^{K-1}} \tag{3.10}$$



(a) Tasa de aprendizaje muy pequeña



(b) Tasa de aprendizaje demasiado grande

Figura 3.7: Consecuencias de usar tanto una tasa de aprendizaje muy pequeña como demasiado grande. Figuras obtenidas de [8]

Aplicando la regla de la cadena para calcular la derivada de la composición de funciones y sabiendo que en los términos de la Ecuación 3.6 el peso w_{ij}^{K-1} solo interviene en $w_{ij}^{K-1}a_i^{K-1}$:

$$\frac{\partial y_j(l)}{\partial w_{ij}^{K-1}} = f'(\sum_{i=1}^{n_{K-1}} w_{ij}^{K-1}a_i^{K-1} + b_j^K)a_i^{K-1}(l) \tag{3.11}$$

Renombrando la Ecuación 3.10 de la siguiente forma y sustituyendo:

$$\delta_j^K(l) = \frac{\partial e_j(l)}{\partial w_{ij}^{K-1}} = -(y_j(l) - \hat{y}_j(l)) \cdot f'(\sum_{i=1}^{n_{K-1}} w_{ij}^{K-1}a_i^{K-1} + b_j^K) \tag{3.12}$$

Por tanto, reemplazando la actualización de los pesos en la Ecuación 3.9:

$$w_{ij}^{K-1}(l) = w_{ij}^{K-1}(l-1) - \alpha\delta_j^K(l)a_i^{K-1}(l) \quad i = 1, ..., n_{K-1};\ j = 1, ..., n_K \tag{3.13}$$

Esta ecuación es aplicable también para actualizar los términos bias. Si se sigue el mismo razonamiento, se obtiene la siguiente fórmula para actualizar los pesos de una capa oculta k:

$$w_{ij}^{k}(l) = w_{ij}^{k}(l-1) - \alpha\delta_j^{k+1}(l)a_i^{k}(l) \quad i = 1, ..., n_{K-1};\ j = 1, ..., n_K;\ k = 1, ..., K-2 \tag{3.14}$$

donde:

$$\delta_j^{k+1}(l) = f'(\sum_{i=1}^{n_k} w_{ij}^k a_i^k + b_j^k) \sum_{s=1}^{n_{k+1}} \delta_s^{k+2}(l)w_{js}^k \tag{3.15}$$

De igual forma se puede extrapolar para los bias.

Para obtener δ es necesario calcular la derivada de la función de activación, f'; por ello se prefieren funciones como la sigmoide o la tangente hiperbólica cuyas derivadas son fáciles de calcular

a partir de la propia f, tal y como se vio en la Sección 3.1.2.

El descenso del gradiente puede ser visto como bajar una montaña: nos encontramos en un punto determinado y tenemos que bajar hasta la falda de la montaña usando las técnicas que podamos, en este caso bajamos por la dirección de máxima pendiente. Imaginemos que tenemos una bola y la dejamos caer por la montaña, una bola más pesada saltará los hoyos, en cambio, una bola más ligera se quedará en cada uno de ellos. Esta idea se puede añadir al algoritmo usando una media móvil para actualizar los pesos:

$$\begin{aligned} z(l) &= \beta \cdot z(l-1) + (1-\beta) \cdot \delta(l)a(l) \\ w(l) &= w(l-1) - \alpha z(l+1) \end{aligned} \tag{3.16}$$

donde $\beta \in [0, 1]$ es el **momento**. Si fuera 0 sería el descenso del gradiente original pero si es cercano a 1 la dirección de descenso es una media de las direcciones tomadas anteriormente.

Descenso estocástico del gradiente

El descenso estocástico del gradiente (SGD) es una adaptación del algoritmo anterior en el cual se realiza el bucle de entrenamiento por cada observación en vez de solo al final del conjunto, como si tuviéramos un tamaño de *batch* de 1. Además se calcula un gradiente aproximado, en vez de calcular el gradiente exacto. Estas dos modificaciones producen que se actualicen más rápidamente los pesos.

Por otra parte, al usar solamente una observación es posible que la función de pérdida fluctúe más bruscamente, lo cual se ha visto que no es un problema a la hora de entrenar, dado que la convergencia a la solución es más rápida aunque la estimación de los pesos tenga una varianza mayor.

Una aproximación intermedia es el **descenso del gradiente por mini-batches**, el cual se parece a SGD en que usa un subconjunto de observaciones para calcular el gradiente pero usa $n' > 1$ en vez de solamente una instancia.

RMSProp

RMSProp [9] es una variante del SGD en la que la diferencia principal es que utiliza una tasa de aprendizaje adaptativa: en vez de usar la misma para cada parámetro, emplea una diferente para cada uno controlada por una tasa global.

Con ello se consigue una velocidad de entrenamiento mayor dando una tasa más grande a los pesos que necesiten cambiar más y una inferior para otros. Para decidir qué tasa dar a cada uno se usan los gradientes: si han sido próximos a 0 en varias iteraciones es necesario aumentar la tasa

de aprendizaje porque la pérdida está siendo plana. En caso contrario, si los gradientes cambian mucho es necesario bajar la tasa de aprendizaje para evitar que diverjan.

Adam

Adam (*adaptative moment estimation*) [10] combina las ideas del SGD con momento y RMSProp. Usa estimaciones de los momentos de primer y segundo orden de los gradientes para adaptar la tasa de aprendizaje a cada peso de la red:

$$\begin{aligned} m(l) &= \beta_1(l)m(l-1) + (1-\beta_1(l))\delta(l)a(l) \\ v(l) &= \beta_2(l)m(l-1) + (1-\beta_2(l))(\delta(l)a(l))^2 \end{aligned} \tag{3.17}$$

En este caso, tenemos unos estimadores sesgados lo cual no solo ocurre en Adam, también en SGD con momento y en RMSProp. Para corregirlo:

$$\begin{aligned} m'(l) &= \frac{m(l)}{1-\beta_1(l)} \\ v'(l) &= \frac{v(l)}{1-\beta_2(l)} \end{aligned} \tag{3.18}$$

Por último, la actualización de los pesos la definimos como:

$$w(l) = w(l-1) - \alpha\frac{m'(l)}{\sqrt{v'(l)}+\epsilon} \tag{3.19}$$

donde ϵ es un valor muy pequeño ($\sim 10^{-8}$) para prevenir divisiones entre 0.

En [10] se demuestra de forma empírica que la convergencia es más rápida que con otros algoritmos para una serie de problemas, por tanto son necesarias menos iteraciones (y tiempo). En cambio, publicaciones recientes como [11] muestra que Adam (junto a otros métodos) no generaliza tan bien como SGD para ciertos casos.

3.1.7. Problemas comunes

Sobreajuste

El sobreajuste es un problema no solo específico de las redes neuronales sino de todo algoritmo de aprendizaje supervisado. Formalmente, si tenemos una hipótesis $h \in H$ generada por un algoritmo supervisado (como puede ser una red neuronal concreta), se dice que hay sobreajuste si existe otra hipótesis $h' \in H$ tal que h tiene menor error que h' sobre el conjunto de datos de entrenamiento pero h' tiene un error menor que h sobre el conjunto global de datos. Esto es, el método de aprendizaje no sabe generalizar (inferir), únicamente acierta con las instancias que ha visto anteriormente.

No hay duda que este puede ser un problema crítico ya que el objetivo fundamental de un algoritmo de aprendizaje es que sepa predecir correctamente sobre datos de los cuales no se tiene dicho conocimiento. Hay una serie de estrategias para calcular el error sobre un conjunto de datos y así comprobar si hay o no sobreajuste. Estas se basan en dividir el conjunto de datos en tres particiones (disjuntas):

- **Entrenamiento**: son los datos con los que se va a ajustar el modelo.
- **Validación**: sirven para evaluar un modelo frecuentemente; no "aprende" de ellos pero sirven para ajustar ciertos hiperparámetros.
- **Test**: es el conjunto más crítico; se usa una sola vez al final de terminar todas las etapas de entrenamiento y validación y es el que da un error más preciso, al ser totalmente nuevo.

Muchas veces, al no disponer de una gran cantidad de datos, no se separa el conjunto de test. Para obtener un error de generalización se utilizan varios métodos sobre el conjunto de validación como pueden ser *holdout*, validación cruzada o *leave-one-out*.

Vistas las formas de obtener el error y comprobar si tenemos sobreajuste, si fuera el caso debemos solucionar este problema. Hay una serie de métodos para paliarlo (o incluso eliminarlo totalmente) como son algunos de los contenidos en el capítulo 7 de [12]:

- **Principio de parsimonia**, también conocido como navaja de Ockham, se basa en que teniendo varias hipótesis que explican los datos de forma similar, deberíamos escoger la más simple (en el caso de las redes neuronales esto es la que menos parámetros tenga). De esta forma se evita el sobreajuste en muchas ocasiones dado que si hay un modelo más simple que generaliza de la misma forma que uno más complejo, el segundo es innecesariamente complicado.
- **Regularización de los pesos**: se usa un tipo de penalización sobre los pesos de la red, añadiéndola a la función a minimizar:
  - L_1 (también conocida como regresión *lasso*) la penalización usada es la suma de los valores absolutos de los pesos

$$\Omega(\theta) = ||w||_1 = \sum_j |w_j| \tag{3.20}$$

  - L_2 (también conocida como regresión *ridge*) se usa como penalización la suma de los pesos al cuadrado

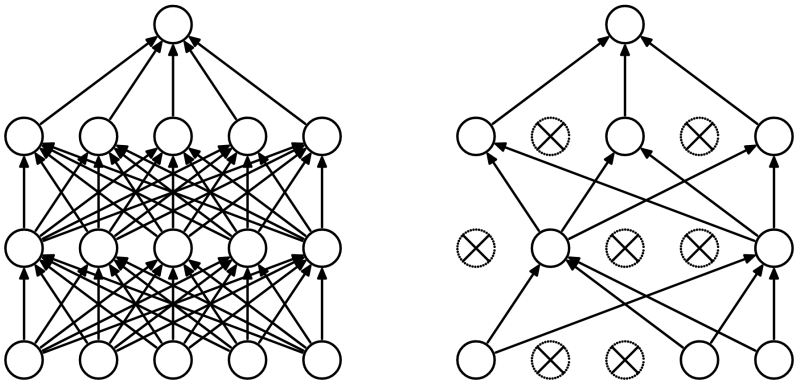
$$\Omega(\theta) = \frac{1}{2}||w||_2^2 = \frac{1}{2}\sum_j w_j^2 \tag{3.21}$$

  De esta forma se consigue que los pesos no sean demasiado grandes, lo cual es un signo de red más compleja (y por tanto de sobreajuste).

- ***Early stopping***: consiste en interrumpir el entrenamiento cuando el error de generalización crece. Dado que en cada época se van a realizar dos etapas, una de entrenamiento y otra de validación (*holdout*), si el rendimiento sobre el segundo conjunto empieza a decrecer entonces el entrenamiento es interrumpido.

- ***Data augmentation***: para conseguir que un modelo generalice mejor la manera más sencilla de hacerlo es entrenarlo con más datos. En la práctica, la cantidad de datos disponibles es limitada. En cambio, podemos generar más observaciones de forma artificial transformando las que tenemos. Por ejemplo, en el caso de imágenes, podemos rotarlas, darles la vuelta, añadir ruido, ... Cualquiera de estas transformaciones crea una "nueva" instancia para la red (lógicamente no es independiente).

- ***Dropout*** [13]: durante el entrenamiento, algunas neuronas de las capas ocultas son ignoradas de forma aleatoria, por lo que no se realizan las etapas de propagación de la entrada ni de retropropagación del error.



(a) Standard Neural Net (b) After applying dropout.

Figura 3.8: *Dropout*, obtenido de [13]

La combinación de redes con *dropout* actúa como un balance y así se reduce el sobreajuste global. Además, se fuerza a que la red aprenda de características más robustas que sean útiles con diferentes subconjuntos de las otras. Por otra parte, para converger, necesita aproximadamente el doble de iteraciones, pero cada una de ellas es más rápida.

Infraajuste

El problema complementario al anterior es que el algoritmo de aprendizaje no realice un ajuste razonable sobre los datos de entrenamiento. Generalmente para solventarlo en las redes neuronales hay que cambiar la arquitectura de la red directamente o añadir más parámetros a la existente para que consiga aprender de los datos correctamente.

Desvanecimiento del gradiente

El desvanecimiento del gradiente ocurre cuando los gradientes de la función de pérdida se aproximan a cero, lo cual hace que a partir de ese punto a la red le cueste mucho aprender. Esto ocurre normalmente cuando se añade más profundidad a la red con ciertas funciones de activación.

La razón por la que se da es que dado que muchas funciones de activación toman una entrada grande y su salida se encuentra en un rango $[a, b]$, un gran cambio en la entrada provoca solo un pequeño cambio en la salida. Por tanto, la derivada es pequeña.

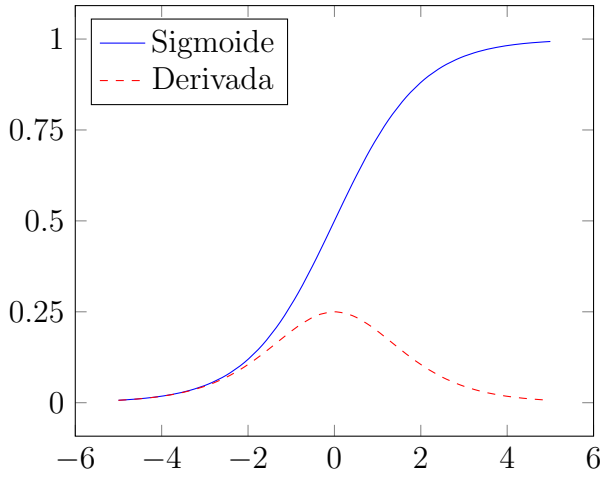


Figura 3.9: Función sigmoide junto a su derivada

Como ejemplo, en la Figura 3.9 se ha representado la función sigmoide junto a su derivada. Se puede apreciar que cuando $|x|$ toma valores grandes la derivada es prácticamente cero.

El desvanecimiento del gradiente es un problema importante en las redes más profundas, en las que tienen pocas capas ocultas normalmente no ocurre. Si tenemos una red con muchas capas con función de activación sigmoide, al hacer la retropropagación usando la regla de la cadena, las derivadas pequeñas se multiplican. Esto provoca que el gradiente decrezca exponencialmente. Un gradiente pequeño hace que los parámetros de las capas iniciales no se actualicen correctamente después de cada época. Estas capas son fundamentales para reconocer los elementos de los datos, por lo que si no se reconocen correctamente hace que la red entera sea poco precisa.

La solución más simple es usar otras funciones de activación como pueden ser el rectificador lineal, en donde no ocurre el problema de una derivada pequeña. Otra solución más elaborada es usar una red residual (ResNet), las cuales se comentarán más detenidamente en la Sección 3.3.

3.2. Mejoras en redes neuronales

En este apartado se van a explicar una serie de mejoras que se han ido produciendo recientemente en el ámbito de las redes neuronales a través de distintas publicaciones.

3.2.1. Política *1cycle*

Dado que los pesos iniciales, muchas veces fijados de forma aleatoria, no son los mejores para resolver el problema podemos pensar en usar una tasa de aprendizaje alta para que entrene más rápido. Esto es peligroso porque tanto en los pasos iniciales como finales podemos divergir muy rápidamente y no llegar al mínimo. Sin embargo, en los pasos intermedios sí que puede ser útil tener esa tasa alta para aproximarnos más rápidamente al punto óptimo.

Usando la idea de empezar y terminar con una tasa de aprendizaje baja y usar una alta en términos medios, se ha creado la política llamada *1cycle* [14]. De este modo se define una programación de la tasa de aprendizaje separada en dos etapas: la etapa de *warmup* (según los autores), donde la tasa va del valor mínimo al máximo y la etapa de *annealing* donde vuelve a decrecer al mínimo (Figura 3.10).

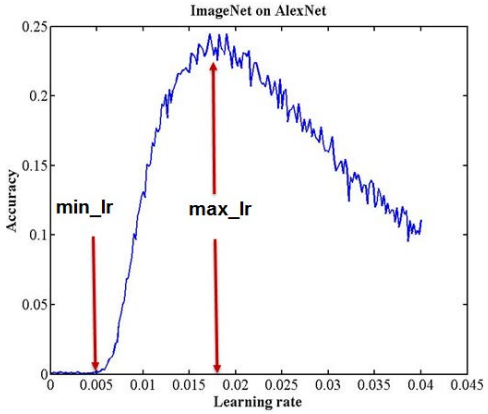


Figura 3.10: Programación de la tasa de aprendizaje, obtenido de [14]

1cycle tiene dos beneficios principalmente al entrenar con tasas altas de aprendizaje:

- Conseguimos un entrenamiento muy rápido (lo cual el autor denomina super-convergencia).
- Se palia el sobreajuste dado que saltamos sobre mínimos locales terminando en una parte más suave (y por tanto que generaliza mejor) de la función de pérdida.

Aparte, esta idea se puede extrapolar también al hiperparámetro momento de SGD [15]. Sugiere variar dicho término de forma opuesta a la tasa de aprendizaje (Figura 3.11).

3.2.2. Normalización del batch

La normalización del batch (*batchnorm*) [16] fue creada para solventar el problema de un alto número de activaciones cercanas a cero e intentar mantener una buena distribución de activaciones

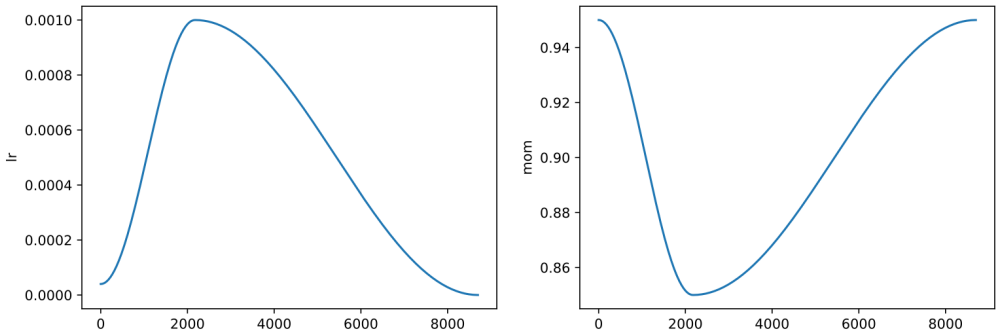


Figura 3.11: Evolución de la tasa de aprendizaje (izquierda) y el momento (derecha)

a lo largo del entrenamiento. La forma de conseguirlo es realizando normalización en las entradas de cada capa en cada mini-batch.

Para normalizar se toman las medias y desviaciones típicas de la capa en cuestión, lo cual puede provocar algún problema por lo que se añaden dos hiperparámetros más, llamados γ y β. Después de normalizar las activaciones se consigue una nueva activación y, por lo que la capa de *batchnorm* devolverá $\gamma \cdot y + \beta$. Es por esto que las activaciones pueden tener cualquier media o varianza independientemente de la salida de la capa anterior (los estadísticos se aprenden de forma independiente).

De esta forma se permite usar tasas de aprendizaje más altas (y por tanto entrenar más rápido) y tener menos en cuenta la inicialización. En la Figura 3.12 se puede ver la mejora que produce usando la arquitectura *Inception*, donde el entrenamiento es mucho más rápido (cinco veces más o menos).

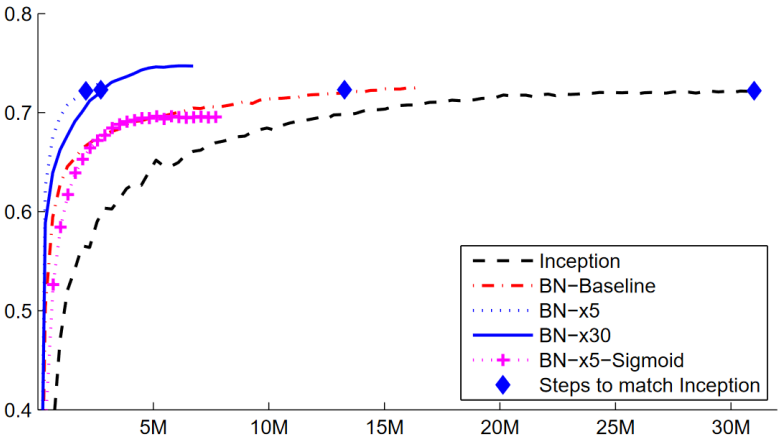


Figura 3.12: Mejora usando *batchnorm*, obtenido de [16]

3.3. Redes neuronales convolucionales

Las redes neuronales convolucionales (CNNs) son un tipo específico de red para el procesamiento de datos con topología de rejilla, como por ejemplo para el análisis de imágenes. Este tipo de redes tienen varios tipos de capas que se detallarán a continuación.

3.3.1. Tipos de capas

Convolucionales

Realizan la operación de **convolución**: tomando una serie de valores de entrada en forma de matriz y aplicando un kernel concreto (matriz también), reduce la entrada. El kernel se va aplicando de forma secuencial a la matriz de entrada de izquierda a derecha y de arriba abajo.

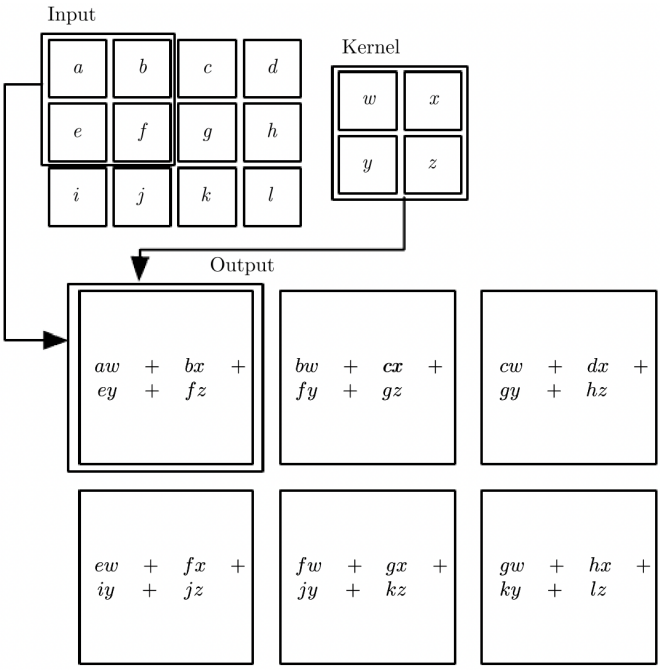


Figura 3.13: Operación de convolución, obtenido de [12]

Como se ve en la Figura 3.13, la aplicación del kernel (en este caso 2×2) produce una salida que se corresponde con la suma de la multiplicación del kernel por la entrada. Se puede aplicar el kernel celda a celda (como se ve en la imagen) o se puede hacer saltando celdas (se suele llamar *stride*). Además, si la dimensión del kernel no es múltiplo de la dimensión de la entrada o viceversa, es necesario añadir un margen (*padding*) alrededor de la entrada, que consiste en una adición de un margen artificial para evitar salirse al aplicar el *kernel*.

La dimensión de salida (n_{out}) se puede calcular de la siguiente forma:

$$n_{out} = \frac{n_{in} - k + 2p}{s} + 1 \tag{3.22}$$

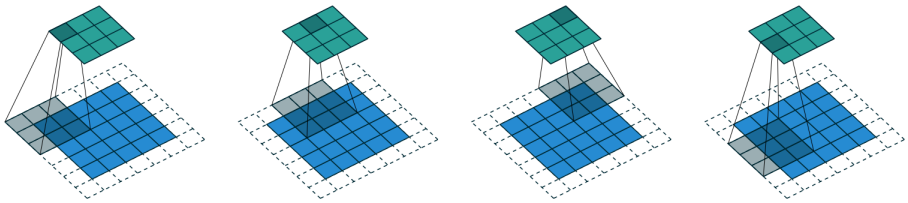


Figura 3.14: Kernel 3×3 aplicado con un *stride* de 1 y un *padding* de 1. Obtenido de [8]

donde n_{in} representa la dimensión de la entrada, k la del *kernel* y p el *padding*. Después de esta capa se suele aplicar una función de activación no lineal, como puede ser la ReLU.

Pooling

Es un tipo de capa que se añade después de las capas convolucionales y de la función de activación no lineal. Sirve para reducir el tamaño de la imagen, reemplazando la salida en cierta localización por un resumen estadístico de las salidas de alrededor. El más común es el *max pooling*, que devuelve el valor del máximo de las casillas vecinas (el número de casillas vecinas a usar lo determina el *kernel*).

Al ser una operación rápida, se consigue ganar velocidad ya que reduce el tamaño de la entrada para otro tipo de capas más costosas. Además, consigue reducir el sobreajuste dado que las capas convolucionales recuerdan de manera exacta la posición de las entradas, por lo que al reducir la resolución de la imagen se consigue mantener las características más importantes.

La operación de *pooling* también consta de un filtro (*kernel*), *stride* y *padding* al igual que la convolución. La salida se puede calcular de la misma forma con la Ecuación 3.22.

Fully connected

Se llaman así a un tipo de capas en los que todos las entradas de una capa están conectadas con todas las unidades de activación de la siguiente capa (como si fuera un perceptrón multicapa, Figura 3.6). Dado que la entrada de un perceptrón multicapa es de un vector en una dimensión (en vez de una matriz como las capas convolucionales) es necesario realizar la operación de aplanado (Figura 3.15).

Este tipo de capas suelen encontrarse al final de la red y su función es recopilar los datos extraídos de las capas anteriores para formar la salida. Se pueden ver como la capa que realiza realmente la clasificación (o regresión) después de que los dos tipos de capas anteriores hayan extraído las características principales. Es una capa que es computacionalmente costosa, aunque algo menos que la operación de convolución.

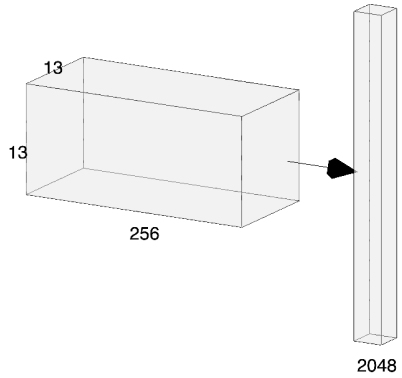


Figura 3.15: Aplanado

Existen muchos tipos de arquitecturas de redes neuronales convolucionales que combinan este tipo de capas como pueden ser las *AlexNet*, *ResNet* o *EfficientNet* (en orden cronológico de aparición).

3.3.2. AlexNet

AlexNet [17], [18] es el modelo que ganó la competición a gran escala ImageNet LSVRC-2012 por un margen bastante amplio. La competición consistía en clasificar imágenes de la vida cotidiana en 200 categorías como pueden ser perros o gatos, con un conjunto de entrenamiento de 1.2 millones de imágenes RGB, 50000 de validación y 150000 de test. Las características fundamentales que lo diferencian de otros modelos son:

- Usar ReLU (Figura 3.5) como función de activación no lineal en vez de la tangente hiperbólica (Figura 3.4).
- Para evitar el sobreajuste se usa *data augmentation* y *dropout* pero no se usa regularización.
- Para reducir el tamaño de la red se usa *overlap pooling* en vez de *max pooling* (es una modificación).

La arquitectura se describe en 3.16, la cual consta de 5 capas convolucionales y 3 de *pooling*:

3.3.3. ResNet

Las redes neuronales residuales [19], de ahí su nombre *ResNet*, es seguramente el tipo de red más utilizado hoy en día. Los autores de la arquitectura se dieron cuenta que no por tener más capas la red funcionaba mejor: una red con menos capas generalizaba mejor que una con muchas capas, tanto en el conjunto de entrenamiento como en el de validación (3.17).

Para probarlo, comenzaron creando una red de 20 capas, la cual se entrena correctamente, y después añadieron otras 36 capas que no realizaban nada, todos los pesos eran 1 y los *bias* de 0. El resultado fue una red de 56 capas que debería hacer lo mismo que la de 20, pero al usar SGD

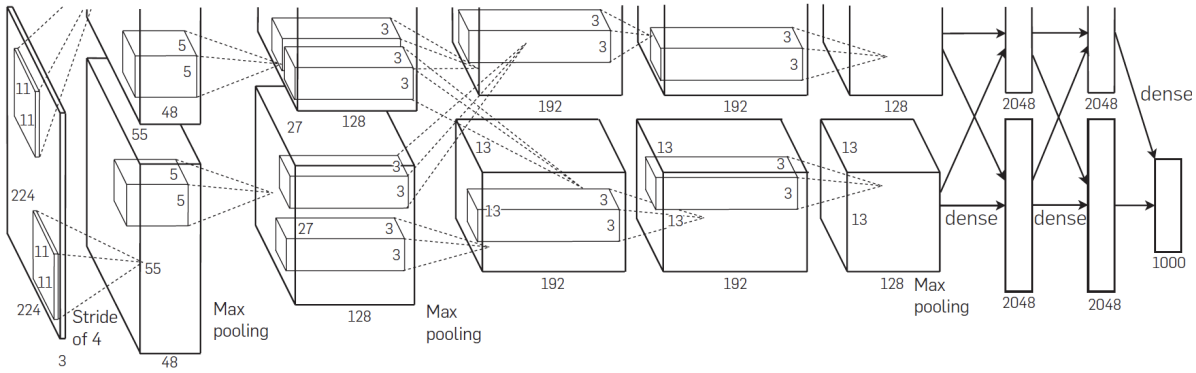


Figura 3.16: Arquitectura de AlexNet, obtenido de [17]

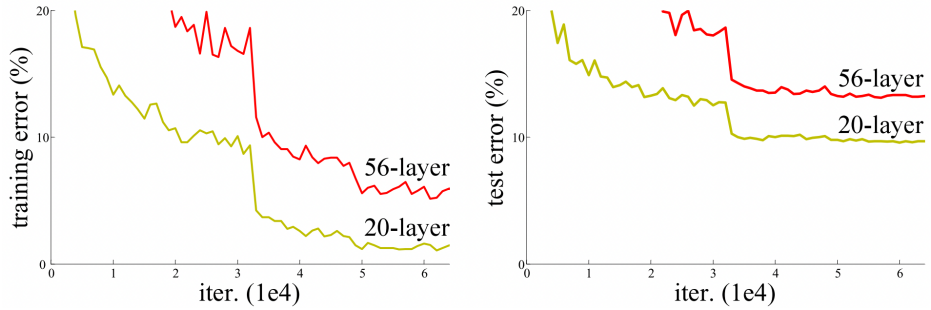


Figura 3.17: Comparación del error de entrenamiento y validación con 20 y 56 capas. Obtenido de [19]

como método de optimización la de 56 funcionaba peor que la de 20 (Figura 3.17).

Otra forma de crear las 36 capas extra es la siguiente: sea `conv(x)` una capa de convolución sobre una entrada x junto a una *batchnorm* y una función de activación ReLU. Como se ha visto en la Sección 3.2.2, la operación de normalización del batch devuelve $\gamma \cdot y + \beta$. Podemos reemplazar la salida de `conv(x)` por `x + conv(x)`, es decir, que la salida sea una suma de la entrada original y la salida de la convolución. Con esto ganamos tener más parámetros entrenables que pueden mejorar el funcionamiento de la red.

En [19] se propone una variante de esta explicación: se tienen dos operaciones de convolución anidadas en vez de una, por lo que en cada bloque la salida es `x + conv2(conv1(x))` como se ve en la Figura 3.18. Usar la entrada (x) como parte de la salida sin modificar se conoce como *identity mapping* (parte derecha). $F(x)$ representa una operación de convolución sobre la entrada x.

La metodología de entrenamiento es la misma que con cualquier red convolucional, solamente que esta red está compuesta por bloques ResNet (en adelante `block(x)`).

Otra forma de entender las ResNets es la siguiente: la salida de cada bloque contando el *identity mapping* es `y = x+block(x)`, por lo que realmente no estamos pidiendo al bloque que prediga

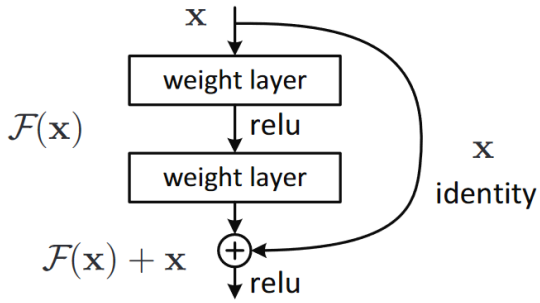


Figura 3.18: Bloque ResNet. Obtenido de [19]

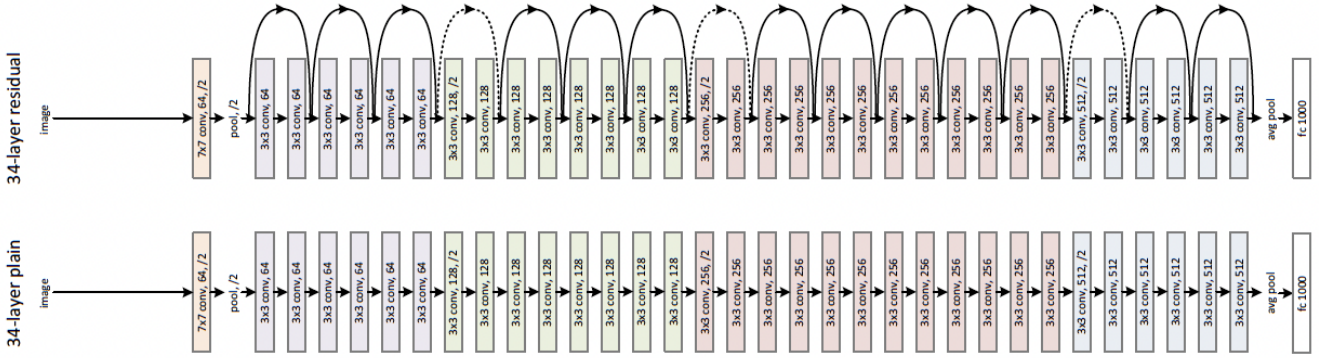


Figura 3.19: Red neuronal de 34 capas (abajo), ResNet del mismo número de capas (arriba). Obtenido de [19]

y, sino la diferencia entre y y x. Por tanto el trabajo de estos bloques no es predecir ciertas características, sino minimizar el error entre la salida y y la entrada x, de ahí su nombre: están prediciendo residuales.

3.3.4. EfficientNet

Las *EfficientNets* [20] se puede decir que hoy en día son la nueva aparición de las arquitecturas de redes neuronales convolucionales, mencionadas por primera vez en 2019. La diferencia con arquitecturas anteriores es la forma de escalar, de tener un número de parámetros mayor para entrenar: por ejemplo las ResNets utilizan un número mayor de capas que se van añadiendo al modelo más sencillo (con menos capas), desde ResNet-18 a ResNet-200 (18 y 200 corresponde al número de capas). Tradicionalmente, la forma de escalar el modelo es sobre su profundidad, añadiendo más capas; sobre su anchura, con un número de neuronas mayor por capa o sobre la resolución de la imagen (menos común). Estos métodos se pueden combinar aunque no es muy recomendable porque requieren un trabajo manual bastante tedioso y normalmente consiguen peor rendimiento en términos de eficiencia y error.

La nueva idea de escalado introducida por los autores de esta arquitectura se denomina compound scaling. Según sus resultados experimentales, es crítico balancear cada una de las dimensiones de la red (anchura/resolución/profundidad), lo cual se puede lograr de forma bastante sencilla

escalándolas por un ratio en particular, en vez del método tradicional que es escalar solamente una de las dimensiones arbitrariamente.

Por ejemplo, si queremos usar 2^N más recursos computacionales, podemos aumentar la profundidad de la red por α^N, la anchura por β^N y el tamaño de las imágenes por γ^N; donde α, β y γ son constantes determinados por una búsqueda inicial sobre el modelo base más sencillo.

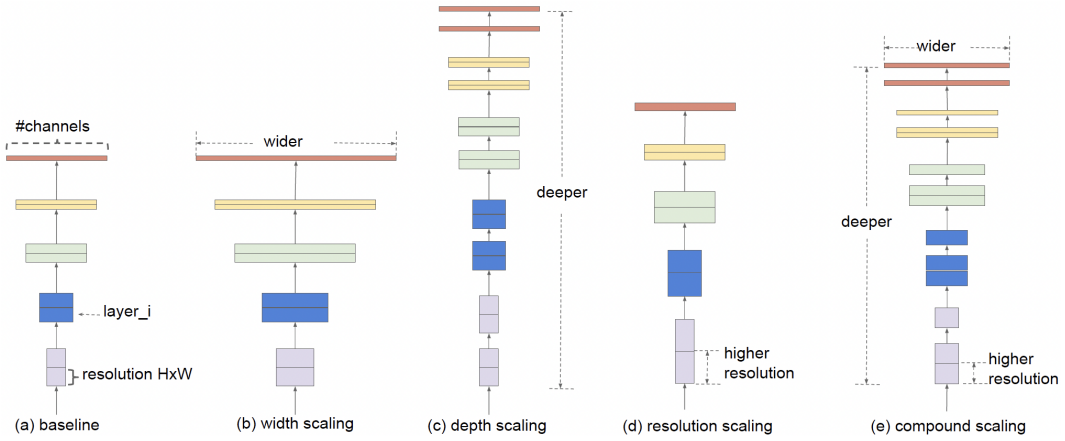


Figura 3.20: Distintas formas de escalar una red neuronal: a) modelo base, b) - d) formas tradicionales de escalar, e) propuesta nueva de EfficientNet. Obtenido de [20]

En el artículo se prueba de forma experimental la eficacia sobre varios conjuntos de datos: ImageNet, CIFAR-100, Flowers y 3 más. En todos consigue mejores resultados que otras arquitecturas como ResNet, DenseNet, Inception o Xception. Tal y como se ve en la Figura 3.21, con un número de parámetros muy inferior a otras arquitecturas consigue un resultado bastante mejor sobre ImageNet. Particularmente, consigue una tasa de acierto del 84.3 %, siendo 8.4 veces más pequeño y 6.1 veces más rápido que el mejor modelo existente (GPipe).

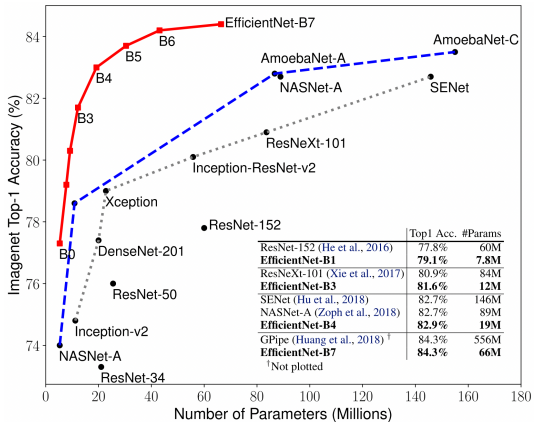


Figura 3.21: Comparación del tamaño del modelo y la tasa de acierto sobre ImageNet. Obtenido de [20]

Los distintos tamaños de EfficientNet se definen desde B0 a B7 (de menor a mayor número de parámetros).

Capítulo 4

Tecnologías utilizadas

4.1. PyTorch

PyTorch [21] es un *framework* de código abierto para la creación de modelos de aprendizaje automático basada en Torch y desarrollado por el Laboratorio de Investigación de Inteligencia Artificial de Facebook (FAIR, por sus siglas en inglés). Tiene dos interfaces diferentes, una para Python y otra para C++ (la primera de ellas es la que se va a usar en este trabajo).

Consiste en un conjunto de bibliotecas de bajo nivel principalmente, aunque tiene un par de características reseñables de alto nivel como son la computación con tensores con aceleración por GPU y una serie de redes neuronales profundas construidas por defecto. Se divide en los siguientes componentes:

- `torch`: operaciones básicas con tensores, con sintaxis similar a NumPy [22]
- `torch.autograd`: diferenciación automática que soporta todas las operaciones de `torch`
- `torch.nn`: biblioteca de redes neuronales
- `torch.optim`: métodos de optimización generales como pueden ser SGD, Adam, ...
- `torch.multiprocessing`: básicamente multiporcesado de Python pero con compartición de memoria para tensores
- `torch.utils`: conjunto de funciones que pueden ser útiles para uno o varios componentes
- `torch.legacy`: componentes que se han eliminado pero que se tienen que guardar por razones de compatibilidad

Además de estos componentes, existen varias bibliotecas especializadas e integradas en el framework para distintas aplicaciones como son:

- `torchaudio`: procesamiento de audio y señales

- `torchtext`: utilidades para el procesamiento de lenguaje natural (NLP)
- `torchvision`: conjunto de funciones para el procesamiento de imágenes

La estructura básica del *framework* es el tensor. Un tensor es una estructura de datos especializada similar a los arrays n-dimensionales (tal que $n > 0$). La forma de trabajar con ellos es similar a los `ndarray` de NumPy, con la salvedad de que los tesores soportan aceleración por GPU. Esta aceleración es crítica en los problemas de aprendizaje automático dado que la capacidad de paralelización de una GPU es mucho mayor que la de las CPU, por lo que la velocidad de cálculo es muy superior al poder realizar más operaciones de forma simultánea.

4.2. fast.ai

fast.ai [23], [8] es una biblioteca de alto nivel para PyTorch, que simplifica la creación de los modelos de redes neuronales y añade una serie de características basadas en las últimas tendencias y artículos publicados (*state-of-the-art*). También permite la modificación del API de bajo y medio nivel, lo cual es útil para la implementación de técnicas que no sean provistas por el API de alto nivel.

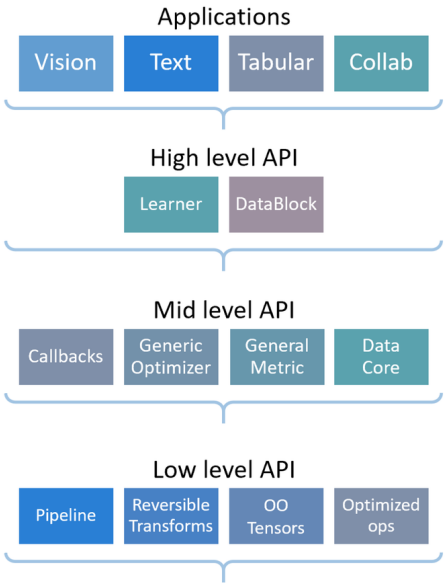


Figura 4.1: Estructura de fast.ai. Extraído de [24]

Haciendo una comparación con otras bibliotecas de *deep learning*, como es la desarrollada por Google (la más popular hoy en día), fast.ai es similar a Keras (API de alto nivel) y PyTorch es como Tensorflow (API de bajo nivel).

A continuación se detallarán distintas clases de fast.ai usadas en este proyecto, cuya documentación se ha extraído de [24] y del propio código fuente [23].

4.2.1. DataBlock

Forma parte de la API de alto nivel y sirve de ayuda para la creación de `Datasets` y `DataLoaders`. Para crear un `DataBlock` son necesarios los siguientes datos:

- Tipos de los datos de entrada y las etiquetas. Para ello, hay definidas una serie de clases, llamadas bloques, que representan los diferentes tipos de datos posibles (además se pueden crear unos propios). Los usados en este trabajo han sido:
  - `ImageBlock`: para los datos de entrada, ya que son imágenes
  - `RegressionBlock`: para los datos de salida, ya que en este caso no son etiquetas sino una regresión de 2 (o 3) valores
- `get_x`: función para obtener los datos de entrada (x), en este caso dicha función devuelve la ruta a las imágenes.
- `get_y`: función para obtener las etiquetas o los valores a predecir (y), en este caso devuelve un array con 2 (o 3) valores
- `splitter`: forma de dividir los datos en entrenamiento y test para ajustar los pesos de la red

4.2.2. DataLoaders

Representa un iterable para el conjunto de datos (creado con un `DataBlock`, por ejemplo). Es el responsable de crear los distintos *batch*, tanto del conjunto de entrenamiento como del de validación y hacer las transformaciones a cada uno de los elementos.

4.2.3. Learner

Es el encargado de realizar el entrenamiento: agrupa un modelo, un conjunto de datos (a través de su `DataLoaders`) y una función de pérdida a optimizar. Tiene varias funciones para entrenar y ajustar los hiperparámetros del modelo, las más comunes son:

- `fit`: ajusta el modelo dada una tasa de aprendizaje y el número de épocas.
- `fit_one_cycle`: no solo entrena el modelo sino que también ajusta la tasa de aprendizaje y el momento, los cuales van cambiando a lo largo de las épocas. Es la implementación de la política *1cycle* (Sección 3.2.1).
- `fine_tune`: consiste en una mejora para realizar transfer learning desde un modelo preentrenado (comunmente desde ImageNet [25]). Realiza dos etapas:
  1. Se entrena la última capa siguiendo la política *1cycle* una serie de épocas (1 por defecto) dado que las demás capas han sido preentrenadas anteriormente ya y se supone que

partimos de unos pesos razonables para nuestro problema (ya que estamos realizando transfer learning).

2. Se aplica *1cycle* en todas las capas.

Según los propios creadores de fast.ai las diferencias entre los dos últimos métodos no son muy grandes, los cuales funcionan de forma similar en la mayor parte de los problemas [26].

Además tiene otras mejoras como permitir personalizar cada etapa del entrenamiento a través de los `Callback` o buscar la tasa de aprendizaje a través de un método automático [14].

4.2.4. Callback

Los `Callback` son un método basado en eventos para realizar acciones en varias etapas del entrenamiento. Es posible crear *callbacks* para, por ejemplo: registrar métricas, guardar el modelo periódicamente, parar el entrenamiento cuando se cumpla una condición, usar múltiples GPUs, ... Esto permite usar una API de alto nivel como la de fast.ai modificándola en los puntos o supuestos que necesitemos, dado que es muy probable que necesitemos cambiar componentes de la API de medio o bajo nivel, lo cual se consigue con este método.

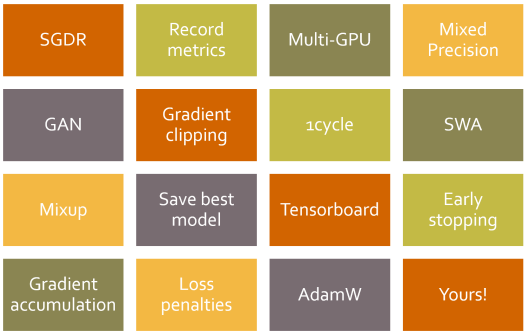


Figura 4.2: Ejemplos de callbacks. Extraído de [8]

Para implementar un callback propio, es necesario crear una clase que herede de `Callback` y definir los eventos donde queremos realizar una acción. Los eventos están acotados por la API y son:

- Antes de: todo el aprendizaje / cada época / entrenamiento en cada época / validación en cada época / cada batch.
- Después de: todo el aprendizaje / cada época / entrenamiento en cada época / validación en cada época / cada batch / calcular las predicciones / calcular la función de pérdida / cada paso (y antes de que los gradientes se pongan a 0) / la propagación hacia atrás.

En cualquiera de esos eventos se pueden realizar acciones, lo cual se consigue leyendo y/o modificando los atributos del `Learner` o de otros `Callback` que se hayan añadido.

4.2.5. Entrenamiento

Una de las ventajas principales de usar fast.ai frente a hacerlo con PyTorch es que no es necesario programar el bucle de entrenamiento "a mano": como se ha visto anteriormente, la clase `Learner` encapsula el entrenamiento y la optimización de una función de pérdida concreta a partir de un conjunto de datos dado en forma de `DataLoaders`. Podemos describir el entrenamiento de la siguiente forma:

1. A partir de un conjunto de datos, se genera un `DataBlock` y después un `DataLoaders` ya dividido en batch de un tamaño determinado.

2. Se crea un `Learner` a partir del `DataLoaders`, de una función de pérdida y un optimizador.

3. Se invoca una función de las especificadas anteriormente para entrenar y validar el modelo. Por ejemplo, la función básica realiza los siguientes pasos:

```
for xb, yb in dl:
        loss = loss_func(model(xb), yb)
        loss.backward()
        opt.step()
        opt.zero_grad()
```

En este proyecto se ha usado principalmente fast.ai aunque algunas definiciones de callbacks se han realizado con PyTorch, modificando directamente tensores.

4.3. JupyterHub

El entorno de desarrollo integrado (IDE) usado en este proyecto ha sido JupyterHub. Es uno de los proyectos de la organización sin ánimo de lucro Jupyter y básicamente consiste de una implementación de JupyterLab (o Jupyter Notebook) para máquinas con recursos compartidos por varios usuarios, facilitando el acceso y mantenimiento.

Jupyter funciona principalmente a base de cuadernos, los cuales son un formato de archivos que combinan código (en lenguaje Python, Julia o R) con texto (en formato Markdown) a través de celdas, lo cual añade una capa de interactividad que no tienen los archivos de código fuente dado que se puede ir ejecutando cada una de las distintas celdas por separado, viendo los resultados al instante (sin tener que ejecutar todo el cuaderno). Cada uno de los cuadernos tienen un kernel de IPython asociado, lo cual permite mantener las variables en tiempo de ejecución.

4.4. Comet.ml

Comet.ml [27] es un servicio en la nube para la gestión de los experimentos de aprendizaje automático. Se organiza en proyectos llamados *workspaces*, los cuales se dividen en distintos

experimentos. Cada experimento es un modelo de aprendizaje con unos parámetros fijos y sus resultados, agrupados en forma de paneles en una interfaz web. Las pestañas por defecto son:

- *Gráficos*: para cada una de las métricas definidas en el experimento, es posible crear un gráfico de barras o de líneas
- *Paneles*: herramienta para crear nuevas gráficas o vistas usando datos guardados en el propio experimento. Es posible crearlos usando JavaScript o Python.
- *Código*: vista del código usado en el experimento, como si fuera un sistema de control de versiones (VCS)
- *Hiperparámetros*: opciones de configuración del experimento, tanto los usados para la red neuronal (tasa de aprendizaje, momento, ...) como otros que queramos guardar (número de épocas usadas, función de pérdida, ...)
- *Definición del modelo*: modelo usado, número de capas y contenido de cada capa
- *Output*: salida de la consola
- *Métricas del sistema*: para cada instante de tiempo, porcentaje de CPU usada, memoria de la GPU usada, ...
- *Paquetes instalados*: lista de los paquetes de Python del sistema junto con su versión
- *Imágenes*
- *Matrices de confusión*
- *Otros*: datos de cualquier tipo, en este caso se han guardado archivos CSV con la predicción para cada una de las observaciones y el archivo pth de fast.ai que almacena los pesos de la red al terminar.

Comet.ml pretende ser similar a un VCS como Git pero orientado a los problemas de aprendizaje, permitiendo guardar todos los elementos necesarios en cada experimento para que pueda ser fácilmente reproducible.

Es fácilmente integrable en un proyecto en curso, basta con añadir las siguientes dos líneas de código:

```
from comet_ml import Experiment
experiment = Experiment(project_name="my-project", workspace="my-workspace")
```

Y ya podremos empezar a guardar información con las funciones que da la API de Comet.ml. Dado que fast.ai no provee de una API para registrar el experimento con esta herramienta, se ha escrito el siguiente Callback para guardar los parámetros mientras se va ejecutando la red neuronal:

```
class LogMetricCometML(Callback):
  order = 100
  def __init__(self, experiment):
    self.experiment = experiment

  def before_fit(self):
    self.experiment.set_model_graph(self.learn.model, overwrite=True)
    self.experiment.log_parameter("n_epoch", self.learn.n_epoch)
    self.experiment.log_parameter("bs", self.learn.dls.bs)
    self.experiment.log_parameter("optimizer",
          str(self.learn.opt_func).split(' at')[0].split('function ')[1]
    )
    self.experiment.log_parameter("loss_function", str(self.learn.loss_func))

  def after_step(self):
    self.experiment.log_metrics(
        self.learn.opt.hypers[0],
        step=self.learn.n_iter*self.learn.epoch + self.learn.iter)

  def after_epoch(self):
    for (n,v) in zip(self.recorder.metric_names[1:-1], self.recorder.final_record):
    if n != "train_loss":
      with self.experiment.validate():
        self.experiment.log_metric(n if n != "valid_loss" else "loss",
              v, epoch=self.learn.epoch
        )
    else:
      self.experiment.log_metric("loss_", v, epoch=self.learn.epoch)
```

4.5. GPU

Aunque las unidades de procesamiento gráfico (GPU, por sus siglas en inglés) se desarrollaron originalmente para acelerar el procesamiento de los gráficos (como su propio nombre indica), pueden mejorar los procesos de entrenamiento de redes neuronales de manera drástica. Esto se debe principalmente a su alto grado de paralelismo (ejecutar múltiples procesos de manera simultánea), conseguido en gran medida a su arquitectura SIMD (*Single Instruction Multiple Data*), lo que permite ejecutar una sola instrucción pero con un nivel de paralelismo muy alto. En cambio, las CPU tienen una arquitectura MIMD (*Multiple Instruction Multiple Data*), lo que les permite ejecutar diferentes instrucciones, consiguiendo cálculos más complejos. La GPU que se ha utilizado es una NVIDIA GeForce RTX 3060, con 12GB de memoria gráfica GDDR6.

Por defecto PyTorch utiliza la GPU si la detecta en el sistema, a través de CUDA (*Compute Unified Device Architecture*), un lenguaje de programación desarrollado por NVIDIA. Es posible activar manualmente su funcionamiento con las siguientes líneas de código:

```
torch.cuda.set_device(0)
device = torch.device(0)
```

Además, para comprobar en todo momento la carga de la GPU así como varios parámetros de temperatura y uso de memoria gráfica, es posible usar la orden `nvidia-smi` en una terminal, arrojando la siguiente información:

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 510.73.05    Driver Version: 510.73.05    CUDA Version: 11.6     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  NVIDIA GeForce ...  Off  | 00000000:04:00.0 Off |                  N/A |
|  0%   44C    P8    14W / 170W |     72MiB / 12288MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A      1155      G   /usr/lib/xorg/Xorg                 55MiB |
|    0   N/A  N/A      1523      G   /usr/bin/gnome-shell               14MiB |
+-----------------------------------------------------------------------------+
```

Capítulo 5

Contexto astronómico

Antes de exponer las características particulares sobre las cuales se ha trabajado se van a definir varios conceptos generales de astrofísica y astroinformática usados posteriormente.

5.1. Conceptos astrofísicos

En esta sección se definen de forma breve los fundamentos de la cosmología moderna que pueden ser útiles para entender la naturaleza de los datos que se van a tratar posteriormente. Se han desarrollado con la ayuda de materiales provistos por Fernando Buitrago Alonso como presentaciones propias de conferencias.

5.1.1. Desplazamiento al rojo

El desplazamiento al rojo, también conocido como *redshift*, consiste en el incremento de la longitud de onda (y por tanto la disminución de la frecuencia) que experimentan los fotones cuando viajan por el espacio o el tiempo dado que se ven afectados por la expansión del universo. Se suele denotar por la letra z, es una magnitud adimensional y se calcula según la longitud de onda (λ) emitidas y observadas por un objeto:

$$z = \frac{\lambda_{obs} - \lambda_{emit}}{\lambda_{emit}} \tag{5.1}$$

En un Universo en expansión, el concepto de desplazamiento al rojo puede ser pensado como el equivalente a la distancia a la que se encuentra la galaxia.

5.1.2. Brillo superficial

El brillo de los objetos astronómicos extensos tales como galaxias y nebulosas se mide con el brillo superficial (denotado con las siglas SB del inglés *surface brightness*). Este cuantifica el brillo aparente por unidad de área angular, por lo que depende de la densidad de luminosidad de su superficie. La unidad de medida son magnitudes por arco-segundo al cuadrado (mag/arcsec^2) y se define por la siguiente fórmula:

$$\mu = -2.5 \cdot \log_{10}(F_0) + zp + 5 \cdot \log_{10}(A) \tag{5.2}$$

donde:

- μ: brillo superficial en mag/arcsec^2
- zp: constante *zeropoint*, dependiente del telescopio. Transforma las unidades de brillo de la imagen (cuentas de cada píxel) a unidades de energía
- F_o : flujo observado desde el detector, energía por unidad de área
- A: área a investigar

Como se puede comprobar leyendo la fórmula y debido al signo menos al principio y el logaritmo, brillos superficiales mayores corresponde con objetos más tenues, y viceversa. Cuando nos alejamos de la parte del universo más cercano (Universo Local), se pierde brillo superficial debido al desplazamiento al rojo, por lo que a la Ecuación 5.2 habría que añadirle el término $+10 \cdot \log_{10}(1+z)$ donde z es el desplazamiento al rojo.

5.1.3. Fotometría

La naturaleza de la información que se recibe del universo es radiativa, es decir, se estudia la luz principalmente. Los métodos para poder observar y analizar todo el espectro electromagnético se han ido perfeccionando. Según sea la longitud de onda detectada se visualiza de una forma u otra.

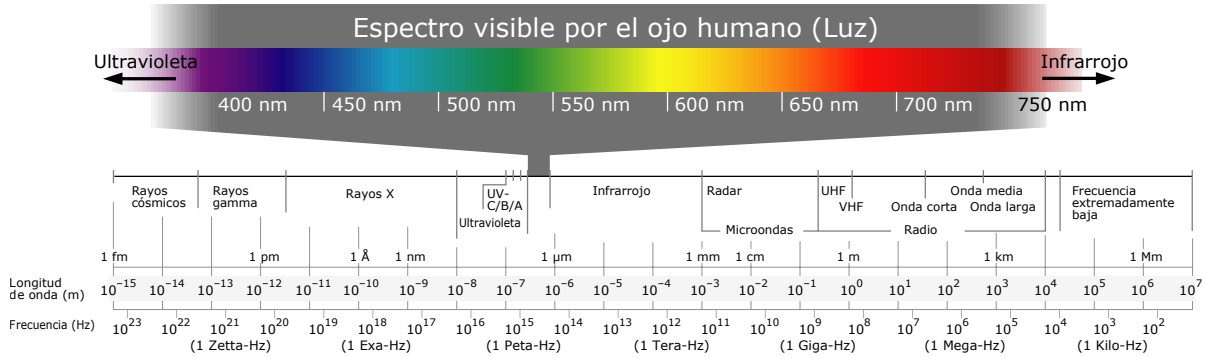


Figura 5.1: Espectro electromagnético, obtenido de [28]

Para conseguir obtener partes de dicho espectro, se utilizan distintos filtros de luz que se colocan delante del detector y con ellos se consigue dividir el espectro dependiendo de una longitud de onda central y una anchura. Con ello se consigue ver todo el espectro "por partes": vamos barriéndolo con distintos filtros. Por ejemplo, los dispositivos de captura habituales cotidianos utilizan filtros RGB (rojo, verde y azul) o CMYK (cian, magenta, amarillo y negro). En el caso de los telescopios no usan los filtros naturales de la luz visible, sino otros que son estándar en astronomía. Por tanto, las imágenes obtenidas del espacio no tienen un color concreto, sino que

depende con que filtro las veamos.

Por ejemplo, el telescopio Hubble (HST por sus siglas en inglés) posee varios filtros cuyos nombres van asociados a cada una de las cámaras del telescopio: WFC3/IR corresponde a una cámara de infrarrojos, ACS a una de luz visible y WFC3/UVIS a luz ultravioleta. Para cada una de estas cámaras hay varios filtros, dependiendo de la longitud de onda que detecten y corresponden con cada una de las líneas de la Figura 5.2.

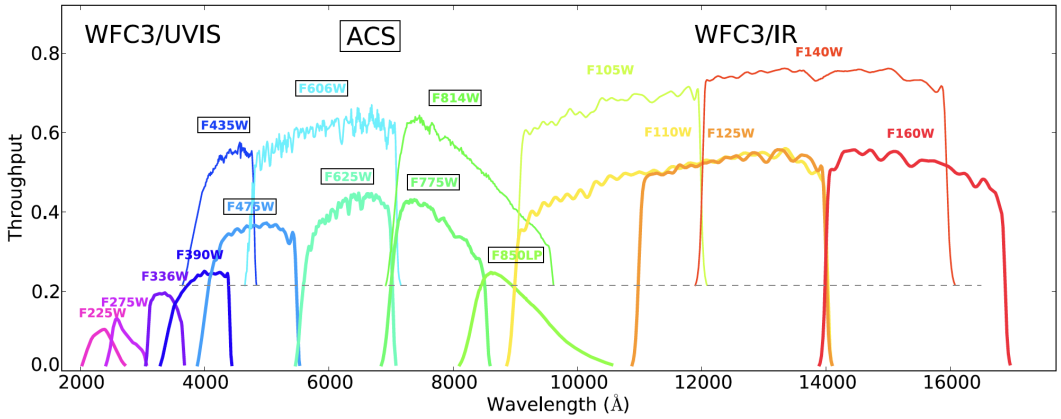


Figura 5.2: Esquema de los filtros usados por el HST, obtenido de [29]

El futuro telescopio Euclid va a tener otros tipos de filtros diferentes: uno de luz visible (VIS) y en el infrarrojo (Y, J, H). Como se puede ver en la Figura 5.3 comprende unas longitudes de onda algo diferentes al de HST: empieza en 4000 Å en vez de en 2000 Å y termina un poco más que el HST, en 20000 Å. Si volvemos a la Figura 5.1, podemos comprobar que aproximadamente ambos telescopios recogen todo el rango visible por el ser humano dado que $4000\text{Å} = 400nm$. El HST además puede detectar parte del espectro ultravioleta e infrarrojo, en cambio Euclid únicamente el infrarrojo pero en mayor cantidad (mayores longitudes de onda).

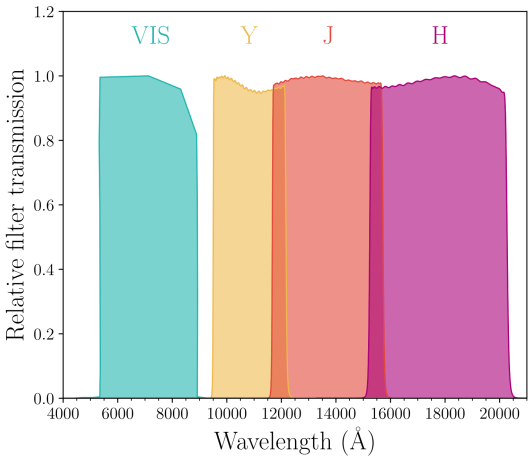


Figura 5.3: Esquema de los filtros que va a usar el telescopio Euclid, obtenido de [30]

Además, cada uno de los filtros del Euclid tiene mayor amplitud que los del HST, la consecuencia fundamental de esto es que por cada filtro se obtiene una cantidad mayor de fotones pero entre

estos no hay diferencias. Es decir, al tener solo un filtro de luz visible (VIS), no diferencia entre los distintos colores que percibimos los humanos, sino que "ve" todo en un solo canal, como si estuviera en escala de grises.

5.2. Astroinformática

La cantidad de datos que se están generando en muchas áreas científicas está llegando a unas dimensiones gigantescas. Por ello, es necesario tener un método de procesamiento y análisis de datos que se pueda usar con cantidades ingentes de datos. Como consecuencia en los últimos años están naciendo muchas áreas científicas orientada al análisis de datos, a veces denominadas como *X-Informatics* donde *X* se refiere a cualquier ciencia (biología, geología, astronomía, ...) e *informatics* a la parte de organización, acceso y minería de datos.

El campo de la astronomía no es ajeno a este crecimiento de datos, por lo que en los últimos años ha surgido la astroinformática. Se define como un subcampo dentro de la astronomía que provee de las metodologías de ciencias de datos necesarias para procesar ingentes cantidades de datos astonómicos entre las que se incluyen la organización, descripción, transformación y normalización de datos así como técnicas de visualización y extracción del conocimiento [31].

5.2.1. Metodología

Normalmente los astrónomos estudian el espacio a través de grandes cartografiados del espacio, las cuales sirven para recoger y medir datos de todos los objetos que son visibles en una región del espacio concreta. Esto se realiza de forma controlada, sistemática y reproducible. Hoy en día estas inspecciones producen terabytes e incluso petabytes de datos como pueden ser la *Sloan Digital Sky Survey* (SDSS), la *2-Micron All-Sky Survey* (2MASS) o *Cosmic Assembly Near-infrared Deep Extragalactic Legacy Survey* (CANDELS). De esta forma, se encuentran objetos de todo tipo: desde los más comunes, de los que hay millones, o muy raros, como pueden ser uno entre mil millones.

El gran crecimiento de datos de las últimas inspecciones permite generar un conocimiento mayor pero también se topa con muchas dificultades, principalmente debidas a la gestión de estos datos. La metodología de trabajo que se suele seguir se denomina *data-driven* (basada en datos), la cual difiere del método científico tradicionalmente usado en astrofísica y es más parecida a la usada en ciencia de datos. Como se puede ver en la Figura 5.4, las etapas de investigación y generación de hipótesis cambian: en primer lugar se realiza un tratamiento y extracción de características interesantes de los datos para después formular una hipótesis que se validará o refutará según los experimentos que se realicen. Este proceso no es en cascada sino que es iterativo e incremental (como se expuso en la Sección 2.3).

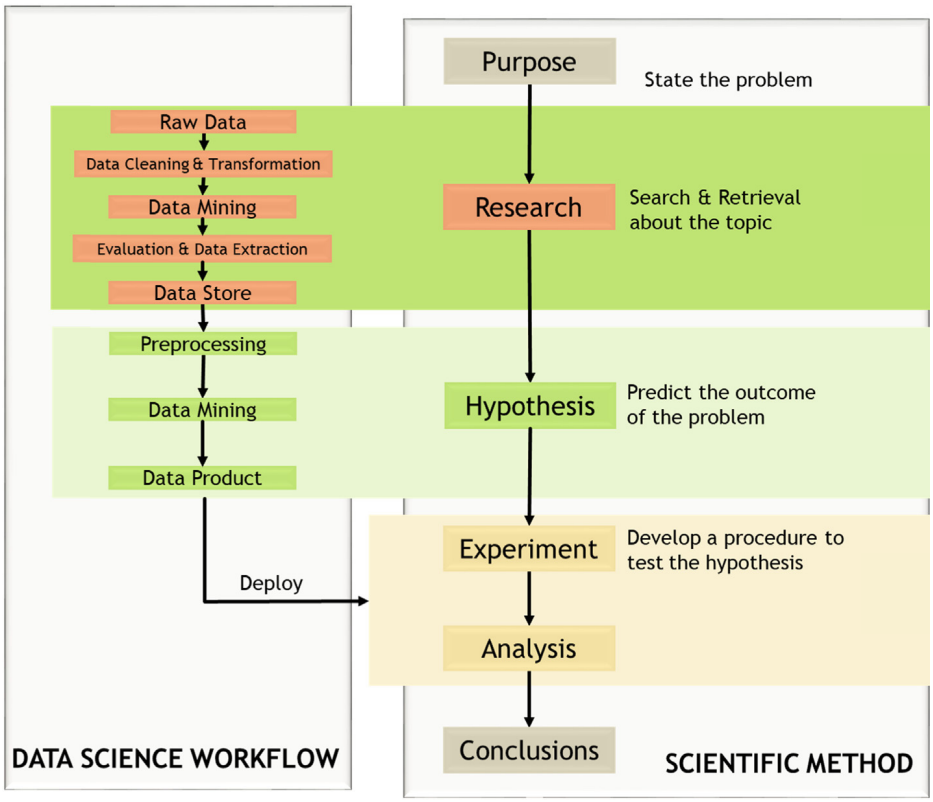


Figura 5.4: Diferencia entre el método científico (derecha) y la metodología basada en datos (izquierda). Adaptado de [32]

5.2.2. Formato FITS

El formato *Flexible Image Transport System* (FITS) [33] es un estándar libre que define un formato de archivo digital para el almacenamiento, transmisión y procesado de datos. Es el formato más usado para imágenes digitales en astronomía, campo específico para el que fue diseñado. Se empezó a desarrollar en 1981 por la NASA y la *International Astronomical Union* y actualmente se encuentra en la versión 4.0, de julio de 2016. El formato de archivo es `.fits`, `.fit` o `.fts`.

Los datos contenidos en este formato se diferencian en dos partes: la cabecera y los datos.

- **Cabecera**: contiene metadatos de la imagen. Consiste en una secuencia de caracteres ASCII en las que cada línea son pares clave-valor y termina siempre con la palabra reservada `END`. Hay ciertas claves obligatorias para que los datos sean interpretables, las cuales son:
  - `SIMPLE`: `T` si es un archivo simple FITS, `F` en caso contrario.
  - `BITPIX`: número de bits por palabra en los datos.
  - `NAXIS`: número de ejes del array.
- **Datos**: es la sección donde realmente se encuentra la imagen en forma de arrays n-dimensionales, donde el número de dimensiones lo especifica el parámetro `NAXIS` (normalmente serán 2 o 3).

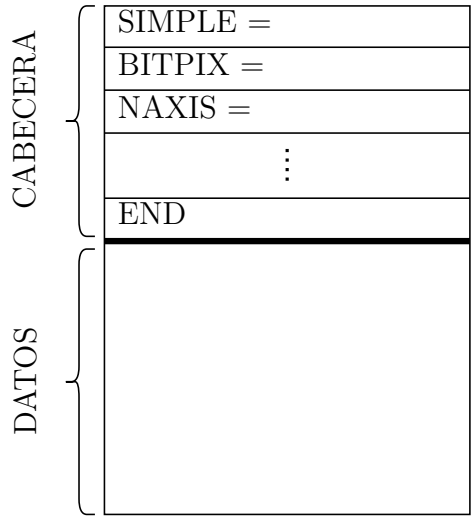


Figura 5.5: Formato de un archivo FITS

Es posible concatenar varias imágenes en un mismo archivo FITS, uniendo la cabecera y datos de cada una de ellas. Con ello se consigue compartir de forma más rápida un conjunto de observaciones.

Este tipo de archivos puede ser leído por programas de uso común como GIMP o Adobe Photoshop, siempre y cuando solamente haya únicamente una imagen y estén en formato simple. No obstante, hay otros programas especializados como SAOImageDS9 que nos dan mucha más información y posibilidades para su tratamiento. Respecto a la lectura en lenguajes de programación, existen librerías para C, FORTRAN, Perl, PDL, Python e IDL.

SAOImageDS9

SAOImageDS9 (comúnmente llamado DS9) [34] es una aplicación multiplataforma para visualizar y tratar imágenes astronómicas programado en C++ y que usa la librería gráfica Tk. No es solamente un programa de visualización de imágenes, sino que también permite realizar transformaciones (como pueden ser logarítmicas o el argumento seno hiperbólico) o aplicar distintos tipos de filtros de color. En la Figura 5.6 se puede ver una captura de su interfaz de usuario.

Para ver el efecto de las distintas escalas y filtros, se ha cogido como ejemplo la Nebulosa del Águila tomada por el telescopio Hubble. Originalmente (Figura 5.7a), no se aprecia prácticamente ninguna forma, parece un cielo totalmente negro. En cambio, si aplicamos un filtro logarítmico (Figura 5.7b) se aprecian varias columnas de gas, que no se veían sin esta escala. Además, se les puede añadir un color siguiendo un filtro, en este caso el *cool* (colores fríos) de DS9 (Figura 5.7c). Si se les aplica distintos filtros de colores como RGB (Figura 5.7d) se puede llegar a conseguir una imagen más artística, siendo realmente una combinación de tres imágenes con un filtro cada una.

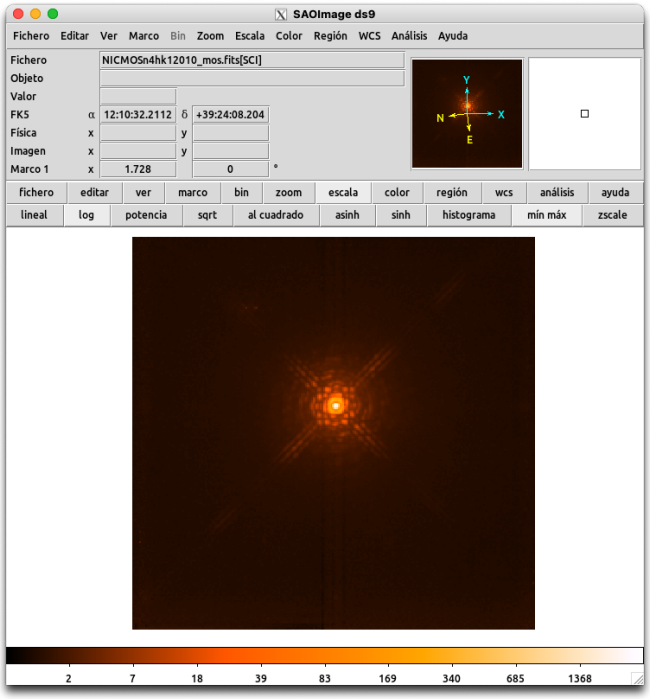
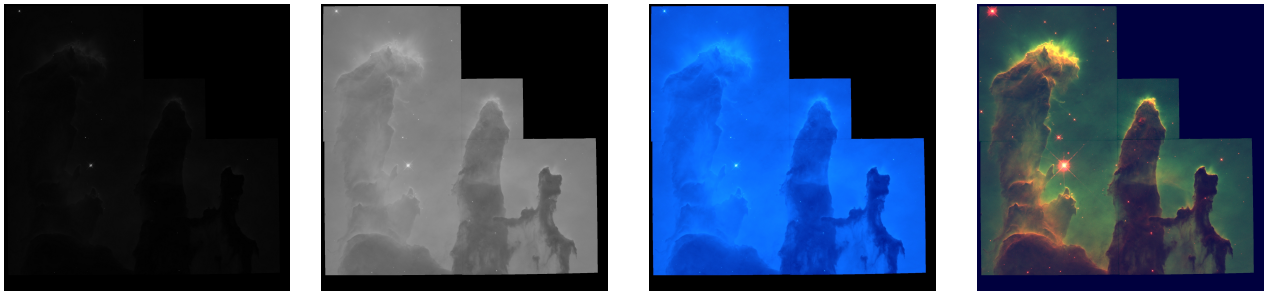


Figura 5.6: Pantalla gráfica de DS9



(a) Escala lineal (b) Escala logarítmica (c) Escala log + *cool* (d) Escala log + RGB

Figura 5.7: Distintas escalas y filtros aplicadas a la imagen de la Nebulosa del Águila tomada por la cámara WFPC2 del HST. Archivo FITS obtenido de [35]

astropy

Existen varias librerías para el tratamiento de imágenes astronómicas dependiendo del lenguaje de programación que se vaya a utilizar. En el caso de este trabajo, al usar Python la librería para la lectura y manipulación de los archivos FITS va a ser *astropy* [36], [37].

Para leer un archivo FITS se puede realizar de forma sencilla con el siguiente fragmento de código:

```
from astropy.io import fits

hdu = fits.open(path)
img = hdu[0].data
```

Como en un mismo archivo FITS puede haber más de una imagen, el resultado es un array. En nuestro caso cada archivo contiene únicamente una galaxia, por lo que basta con acceder al primer elemento. Como se explicó anteriormente, el formato consta de dos partes, cabecera y datos, a los que se puede acceder a través de los atributos `header` y `data` respectivamente.

5.3. Euclid

Euclid consiste en un telescopio espacial desarrollado por la Agencia Espacial Europea (ESA) cuya misión principal es mejorar la comprensión sobre la energía oscura y la materia oscura, midiendo la aceleración del Universo. Para conseguirlo, va a realizar un cartografiado continuo del cielo, capturando información tanto de carácter fotométrico como espectral (ver Figura 5.3). Así se obtendrá la posición de las galaxias y otros objetos astronómicos, pudiendo medir las variaciones con respecto a donde se pensaba que se encontraba dicho objeto. Como la energía oscura contribuye a la aceleración de la expansión del Universo, se puede cuantificar a través de estas variaciones.

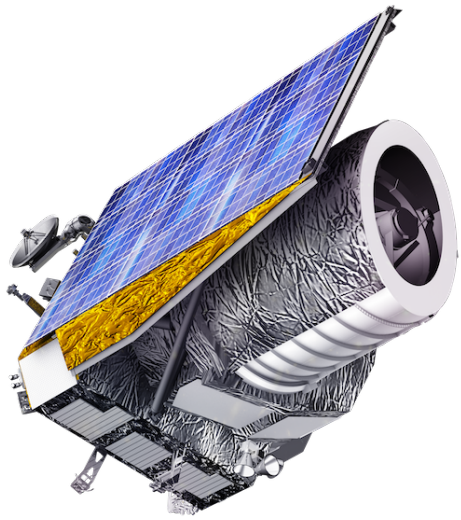


Figura 5.8: Representación del telescopio espacial Euclid. Obtenido de [38]

Este proyecto es una misión de investigación de la ESA de tamaño medio (*M-class*), con un presupuesto aproximado de 500 millones de euros. Se empezó a desarrollar hace 10 años y se tenía pensado enviar en 2023 en un cohete Soyuz ST 2-1b pero por culpa de la reciente invasión de Rusia a Ucrania se ha tenido que retrasar hasta 2024, al cambiar la lanzadera por un Ariane 62 (con las modificaciones técnicas que supone).

En el marco de la colaboración Euclid, uno de los objetivos secundarios que han surgido es el estudio de la morfología de galaxias, en el que participan investigadores y estudiantes de diversos puntos de Europa, como la Universidad de Nápoles, el Instituto de Ciencias del Espacio de Portugal o el Instituto de Astrofísica de Canarias. En este grupo de trabajo es donde participan tanto

Fernando Buitrago como Benjamín Sahelices, creando modelos artificiales para la clasificación morfológica de galaxias distantes usando únicamente la información fotométrica (imágenes en el rango visible).

Una galaxia se puede definir como una agrupación de planetas y estrellas principalmente, las cuales están unidas gravitacionalmente. Según la forma con la que se observen, las galaxias se pueden clasificar en diversos tipos. Dado que es un esquema puramente observacional, es una clasificación subjetiva, según como se vea por el ojo humano. La primera clasificación corresponde con los estudios realizados por E. Hubble [39], [40] el cual clasificó las galaxias en tres tipos principalmente según el ciclo de vida de las mismas (Figura 5.9).

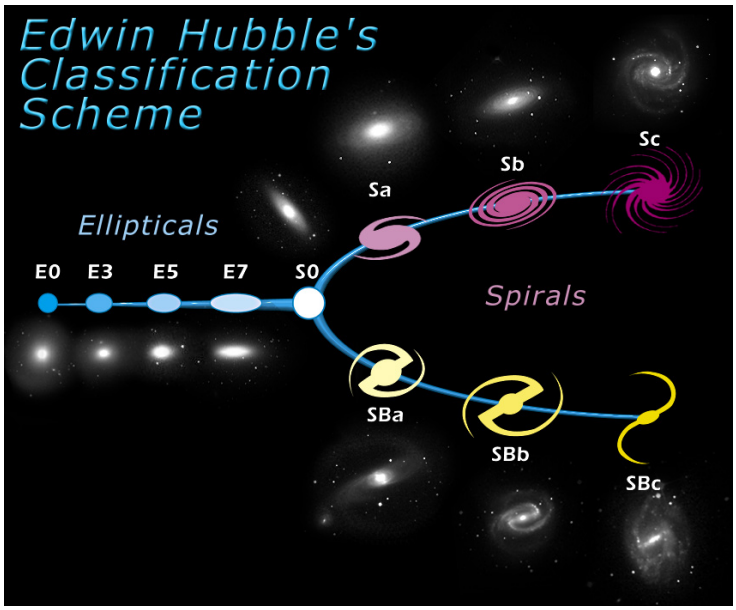


Figura 5.9: Esquema de clasificación propuesto por E. Hubble en [39]. Imagen de dominio público

- **Elípticas**, también conocidas como *early-type* dado que se pensaba que eran las primeras formas en las que se manifiesta una galaxia. La estructura que muestran es escasa, con una distribución bastante uniforme de estrellas por toda su extensión en forma de elipse. Según lo ovaladas que sean, se clasifican con siete números, de E0 (más circular) a E7 (más ovalada).
- **Lenticulares**, se consideran de transición desde las elípticas a las espirales. Se caracterizan por tener forma de disco pero con una concentración de estrellas en su centro muy importante y una extensa envoltura.
- **Espirales**, también conocidas como *late-type* dado que son las más jóvenes, tienen brazos espirales bien marcados (formados también de estrellas y materia interestelar) con forma de disco. Tienen un bulbo central formado por las estrellas más viejas, en una gran concentración. Dependiendo de la forma que tenga la parte central, se pueden dividir en barradas, si tiene forma de barra o regulares (si no lo tiene). Según lo disperso que tenga los brazos, se clasifican de la letra a (más apretados) a la c (más dispersos).

En la actualidad se considera este esquema como demasiado simple, pero las ideas fundamentales son la base de los que se han desarrollado con posterioridad. El usado en este trabajo es una modificación, que se explicará más en detalle en el Capítulo 6.

Saber la morfología de las galaxias es útil dado que da información de los parámetros de órbita y la historia de su generación. También está relacionada con el entorno cercano de la galaxia, ya que las interacciones con otras galaxias como las mareas, los choques y las fusiones pueden cambiar la forma de la galaxia.

5.4. Generación sintética

La segunda parte de este trabajo trata sobre datos generados de forma sintética, con simulaciones hidrodinámicas. Estas simulaciones consisten en definir un cubo e introducir una serie de partículas para ver cómo evolucionan e interaccionan entre ellas. De esta forma se consigue simular el comportamiento de una parte del universo y tener una cantidad de información mayor.

Un ejemplo de simulación es NewHorizon [41], donde se define un volumen de aproximadamente 16 Mpc^3 (megapársecs cúbicos, $16\text{Mpc}^3 \simeq 4.7 \cdot 10^{68}\text{m}^3$) con una resolución de 34pc. Se han usado modelos *state-of-the-art* físicos para emular galaxias en sus entornos cosmológicos. Esto incluye modelado del enfriamiento de gases, formación de estrellas, retroalimentación de estrellas masivas y agujeros negros supermasivos. En la Figura 5.10 se puede observar ejemplos de imágenes captadas de esta simulación.

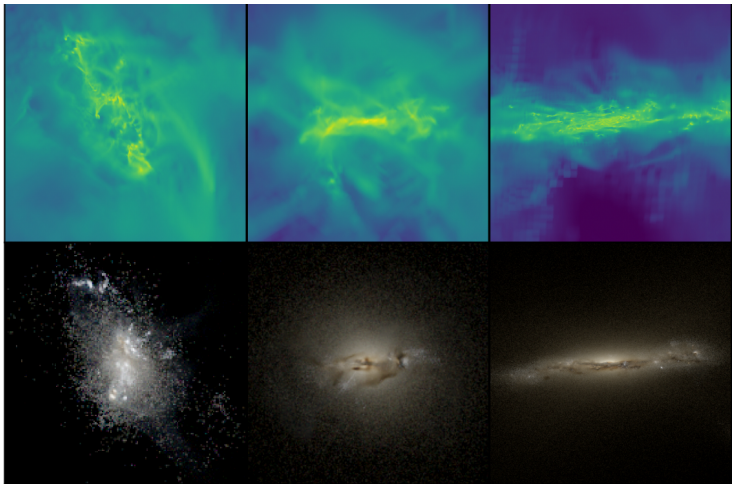


Figura 5.10: Evolución de la formación de una galaxia en NewHorizon. La parte superior muestra las proyecciones de densidades de gas hidrógeno y la inferior cómo se vería por parte de un telescopio aplicando color. Obtenido de [41]

De esta forma se consigue obtener información de las interacciones de galaxias, es decir, qué ocurre cuando dos galaxias "chocan" entre ellas. Este fenómeno es muy complicado de captar en la realidad, lo que hace más interesante este tipo de simulaciones.

Capítulo 6

Conjunto de datos de Euclid

Esta primera parte del trabajo se enmarca en la colaboración Euclid, como se ha descrito en la Sección 5.3. El objetivo principal es caracterizar ciertas cualidades morfológicas de las galaxias de manera automática, dado que en un futuro cercano, cuando el telescopio Euclid sea lanzado, la cantidad de datos va a ser inabarcable por el ser humano dado que este telescopio va a realizar una cartografía sistemática de parte del universo y tenemos que preparar los algoritmos de identificación y análisis de galaxias existentes para ese momento.

La aproximación utilizada se basa en usar redes neuronales convolucionales para realizar esta clasificación, dado que en trabajos anteriores ha dado muy buenos resultados [42], [43], [44], [45], [46], [47]. Como se verá en la siguiente sección no consiste exactamente en el mismo problema de otros trabajos pero es una tarea parecida en cierta medida.

6.1. Descripción del conjunto de datos

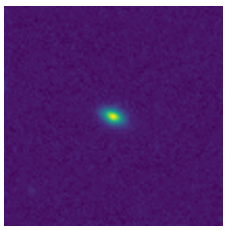
El objetivo final es clasificar imágenes tomadas por el telescopio Euclid mimetizando el comportamiento humano, por lo que sería ideal tener un conjunto de datos de este tipo. Sin embargo, esto no es posible dado que aún no se ha lanzado (es un telescopio espacial), por lo que en su lugar se han usado imágenes del HST transformadas de tal forma que se simulen los detectores de Euclid. Se ha utilizado estas imágenes principalmente porque ambos telescopios utilizan el mismo tamaño de píxel, además de que al ser también espacial se minimiza mucho el ruido (los telescopios terrestres tienen un ruido muy superior). Aparte de las diferencias vistas entre los detectores de los dos telescopios, cabe destacar también la diferencia de canales (HST tiene 3 y Euclid 1 en el rango visible del espectro electromagnético) y el objetivo de cada proyecto: HST toma imágenes multipropósito mientras que Euclid cartografía por igual todo el cielo. La fuente de los datos ha sido *Galaxy Zoo: Hubble* [48], tomadas por la cámara ACS del Hubble junto a otros datos de inspecciones anteriores.

El conjunto de datos total contiene 23539 observaciones de tamaño 200×200 píxeles en formato FITS y con un solo canal. Cada imagen contiene una sola galaxia y puede no estar centrada pero

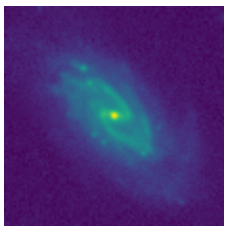
se ve en su totalidad. Estos datos han sido preprocesados usando un escalado logarítmico (en base 10) y un filtro de color *viridis*.

En el marco de la colaboración, se decidió empezar por las tareas denominadas T01, T12, T02, T03 y T04; para comprobar los resultados obtenidos por distintos métodos de *deep learning*. Las 5 cuestiones que se trataron fueron:

- **T01**: ¿La galaxia es lisa y redondeada, sin señales de un disco? Originalmente era un problema multiclase pero se decidió tratarlo como binario dado que la categoría estrella no es útil desde el punto de vista de esta colaboración, por lo que únicamente se quiere determinar si tiene forma de disco o no.



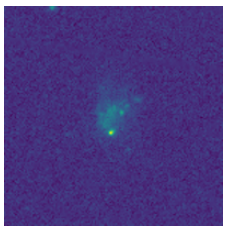
(a) *A1 - smooth*



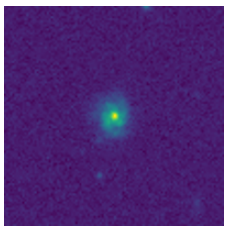
(b) *A2 - features or disk*

Figura 6.1: Ejemplos de imágenes para la tarea T01

- **T12**: ¿La galaxia tiene una apariencia mayoritariamente grumosa?



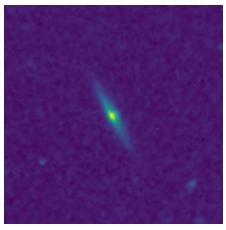
(a) *A1 - yes*



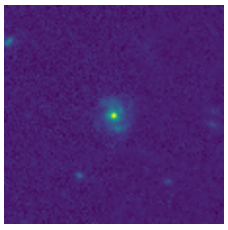
(b) *A2 - no*

Figura 6.2: Ejemplos de imágenes para la tarea T12

- **T02**: ¿Puede ser un disco visto de canto?



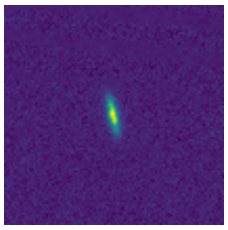
(a) *A1 - yes*



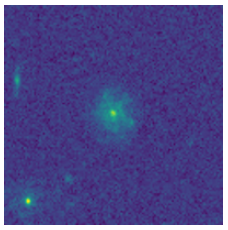
(b) *A2 - no*

Figura 6.3: Ejemplos de imágenes para la tarea T02

- **T03**: ¿Se percibe una forma de barra en el centro de la galaxia?



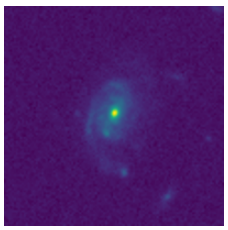
(a) *A1 - bar*



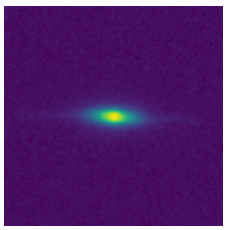
(b) *A2 - no bar*

Figura 6.4: Ejemplos de imágenes para la tarea T03

- **T04**: ¿Hay algún signo de brazos de espiral?



(a) *A1 - spiral*



(b) *A2 - no spiral*

Figura 6.5: Ejemplos de imágenes para la tarea T04

Estas tareas se enmarcan en un contexto más grande (con más tareas) que se encuentran definido en el árbol de decisión (Figura 6.6). Se organiza en cuatro niveles dependiendo de la profundidad de cada cuestión: en el nivel 1 se encuentran las preguntas que no dependen de ninguna respuesta, en el nivel 2 las que dependen de una respuesta y así sucesivamente. En total hay 18 cuestiones con diferente número de respuestas: en todos los casos es un problema de clasificación sobre distintos aspectos de la morfología de las galaxias, en algunos es binaria y en otros multiclase.

El conjunto de datos ha sido etiquetado por distintos astrónomos, los cuales han respondido a estas preguntas. No todas las galaxias tienen el mismo número de evaluaciones y dentro de una misma imagen algunas preguntas pueden tener más evaluaciones que otras. Como se puede ver en la Figura 6.7 difieren bastante unas tareas de otras: T01 es la que tiene un número mayor de galaxias con más clasificadores, seguida de T12 (por bastante diferencia), después T02, siendo las dos últimas las que menos tienen.

En términos formales, estamos buscando una hipótesis $h_i : X \to Y_i$ donde X es la variable independiente y en este caso está formada por matrices bidimensionales $p \times p$, $p = 200$ de valores reales. Para cada tarea i, tenemos un conjunto de respuestas $Y_i = \{y_{i1}, y_{i2}, ..., y_{ik}\}$, $i \in \{\text{T01, T12, T02,...}\}$ donde cada $y_{ij} \in [0, 1]$ y corresponde a la frecuencia con la que se ha observado la característica j. Es decir, dados los distintos evaluadores, los cuales han respondido a cada característica j de la

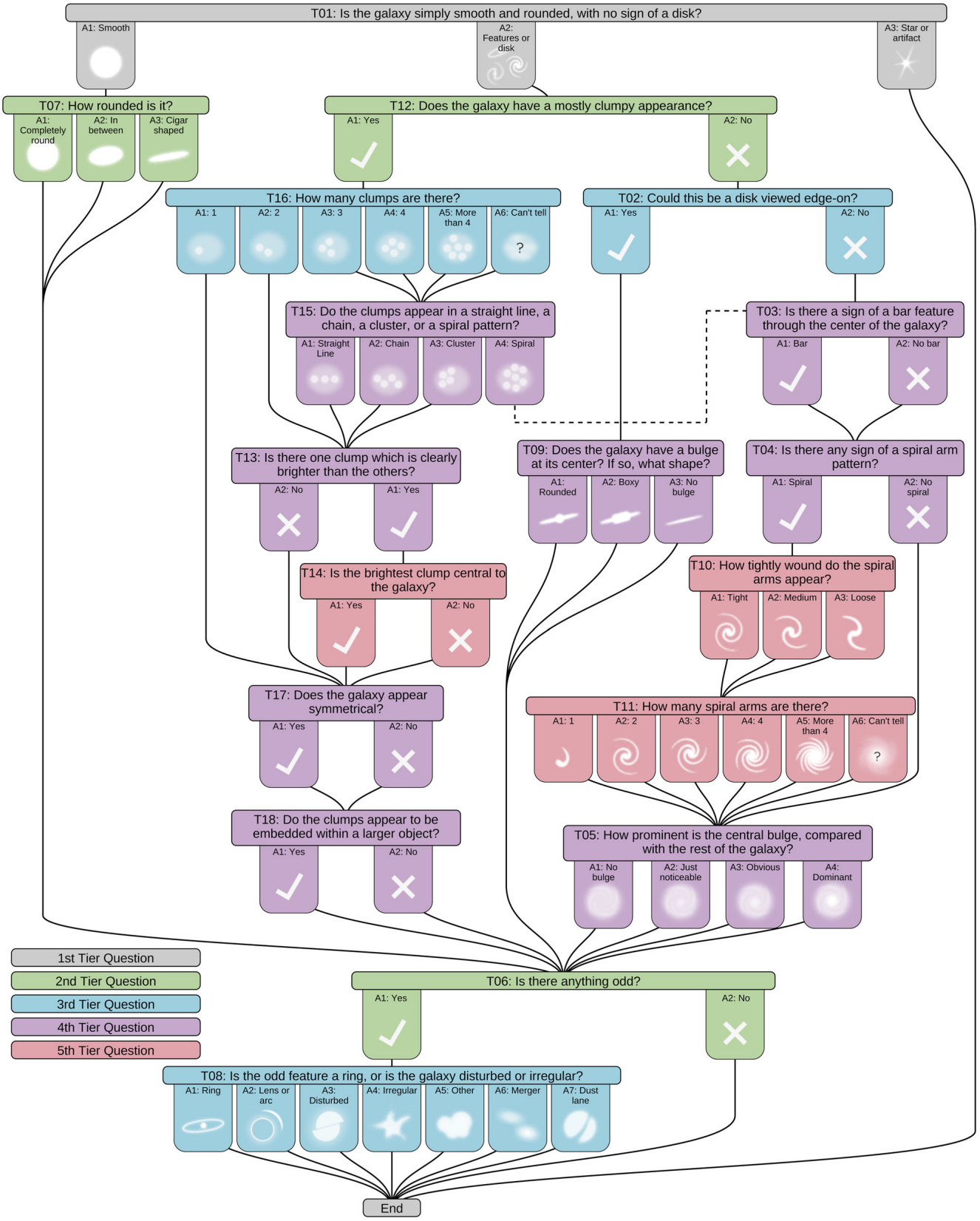


Figura 6.6: Árbol de decisión de las diferentes tareas a tratar, obtenido de [48]

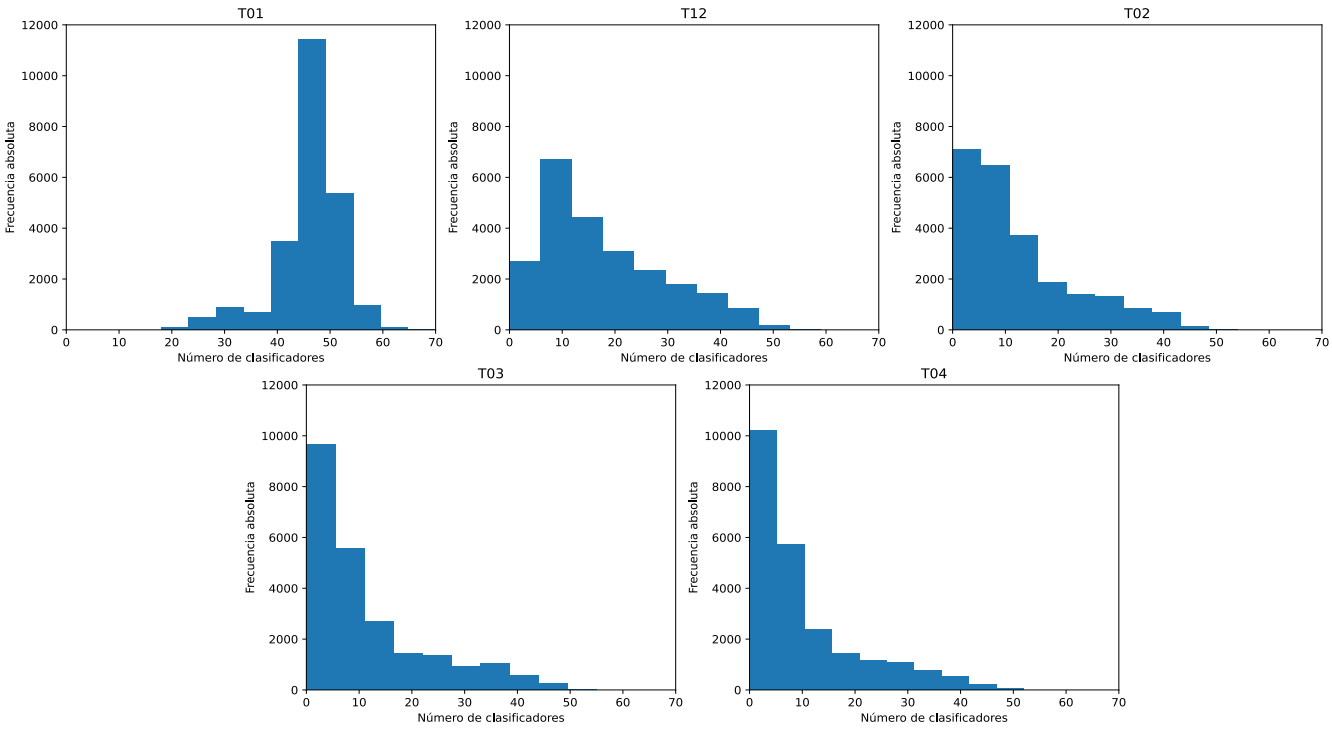


Figura 6.7: Número de clasificadores por tarea

tarea i con 1 o 0 (si está presente o no, respectivamente) y se realiza una media de todas ellas para así tener la frecuencia de cada característica. En cada pregunta las características son excluyentes, es decir, si se responde que hay una de ellas las demás tendrán una respuesta negativa (por ejemplo, en la pregunta T03 las galaxias tienen o no una forma de barra en el centro, pero no ambas) y al ser frecuencias $\sum_j y_{tj} = 1$, para cada problema t. Por tanto cada instancia se supondrá de la clase con mayor frecuencia.

Para cada una de las respuestas, se ha realizado una ponderación según los evaluadores que la han clasificado: para cada evaluador se asigna un peso dependiendo de su fiabilidad. Dicho peso se ha calculado según cuánto se alejen las clasificaciones de dicha persona con respecto a la media general. Sea $y_{\mathrm{T},i,k,e}$ la respuesta k de un evaluador e sobre una galaxia i dada una tarea T, la frecuencia de dicha respuesta sobre la instancia se calcularía como:

$$y_{\mathrm{T},i,k} = \frac{\sum_e w_e \cdot y_{\mathrm{T},i,k,e}}{\sum_k \sum_e w_e \cdot y_{\mathrm{T},i,k,e}} \tag{6.1}$$

donde w_e es la ponderación que se le da a cada evaluador e.

El conjunto de datos está dividido originalmente en dos partes, una parte para usar en entrenamiento y otra para test, con la que no se entrenará y solo sirve para comprobar si los modelos generalizan correctamente. El primero está formado por 19962 observaciones y el segundo por 3577.

Para muchas galaxias el número de clasificaciones es bajo, por lo que es más probable que

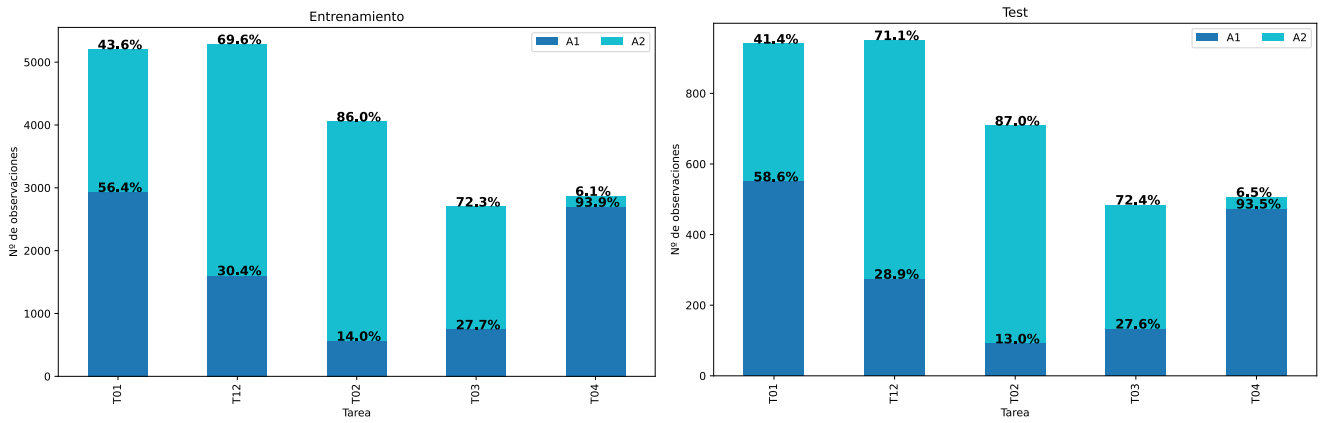
dichas clasificaciones sean inexactas o incorrectas. Por ello, en el marco de la colaboración se ha decidido tomar dos subconjuntos de datos para cada uno de los problemas:

- ***Clean dataset***: solo se seleccionan las observaciones que tengan al menos 20 evaluaciones ($n_{\text{votes}} \geq 20$). De esta forma se supone que las estimaciones son más correctas al haber estado expuestas a un número mayor de personas. Además, dependiendo de la tarea a tratar se seleccionan únicamente las galaxias con mayor frecuencia, con los siguientes criterios para tratar de balancear las clases:

  - T01: se seleccionan únicamente las dos primeras clases, la tercera (*A3 - star or artifact*) no se considera relevante. Para las dos clases, únicamente se toman los valores de cada una mayores de 0.8 ($y_{\text{T01,1}} > 0.8$ y $y_{\text{T01,2}} > 0.8$). De esta forma, tenemos en la respuesta *A1 - smooth* 2935 instancias de entrenamiento y 551 de test y en la respuesta *A2 - features or disk* 2271 y 390 respectivamente.
  - T12: para la primera respuesta se toman los valores mayores que 0.5 ($y_{\text{T12,1}} > 0.5$) y en la segunda los mayores de 0.8 ($y_{\text{T12,2}} > 0.8$). Para la respuesta *A1 - yes* quedan 1610 instancias de entrenamiento y 275 de test y para *A2 - no* tenemos 3678 y 676, respectivamente.
  - T02: para la primera respuesta se toman los valores mayores que 0.5 ($y_{\text{T02,1}} > 0.5$) y en la segunda los mayores de 0.8 ($y_{\text{T02,2}} > 0.8$). Para la respuesta *A1 - yes* quedan 566 instancias de entrenamiento y 92 de test y para *A2 - no* tenemos 3491 y 617, respectivamente.
  - T03: para la primera respuesta se toman los valores mayores que 0.5 ($y_{\text{T03,1}} > 0.5$) y en la segunda los mayores de 0.8 ($y_{\text{T03,2}} > 0.8$). Para la respuesta *A1 - bar* quedan 749 instancias de entrenamiento y 134 de test y para *A2 - no bar* tenemos 1954 y 351, respectivamente.
  - T04: para la primera respuesta se toman los valores mayores que 0.8 ($y_{\text{T04,1}} > 0.8$) y en la segunda los mayores de 0.5 ($y_{\text{T04,2}} > 0.5$). Para la respuesta *A1 - spiral* quedan 2699 instancias de entrenamiento y 474 de test y para *A2 - no spiral* tenemos 174 y 33, respectivamente.

  Tanto en los conjuntos de entrenamiento como de test, para las 4 cuatro últimas tareas se comprueba que las clases están desbalanceadas, es decir, hay una clase que concentra muchas más instancias que la otra. Además, el número de instancias va disminuyendo conforme vamos bajando en el árbol de decisión de la Figura 6.6.

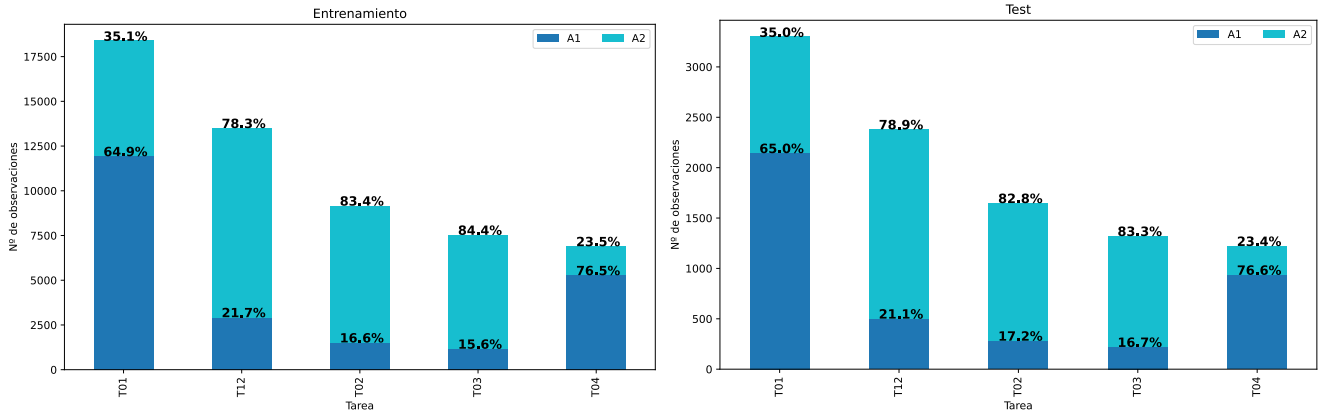
- ***All galaxies dataset***: se seleccionan las observaciones que tengan al menos 10 evaluaciones ($n_{\text{votes}} \geq 10$). En este caso para elegir los criterios se toman los que sean mayores de 0.5 para todas las clases de todos los problemas, quedando el número de instancias de entrenamiento y test de la siguiente forma:



(a) Distribución de clases de entrenamiento (b) Distribución de clases de test

Figura 6.8: Distribución de clases para entrenamiento y test de *clean dataset*

- T01: *A1* - *smooth* 11937 / 2145, *A2* - *features or disk* 6449 / 1153.
- T12: *A1* - *yes* 2934 / 502, *A2* - *no* 10576 / 1879.
- T02: *A1* - *yes* 1523 / 283, *A2* - *no* 7650 / 1366.
- T03: *A1* - *bar* 1173 / 221, *A2* - *no bar* 6367 / 1101.
- T04: *A1* - *spiral* 5285 / 937, *A2* - *no spiral* 1619 / 286.



(a) Distribución de clases de entrenamiento (b) Distribución de clases de test

Figura 6.9: Distribución de clases para entrenamiento y test de *all galaxies dataset*

Al igual que con el conjunto de datos anterior se verifica en la Figura 6.9 que las cuatro últimas clases están desbalanceadas tanto en entrenamiento como en test, aunque algunas en menor medida que en *clean dataset* como T02 o T04.

En la Figura 6.10 se muestra el esquema de la división en los distintos conjuntos de datos por tareas.

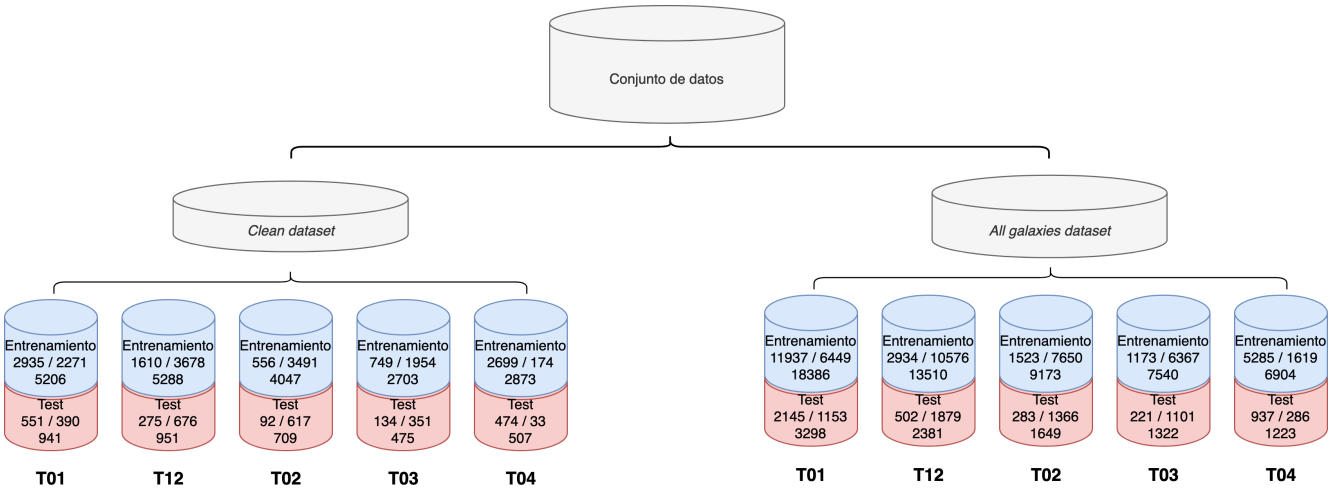


Figura 6.10: Esquema de la división en conjuntos, por tareas y partición en entrenamiento/validación y test. La primera fila de números en cada conjunto indica el número de elementos en cada una de las clases (A1 / A2) y la segunda la suma de las dos clases (número total de instancias)

6.2. Preprocesado y data augmentation

Sobre el preprocesado, en primer lugar se ha aplicado una escala logarítmica en base 10 a las observaciones. De esta forma se consigue aumentar el contraste de ciertos píxeles (debido a que el centro de las galaxias es muy luminoso y sus partes externas muy tenues) y así conseguir ver características que sin esta escala no se verían (ver Figura 5.7). Al estar en un solo canal, se ha aplicado un filtro de color "viridis" antes de guardar la imagen en formato PNG, el cual no sería necesario pero sirve para visualizar mejor los resultados. Además, en cada batch los datos se han normalizado aplicando estandarización, para conseguir media 0 y varianza 1.

Para aumentar de forma artificial el conjunto de datos y de esta forma tener más observaciones "diferentes" de las que el modelo pueda aprender se van a usar técnicas de *data augmentation*. Es una técnica ampliamente usada en la actualidad dado que los conjuntos de datos son finitos y, en general, las redes neuronales (y la mayor parte de los algoritmos de aprendizaje) aprenden mejor cuánta más información tengan dado que tienen que ajustar un número muy grande de parámetros.

Hay que tener cuidado con las técnicas que se usen dado que pueden no valer todas en todos los contextos. En el campo que nos ocupa, es fácil comprobar que, por ejemplo, al girar una galaxia no cambian ninguna de sus propiedades morfológicas. En cambio, si aumentamos la imagen de más es posible que parte de la galaxia quede fuera, por lo que hay que tener cuidado con el *zoom* que se aplica dado que puede recortar información valiosa. Las técnicas que se van a usar han sido extraídas de trabajos anteriores relacionados en los que han dado buenos resultados, por lo que se comprueba que son aptas para nuestro campo de aplicación [42], [43], [44], [46], [49], [47].

- **Rotaciones**: con probabilidad uniforme se han rotado las imágenes 0^{o}, 90^{o}, 180^{o} y 270^{o} en

sentido horario (probabilidad $\frac{1}{4}$ de cada tipo). Aplicar esta transformación no cambia las características visibles de la galaxia, tan solo su orientación.

- **Voltear** respecto al eje vertical. Nuevamente esta transformación no cambia la morfología de la galaxia.
- **Ampliación** con zoom entre 0.75x (más alejado) y 1.3x (más cercano). Esta transformación podría dar lugar a recortes en la imagen pero con los valores usados no ocurre dado que las galaxias están centradas y no ocupan la totalidad del marco original.
- **Ruido gausiano**, de tal forma que sea tan pequeño que no afecte a las características visuales de la galaxia. Por tanto, a cada píxel se le ha sumado un valor $x \sim N(0, \sigma)$, con $\sigma = 0.002$.

Estas transformaciones se han aplicado únicamente al conjunto de entrenamiento y no al de test. Si las aplicáramos al de test tendríamos varias imágenes que no serían totalmente independientes entre sí, dado que aunque se apliquen estas operaciones la imagen guarda relación con la original. Por tanto cometeríamos un error al calcular las métricas sobre el conjunto de test, por lo que es una mala práctica.

6.3. Cuestiones a estudiar

Dentro del marco de la colaboración, para cada una de las tareas descritas y para cada uno de los conjuntos de datos, se plantean las siguientes dos cuestiones a tratar:

- **Q1 - Rendimiento**: dado que es un problema de clasificación binario, para ver cómo funcionan los modelos sobre los conjuntos se van a calcular las matrices de confusión, la tasa de acierto (*accuracy*), el *recall*, la precisión (*precision*) y el *F1 score*.

  La matriz de confusión (Tabla 6.1) es una disposición concreta en forma de tabla para ver gráficamente el rendimiento de un algoritmo de clasificación (en este caso binario dado que tenemos dos clases). Denotamos con TP a los verdaderos positivos (instancias positivas que se clasifican como tal), TN a los verdaderos negativos (observaciones negativas que se clasifican como tal), FP a los falsos positivos (instancias negativas que se clasifican como negativas) y FN a los falsos negativos (observaciones positivas que se clasifican como negativas).

		Predicción	
		0	**1**
Real	**0**	TN	FP
	1	FN	TP

Tabla 6.1: Matriz de confusión en un problema de clasificación binario

Las métricas descritas se calculan como:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{6.2}$$

$$recall = \frac{TP}{TP + FN} \tag{6.3}$$

$$precision = \frac{TP}{TP + FP} \tag{6.4}$$

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{6.5}$$

El *accuracy* nos da información sobre el ajuste del modelo, pero en ciertas ocasiones puede no ser la mejor métrica. Por ejemplo, imaginemos que queremos crear un método de clasificación para detectar a los terroristas de entre la población general. Típicamente, el mayor porcentaje de las personas no son terroristas, pongamos que un 99 % no lo son, mientras el 1 % restante sí lo son. Si usáramos solo la tasa de acierto como medida, clasificando a todo el mundo como no terrorista conseguiríamos un acierto del 99 %, tan solo fallaríamos en el 1 %. Sin embargo este método de clasificación no tiene ninguna utilidad, ya que no aporta información alguna.

Por ello, es útil fijarnos en el *recall*, la precisión y el valor F1. La primera de ellas corresponde a la fracción de instancias relevantes (TP y FN) que fueron predichas correctamente (TP). En el segundo caso, corresponde a la fracción de observaciones relevantes (TP) sobre todo el conjunto de instancias predichas (TP y FP). Por último, el valor F1 nos da una medida combinada de los dos anteriores, dando el mismo peso a ambas.

- **Q2 - Número de galaxias necesarias para obtener el máximo rendimiento**. Dado que muchas veces el conjunto de datos etiquetados es limitado, como se puede ver cuando descendemos en el árbol de decisión (Figura 6.6). Por ello es útil tener una medida del número de instancias necesarias para llegar al rendimiento que da el modelo que mejor ajusta a los datos, en términos de las métricas definidas.

  Para ello, se van a crear modelos con los parámetros óptimos encontrados en el anterior apartado usando de mil en mil observaciones. Es decir, se va a comenzar con un modelo de 1000 datos, después otro con 2000, ...

Capítulo 7

Resultados Euclid

Como se ha detallado en el capítulo anterior, el objetivo principal es crear una serie de modelos de *deep learning* para la clasificación morfológica de galaxias distantes. Para ello se usarán datos del HST que han sido modificados para simular las lentes del futuro telescopio Euclid. En esta sección se detallarán los experimentos que se han desarrollado para cada una de las dos cuestiones, con los dos conjuntos de datos propuestos y para cada una de las tareas descritas.

A la hora de realizar el entrenamiento en todos los casos se seguirá el mismo procedimiento. En primer lugar, con el conjunto de entrenamiento, se separará en dos usando *holdout*: $\frac{2}{3}$ para entrenar los parámetros y $\frac{1}{3}$ de validación, con la cual no se entrena, solo sirve para comprobar que sea razonable la fase anterior. Con el ajuste obtenido, se usará el conjunto de test únicamente para generar las métricas de generalización, dado que son instancias nuevas con las que la red no ha entrenado y desconoce. El resumen de la división de los conjuntos de datos junto con lo comentado en el Capítulo 6 se puede ver en la Figura 7.1.

7.1. Cuestión 1

Para abordar esta cuestión, en primer lugar se van a hacer una serie de experimentos para contrastar las siguientes hipótesis:

- **Ponderación**. Como se ha visto anteriormente, tenemos dos tipos de variables por cada respuesta (en cada tarea T): una de ellas está ponderada según si el evaluador es "fiable" y otra está sin ponderar.
- **Modelo**. Se han escogido una serie de modelos que han demostrado funcionar correctamente con problemas similares, tanto del ámbito de la astronomía como en otros ámbitos. Concretamente, se han seleccionado varias configuraciones de ResNet, EfficientNet y un modelo concreto para la clasificación morfológica de galaxias llamado GaMorNet [45] (basado en AlexNet).

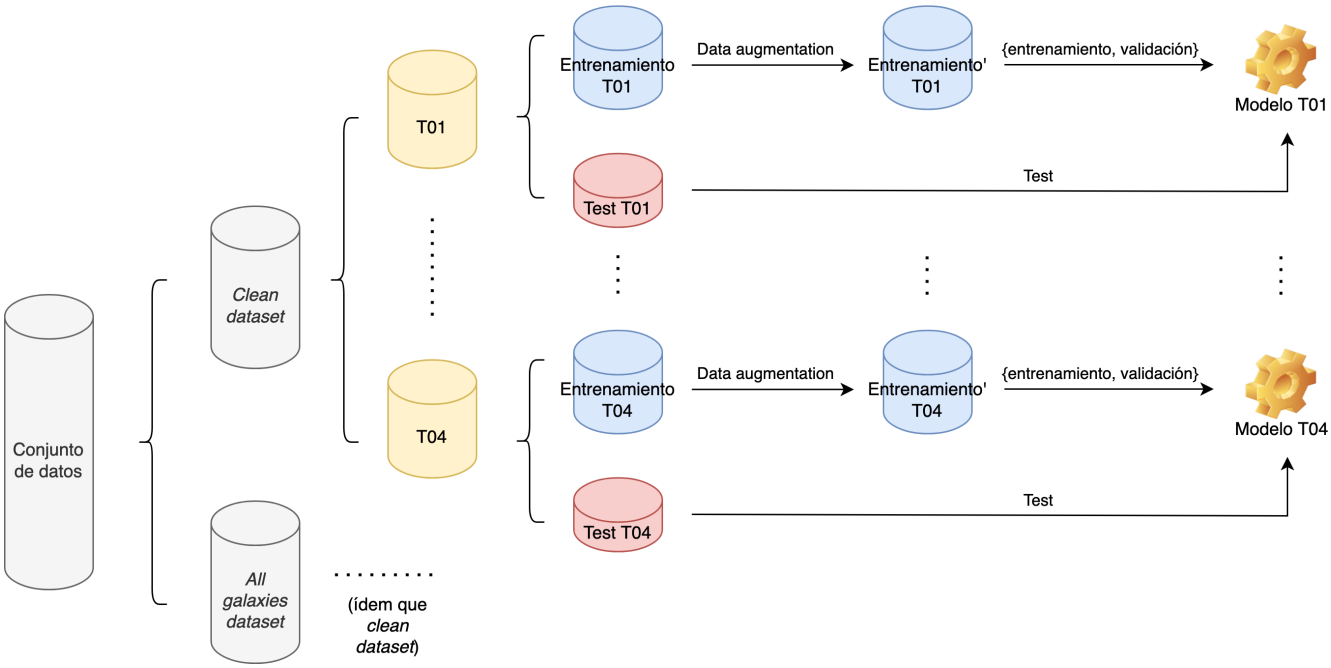


Figura 7.1: Esquema general de la división de los conjuntos de datos

Se han seleccionado distintas configuraciones atendiendo al número de parámetros, para comprobar si es necesario una red más profunda o menos. Por ello, de ResNet se han usado ResNet-18, ResNet-50 y ResNet-101; de EfficientNet los modelos EfficientNet-B0, EfficientNet-B1, EfficientNet-B4 y EfficientNet-B7.

- **Preentrenamiento**. La forma general de obtener la inicialización de los pesos de la red es hacerlo de forma aleatoria según una distribución estadística concreta. Sin embargo, también es posible obtenerlos habiendo entrenado otra red anteriormente y coger los pesos que nos da el resultado de dicho entrenamiento (conocido como *transfer learning*). Para cada configuración de red, PyTorch nos permite obtener el resultado de los parámetros después de haberlo entrenado con ImageNet.

  ImageNet [25] es una gran base de datos que contiene imágenes sobre objetos de la vida cotidiana clasificados en unas 200 categorías, como pueden ser perros, gatos, semáforos, autobuses... Preentrenando de esta forma teóricamente se consigue que el modelo ya sea capaz de distinguir formas y bordes.

Dado que es una batería de experimentos bastante extensa (la combinación de todos los modelos para un conjunto de datos resulta en 150 experimentos diferentes), en primer lugar se va a probar la primera de las tareas T01 con todos los modelos, tanto ponderados como no, únicamente para el primer conjunto de datos *clean dataset*. Se ha elegido este conjunto dado que es el más pequeño pero a la vez el más refinado, por lo que en principio debería dar los mejores resultados. En la Tabla 7.1 se muestra un resumen de los experimentos propuestos en esta primera fase para un modelo, en concreto ResNet-18.

Modelo	Preentrenado	Ponderado
ResNet-18	no	no
ResNet-18	no	sí
ResNet-18	sí	no
ResNet-18	sí	sí

Tabla 7.1: Resumen de la primera batería de experimentos para ResNet-18

A continuación, con los resultados de esta primera batería se tratarán las siguientes cuatro tareas. Como se presentó en la Sección 2.3, el enfoque que se va a seguir es iterativo, por lo que dependiendo de estos primeros resultados se seguirá por un camino u otro.

7.1.1. Regresión

Como se ha descrito en el Capítulo 6 la variable respuesta es continua en el intervalo $[0, 1]$ correspondiente a la frecuencia de cada clase. Para un primer modelo, se ha decidido realizar regresión, una por cada clase, dado que en trabajos anteriores aplicados a problemas de morfología de galaxias daban mejores resultados que haciendo clasificación directamente [47], [50]. Como función de pérdida se ha utilizado el error cuadrático medio (MSE) definido en la Sección 3.1.6.

El entrenamiento de los distintos modelos se realizó con un tamaño de batch y número de épocas diferente para cada uno de ellos, los cuales se resumen en la Tabla 7.2. El criterio para decidir estos hiperparámetros fue la capacidad computacional, dado que al tener 12GB de memoria gráfica usando modelos más profundos se producía un desbordamiento de la misma, por lo que se necesitaba bajar el tamaño del batch. Para elegir el número de épocas, se ajustó para que cada modelo tardara en entrenar y validar entre 2 y 3 horas, por lo que modelos con más capas tendrán un número menor de épocas.

	ResNet-18	ResNet-50	ResNet-101	EfficientNet-B0	EfficientNet-B1	EfficientNet-B4	EfficientNet-B7	GaMorNet
Tamaño del batch	64	64	50	64	64	45	16	64
Número de épocas	300	200	100	240	180	120	100	300

Tabla 7.2: Tamaño del batch y número de épocas usado para entrenar cada arquitectura de red usando regresión

Este segundo criterio puede provocar que modelos más complicados no lleguen a aprender tan rápido (necesiten más épocas) dado que el número de parámetros es muy superior. En cambio, como se verá a continuación, esto no ha sido un problema ya que con el número de épocas propuesto todos los modelos han convergido (se ha estabilizado su función de pérdida). El optimizador usado para todos ellos es Adam, excepto para GaMorNet que ha sido SGD dado que así lo sugieren sus autores [45]. A la hora de entrenar los modelos se han usado varias de las mejoras descritas en el Capítulo 3 como la política *1cycle*, la normalización del batch después de cada capa convolucional

y aplicar dropout con probabilidad 0.5 en las últimas capas (*fully connected*).

En primer lugar, se observan las gráficas de pérdida tanto para la etapa de entrenamiento como la de validación para comprobar si hay sobreajuste. Un indicador de este problema sería que la función de pérdida para el conjunto de entrenamiento sería menor que para el conjunto de validación, dado que aprende "de memoria" las instancias del primero de ellos. Para empezar, únicamente se ha analizado la primera de las tareas con *clean dataset* para comprobar si hay diferencias entre aplicar ponderación a las frecuencias según el evaluador o no.

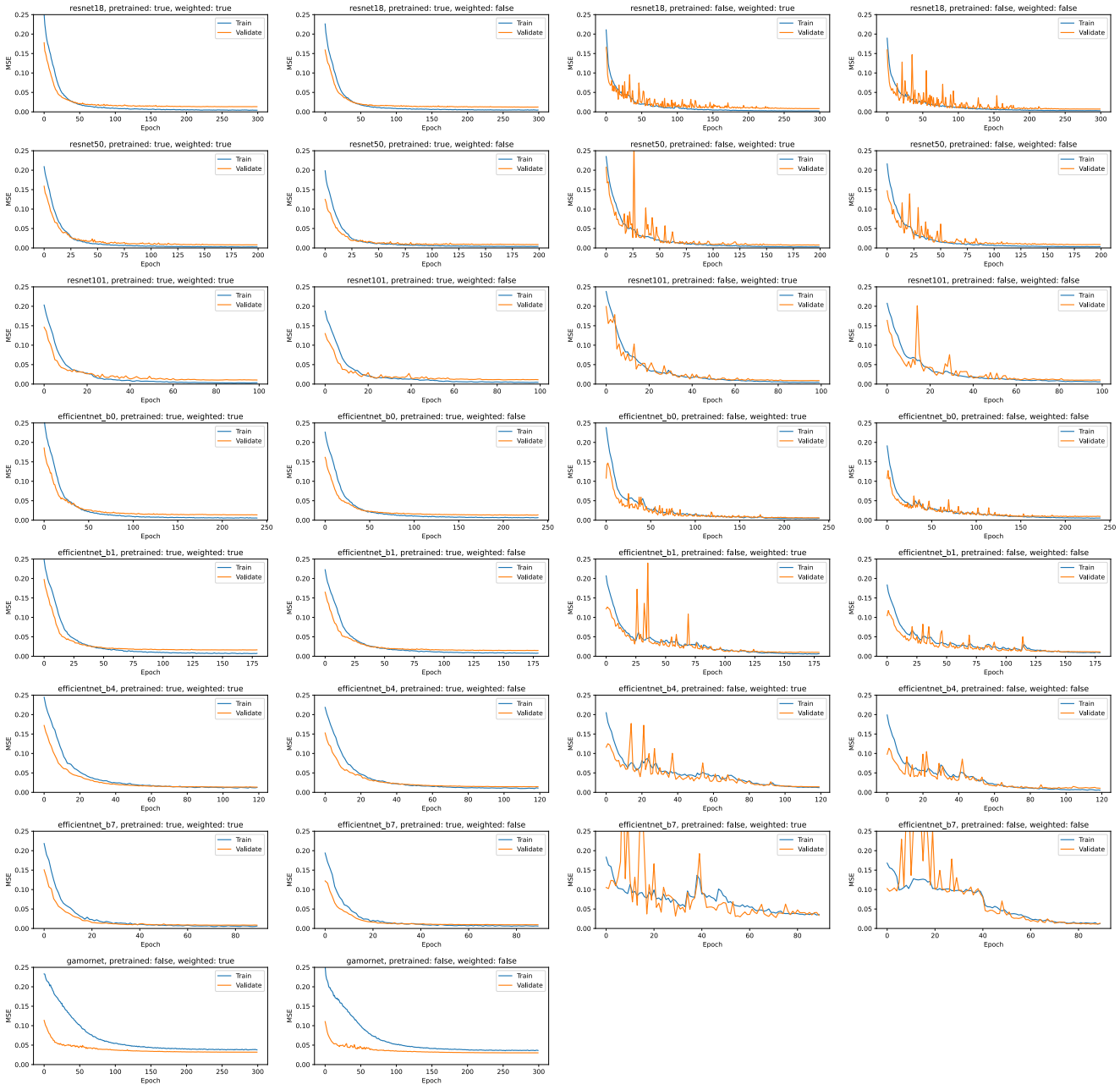


Figura 7.2: Evolución de la función de pérdida de entrenamiento y validación en el primer experimento con T01 y *clean dataset*. Cada fila corresponde con un modelo en todas las combinaciones posibles de preentrenamiento y ponderación

De la Figura 7.2 se pueden extraer varias conclusiones:

- Los modelos preentrenados se estabilizan más rápidamente
- No hay diferencias entre usar la respuesta ponderada o no o entre usar preentrenamiento o no: todos llegan más o menos al mismo valor del MSE, aproximadamente 0.015 en validación y 0.01 en entrenamiento.
- Prácticamente no hay sobreajuste en ninguno de los modelos: se ha puesto un número alto de iteraciones para ver el comportamiento de entrenamiento y validación mucho más tarde de que la pérdida se haya estabilizado. Prácticamente en todos ambas curvas van a la par, sin grandes diferencias

Acto seguido, se ha medido el *accuracy* para todos ellos sobre el conjunto de test. Para transformar la regresión a clasificación, se ha escogido la clase con el valor máximo, dado que como son frecuencias la suma por clase debe de ser 1 (Ecuación 6.1). Por tanto, la clase asignada para una observación i sobre una tarea T concreta se obtiene como:

$$y_{\mathrm{T},i} = \arg\max_{k \in K} y_{\mathrm{T},i,k}, \quad y_{\mathrm{T},i} \in \{1, ..., K\} \tag{7.1}$$

Como se comprueba en la Figura 7.3, no hay diferencias entre las distintas configuraciones sobre los modelos, incluso tampoco entre las distintas arquitecturas de red excepto con GaMorNet, que se obtienen unos resultados ligeramente inferiores.

A la vista de los resultados, las configuraciones que se van a elegir son las siguientes:

- **No preentrenar**: dado que no hay diferencias entre usar *transfer learning* desde ImageNet o no, se ha decidido no usarlo dado que no da una clara ventaja.
- **Usar ponderación**: tampoco había diferencias entre usarla o no pero dado que estas ponderaciones han sido realizadas por astrónomos se va a seguir su criterio.

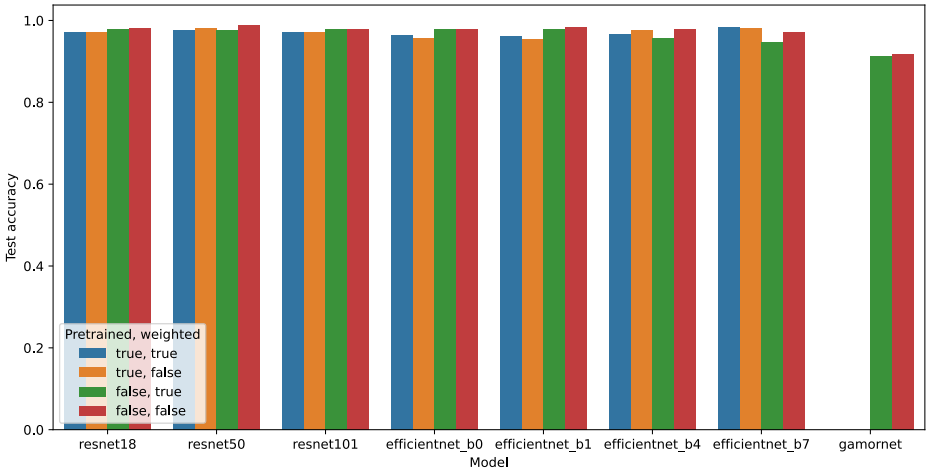


Figura 7.3: Comparación de la tasa de acierto sobre T01 y *clean dataset* de los modelos probados

Habiendo fijado estos dos parámetros, se va a proseguir a analizar las siguientes tareas bajo el mismo conjunto de datos. Como figuras en línea solo se mostrarán un subconjunto de ellos, los correspondientes a ResNet-18, ResNet-50, EfficientNet-B0, EfficientNet-B1 y GaMorNet. No obstante, en el Apéndice A se adjuntan los gráficos para todos los modelos probados.

T12 - ¿La galaxia tiene una apariencia mayoritariamente grumosa?

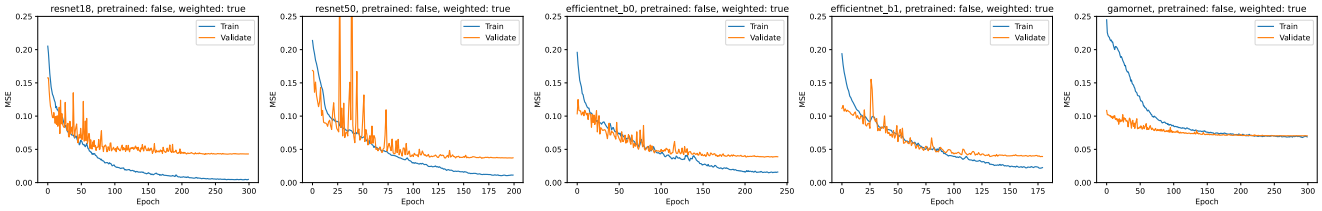


Figura 7.4: Evolución de la función de pérdida de entrenamiento y validación en el primer experimento con T12 y *clean dataset* para algunos de los modelos probados

En este caso sí que hay indicios de sobreajuste, dado que la curva de validación termina estando por encima de la de test en prácticamente todos los modelos excepto en GaMorNet (Figura 7.4). En este caso la función de pérdida se encuentra en torno al 0.05 del conjunto de validación.

Con respecto al rendimiento sobre el conjunto de test, en la Figura 7.5 se han representado las matrices de confusión para los mismos modelos (las matrices de confusión completas se encuentran en el anexo) junto con el *accuracy* y el *F1 score*. En todos los casos es rendimiento es bueno aunque peor que en la primera de las cuestiones y bastante parecido entre ellos, excepto en GaMorNet que vuelve a tener unos resultados peores.

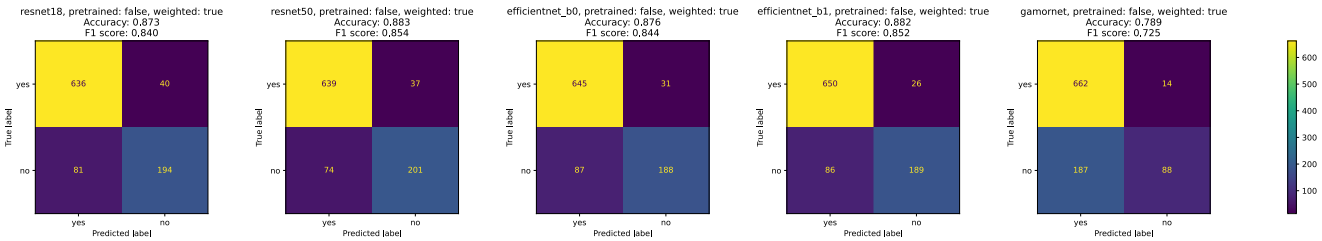
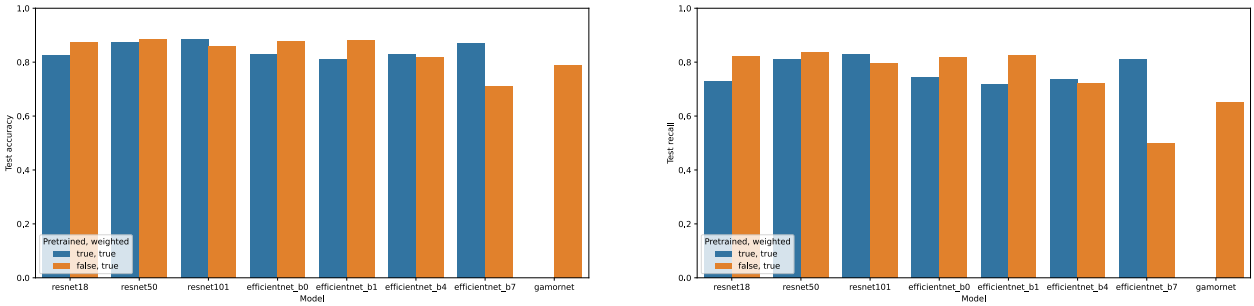


Figura 7.5: Matrices de confusión en el primer experimento con T12 y *clean dataset* para algunos de los modelos probados

Para comparar todos los modelos, se han realizado dos gráficos de barras con métricas sobre el conjunto de test, concretamente *accuracy* y *F1 score* (Figura 7.6). Todos los modelos tienen un rendimiento similar, excepto GaMorNet y EfficientNet-B7 sin preentrenar.



(a) *Accuracy* (b) *F1 score*

Figura 7.6: Métricas sobre el conjunto de test para T12 con regresión y *clean dataset*

T02 - ¿Puede ser un disco visto de canto?

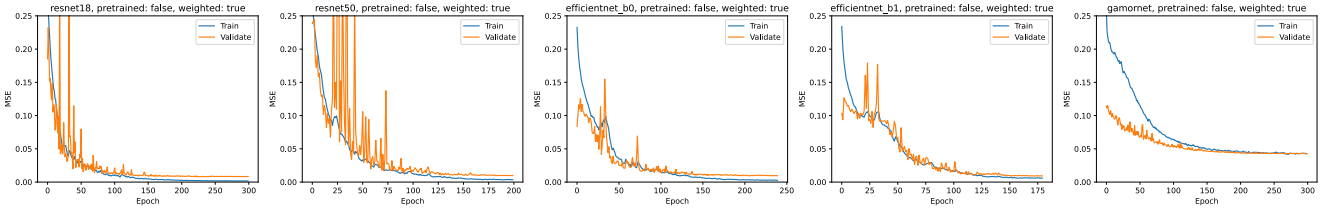


Figura 7.7: Evolución de la función de pérdida de entrenamiento y validación en el primer experimento con T02 y *clean dataset* para algunos de los modelos probados

A la vista de Figura 7.7 en este caso no hay signos claros de sobreajuste, en contraposición de lo que se veía en T12, con una función de pérdida del 0.012 en el conjunto de validación en la mayor parte de los modelos. En Figura 7.8 se comprueba que aunque hay algo de desbalanceo en los datos es una característica que los modelos distinguen correctamente como se puede comprobar con el valor F1 (aunque es inferior que la tasa de acierto).

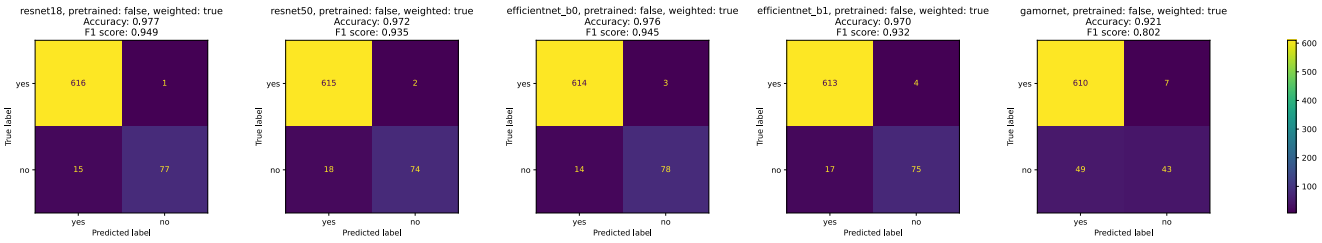
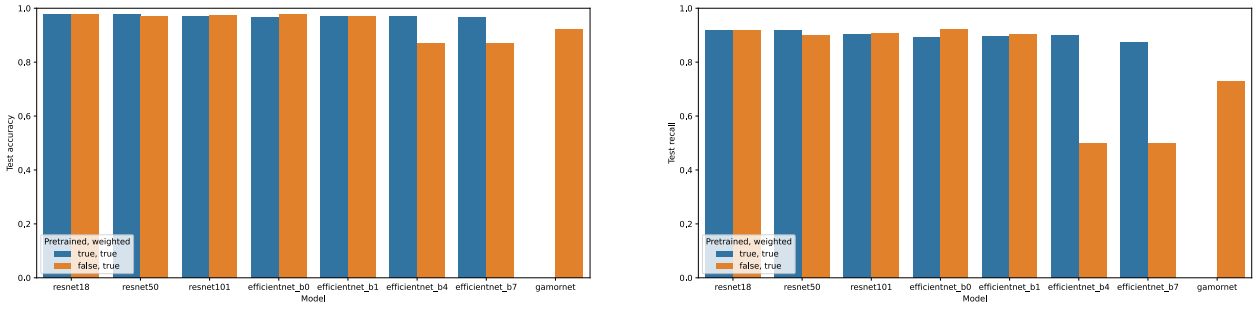


Figura 7.8: Matrices de confusión en el primer experimento con T02 y *clean dataset* para algunos de los modelos probados

Comparando todos los modelos (Figura 7.9), cabe destacar que las EfficientNet más profundas sin preentrenar funcionan algo peor.



(a) *Accuracy* (b) *F1 score*

Figura 7.9: Métricas sobre el conjunto de test para T02 con regresión y *clean dataset*

T03 - ¿Se percibe una forma de barra en el centro de la galaxia?

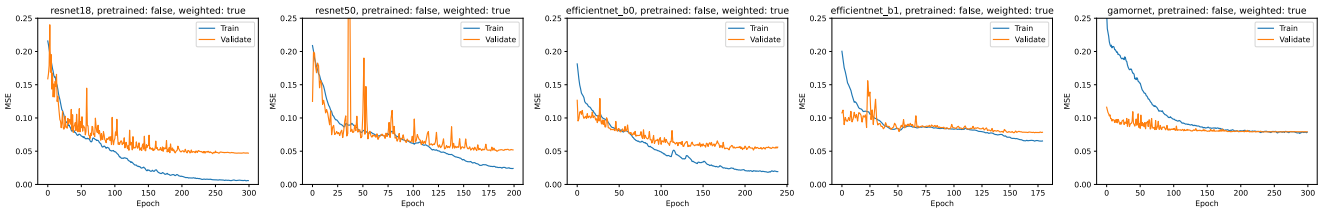


Figura 7.10: Evolución de la función de pérdida de entrenamiento y validación en el primer experimento con T03 y *clean dataset* para algunos de los modelos probados

En este caso hay algo de sobreajuste también (Figura 7.10), más que en el caso anterior y parecido a T12, el MSE sobre validación ronda el 0.06. Esto se explica bastante mejor viendo las matrices de confusión (Figura 7.11), en la primera de las clases hay bastantes instancias mal clasificadas como si fueran de la segunda clase. Parece que los dos modelos últimos tienen peor rendimiento, dado que el número de mal clasificadas es muy superior en estos dos aunque son dos modelos con distinto número de parámetros: GaMorNet es el más sencillo de todos y EfficientNet-B1 de los expuestos en esa gráfica es de los más profundos.

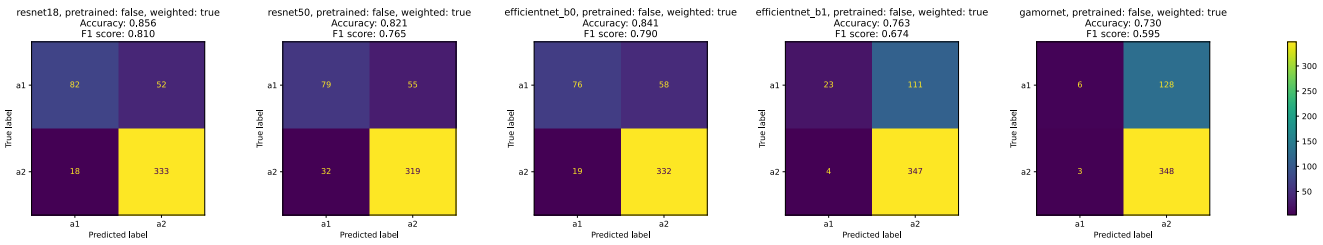
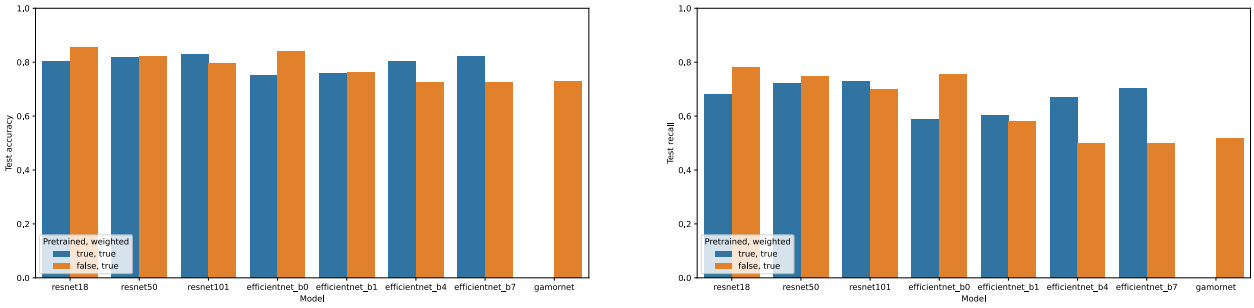


Figura 7.11: Matrices de confusión en el primer experimento con T03 y *clean dataset* para algunos de los modelos probados

Comparando todos los modelos, de nuevo y como ocurría en T12 el F1 score no va a la par con el *accuracy*, por lo que una de las clases está peor predicha que la otra (la primera como se ha

visto con las matrices de confusión). Los modelos más profundos de EfficientNet vuelven a ser los que peores rendimiento tienen (sin preentrenar) considerando el valor F1 junto con GaMorNet.



(a) *Accuracy* (b) *F1 score*

Figura 7.12: Métricas sobre el conjunto de test para T03 con regresión y *clean dataset*

T04 - ¿Hay algún signo de brazos en espiral?

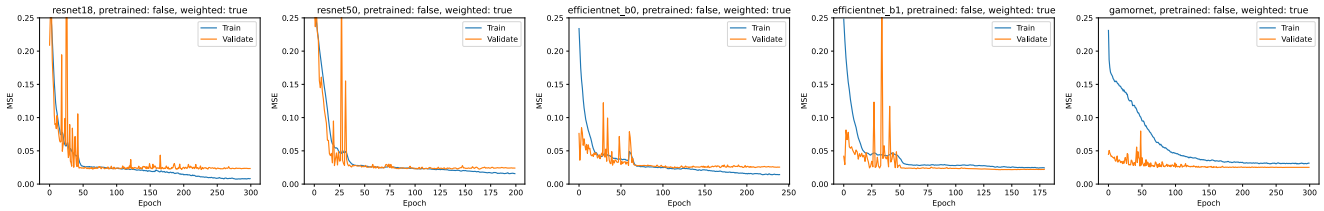


Figura 7.13: Evolución de la función de pérdida de entrenamiento y validación en el primer experimento con T03 y *clean dataset* para algunos de los modelos probados

Viendo la evolución de la pérdida (Figura 7.13), parece que hay un poco de sobreajuste en algunos de los modelos pero en general no parece ser un problema importante. Pero a la hora de comprobar las matrices de confusión (Figura 7.14) el panorama es bastante diferente: prácticamente nunca se predice la segunda clase y, al haber más del 90 % de instancias de la primera clase la función de pérdida no refleja este comportamiento. Este es el caso más claro de los que se han analizado de que no solamente podemos fijarnos en la tasa de acierto y menos en un conjunto tan desequilibrado. Por ejemplo en los últimos dos modelos no se predice nunca la segunda clase, por lo que estos clasificadores no tienen ningún sentido, son totalmente inútiles.

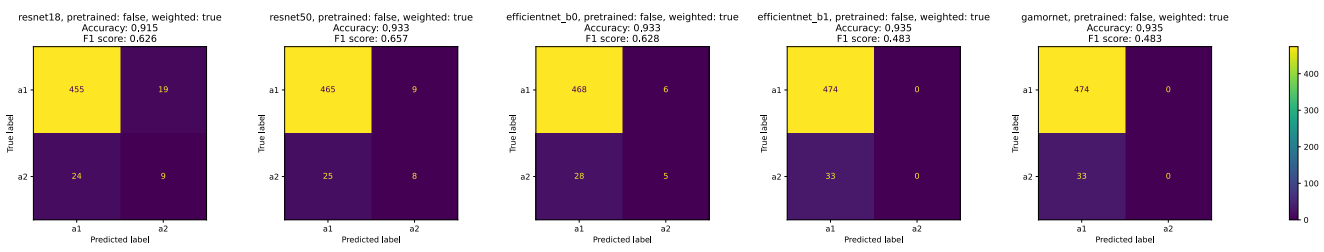
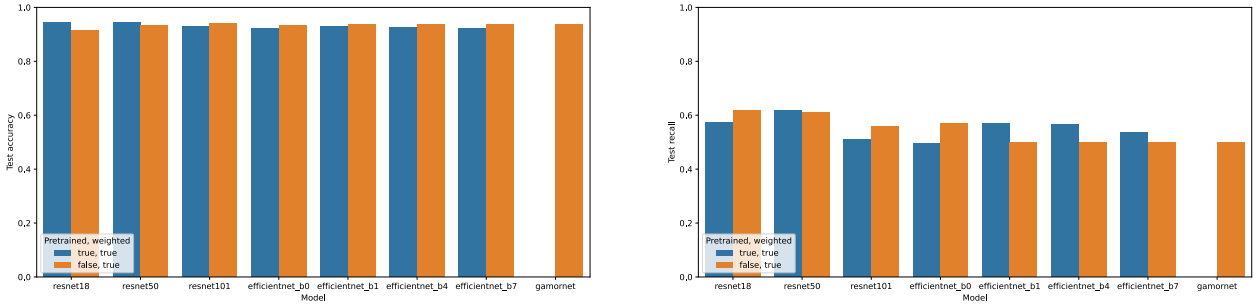


Figura 7.14: Matrices de confusión en el primer experimento con T04 y *clean dataset* para algunos de los modelos probados

El comportamiento visto con los modelos anteriores a través de las matrices de confusión parece que se repite con los demás modelos: (Figura 7.15) aunque el accuracy es muy bueno, el valor F1 es prácticamente 0.5, por lo que estos modelos no son demasiado útiles en la práctica. Los cuatro primeros parecen algo mejores, pero en todo caso no superan el 0.6 en el F1 score.



(a) *Accuracy* (b) *F1 score*

Figura 7.15: Métricas sobre el conjunto de test para T04 con regresión y *clean dataset*

Discusión

En general, parece que todos los modelos tienen un comportamiento más o menos similar sobre cada tarea según las métricas definidas: en T01 tienen el mejor rendimiento, seguido por T02. T12 y T03 parecen sobreajustar algo más los datos y no comportarse tan bien como en los dos casos anteriores. Por último en T04 se obtienen modelos bastante malos, algunos de ellos no clasificando en ningún caso a la segunda clase, lo que carece de interés.

Vistos los resultados, no parece que haya diferencias significativas entre la mayor parte de los modelos, en términos generales funcionan de forma parecida excepto GaMorNet, que tiene un rendimiento bastante inferior a los demás en todas las cuestiones por lo que se va a descartar el primero.

Para comparar los demás de manera más formal se ha realizado un test estadístico para detectar diferencias significativas entre varios modelos con distintos conjuntos de datos [51]. En nuestro caso los conjuntos son las distintas tareas, dado que se consideran distintas características de una galaxia. Se ha usado el test de Friedman [52] basado en rangos: a cada observación se le asigna un rango dependiendo del valor que ha arrojado en una métrica en concreto. Por ejemplo, usando el *accuracy* para cada modelo y tarea se realiza un ranking ordenando los modelos de mejor a peor y asignándoles números de 1 (el mejor) hasta el número de modelos que haya.

Sea r_i^j el ranking del algoritmo j-ésimo de un total de k algoritmos sobre el conjunto de datos i-ésimo de un total de N conjuntos de datos. Se calcula la media de los rankings de los algoritmos como $R_j = \frac{1}{N}\sum_{i=1}^{N} r_i^j$. Bajo la hipótesis nula de igualdad entre los distintos modelos y por tanto R_j iguales, el estadístico se define como:

$$\chi_F^2 = \frac{12N}{k(k+1)}\left[\sum_j R_j^2 - \frac{k(k+1)^2}{4}\right] \tag{7.2}$$

El cual sigue una distribución de Friedman. En este caso y, como es habitual, se va a usar la corrección de Iman y Davenport [53], que consiste en:

$$F_F = \frac{(N-1)\chi_F^2}{N(k-1)-\chi_F^2} \tag{7.3}$$

Que sigue una distribución F de Snedecor con $k-1$ y $(k-1)(N-1)$ grados de libertad. En el caso concreto que se está tratando $k = 15$ y $N = 5$, por lo que $F_F \sim F_{14,56}$ (Figura 7.16). Los valores críticos que se van a usar van a ser $\alpha = 0.1$ (significativo) y $\alpha = 0.05$ (muy significativo).

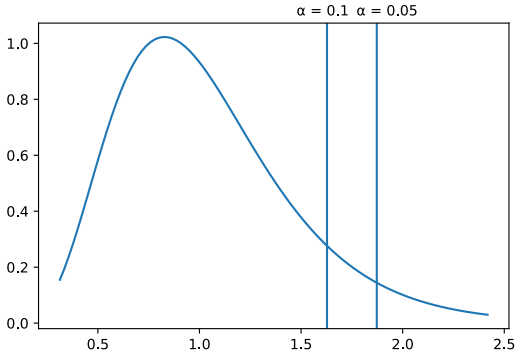


Figura 7.16: F de Snedecor de 14 y 56 grados de libertad. Se han señalado los valores que dejan una probabilidad $1-\alpha$ por debajo, con $\alpha = 0.1$ y $\alpha = 0.05$

Como métricas para realizar los rankings se van a usar tanto el *accuracy* como el *F1 score* dado que viendo lo desbalanceado de algunas de las tareas (como T04). Para cada una se han realizado los rankings, obtenido el valor de Friedman y aplicado la corrección de Iman y Davenport, con los valores mostrados en la Tabla 7.3. Ninguno de ellos es inferior al valor crítico de la F en $\alpha = 0.1$, el cual es 1.87 (y por tanto tampoco a $\alpha = 0.05$), por lo que en ninguno de los casos se rechaza la hipótesis nula de igualdad de modelos.

	Accuracy	***F1 score***
Valor Iman-Davenport	0.019	0.072

Tabla 7.3: Valores del test de Iman y Davenport usando el *accuracy* y el valor F1 como criterios a la hora de realizar los rankings respectivamente

Por lo que este método no encuentra diferencias significativas estadísticamente entre los distintos modelos, lo cual aunque era de esperar viendo las métricas, se ha querido corroborar.

A la vista de los resultados obtenidos en esta fase, se han planteado posibles experimentos que realizar a continuación, los cuales son:

- **Clasificación ponderada**: en vez de tratar el problema como regresión, habría que tratarlo como uno de clasificación. A cada una de las clases se les asignaría un peso w_c, $c = 1, ..., C$ (en nuestro caso $C = 2$, clasificación binaria). Hay varias estrategias posibles para fijar estos pesos, las más comunes son:

  - Inverso del número de instancias de la clase c (*INS*, por sus siglas en inglés): $w_c = \frac{1}{n_c}$
  - Inverso de la raíz cuadrada del número de instancias de la clase c (*ISNS*): $w_c = \frac{1}{\sqrt{n_c}}$
  - Número de instancias efectivo (*ENS*) [54]: $w_c = \frac{1}{E_{n_c}}$, $E_{n_c} = \frac{1-\beta^{n_c}}{1-\beta}$

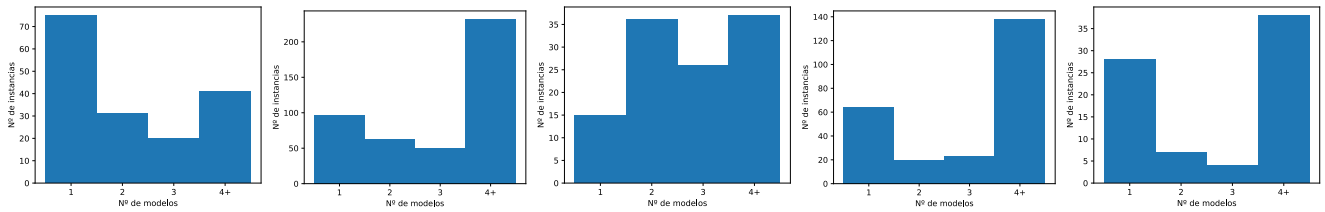
  n_c corresponde con el número de muestras en el conjunto de entrenamiento de la clase c y β es un hiperparámetro a configurar, los autores recomiendan $\beta = 0.99$.

  De esta forma previsiblemente se conseguiría paliar el problema de tener conjuntos de datos con clases desbalanceadas. Es la primera de las estrategias que se van a probar, en primer lugar se realizará la clasificación sin ponderar para comprobar que los resultados son parecidos a los obtenidos con regresión y después se probarán los tres esquemas de ponderación propuestos.

- ***Oversampling***: esta técnica se basa en remuestrear (con reemplazamiento) el conjunto de datos, tomando la misma proporción (o una especificada) de una clase y de otra. Para lograrlo se toman con reemplazamiento las observaciones de la clase más pequeña. En principio de esta forma también se conseguiría balancear el problema pero por contrapartida es mucho más sensible al sobreajuste.

- **Construcción de ensembles**: construir un ensemble de clasificadores, es decir, agrupar uno o varios modelos y tratarlos como si fueran un modelo base. Los ensembles pueden ser de muchos tipos pero en este caso se han considerado dos viendo el desarrollo de los modelos anteriores:

  - **Voto ponderado**: entrenar varios modelos de los anteriores, a ser posible los que fallen en distintas instancias para ganar diversidad, y con la salida de ellos se realiza un voto ponderado según unos pesos que hay que entrenar. Para comprobar si este ensemble es adecuado se han realizado los siguientes gráficos, uno por cada tarea, que muestran el número de observaciones sobre el conjunto de test en las que falla cada modelo. En el eje horizontal se han representado el número de modelos y en el vertical las observaciones sobre el conjunto de test de *clean dataset* en que falla cada modelo.



(a) T1 (b) T12 (c) T02 (d) T03 (e) T04

Figura 7.17: Histograma con el número de modelos que fallan al predecir del conjunto de test de *clean dataset*. Se han eliminado las observaciones que aciertan todos los modelos en cada tarea.

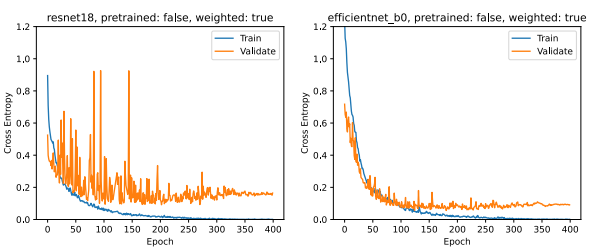
Fijándonos en la Figura 7.17 se puede apreciar que en la mayor parte de las tareas el mayor número de instancias corresponde a observaciones en las que fallan cuatro modelos o más (excepto en T01). Esto indica que habría que hacer un ensemble de más modelos que 4 para poder conseguir un rendimiento sobre todo el conjunto de test el cual seguramente sea inferior porque es posible que no sean los mismos modelos los que acierten en unas y en otras.

- ***Boosting***: se seleccionaría un único modelo y por cada instancia de entrenamiento se tendrían unos pesos. Consiste en un método iterativo en el que cada vez se entrena y a la función de pérdida de cada instancia se multiplica por sus pesos. Al final de la iteración, se actualizan los pesos de cada observación dando un mayor peso a las mal clasificadas, para que en la siguiente iteración tenga más en cuenta dichas galaxias. De esta forma no solo se paliaría el problema del desbalanceo en ciertas de las tareas, sino que tendríamos un indicador de las imágenes que son más conflictivas (las de mayor peso al final de todas las iteracciones). Por contrapartida, seguramente se aumentaría el sobreajuste pero en general viendo las gráficas de la evolución de la pérdida anteriores no es un problema demasiado grande (previsiblemente).

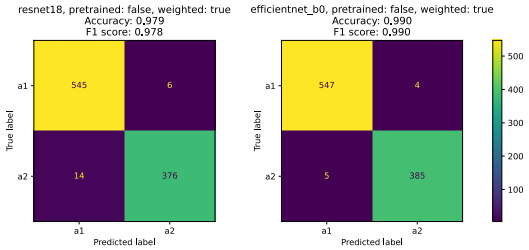
Siguiendo con la experimentación, en primer lugar se va a probar a realizar una clasificación (sin pesos) y seguidamente se van a aplicar los pesos en la siguiente etapa con los dos modelos seleccionados, sobre *clean dataset* y para todas las tareas. Con los resultados de esta fase se reformularán las posibles opciones a seguir.

7.1.2. Clasificación

En esta fase cada una de las cuestiones se va a tratar desde el punto de vista de la clasificación. Por tanto se usará la entropía cruzada (*Cross Entropy*) como función de pérdida en vez del MSE. De nuevo se han entrenado los modelos con un número suficiente alto de épocas para que converjan la pérdida y el entrenamiento, 400 en ambos casos. El tamaño de batch se ha fijado en 64.



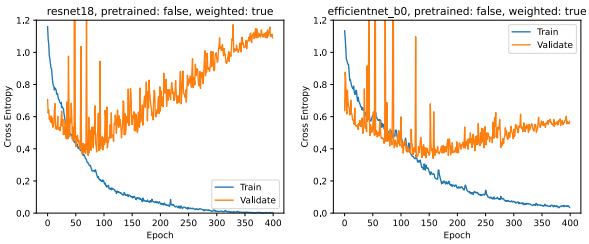
(a) Función de pérdida



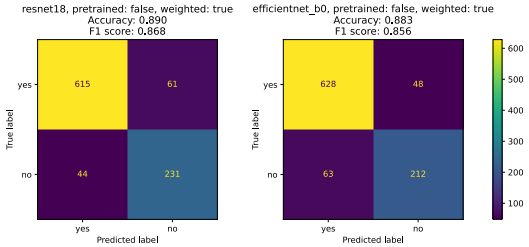
(b) Matriz de confusión con *accuracy* y *F1 score*

Figura 7.18: Tarea T01 con *clean dataset* realizando clasificación con los modelos seleccionados. Evolución de la función de pérdida en entrenamiento y validación y matriz de confusión con *accuracy* y *F1 score* sobre test

Según la evolución de la función de pérdida (Figura 7.18a) se aprecia mayor sobreajuste que realizando regresión, la curva de pérdida de validación está prácticamente siempre por encima de la de entrenamiento. Esto se hace más evidente en ResNet-18 que en EfficientNet-B0 y además en el primer caso hay más variación en validación. Comprobando este posible sobreajuste con el conjunto de test por medio de las matrices de confusión (Figura 7.18b) no parece que se esté dando al tener un rendimiento muy bueno: en ResNet-18 solo falla en 20 instancias de un total de 941 y en EfficientNet-B0 en 9 (menos de la mitad). Por tanto los resultados son aún mejores que en el caso de regresión, que ya eran buenos en un principio y no parece que sobreajuste viendo las métricas sobre test.



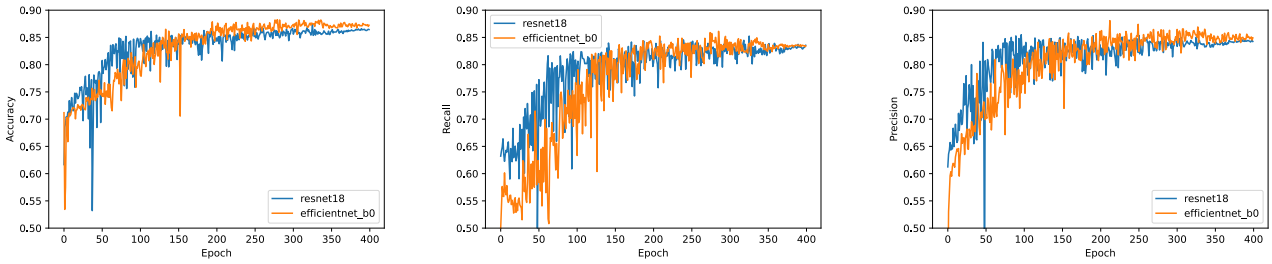
(a) Función de pérdida



(b) Matriz de confusión con *accuracy* y *F1 score*

Figura 7.19: Tarea T12 con *clean dataset* realizando clasificación con los modelos seleccionados. Evolución de la función de pérdida en entrenamiento y validación y matriz de confusión con *accuracy* y *F1 score* sobre test

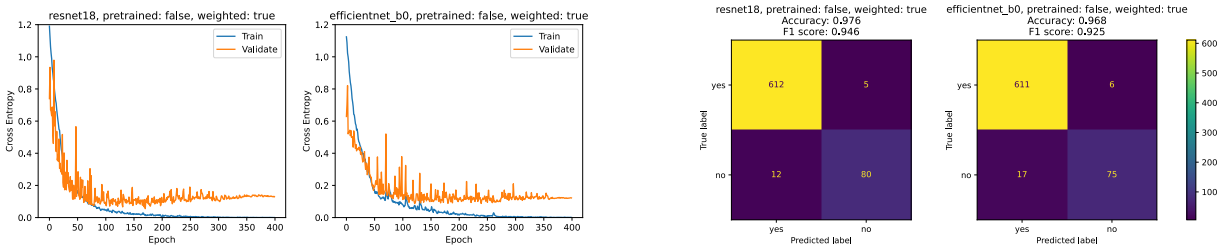
En la Figura 7.19 podemos observar un comportamiento bastante curioso a primera vista: mientras las funciones de pérdida de validación se incrementan, al final los modelos obtenidos tienen unos resultados bastante buenos en el conjunto de test, mejores que realizando regresión (Figura 7.5). En cambio en la regresión la pérdida sí que disminuía (Figura 7.4) en vez de aumentar como en este caso. Es decir, parece que aunque se está sobreajustando en los datos de entrenamiento, se siguen aprendiendo características importantes para generalizar, al tener un buen rendimiento en el conjunto de test.



(a) *Accuracy* (b) *Recall* (c) *Precision*

Figura 7.20: Evolución de varias métricas sobre el conjunto de entrenamiento y validación para T12 y *clean dataset*

En la Figura 7.20 se ha representado la evolución de las métricas en el conjunto de entrenamiento y de validación. En efecto se comprueba que aunque el modelo parezca sobreajustar el valor de las métricas sigue aumentando solo que de forma más lenta. Una solución a este comportamiento podría ser parar el entrenamiento a las 150-200 épocas, al tener unos resultados parecidos a 400 épocas según las métricas usadas.

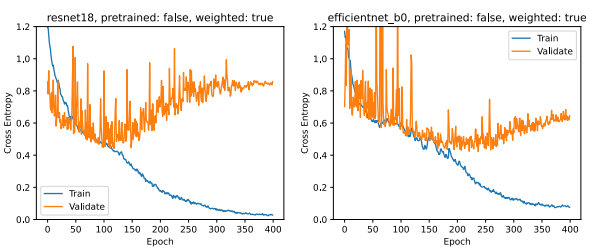


(a) Función de pérdida (b) Matriz de confusión con *accuracy* y *F1 score*

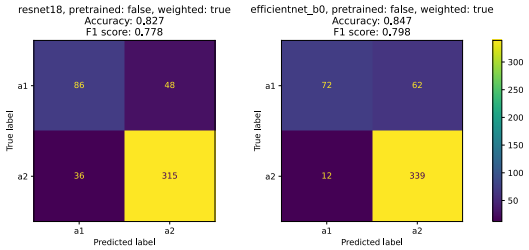
Figura 7.21: Tarea T02 con *clean dataset* realizando clasificación con los modelos seleccionados. Evolución de la función de pérdida en entrenamiento y validación y matriz de confusión con *accuracy* y *F1 score* sobre test

En el caso de la tarea T02 (Figura 7.21) el rendimiento que se obtiene es muy parecido al caso de regresión, no hay grandes diferencias ni en la evolución de las gráficas ni el comportamiento sobre el conjunto de test.

Sobre T03 (Figura 7.22) sin embargo sí que se aprecian varias diferencias. En primer lugar y como ocurría realizando clasificación con T12 (Figura 7.19) llama la atención la evolución de la función de pérdida con respecto al conjunto de validación, la cual se incrementa a partir de 100 épocas en el caso de ResNet-18 y de 200 en EfficientNet-B0. En cambio el rendimiento sobre el conjunto de test es también bueno, similar al caso de regresión, por lo que parece que en realidad no se está produciendo el sobreajuste a la vista de los resultados de test (al igual que en T12).



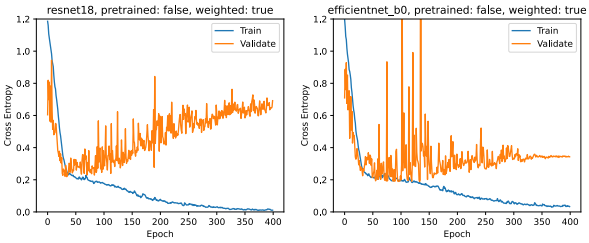
(a) Función de pérdida



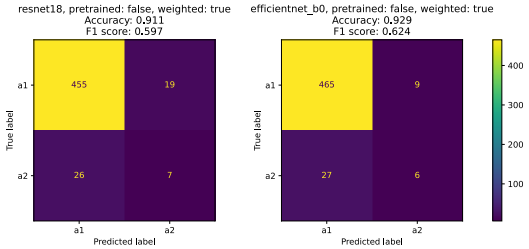
(b) Matriz de confusión con *accuracy* y *F1 score*

Figura 7.22: Tarea T03 con *clean dataset* realizando clasificación con los modelos seleccionados. Evolución de la función de pérdida en entrenamiento y validación y matriz de confusión con *accuracy* y *F1 score* sobre test

En relación a la última de las tareas (T04, Figura 7.23) se mantiene el mismo comportamiento que con T03 y T12 en el caso de clasificación: aunque parezca que se sobreajusta en validación al comprobarlo sobre el conjunto de test se obtienen resultados similares que en regresión (la evolución de las métricas sobre test se encuentra en el Apéndice B). Igual que ocurría en regresión, el resultado de esta última cuestión es bastante deficitario, fallando la segunda clase seguramente por el desbalanceo del conjunto de datos.



(a) Función de pérdida



(b) Matriz de confusión con *accuracy* y *F1 score*

Figura 7.23: Tarea T04 con *clean dataset* realizando clasificación con los modelos seleccionados. Evolución de la función de pérdida en entrenamiento y validación y matriz de confusión con *accuracy* y *F1 score* sobre test

En este caso se ha usado el conjunto *all galaxies dataset* para comprobar que los modelos ajustados funcionan con un conjunto mayor pero menos refinado de galaxias. En la Tabla 7.4 se muestran los resultados de las principales métricas, tanto de entrenamiento como de validación y test En el Apéndice B se encuentran las gráficas completas de la evolución de la pérdida en entrenamiento y validación y matrices de confusión sobre test. Comparando con el anterior conjunto:

- **T01**: sobre los parámetros de evolución de la función de pérdida, en este caso se ve un posible sobreajuste dado que al final la pérdida de entrenamiento de cada modelo ha disminuido hasta 0.01 o 0.025 respectivamente, en cambio la validación se incrementa llegando a alcanzar 1.015 y 0.855. Como pasaba antes esto parece no reflejarse sobre el conjunto de test al que se llega a un 87 % de *accuracy* y 86 % de *F1 score*, resultados que aunque peores que en el otro conjunto bastante buenos.

- **T12**: ocurre el mismo fenómeno en las curvas de pérdida y obteniendo resultados buenos sobre el de test. De nuevo son algo peores que en el caso de *clean dataset*, aproximadamente un 2 % peores.

- **T02**: en este caso el loss de validación no crece tanto, pero sigue teniendo un valor muy superior al de entrenamiento. Los resultados son algo inferiores al anterior conjunto, especialmente sobre *F1 score* aunque manteniéndose por encima del 88 %.

- **T03**: de nuevo ocurre que la pérdida se incrementa aunque no se refleja en el conjunto de test. En este caso el accuracy es algo mejor pero *F1 score* son peores, bajando del 78 % al 69 % aproximadamente.

- **T04**: cabe destacar en esta tarea que los resultados son algo mejores: aunque el *accuracy* es inferior, el valor del *recall* y la *precision* son mejores pasando en ambos modelos de 60 % y 62 % al 68 % y 69 % respectivamente.

Tarea	Modelo	Train loss	Validate loss	Validate accuracy	Validate recall	Validate precision	Test accuracy	Test F1 score
t01	**resnet18**	0.01	1.015	0.865	0.852	0.85	0.876	0.862
	efficientnet_b0	0.025	0.855	0.87	0.857	0.858	0.875	0.862
t12	**resnet18**	0.013	1.13	0.839	0.735	0.762	0.841	0.746
	efficientnet_b0	0.061	0.89	0.831	0.722	0.748	0.843	0.752
t02	**resnet18**	0.002	0.43	0.943	0.892	0.89	0.942	0.9
	efficientnet_b0	0.011	0.36	0.944	0.891	0.907	0.932	0.881
t03	**resnet18**	0.012	1.068	0.858	0.689	0.748	0.852	0.705
	efficientnet_b0	0.127	0.454	0.875	0.716	0.759	0.849	0.696
t04	**resnet18**	0.054	1.256	0.791	0.676	0.718	0.787	0.682
	efficientnet_b0	0.118	0.934	0.797	0.677	0.707	0.796	0.699

Tabla 7.4: Resultados de los modelos de clasificación sobre todas las tareas en *all galaxies dataset*

Discusión

Sobre el fenómeno que se percibía en varias de las tareas anteriores, que la curva de pérdida sobre validación se incremente (empeore) mientras las demás métricas sobre el mismo conjunto mejoraban puede deberse a la forma con la que se calculan ambas funciones. En primer lugar recordemos que la pérdida está dada por la entropía cruzada, cuya fórmula es:

$$e = -(y \log p + (1 - y) \log(1 - p))$$

donde p es la salida de la capa softmax, anterior a la clasificación y siendo una capa de regresión. En cambio, tanto el *accuracy*, *recall* y *precision* (y por tanto el *F1 score*) se calculan al haber discretizado dichos valores. Por tanto, aunque en la mayoría de los casos ocurre, no tienen porqué estar correladas ambas funciones y cuando se incrementa una no tiene porqué decrementar la otra (y viceversa). Además, la entropía cruzada no es una función acotada, como se puede observar en la Figura 7.24, donde se ha representado los valores que toma la función de pérdida dependiendo de la probabilidad predicha cuando la clase real es 1. La consecuencia de ello es que si la red falla

en observaciones que estaba "muy segura" que clasificaba correctamente la pérdida crece de forma muy rápida, lo cual es lo que probablemente esté ocurriendo.

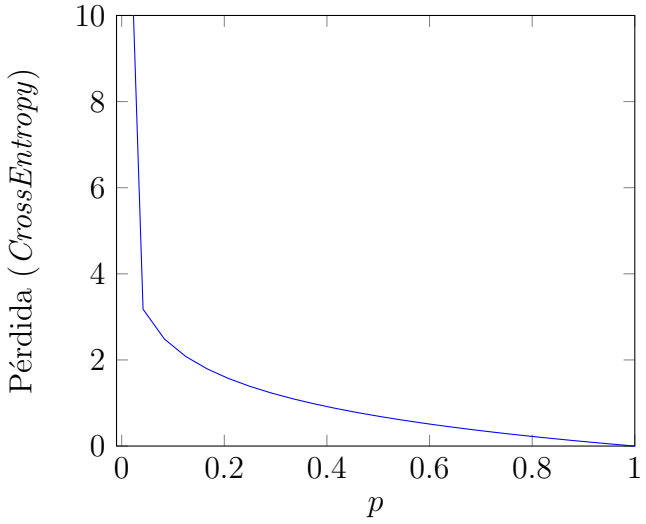


Figura 7.24: Entropía cruzada según valores predichos para una instancia de clase real 1

No obstante, sobre el conjunto de test se obtienen resultados al menos igual de buenos que en regresión, por lo que se va a proseguir con este modelo para compararlo con la clasificación ponderada.

Como conclusión sobre ambos conjuntos de datos es que los resultados en general para el mayor de ellos son peores, lo cual en un principio podría parecer ilógico dado que se tienen más datos y por tanto debería aprender más (además *clean dataset* está incluído en *all galaxies dataset*). Sin embargo, como el más pequeño de ellos se ha generado usando conocimiento sobre las que son en principio las galaxias mejor clasificadas (recordemos que están evaluadas por humanos) es normal que tenga mejores resultados, al ser un conjunto más refinado.

En la siguiente fase se va a intentar corregir el problema con las dos últimas tareas, con el objetivo de paliar el desbalanceo de los datos utilizando pesos según la cantidad de observaciones por cada clase, con 3 esquemas distintos: INS, ISNS y ENS.

7.1.3. Clasificación ponderada

En la Tabla 7.5 se muestran las métricas medidas para cada uno de los esquemas de ponderación, para cada una de las tareas junto con la clasificación sin ponderación para facilitar la comparación. Las gráficas completas de la evolución de la pérdida en entrenamiento y validación y las matrices de confusión sobre el conjunto de test se encuentran en el Apéndice C.

Dado que en T01 y T02 los resultados eran de por sí ya muy buenos las conclusiones que se obtienen ponderando son las mismas. Sobre el posible desbalanceo en las demás tareas, parece que

Tarea	Pesos	Modelo	Train loss	Validate loss	Validate accuracy	Validate recall	Validate precision	Test accuracy	Test F1 score
t01	none	resnet18	6.238e-04	0.164	0.973	0.971	0.974	0.979	0.978
		efficientnet_b0	1.286e-03	0.091	0.984	0.984	0.984	0.990	0.990
	ins	resnet18	4.691e-04	0.148	0.974	0.973	0.974	0.976	0.975
		efficientnet_b0	1.603e-03	0.099	0.979	0.978	0.980	0.982	0.981
	isns	resnet18	5.338e-04	0.160	0.974	0.972	0.974	0.988	0.988
		efficientnet_b0	1.704e-03	0.105	0.976	0.976	0.976	0.986	0.986
	ens	resnet18	1.389e-03	0.123	0.980	0.979	0.981	0.981	0.980
		efficientnet_b0	1.502e-03	0.126	0.982	0.981	0.982	0.986	0.986
t12	none	resnet18	3.244e-03	1.090	0.864	0.836	0.843	0.890	0.868
		efficientnet_b0	3.577e-02	0.570	0.872	0.834	0.849	0.883	0.856
	ins	resnet18	4.082e-03	1.403	0.871	0.836	0.854	0.877	0.850
		efficientnet_b0	3.409e-02	0.833	0.868	0.836	0.848	0.875	0.848
	isns	resnet18	3.808e-03	1.203	0.858	0.814	0.835	0.875	0.848
		efficientnet_b0	2.241e-02	0.820	0.863	0.828	0.843	0.876	0.848
	ens	resnet18	3.911e-03	1.121	0.868	0.840	0.850	0.882	0.856
		efficientnet_b0	2.642e-02	1.130	0.860	0.823	0.848	0.890	0.864
t02	none	resnet18	2.403e-04	0.129	0.981	0.955	0.967	0.976	0.946
		efficientnet_b0	8.740e-04	0.122	0.972	0.936	0.954	0.968	0.925
	ins	resnet18	7.869e-04	0.385	0.979	0.947	0.971	0.973	0.939
		efficientnet_b0	5.951e-03	0.298	0.978	0.945	0.965	0.963	0.916
	isns	resnet18	5.110e-04	0.306	0.977	0.932	0.969	0.979	0.952
		efficientnet_b0	9.569e-04	0.221	0.979	0.943	0.962	0.970	0.932
	ens	resnet18	9.171e-04	0.330	0.979	0.950	0.962	0.972	0.935
		efficientnet_b0	2.795e-03	0.465	0.968	0.922	0.936	0.972	0.935
t03	none	resnet18	2.751e-02	0.857	0.812	0.766	0.760	0.827	0.778
		efficientnet_b0	7.621e-02	0.645	0.842	0.763	0.847	0.847	0.798
	ins	resnet18	1.084e-02	1.156	0.859	0.805	0.826	0.835	0.789
		efficientnet_b0	6.301e-02	0.876	0.841	0.791	0.801	0.860	0.820
	isns	resnet18	1.527e-02	0.966	0.858	0.805	0.842	0.856	0.813
		efficientnet_b0	2.449e-01	0.665	0.782	0.729	0.738	0.802	0.741
	ens	resnet18	3.408e-02	1.037	0.818	0.758	0.783	0.816	0.757
		efficientnet_b0	2.611e-01	0.629	0.805	0.760	0.765	0.800	0.755
t04	none	resnet18	1.082e-02	0.692	0.920	0.572	0.597	0.911	0.597
		efficientnet_b0	3.296e-02	0.343	0.941	0.578	0.681	0.929	0.624
	ins	resnet18	8.101e-02	1.533	0.920	0.620	0.594	0.901	0.558
		efficientnet_b0	2.516e-01	0.898	0.888	0.663	0.610	0.854	0.586
	isns	resnet18	4.857e-02	0.972	0.938	0.611	0.631	0.931	0.641
		efficientnet_b0	1.323e-01	0.719	0.923	0.582	0.644	0.913	0.574
	ens	resnet18	7.162e-02	2.041	0.921	0.673	0.653	0.907	0.645
		efficientnet_b0	2.236e-01	1.067	0.891	0.652	0.607	0.862	0.664

Tabla 7.5: Resultados de los modelos de clasificación (ponderada o no) sobre todas las tareas en *clean dataset*

en T12 no se consigue mejorar en ninguno de los dos modelos. Sobre T03, en ResNet-18 con ISNS aparece una mejora del 3 % en ambas métricas de test, las cuales no son despreciables. En T04 también se muestra una mejora con el mismo modelo y ponderación, del 2 % sobre el *accuracy* y 4 % del *F1 score* (test ambos).

Estas mejoras comentadas es posible que se deban a la variabilidad de los datos. Otra comparación posiblemente más interesante sea para elegir el mejor modelo entre todos los descritos. De nuevo, si eligiéramos la regla de la navaja de Ockham comparando únicamente los modelos sin ponderación, nos quedaríamos con ResNet-18 dado que los resultados no difieren mucho con los de EfficientNet-B0 y es un modelo bastante más sencillo. Sin embargo, en algunas de las cues-

tiones como en T01 perderíamos rendimiento, concretamente un 2 % que aunque pueda parecer insignificante en los términos que estamos, rondando el 97 − 99 % es bastante significativo. Si nos comparamos ponderación con no ponderación, en las mismas tareas donde EfficientNet-B0 es mejor (T01, T03 y T04), usando el esquema ISNS con ResNet-18 se igualan o incluso mejoran los resultados. Por tanto, añadiendo estos pesos (ISNS) conseguimos un rendimiento muy parecido al de EfficientNet-B0 pero usando un modelo más sencillo como lo es ResNet-18.

7.1.4. Modelo final

El modelo escogido es una red neuronal residual con 18 capas (ResNet-18), como se muestra en la Figura 7.25. La representación de cada bloque (en gris) consta de: una capa convolucional (2D), una de normalización del batch (2D) y por último una activación ReLU. Además, después de la capa *AvgPool* hay una capa de normalización del batch (1D) y una *Dropout* con probabilidad 0.5. Los parámetros utilizados para entrenar la red se encuentran en la Tabla 7.6.

Preentrenado	**Función de pérdida**	**Ponderación**	**Optimizador**	**Tamaño del batch**	**Épocas**
Sí	*CrossEntropy*	ISNS	Adam	64	250

Tabla 7.6: Parámetros del modelo escogido

Se ha intentado paliar el efecto de un posible sobreajuste que se veía en alguna gráfica como la Figura 7.19a usando regularización. Se ha probado con la regresión *ridge* (regularización L_2) explicada en la Sección 3.1.7 usando 0.1 como parámetro de penalización. Los resultados obtenidos han sido iguales de los del modelo sin regularizar, por lo que se ha descartado.

Como prueba de los resultados obtenidos en el modelo y para interpretarlos, se va a utilizar el método denominado *class activation map* (CAM) [55]. De esta forma se intentará comprender cómo toma la red neuronal la decisión de clasificar cada instancia, dependiendo de la parte de la imagen que tiene en cuenta.

Este método utiliza la salida de la última capa convolucional de la red junto con las predicciones para proporcionar un mapa de calor como justificación de la decisión tomada. Para cada posición de la última capa convolucional, tenemos el mismo número de neuronas que en la última capa *fully-connected*. Por tanto, si realizamos el producto de dichas activaciones con los pesos finales se obtiene una puntuación para cada píxel de la imagen, lo que se puede usar para dar una justificación de la clasificación realizada.

Existe una mejora de este procedimiento llamada *Gradient CAM* (Grad-CAM) [56] que permite hacer mapas de calor de todas las capas, no solo la última. Esto puede ser interesante dado que cada capa tiene la capacidad de detectar ciertas formas, independientemente de la posición donde se encuentren [57].

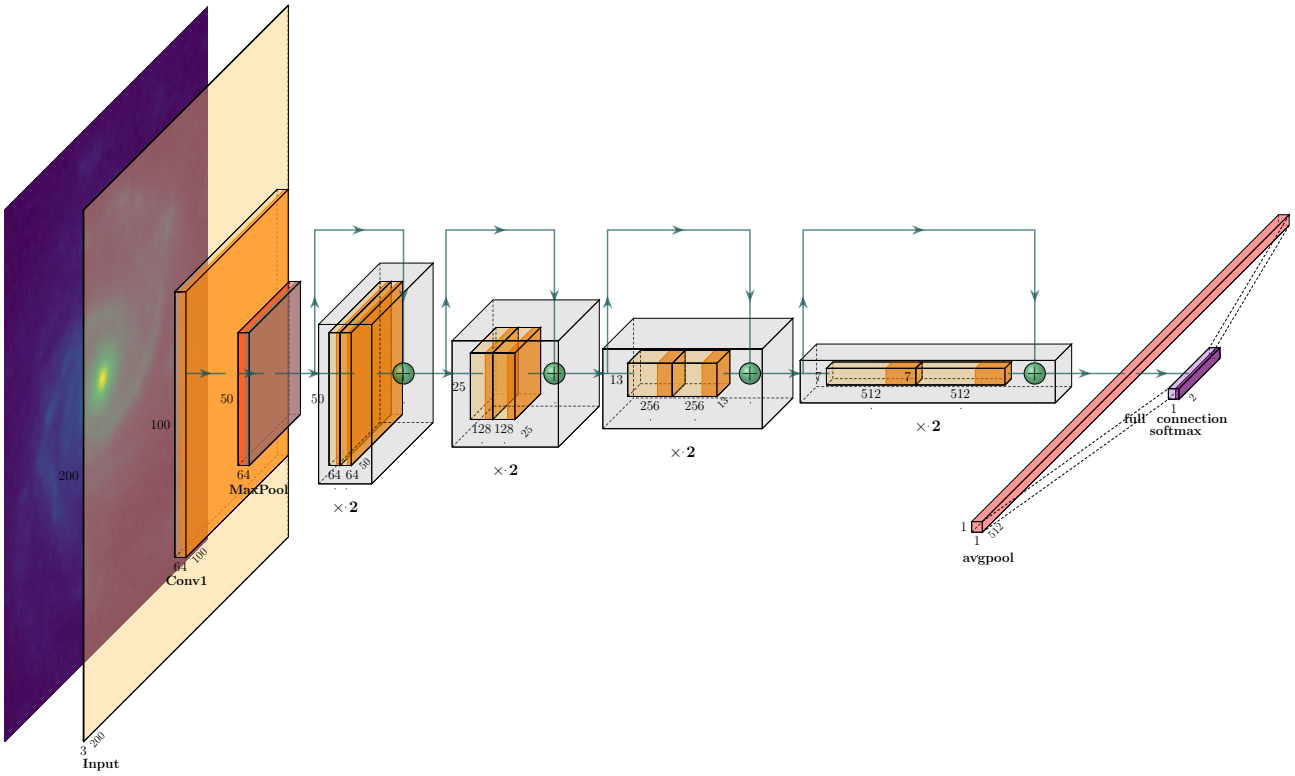
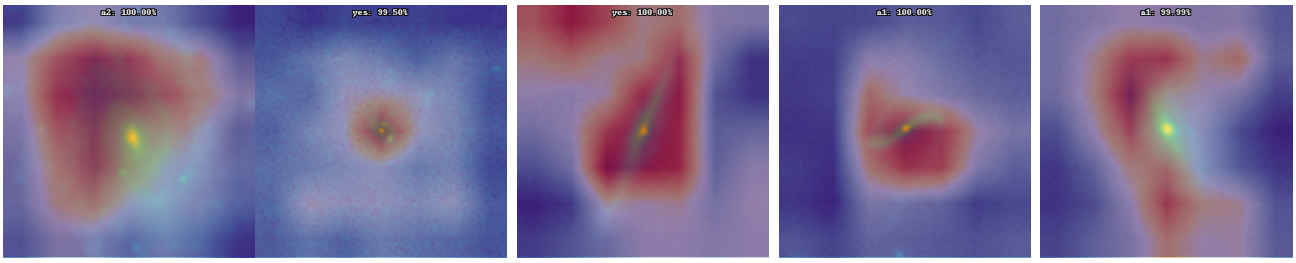


Figura 7.25: Arquitectura de capas de ResNet-18

En la Figura 7.26 se muestran varios ejemplos de galaxias correctamente clasificadas por la red. En todas ellas parecen centrarse en las características propias de cada pregunta: por ejemplo, en T01 se quería comprobar si la galaxia tenía forma de disco, por lo que en vez de concentrarse en el centro lo hace en una esquina de la galaxia (Figura 7.26a). En T12 se quería detectar si tenía apariencia grumosa, por lo que es importante tanto el centro como el entorno, como se muestra en la Figura 7.26b. En el caso de T02, el objetivo era saber si podía tener forma de disco visto de canto, por lo que la zona en la que se interesa la red es alargada, siguiendo la forma de la galaxia (Figura 7.26c). La tarea T03 trataba sobre si puede existir una forma de barra en el núcleo, por lo que principalmente hay que poner la atención en el centro y su entorno (Figura 7.26d). Por último, en T04 se quería comprobar si podía tener brazos en espiral, por lo que es más importante la zona alrededor del centro que el propio núcleo (Figura 7.26e).

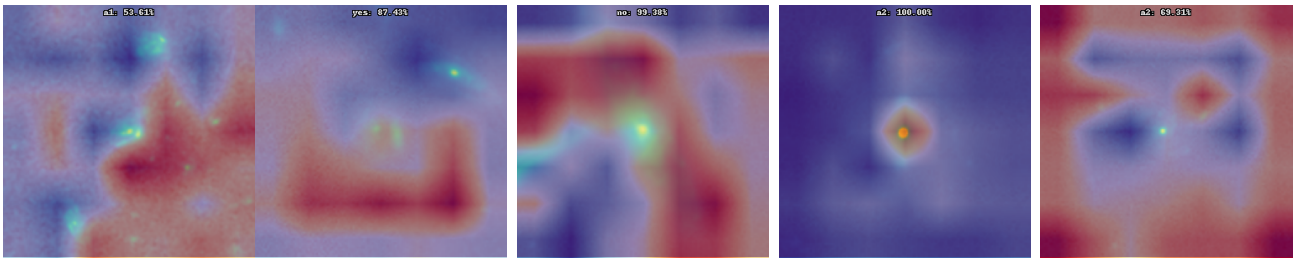
En contraposición, en la Figura 7.27 se muestran imágenes que han sido mal clasificadas por la red. En todas ellas se aprecia que la red no se centra en las características más importantes, sino en zonas de la imagen que no guardan relación con las preguntas. En la primera, se confunde principalmente porque hay otros focos de luz aparte de la galaxia central. En T12, T02 y T04 no consigue enfocar la galaxia y por tanto la clasificación la realiza otras partes de la imagen. En cambio, en T03 se clasifica la galaxia como que no tiene barra central, lo cual a simple vista parece que así es, pero en el conjunto de datos se había marcado que sí tenía, por lo que seguramente



(a) T01 (b) T12 (c) T02 (d) T03 (e) T04

Figura 7.26: Mapas de calor correspondientes a observaciones bien clasificadas

haya sido un error por parte de los evaluadores y la red lo ha predicho correctamente.



(a) T01 (b) T12 (c) T02 (d) T03 (e) T04

Figura 7.27: Mapas de calor correspondientes a observaciones mal clasificadas

7.2. Cuestión 2

Dado que normalmente es muy complicado tener un gran conjunto de datos de una buena calidad para entrenar el modelo, es interesante conocer el número mínimo de instancias necesarias para llegar a un rendimiento similar que con el máximo probado. Para ello, partiendo del modelo escogido, se irá evaluando con tamaños de 1000 en 1000 observaciones (1000, 2000, 3000, ...) para cada uno de los conjuntos y tareas. De nuevo para probar el rendimiento se usará tanto el *accuracy* como el *F1 score* evaluados sobre el conjunto de test, lo que nos dará un valor de generalización.

Para comparar entre los distintos tamaños es útil tener una medida de la variabilidad. Dado que cada observación es independiente de las demás (fijados un conjunto y una tarea) y el tamaño de los datos de test es mayor a 30 instancias es posible utilizar el procedimiento desarrollado en el Capítulo 5 de [3]. Bajo estas condiciones, el valor más probable del error de generalización disponible es el estimado con el conjunto de test y se encuentra en el siguiente intervalo con una confianza de $1 - \alpha$:

$$e_S \pm z_{1-\frac{\alpha}{2}} \sqrt{\frac{e_S(1-e_S)}{n_S}} \tag{7.4}$$

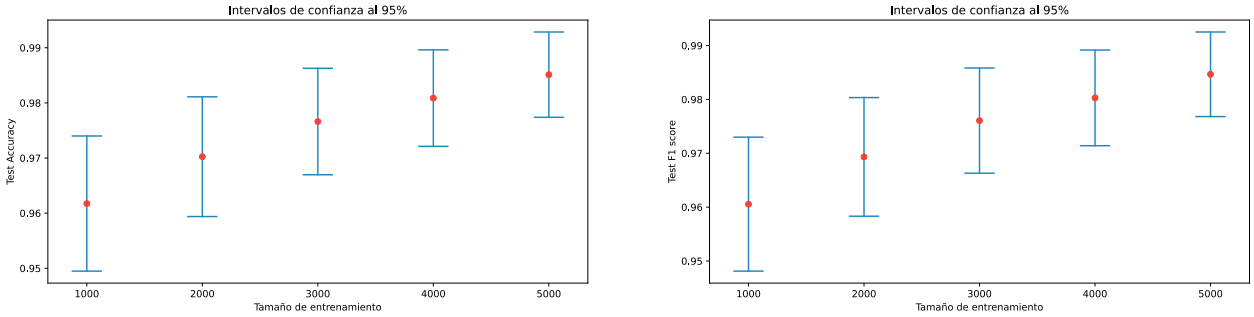
donde e_S es una medida del error bajo el conjunto S, n_S es el número de instancias de S, $z_{1-\frac{\alpha}{2}}$ corresponde el valor de una distribución Normal con media 0 y desviación típica de 1 que encierra una probabilidad de al menos $1-\frac{\alpha}{2}$. Comúnmente se toma como $\alpha = 0.05$, por lo que $z \simeq 1.96$.

El fundamento teórico detrás de este resultado es el siguiente: consideremos que S es una muestra aleatoria de galaxias de la distribución real $\mathcal{D}$, la cual no es conocida. Si consideramos S_i muestras diferentes, $i = \{1, ..., k\}$, e_{S_i} es una variable aleatoria. Si k es lo suficientemente grande, la distribución del error se aproximará a una Binomial $b(n, p)$, cuya función de probabilidad es:

$$P(r) = \frac{n!}{r!(n-r)!} p^r (1-p)^{n-r} \tag{7.5}$$

cuya media es np y varianza $np(1-p)$. Para valores suficientemente grandes de n (se considera suficientemente grande a $n > 30$) es posible aproximar la distribución Binomial a una Normal, siempre que $np(1-p) \geq 5$. En nuestro caso se tendrá que cumplir que $n_S e_S(1-e_S) \geq 5$ (en todos los casos se cumple).

Para cada tarea sobre el conjunto *clean dataset* se ha calculado el *accuracy* y el valor F1 sobre el conjunto de test. Con ambos valores se han calculado intervalos de confianza según el método descrito, con el objetivo de comprobar el mínimo de galaxias necesarias para obtener el máximo rendimiento.



(a) *Accuracy* (b) *F1 score*

Figura 7.28: Comparación de las métricas de test en la tarea T01 con *clean dataset* según el número de instancias de entrenamiento

Para ambas métricas el mejor valor se obtiene entrenando con el número máximo de observaciones (Figura 7.28). Comparando los intervalos, en el accuracy el último intervalo (5000) se solapa con el segundo (2000), por lo que con una confianza del 95 % no hay diferencias significativas entre ambos. De igual forma ocurre con el valor F1, se solapan los mismos intervalos. Por tanto, en esta tarea con 2000 galaxias de entrenamiento podemos conseguir unos resultados similares a los que se consiguen con 5000 galaxias, con una confianza del 95 %.

De igual manera se ha procedido con las demás tareas en *clean dataset*, cuyos gráficos completos

se encuentran en el Apéndice D. En la Tabla 7.7 se muestra un resumen en forma de tabla con los intervalos de confianza: en verde se han señalado los intervalos correspondientes a los mejores valores para cada tarea y métrica, y en azul el mínimo número de instancias necesarias cuyo intervalo se solapa con el mejor.

	T12		**T02**		**T03**		**T04**	
n	***Accuracy***	***F1 score***	***Accuracy***	***F1 score***	***Accuracy***	***F1 score***	***Accuracy***	***F1 score***
1000	(0.757, 0.810)	(0.706, 0.762)	(0.936, 0.968)	(0.864, 0.911)	(0.736, 0.811)	(0.666, 0.747)	(0.902, 0.948)	(0.633, 0.715)
2000	(0.816, 0.863)	(0.769, 0.820)	(0.963, 0.986)	(0.925, 0.960)	(0.751, 0.824)	(0.681, 0.760)	(0.898, 0.945)	(0.547, 0.632)
3000	(0.845, 0.888)	(0.814, 0.861)	(0.968, 0.989)	(0.936, 0.968)				
4000	(0.840, 0.884)	(0.810, 0.857)	(0.955, 0.980)	(0.907, 0.946)				
5000	(0.862, 0.903)	(0.836, 0.881)						

Tabla 7.7: Comparación de las métricas de test para las tareas T12 - T04 con *clean dataset* según el número de instancias de entrenamiento

Para la tarea T12, bastaría entrenar con 3000 galaxias (con 2000 se obtendría parecido *accuracy* pero no el valor F1). En el caso de T02 con 2000 serían suficientes y tanto para T03 como para T04 con 1000 galaxias se obtendrían resultados similares que con 2000.

	T01		**T12**		**T02**		**T03**		**T04**	
n	***Accuracy***	***F1 score***	***Accuracy***	***F1 score***	***Accuracy***	***F1 score***	***Accuracy***	***F1 score***	***Accuracy***	***F1 score***
1000	(0.751, 0.779)	(0.735, 0.764)	(0.766, 0.799)	(0.606, 0.645)	(0.895, 0.922)	(0.821, 0.857)	(0.767, 0.811)	(0.567, 0.620)	(0.717, 0.766)	(0.603, 0.657)
2000	(0.793, 0.820)	(0.767, 0.795)	(0.761, 0.795)	(0.646, 0.684)	(0.912, 0.938)	(0.850, 0.882)	(0.774, 0.817)	(0.584, 0.636)	(0.697, 0.747)	(0.597, 0.651)
3000	(0.810, 0.836)	(0.789, 0.816)	(0.781, 0.813)	(0.670, 0.707)	(0.923, 0.947)	(0.869, 0.900)	(0.729, 0.776)	(0.603, 0.655)	(0.756, 0.802)	(0.653, 0.706)
4000	(0.822, 0.848)	(0.804, 0.831)	(0.808, 0.839)	(0.698, 0.734)	(0.910, 0.936)	(0.852, 0.884)	(0.787, 0.829)	(0.589, 0.642)	(0.762, 0.808)	(0.644, 0.697)
5000	(0.837, 0.862)	(0.820, 0.845)	(0.798, 0.830)	(0.696, 0.732)	(0.931, 0.954)	(0.883, 0.912)	(0.797, 0.839)	(0.608, 0.660)	(0.706, 0.756)	(0.611, 0.665)
6000	(0.829, 0.854)	(0.815, 0.841)	(0.813, 0.843)	(0.715, 0.751)	(0.925, 0.949)	(0.876, 0.906)	(0.792, 0.834)	(0.622, 0.674)	(0.748, 0.795)	(0.639, 0.692)
7000	(0.836, 0.860)	(0.818, 0.843)	(0.808, 0.839)	(0.720, 0.755)	(0.927, 0.950)	(0.874, 0.904)	(0.801, 0.842)	(0.630, 0.681)		
8000	(0.839, 0.863)	(0.820, 0.846)	(0.812, 0.842)	(0.714, 0.750)	(0.936, 0.957)	(0.893, 0.921)				
9000	(0.846, 0.870)	(0.829, 0.854)	(0.831, 0.860)	(0.749, 0.783)	(0.926, 0.949)	(0.875, 0.905)				
10000	(0.839, 0.864)	(0.823, 0.848)	(0.819, 0.849)	(0.721, 0.756)						
11000	(0.842, 0.866)	(0.826, 0.851)	(0.826, 0.856)	(0.735, 0.770)						
12000	(0.836, 0.860)	(0.820, 0.845)	(0.809, 0.840)	(0.719, 0.754)						
13000	(0.839, 0.863)	(0.823, 0.848)	(0.827, 0.856)	(0.742, 0.776)						
14000	(0.855, 0.878)	(0.841, 0.865)								
15000	(0.848, 0.872)	(0.835, 0.860)								
16000	(0.852, 0.876)	(0.836, 0.861)								
17000	(0.849, 0.873)	(0.834, 0.858)								
18000	(0.854, 0.877)	(0.838, 0.862)								

Tabla 7.8: Comparación de las métricas de test para todas las tareas con *all galaxies dataset* según el número de instancias de entrenamiento

Este procedimiento se ha repetido para el otro conjunto de datos más grande (*all galaxies dataset*), cuyos resultados se muestran en la Tabla 7.8 y gráficos completos en el Apéndice D. Para la primera de las tareas con 7000 instancias valdría para obtener un rendimiento similar al máximo, situado en 14000. Con respecto a T12, se necesitarían al menos 6000 galaxias para obtener en ambas métricas un valor similar al mejor. En T02 bastaría con 3000, en T03 con 2000 y en T04 únicamente con 1000.

Para concluir, resulta interesante observar que el máximo (valor en verde) en cada métrica no se

alcanza con el máximo de instancias de entrenamiento, sino con un valor intermedio. Por ejemplo, en T01 en vez de obtenerse con 18000 se consigue con 14000 o en T12 con 9000 (en vez de 13000).

Capítulo 8

Conjunto de datos y resultados generación sintética

Por último, se presentará un problema de astroinformática algo diferente pero también relacionado con la morfología de las galaxias. En este capítulo se describirá el conjunto de datos y algunos resultados preliminares que se han ido obteniendo.

8.1. Descripción del conjunto de datos

Como se adelantó en la Sección 5.4, el conjunto de datos que se va a detallar ha sido generado de forma artificial por medio de simulaciones hidrodinámica. Estos datos provienen concretamente de la simulación realizada en [58] y han sido proporcionados por la investigadora Helena Domínguez Sánchez del Instituto de Ciencias del Espacio (ICE-CSIC).

Al ser una simulación, es posible realizar más fotografías de cada galaxia y en distintos instantes de tiempo. En este caso se han realizado tres capturas de cada galaxia en distintos instantes de tiempo, una por cada proyección (π) en 3D: xy, yz, xz. Para medir la distancia se utiliza el desplazamiento al rojo (z), donde el valor cero corresponde al momento actual y valores cada vez más grandes indica que la luz ha recorrido más distancia hasta poder verla, por lo que son más antiguas: por ejemplo $z = 0.10$ equivale a una galaxia de hace 1286 mil millones de años y una con $z = 1$ de 7731 mil millones de años.

Aparte del desplazamiento al rojo también se ha tomado una captura en distintos instantes (*snapshot*). Esto podría parecer lo mismo que el desplazamiento al rojo pero son conceptos distintos. Para ilustrarlo, podemos hacer la siguiente analogía: imaginemos que nos encontramos en el momento actual y que tenemos una persona que acaba de cumplir 66 años. Si pudiéramos viajar en el tiempo y trasladar a dicha persona cuando era un niño, con 7 años, al momento actual. Entonces, la persona adulta tendría un desplazamiento al rojo de $z = 0$ pero el niño (que ha viajado en el tiempo) tendría un desplazamiento al rojo $z > 0$ (estrictamente mayor). En cambio,

las dos personas están en este momento juntas, por lo que la captura se ha realizado en el mismo momento (y por tanto tendrá el mismo valor de *snapshot*).

Por último, cada galaxia, en cada proyección, cada instante de tiempo y desplazamiento al rojo se ha tomado con varios brillos superficiales distintos (μ).

Parámetros	**Valores**
μ	28, 29, 30, 31, 35
z	0.05, 0.1, 0.2, 0.4, 0.8
π	xy, yz, xz
snapshot	545, 658, 791, 904

Tabla 8.1: Distintos parámetros utilizados para capturar las galaxias

En la Tabla 8.1 se muestran los valores de los parámetros utilizados. En concreto, los brillos superficiales se corresponden con los límites que pueden detectar la mayor parte de los detectores de los telescopios, entre 29 y 31 (Euclid por ejemplo) como se puede observar en la Figura 8.1.

La muestra contiene 36 galaxias distintas, las cuales se han tomado en todas las combinaciones de μ, z, π y *snapshot* descritas, por lo que en total resultan 10800 imágenes distintas provistas en archivos FITS (uno por cada toma). Sin embargo, los datos disponibles de los que tenemos información y el archivo FITS son muchos menos: en total, en el archivo CSV con las etiquetas hay 5882; y archivos FITS 9432, pero muchos de ellos no están clasificados en el CSV. Por tanto, como combinación (tener información de la clasificación y el archivo de imagen) únicamente se encuentran disponibles 5835 instancias.

El objetivo de este estudio es encontrar, por medios de aprendizaje automático, formas de bajo brillo superficial que ocurren en las interacciones entre galaxias. Concretamente, las formas que se están buscando son: dobles núcleos, remanentes de fusión, puentes, colas de marea, *streams* y *shells* (nombradas de izquierda a derecha en la Figura 8.2, la categoría *plume* no se tiene en cuenta). Aparte, existe otra categoría denominada "miscelánea" que contiene imágenes sobre interacciones que no se ha sabido muy bien cómo catalogar o no está lo suficientemente claro.

A la hora de transformar las imágenes en primer lugar se utilizó una escala logarítmica pero cuando se consultó a los astrónomos decidieron utilizar una escala de argumento seno hiperbólico preferiblemente y después seleccionar el 97 % de los píxeles, eliminando el 3 % restante perteneciente a las colas de una Normal (los más atípicos, tanto los más luminosos como los menos).

Para cada combinación descrita en los valores de la Tabla 8.1 varios astrónomos han catalogado cada una de estas combinaciones (correspondientes a cierta galaxia, proyección, *snapshot*, *redshift* y μ determinado). En cada evaluación se señala cuántas características de las dichas anteriormente se

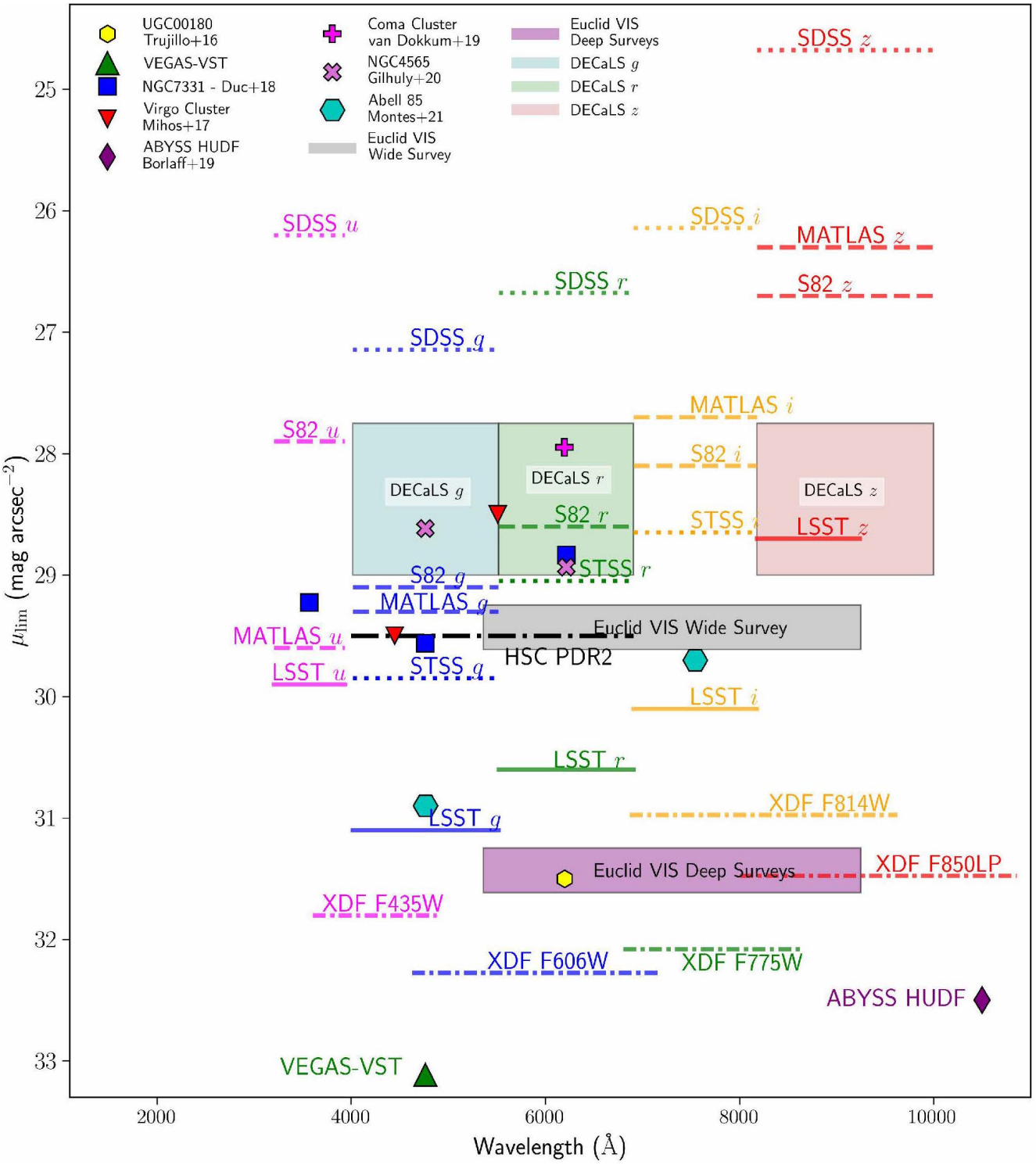


Figura 8.1: Límites del brillo superficial μ detectado por cada telescopio. Obtenido de [59]

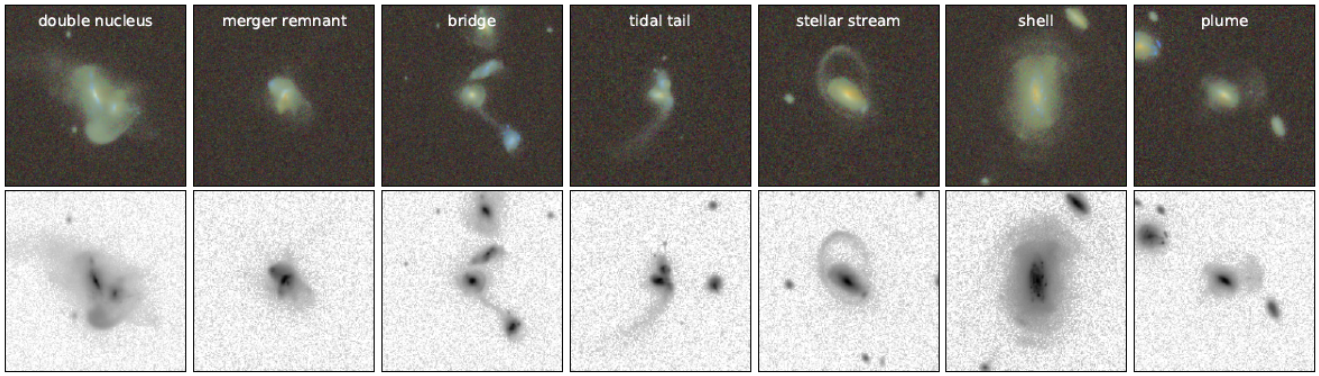


Figura 8.2: Ejemplos de imágenes de cada categoría, con dos filtros de color. Obtenido de [58]

encuentran en la imagen. Sea $y_{i,c,e}$ la clasificación de la característica $c \in \{\text{doble\_nucleo}, \text{puente}, ...\}$ de un evaluador e sobre la imagen i-ésima; entonces $y_{i,c,e} \in \mathbb{N} \cup \{0\}$. Nótese que una galaxia puede tener más de una característica del mismo o de distintos tipos.

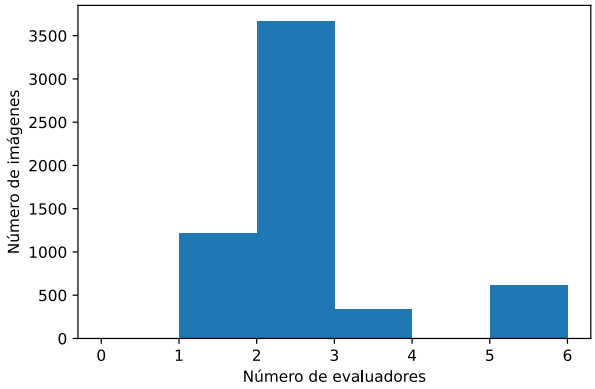
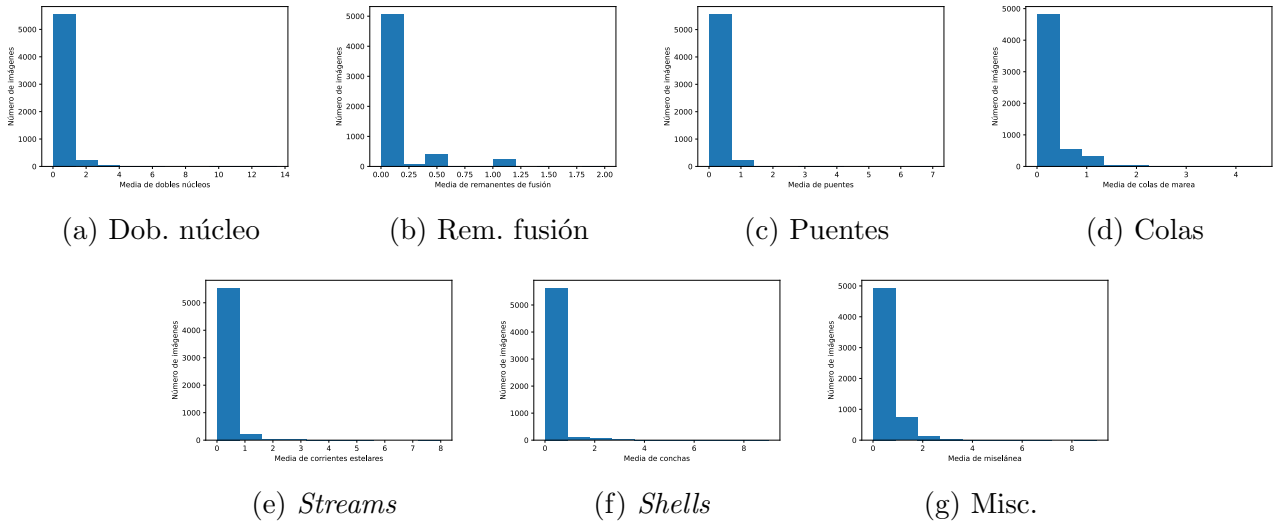


Figura 8.3: Histograma del recuento de imágenes por el número de evaluadores

En la Figura 8.3 se muestra un histograma con el recuento de imágenes evaluadas según el número de evaluadores. Se observa que la mayor parte de ellos están solo evaluadas por dos o una persona, es un número bastante bajo dado que la naturaleza de las características que estamos midiendo es algo subjetiva.

Para comprobar la distribución de cada característica, se ha calculado la media de cada c: $\bar{y}_{i,c} = \frac{1}{n_e} \sum_e y_{i,c,e}$ (n_e denota el número de evaluadores de la imagen i-ésima). Con estas medias se ha realizado un histograma para ver el recuento de imágenes frente a la media por característica (Figura 8.4). En esta figura se aprecia que la mayor parte de las imágenes no tienen ninguna característica de cada tipo, lo cual es un problema al tener una distribución desbalanceada.

Aparte de las dificultades anteriormente vistas, se añade que las imágenes tienen distintos tamaño de píxeles. En la Figura 8.3 se muestra un histograma con los distintos tamaños y el número



(a) Dob. núcleo (b) Rem. fusión (c) Puentes (d) Colas

(e) *Streams* (f) *Shells* (g) Misc.

Figura 8.4: Histogramas del recuento de imágenes según la media de cada categoría

de imágenes de cada uno. Dado que para entrenar un modelo CNN se necesita que todas tengan el mismo tamaño, se ha decidido aumentarlas o reducirlas a la mediana de los valores, que es 60×60 píxeles. Cuando se reduce la dimensión, se utiliza un suavizado gaussiano para evitar el *aliasing*. A la hora de aumentarla, los píxeles nuevos se generan realizando una media de los de su entorno. Para ambos se ha utilizado el paquete `scikit-image` [60].

Para "generar" más datos de forma artificial se han utilizado las mismas operaciones de *data augmentation* que las expuestas en la Sección 6.2.

La primera aproximación que se va a probar es realizar regresión. Para empezar, se ha simplificado el problema: en un primer momento se tenía la media de características encontradas en una imagen ($\bar{y}_{i,c}$), el hecho de que la red tenga que contar el número de características que aparecen puede ser complicado. Por ello, se ha facilitado realizando la siguiente transformación de las etiquetas: por cada imagen solamente nos interesa saber si hay o no cierta característica (no importa las veces que aparezca).

De esta forma, si partimos de los datos originales $y_{i,c,e}$ (antes de realizar la media por fotografía), generamos unos $y'_{i,c,e}$ tales que:

$$y'_{i,c,e} = \begin{cases} 1 & \text{si } y_{i,c,e} \geq 1 \\ 0 & \text{si } y_{i,c,e} = 0 \end{cases}$$

A continuación, como hay más de una evaluación por instancia, se ha calculado la media de ellas: $\bar{y}'_{i,c} = \frac{1}{n_e} \sum_e y_{i,c,e}$, que se usará como variable a predecir por la regresión. Como hay siete características distintas, la salida de cada modelo son siete valores en el rango $[0, 1]$.

Para comprobar que los modelos entrenen bien, se ha partido el conjunto de datos en dos:

uno para entrenamiento/validación y otro para test, el cual solo se utilizará para calcular las métricas finales. Esta separación se ha realizado de forma aleatoria, tomando 80 % para entrenamiento/validación y 20 % para test.

8.2. Resultados

En primer lugar e igual que con los datos de Euclid, el objetivo del primer experimento es determinar si hay diferencias significativas entre usar una arquitectura de red u otra. Para ello, se han considerado: de ResNet, 18, 50 y 101 capas; y de EfficientNet las configuraciones B0, B4 y B7. Se ha descartado la arquitectura GaMorNet al no dar buenos resultados en el conjunto de datos anterior.

De nuevo, se han utilizado tamaños de batch lo suficientemente grandes para que la pérdida converja y tardara en total cada modelo unas dos horas (Tabla 8.2)

	ResNet-18	**ResNet-50**	**ResNet-101**	**EfficientNet-B0**	**EfficientNet-B4**	**EfficientNet-B7**
Tamaño del batch	64	64	64	64	64	30
Número de épocas	700	420	300	700	300	150

Tabla 8.2: Tamaño del batch y número de épocas usado para entrenar cada arquitectura de red usando regresión

Al ser un problema de regresión, se ha utilizado el MSE como función de pérdida, RMSE como métrica (dado que es más interpretable que el MSE) y Adam como optimizador, al funcionar correctamente en el capítulo anterior.

Modelo	**RMSE**	**RMSE streams**	**RMSE shells**	**RMSE tails**	**RMSE bridge**	**RMSE merger remnant**	**RMSE double nuclei**	**RMSE misc**
resnet18	0.24	0.189	0.134	0.266	0.237	0.2	0.301	0.356
resnet50	0.236	0.196	0.136	0.26	0.223	0.196	0.283	0.361
resnet101	0.238	0.18	0.136	0.264	0.231	0.202	0.296	0.358
efficientnet_b0	0.238	0.196	0.134	0.264	0.228	0.195	0.291	0.355
efficientnet_b4	0.22	0.171	0.123	0.242	0.214	0.19	0.276	0.326
efficientnet_b7	0.218	0.169	0.119	0.245	0.21	0.183	0.275	0.323
Media	0.232	0.184	0.130	0.257	0.224	0.194	0.287	0.346

Tabla 8.3: Métricas calculadas sobre el conjunto de test

En la Tabla 8.3 se muestran los valores de la raíz cuadrada del error cuadrático medio (RMSE) sobre el conjunto de test para cada una de las características. En general, todos los modelos arrojan resultados bastante pobres: el RMSE promedio mejor es del 0.218, correspondiente a EfficientNet-B7. Si nos fijamos en cada categoría, se aprecia que algunas ajustan mejor que otras: en primer lugar se encuentran *shells* con un RMSE medio de 0.13, seguidas de *streams* (RMSE medio de 0.184). Por contraposición, la que peor predice sería las de la categoría miscelánea, obteniendo un RMSE medio de 0.346. Este comportamiento es lógico al ser una categoría que agrupa los objetos que no se sabe muy bien cómo catalogar. La categoría de doble núcleo es la segunda peor con respecto al RMSE, con una media de 0.287.

Como con ninguna arquitectura de red se consiguen buenos resultados, se ha decidido cambiar la forma de atacar el problema. En vez de realizar una regresión múltiple, se van a crear tantos modelos como características que queramos predecir (siete en total) dada una arquitectura de red. Como hay seis arquitecturas distintas, en total se crearán 42 modelos, cada uno con una regresión simple en el rango $[0, 1]$.

En la Figura 8.5 se muestran ejemplos de predicciones del modelo ResNet-50 para el conjunto de test. Para alguna de las imágenes sí que se parecen en gran medida a las reales, en general para aquellas que tienen valores de 0 en cada característica (consecuencia del desequilibrio en el conjunto de datos). También se aprecia el ruido introducido por la ampliación de alguna fotografía, haciendo muy complicada la clasificación, incluso manual.

Modelo	RMSE streams	RMSE shells	RMSE tails	RMSE bridge	RMSE merger remnant	RMSE double nuclei	RMSE misc
resnet18	0.223	0.161	0.277	0.242	0.228	0.308	0.371
resnet50	0.217	0.172	0.273	0.252	0.217	0.317	0.376
resnet101	0.199	0.146	0.275	0.245	0.222	0.301	0.374
efficientnet_b0	0.211	0.163	0.273	0.238	0.22	0.306	0.367
efficientnet_b4	0.19	0.14	0.25	0.234	0.201	0.306	0.355
efficientnet_b7	0.196	0.161	0.275	0.235	0.211	0.291	0.327

Tabla 8.4: Métricas calculadas sobre el conjunto de test usando un modelo distinto por cada característica

Al realizar una regresión simple por cada modelo se han obtenido los resultados mostrados en la Tabla 8.4. Para las cuatro primeras características (*streams*, *shells*, *tails* y *bridge*) comparando con los datos anteriores se reduce el RMSE. Por ejemplo en ResNet-18 para *shells* se ha reducido el RMSE en 0.028 y en *streams* (mismo modelo) en 0.017. Conseguir buenos resultados en estas dos características seguramente se deba a que son más sencillas de caracterizar y menos difusas. En cambio, para las últimas características (*merger remnant*, *double nuclei* y *misc*) parece que las métricas aumentan, como por ejemplo en el caso de EfficientNet-B7, cuyo aumento sobre *merger remnant* es de 0.028 o en *double nuclei* de 0.016.

El modelo que parece funcionar mejor que los demás es EfficientNet-B4, pero sin grandes diferencias, siendo el que mejor funciona según la métrica considerada para todas las características excepto *double nuclei* y *misc*.

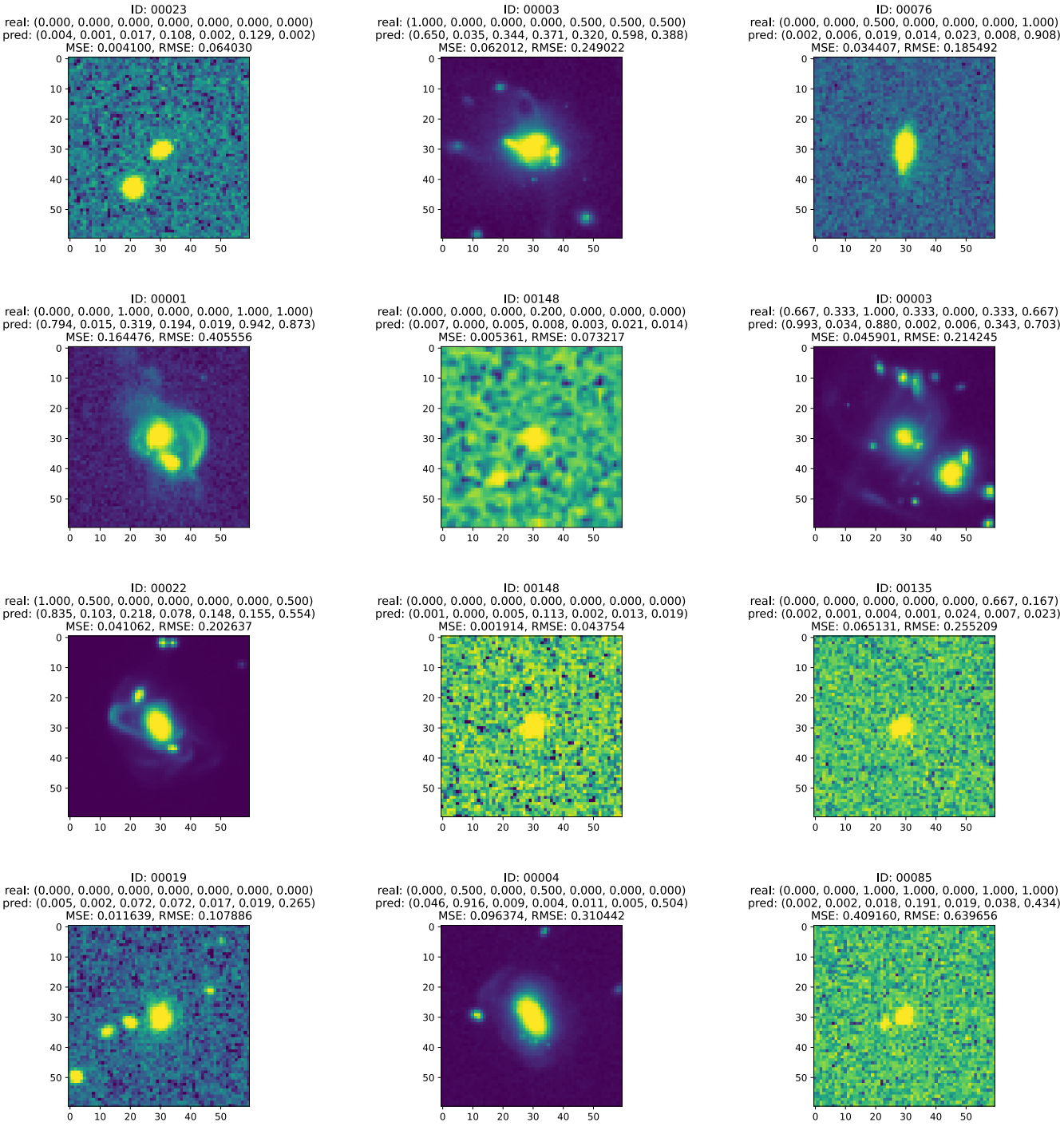


Figura 8.5: Ejemplo de predicciones de ResNet-50 sobre el conjunto de test para el modelo de regresión múltiple

Capítulo 9

Conclusiones y líneas futuras

9.1. Conclusiones

En este trabajo se han estudiado, analizado e implementado soluciones basadas en algoritmos de *deep learning*, en particular redes neuronales convolucionales aplicadas al ámbito de la astronomía. Se han aplicado sobre dos conjuntos de datos diferentes: el primero de ellos obtenido a través de la modificación de imágenes del telescopio espacial Hubble para emular las lentes del nuevo telescopio Euclid y poder entrenar modelos cuando dicho satélite esté en órbita y envíe fotografías. El segundo de ellos, se ha generado a través de simulaciones hidrodinámicas para emular los eventos que ocurren en la interacción entre dos o más galaxias.

Sobre el conjunto de datos de la colaboración Euclid se han obtenido buenos resultados, llegando a conseguir una tasa de acierto y valor F1 superiores al 98 % para la cuestión de determinar la forma de la galaxia (T01). A la hora de reconocer si una galaxia podía tener forma de disco visto de canto (T02) también se obtenían resultados bastante buenos, del 97 % de tasa de acierto y 95 % en el valor F1. Sin embargo, para ciertos problemas los resultados han sido algo peores como determinar si la galaxia tiene una apariencia grumosa (T12) y comprobar si tiene una forma de barra en su centro (T03). No obstante, en todo caso se consiguen una tasa de acierto y valor F1 superiores al 80 % sobre el conjunto de test. Por último, a la hora de identificar si hay signos de brazos en espiral (T04) los resultados han sido pobres, especialmente sobre sobre la segunda clase (*no espiral*), consiguiéndose un valor F1 del 64 % y una tasa de acierto del 93 %. Esto se debe al desbalanceo del conjunto de datos, con apenas instancias de dicha clase.

En la segunda parte tratada de este mismo conjunto (*Cuestión 2*), sobre el número mínimo de galaxias necesario para obtener el máximo rendimiento (según las métricas escogidas), se ha llegado a la conclusión que es necesario un número bastante bajo al del conjunto de datos completo, tanto para *clean dataset* como para *all galaxies dataset*. Por ejemplo, para *all galaxies dataset* y la cuestión correspondiente a determinar la forma de la galaxia (T01), con 7000 observaciones se obtienen resultados similares (con una confianza del 95 %) a entrenar el modelo con 14000, lo

que supone la mitad. Este resultado es de gran utilidad al poder escoger las evaluaciones sobre imágenes más fiables, pudiendo concentrar los esfuerzos de etiquetado, muchas veces limitados dado al bajo número de profesionales disponibles para estas tareas.

Después se han generado unos resultados preliminares sobre datos de generación sintética, obtenidos de una simulación hidrodinámica. En este caso el objetivo no era responder a una serie de preguntas binarias, sino detectar la aparición de estructuras de bajo brillo superficial como pueden ser colas de marea, dobles núcleos o *streams*. En este caso los resultados han sido más pobres (aunque no dejan de ser preliminares), en el mejor de los casos con un RMSE sobre el conjunto de test del 0.14 en la característica *shells*. Estos resultados se deben a dos cuestiones fundamentalmente: la primera por culpa de las etiquetas originales, las cuales no han sido tan refinadas como en el caso de Euclid y pueden ser incorrectas en algún caso, provocando que la red no aprenda correctamente. La segunda se debe a la calidad de las imágenes dado que muchas han tenido que ser "agrandadas" para que tengan el mismo tamaño que las demás (y la otra mitad han tenido que ser reducidas). Si a esto le añadimos que el objetivo es buscar formas confusas y subjetivas, que dos expertos pueden catalogar como diferentes, nos encontramos ante un problema de una dificultad importante.

9.2. Líneas futuras

Los buenos resultados sobre el conjunto de datos de la colaboración Euclid demuestra el potencial uso que tiene la informática y en concreto la inteligencia artificial para catalogar las galaxias según su morfología, a través de preguntas binarias. No obstante, sobre algunas de las tareas aún existe un amplio margen de mejora que se puede acortar utilizando modelos más elaborados como puede ser la construcción de ensembles, por ejemplo utilizando *boosting*.

También se pueden explorar otros algoritmos de *active learning* para identificar las galaxias más complicadas de clasificar y así centrar la atención en mejorar las etiquetas de dichas observaciones, dado que la mayor parte de ellas parecen etiquetarse correctamente. Aunque es complicado aplicar esta técnica en las redes neuronales dado que no dan una medida de la incertidumbre de la clasificación, es posible utilizar otros modelos de redes como son las bayesianas para obtener dicha medida [61]. No obstante, la continuación sobre estos resultados está a expensas de qué línea de trabajo se determine a seguir por parte de la colaboración Euclid.

Sobre la detección de estructuras de bajo brillo superficial queda como trabajo pendiente continuar en la exploración de técnicas para mejorar los resultados, para en un futuro publicar un *paper* conjunto con Helena Domínguez. En primer lugar se intentará simplificar el problema a una clasificación en tres niveles para cada característica: si se está seguro que no existe en la imagen, por ejemplo si la media de los evaluadores es cercana a 0 (límite en < 0.2, por ejemplo); si se está seguro de que existe en la galaxia observada, pudiendo tomar la media de los evaluadores

superior al 0.8 (cercano a 1) y si existen dudas, que son los casos intermedios. Si dicho modelo aún no funcionara correctamente según las métricas de la tasa de acierto y el valor F1, se puede simplificar aún más el problema, eliminando la clase intermedia y entrenando únicamente con los extremos.

Otra posibilidad es investigar sobre el preprocesamiento de las imágenes, dado que en este caso se tienen fotografías de distintos tamaños y se puedan ampliar (y reducir) utilizando otros algoritmos o escalas o utilizar solo las de mayor calidad porque al aumentarlas se añade ruido y esto puede ser lo que esté empeorando la capacidad de generalización del modelo.

Bibliografía

[1] Project Management Institute, *A guide to the project management body of knowledge (PMBOK guide)*, eng, 6th ed. Newtown Square, Penn., USA: Project Management Institute, 2017, pág. 756, ISBN: 978-1-62825-184-5.

[2] Agencia Tributaria, *3.5.4 Tabla de amortización simplificada*. dirección: https://sede.agenciatributaria.gob.es/Sede/ayuda/manuales-videos-folletos/manuales-practicos/folleto-actividades-economicas/3-impuesto-sobre-renta-personas-fisicas/3_5-estimacion-directa-simplificada/3_5_4-tabla-amortizacion-simplificada.html (visitado 25-04-2022).

[3] T. M. Mitchell, *Machine learning*. New York: MacGraw-Hill, 1997, ISBN: 0070428077.

[4] S. J. Russell, *Artificial Intelligence : A modern approach* (Prentice Hall series in artificial intelligence), eng, 3rd. ed., P. Norving, ed. Upper Saddle River, New Jersey: Prentice-Hall, 2010, ISBN: 978-0-13-207148-2.

[5] M. A. Nielsen, *Neural Networks and Deep Learning*, 2015. dirección: http://neuralnetworksanddeeplearning.com (visitado 16-09-2021).

[6] P. Isasi Viñuela e I. M. Galván León, *Redes de neuronas artificiales: un enfoque Práctico*, I. M. Galván León, ed. Madrid: Pearson, 2003, pág. 248, ISBN: 84-205-4025-0.

[7] W. S. McCulloch y W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The Bulletin of Mathematical Biophysics*, vol. 5, n.º 4, págs. 115-133, 1943, ISSN: 0007-4985. DOI: 10.1007/BF02478259.

[8] J. Howard y S. Gugger, *Deep Learning for Coders with Fastai and Pytorch: AI Applications Without a PhD*. O'Reilly Media, Incorporated, 2020, ISBN: 9781492045526.

[9] G. Hinton, *Lecture 6e - rmsprop: Divide the gradient by a running average of its recent magnitude*, 2012. dirección: http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf (visitado 27-04-2022).

[10] D. P. Kingma y J. Ba, "Adam: A Method for Stochastic Optimization," 2014. arXiv: 1412.6980.

[11] A. C. Wilson, R. Roelofs, M. Stern, N. Srebro y B. Recht, "The Marginal Value of Adaptive Gradient Methods in Machine Learning," 2017. arXiv: 1705.08292.

[12] I. Goodfellow, Y. Bengio y A. Courville, *Deep learning* (Adaptive computation and machine learning), eng. Cambridge (Massachusetts): MIT Press, 2016, pág. 775, ISBN: 978-0-262-03561-3.

[13] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever y R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, págs. 1929-1958, 2014, ISSN: 15337928. dirección: https://jmlr.org/papers/v15/srivastava14a.html.

[14] L. N. Smith, "Cyclical Learning Rates for Training Neural Networks," en *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, IEEE, 2017, págs. 464-472. DOI: 10.1109/WACV.2017.58. arXiv: 1506.01186.

[15] L. N. Smith, "A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay," 2018. arXiv: 1803.09820.

[16] S. Ioffe y C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," 2015. arXiv: 1502.03167.

[17] A. Krizhevsky, I. Sutskever y G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," en *Advances in Neural Information Processing Systems*, F. Pereira, C. J. Burges, L. Bottou y K. Q. Weinberger, eds., vol. 25, Curran Associates, Inc., 2012. dirección: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[18] A. Krizhevsky, "One weird trick for parallelizing convolutional neural networks," 2014. arXiv: 1404.5997.

[19] K. He, X. Zhang, S. Ren y J. Sun, "Deep Residual Learning for Image Recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 2016-Decem, págs. 770-778, 2016, ISSN: 10636919. DOI: 10.1109/CVPR.2016.90. arXiv: 1512.03385.

[20] M. Tan y Q. V. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," 2019. arXiv: 1905.11946.

[21] A. Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library," en *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox y R. Garnett, eds., Curran Associates, Inc., 2019, págs. 8024-8035. dirección: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[22] C. R. Harris et al. "Array programming with NumPy," *Nature*, vol. 585, n.º 7825, págs. 357-362, 2020. DOI: 10.1038/s41586-020-2649-2.

[23] J. Howard et al. *fastai*, 2018. dirección: https://github.com/fastai/fastai.

[24] *FastAI Documentation*. dirección: https://docs.fast.ai/ (visitado 20-04-2022).

[25] J. Deng, W. Dong, R. Socher, L.-J. Li, Kai Li y Li Fei-Fei, "ImageNet: A large-scale hierarchical image database," págs. 248-255, 2010. DOI: 10.1109/CVPR.2009.5206848.

[26] Z. Mueller, *"fine_tune" vs. "fit_one_cycle"*, 2020. dirección: https://forums.fast.ai/t/fine-tune-vs-fit-one-cycle/66029/2 (visitado 19-04-2022).

[27] Comet.ML. "Comet.ML home page." (2021), dirección: https://www.comet.ml/ (visitado 29-03-2022).

[28] J. Horst Frank. "Espectro electromagnético." (2007), dirección: https://commons.wikimedia.org/wiki/File:Electromagnetic_spectrum-es.svg.

[29] M. Postman et al. "THE CLUSTER LENSING AND SUPERNOVA SURVEY WITH HUBBLE: AN OVERVIEW," *The Astrophysical Journal Supplement Series*, vol. 199, n.º 2, pág. 25, 2012, ISSN: 0067-0049. DOI: 10.1088/0067-0049/199/2/25.

[30] C. Inserra et al. "Euclid: Superluminous supernovae in the Deep Survey," *Astronomy & Astrophysics*, vol. 609, A83, 2018, ISSN: 0004-6361. DOI: 10.1051/0004-6361/201731758.

[31] K. D. Borne, "Astroinformatics: data-oriented astronomy research and education," *Earth Science Informatics*, vol. 3, n.º 1-2, págs. 5-17, 2010, ISSN: 1865-0473. DOI: 10.1007/s12145-010-0055-2.

[32] A. Carballo-Meilan, L. McDonald, W. Pragot, L. M. Starnawski, A. N. Saleemi y W. Afzal, "Development of a data-driven scientific methodology: From articles to chemometric data products," *Chemometrics and Intelligent Laboratory Systems*, vol. 225, pág. 104 555, 2022, ISSN: 0169-7439. DOI: 10.1016/J.CHEMOLAB.2022.104555.

[33] NASA/GSFC, *FITS Support Office*. dirección: https://fits.gsfc.nasa.gov/ (visitado 18-05-2022).

[34] W. A. Joye y E. Mandel, "New Features of SAOImage DS9," en *Astronomical Data Analysis Software and Systems XII*, vol. 295, 2003, pág. 489, ISBN: 1050-3390. dirección: https://ui.adsabs.harvard.edu/abs/2003ASPC..295..489J.

[35] ESA, *Datasets for education and for fun — ESA/Hubble*. dirección: https://esahubble.org/projects/fits_liberator/eagledata/ (visitado 19-05-2022).

[36] T. P. Robitaille et al. "Astropy: A community Python package for astronomy," *Astronomy & Astrophysics*, vol. 558, A33, 2013, ISSN: 0004-6361. DOI: 10.1051/0004-6361/201322068. arXiv: 1307.6212 [astro-ph.IM].

[37] A. M. Price-Whelan et al. "The Astropy Project: Building an Open-science Project and Status of the v2.0 Core Package," *The Astronomical Journal*, vol. 156, n.º 3, pág. 123, 2018, ISSN: 1538-3881. DOI: 10.3847/1538-3881/aabc4f. arXiv: 1801.02634 [astro-ph.IM].

[38] E. S. Agency, *ESA Science & Technology - Euclid*. dirección: https://sci.esa.int/web/euclid (visitado 15-06-2022).

[39] E. P. Hubble, "Extragalactic nebulae.," *The Astrophysical Journal*, vol. 64, pág. 321, 1926, ISSN: 0004-637X. DOI: 10.1086/143018.

[40] E. P. Hubble, *Realm of the Nebulae*. 1936, ISBN: 9780300025002.

[41] Y. Dubois et al. "Introducing the NEWHORIZON simulation: Galaxy properties with resolved internal dynamics across cosmic time," *Astronomy and Astrophysics*, vol. 651, A109, 2021, ISSN: 0004-6361. DOI: 10.1051/0004-6361/202039429.

[42] M. Huertas-Company et al. "A CATALOG OF VISUAL-LIKE MORPHOLOGIES IN THE 5 CANDELS FIELDS USING DEEP LEARNING," *The Astrophysical Journal Supplement Series*, vol. 221, n.º 1, pág. 8, 2015, ISSN: 0067-0049. DOI: 10.1088/0067-0049/221/1/8.

[43] S. Dieleman, K. W. Willett y J. Dambre, "Rotation-invariant convolutional neural networks for galaxy morphology prediction," *Monthly Notices of the Royal Astronomical Society*, vol. 450, n.º 2, págs. 1441-1459, 2015, ISSN: 0035-8711. DOI: 10.1093/MNRAS/STV632. arXiv: 1503.07077.

[44] H. Domínguez Sánchez, M. Huertas-Company, M. Bernardi, D. Tuccillo y J. L. Fischer, "Improving galaxy morphologies for SDSS with Deep Learning," *Monthly Notices of the Royal Astronomical Society*, vol. 476, n.º 3, págs. 3661-3676, 2018, ISSN: 0035-8711. DOI: 10.1093/MNRAS/STY338. arXiv: 1711.05744.

[45] A. Ghosh, C. M. Urry, Z. Wang, K. Schawinski, D. Turp y M. C. Powell, "Galaxy Morphology Network: A Convolutional Neural Network Used to Study Morphology and Quenching in 100,000 SDSS and 20,000 CANDELS Galaxies," *The Astrophysical Journal*, vol. 895, n.º 2, pág. 112, 2020, ISSN: 0004-637X. DOI: 10.3847/1538-4357/AB8A47. arXiv: 2006.14639.

[46] J. Vega-Ferrero et al. "Pushing automated morphological classifications to their limits with the Dark Energy Survey," *Monthly Notices of the Royal Astronomical Society*, vol. 506, n.º 2, págs. 1927-1943, 2021, ISSN: 0035-8711. DOI: 10.1093/MNRAS/STAB594.

[47] J. Fernández Iglesias, "Desarrollo de Técnicas Basadas en Frameworks Deep Learning para la Caracterización de Galaxias Distantes," 2021. dirección: https://uvadoc.uva.es/handle/10324/50236.

[48] K. W. Willett et al. "Galaxy Zoo: morphological classifications for 120 000 galaxies in HST legacy imaging," *Monthly Notices of the Royal Astronomical Society*, vol. 464, n.º 4, págs. 4176-4203, 2017, ISSN: 0035-8711. DOI: 10.1093/mnras/stw2568.

[49] M. Walmsley et al. "Galaxy Zoo: probabilistic morphology through Bayesian CNNs and active learning," *Monthly Notices of the Royal Astronomical Society*, vol. 491, n.º 2, págs. 1554-1574, 2020, ISSN: 0035-8711. DOI: 10.1093/mnras/stz2816. arXiv: 1905.07424.

[50] J. Barrio Conde, "Análisis y experimentación práctica de Frameworks Deep Learning aplicados a la astronomía," 2021. dirección: https://uvadoc.uva.es/handle/10324/50088.

[51] J. Demšar, "Statistical Comparisons of Classifiers over Multiple Data Sets," *Journal of Machine Learning Research*, vol. 7, págs. 1-30, 2006. DOI: 10.5555/1248547.1248548.

[52] M. Friedman, "The Use of Ranks to Avoid the Assumption of Normality Implicit in the Analysis of Variance," *Journal of the American Statistical Association*, vol. 32, n.º 200, pág. 675, 1937, ISSN: 01621459. DOI: 10.2307/2279372.

[53] R. L. Iman y J. M. Davenport, "Approximations of the critical region of the fbietkan statistic," *Communications in Statistics - Theory and Methods*, vol. 9, n.º 6, págs. 571-595, 1980, ISSN: 0361-0926. DOI: 10.1080/03610928008827904.

[54] Y. Cui, M. Jia, T. Y. Lin, Y. Song y S. Belongie, "Class-Balanced Loss Based on Effective Number of Samples," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2019-June, págs. 9260-9269, 2019, ISSN: 10636919. DOI: 10.1109/CVPR.2019.00949. arXiv: 1901.05555.

[55] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva y A. Torralba, "Learning Deep Features for Discriminative Localization," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2016-Decem, págs. 2921-2929, 2015, ISSN: 10636919. arXiv: 1512.04150.

[56] R. R. Selvaraju, A. Das, R. Vedantam, M. Cogswell, D. Parikh y D. Batra, "Grad-CAM: Why did you say that?," 2016. arXiv: 1611.07450.

[57] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva y A. Torralba, "Object Detectors Emerge in Deep Scene CNNs," *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 2014. arXiv: 1412.6856.

[58] G. Martin et al. "Preparing for low surface brightness science with the Vera C. Rubin Observatory: characterisation of tidal features from mock images," 2022. DOI: 10.1093/mnras/stac1003. arXiv: 2203.07675.

[59] A. S. Borlaff et al. "Euclid preparation: XVI. Exploring the ultra-low surface brightness Universe with Euclid /VIS," *Astronomy and Astrophysics*, vol. 657, A92, 2022, ISSN: 14320746. DOI: 10.1051/0004-6361/202141935. arXiv: 2108.10321.

[60] S. Van Der Walt et al. "Scikit-image: Image processing in python," *PeerJ*, vol. 2014, n.º 1, e453, 2014, ISSN: 21678359. DOI: 10.7717/PEERJ.453/FIG-5.

[61] Y. Gal, R. Islam y Z. Ghahramani, "Deep Bayesian Active Learning with Image Data," *34th International Conference on Machine Learning, ICML 2017*, vol. 3, págs. 1923-1932, 2017. arXiv: 1703.02910.

Apéndice A

Gráficos de regresión

A.1. Evolución de la función de pérdida

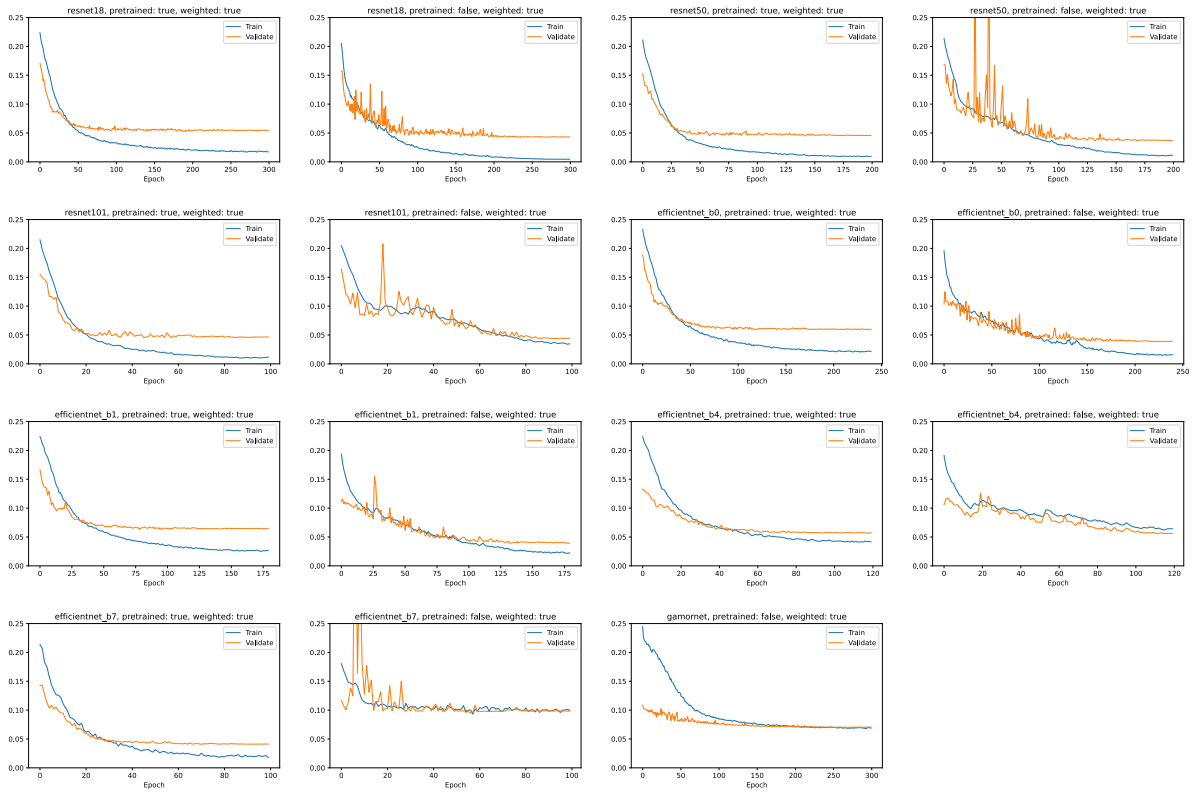


Figura A.1: Evolución de la función de pérdida de entrenamiento y validación con T12 y *clean dataset*

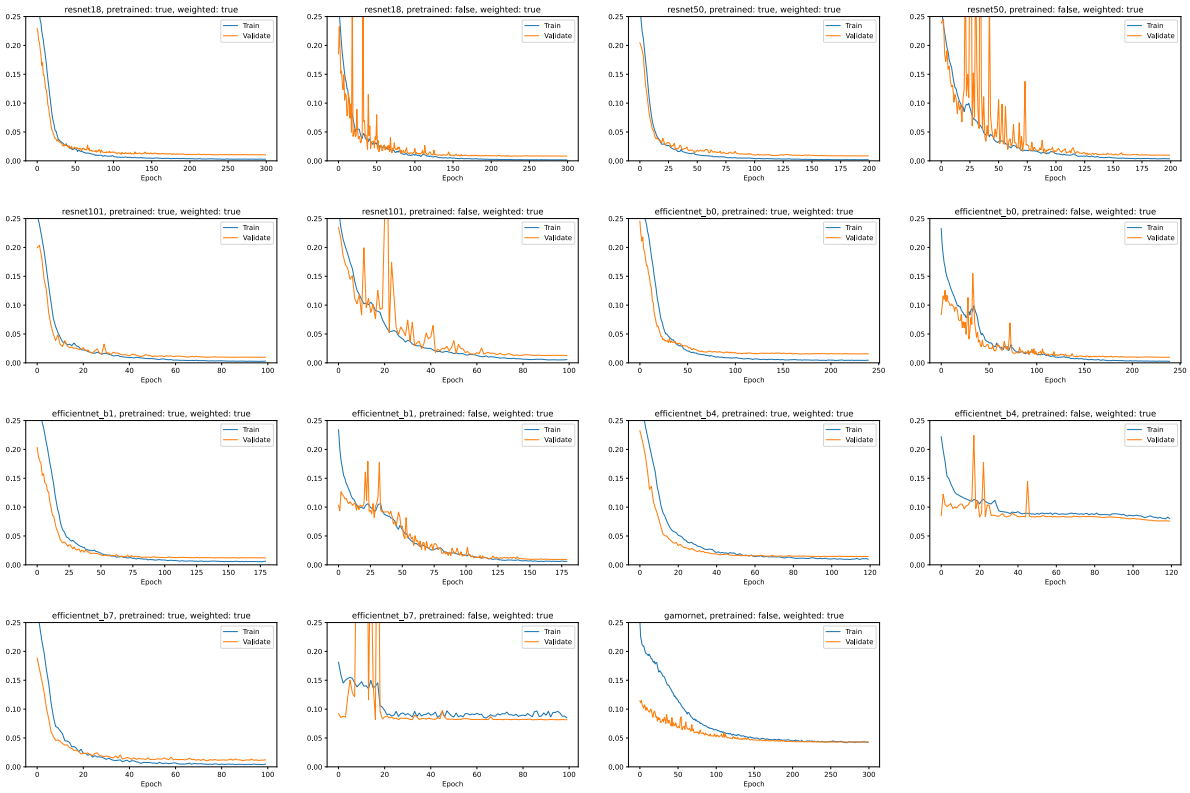


Figura A.2: Evolución de la función de pérdida de entrenamiento y validación con T02 y *clean dataset*

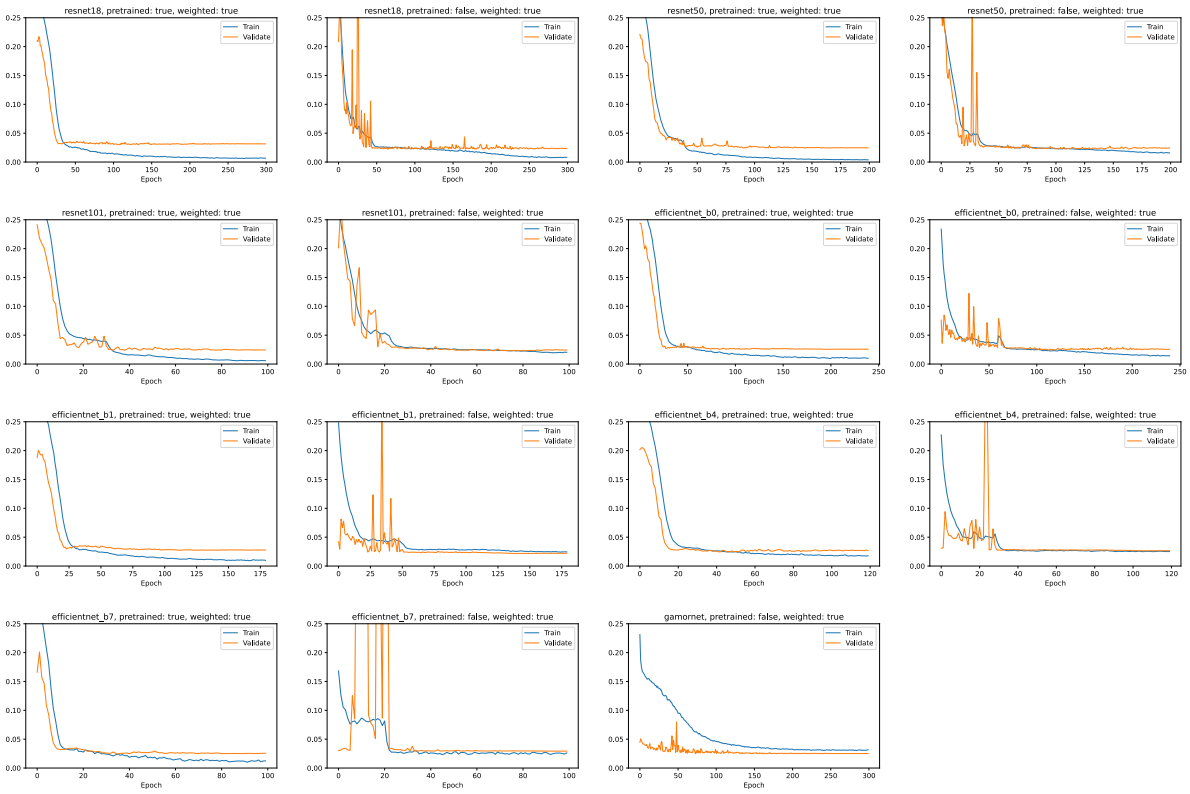


Figura A.4: Evolución de la función de pérdida de entrenamiento y validación con T04 y *clean dataset*

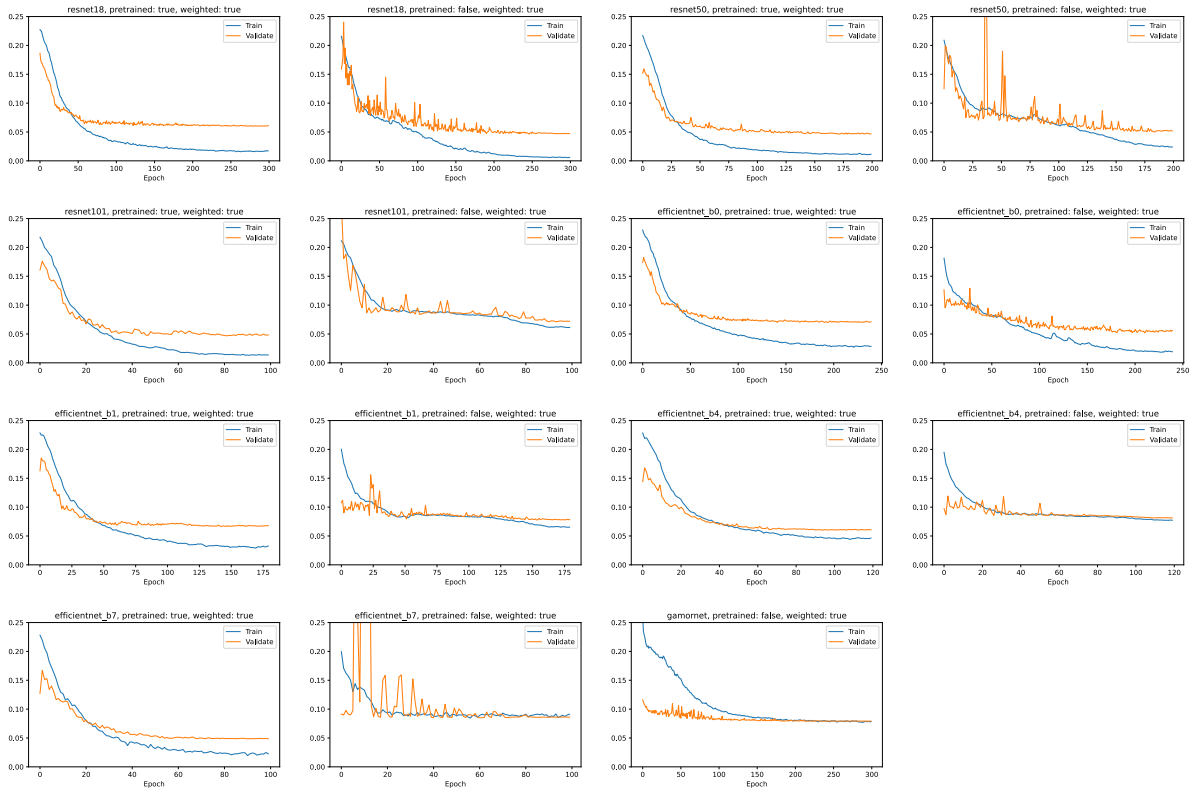


Figura A.3: Evolución de la función de pérdida de entrenamiento y validación con T03 y *clean dataset*

A.2. Matrices de confusión

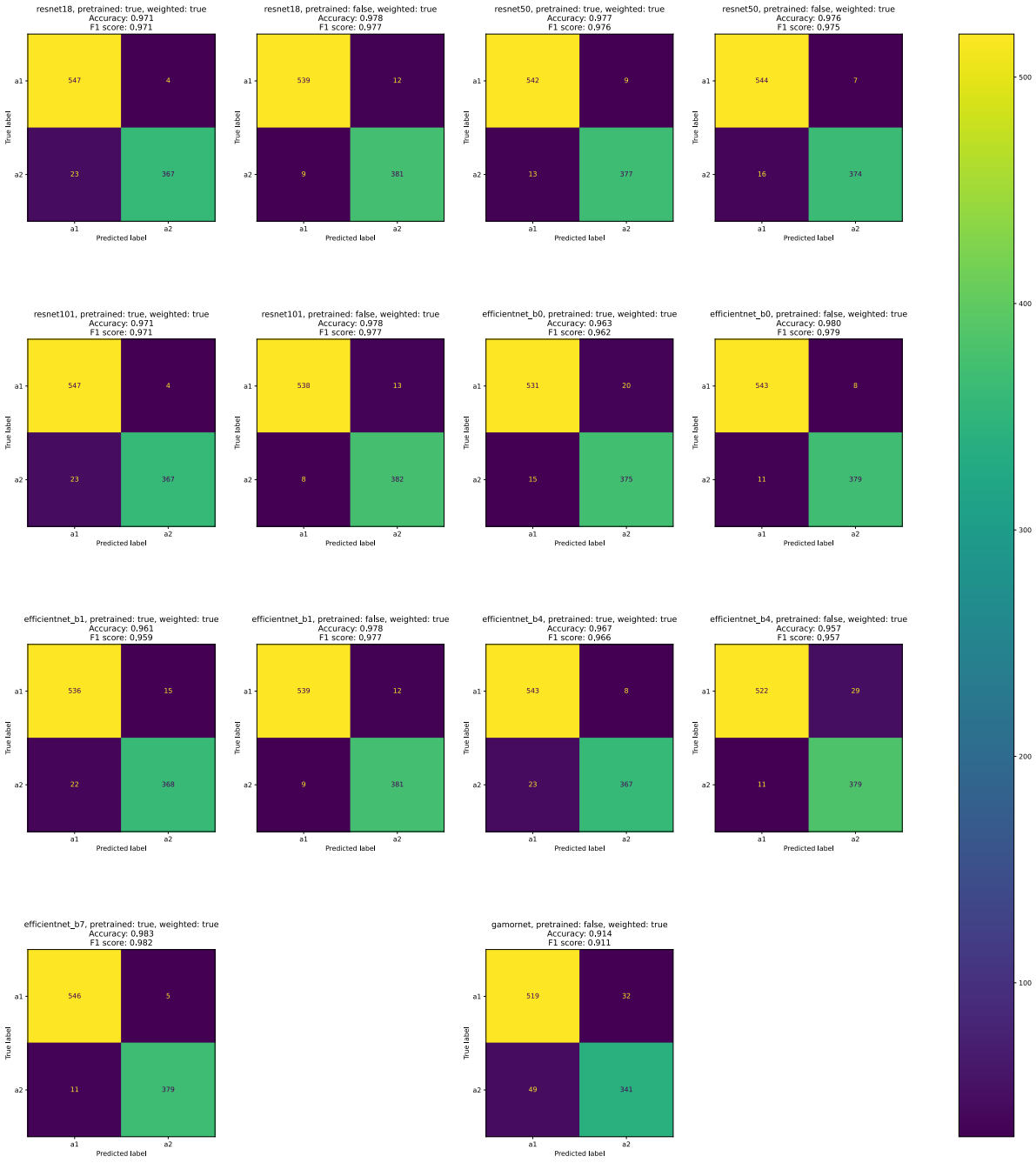


Figura A.5: Matrices de confusión sobre test para T01 y *clean dataset*

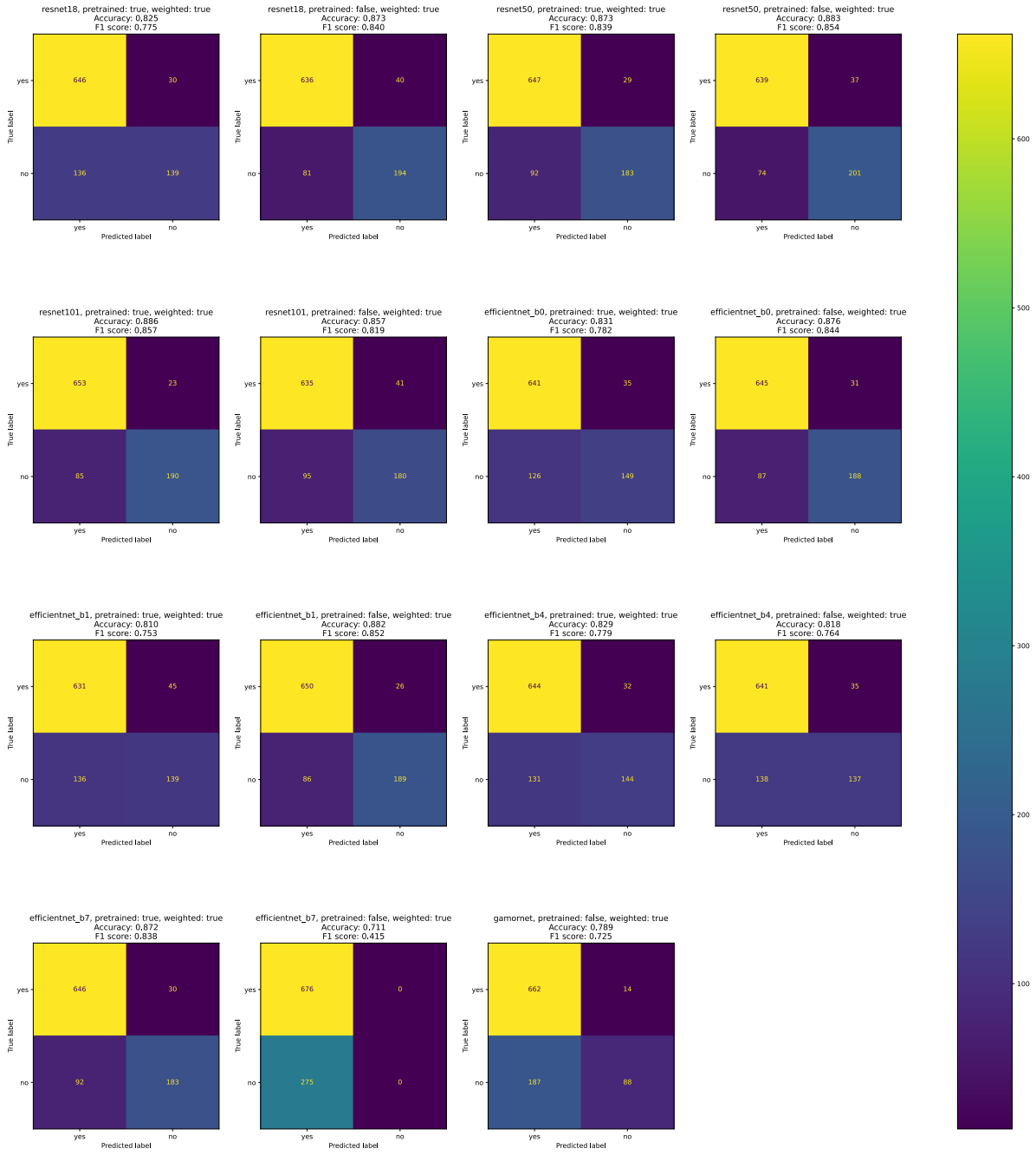


Figura A.6: Matrices de confusión sobre test para T12 y *clean dataset*

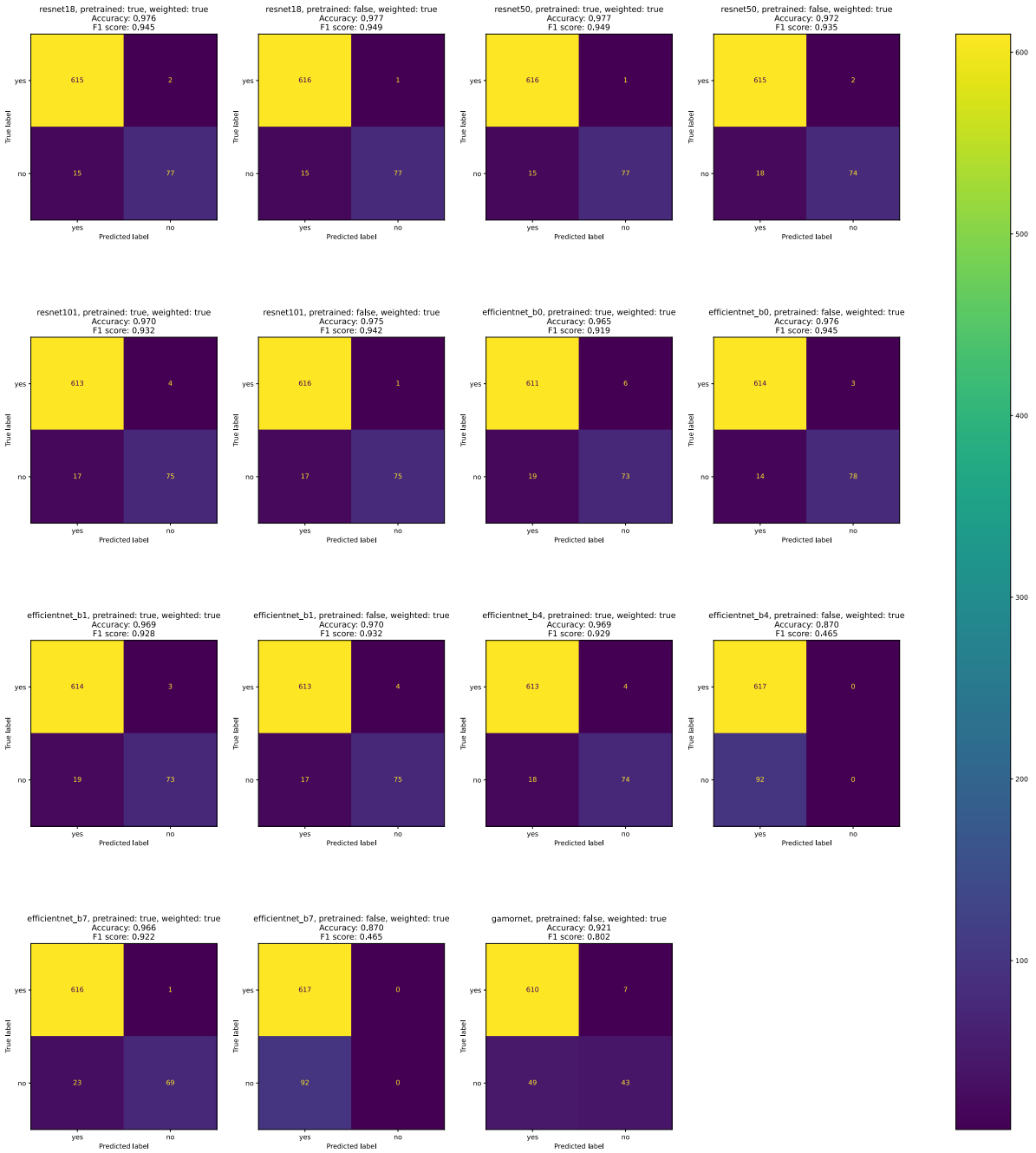


Figura A.7: Matrices de confusión sobre test para T02 y *clean dataset*

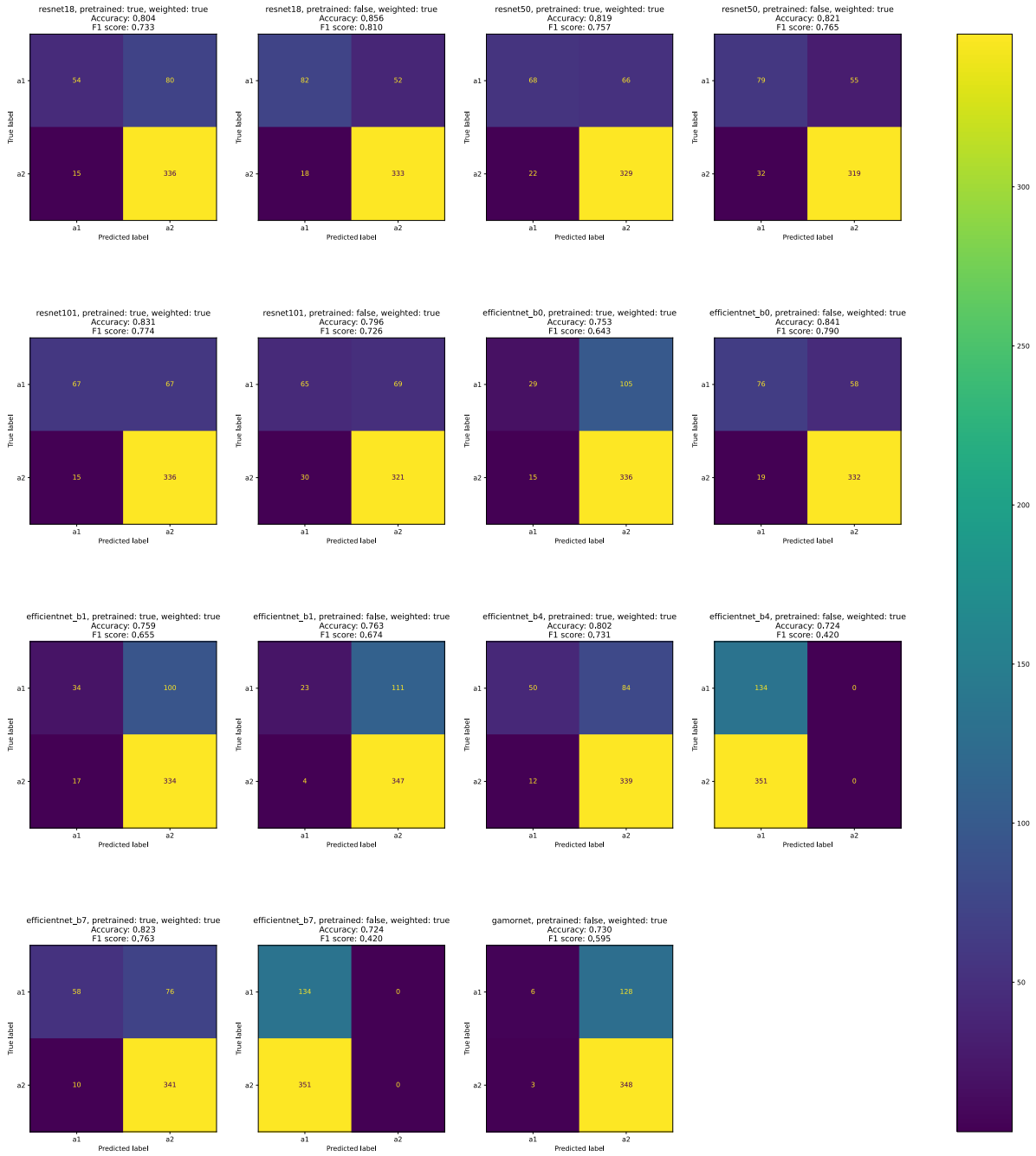


Figura A.8: Matrices de confusión sobre test para T03 y *clean dataset*

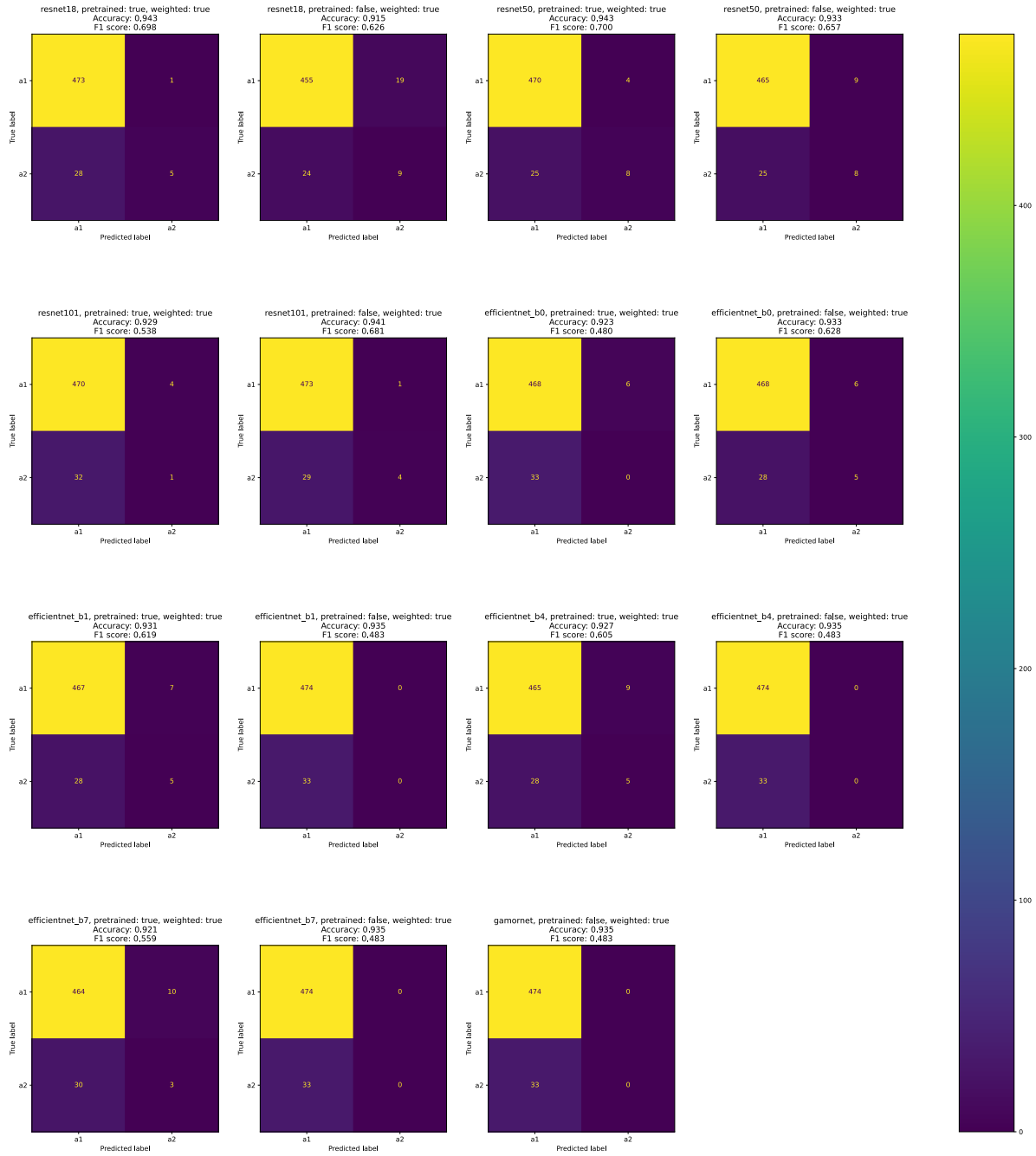
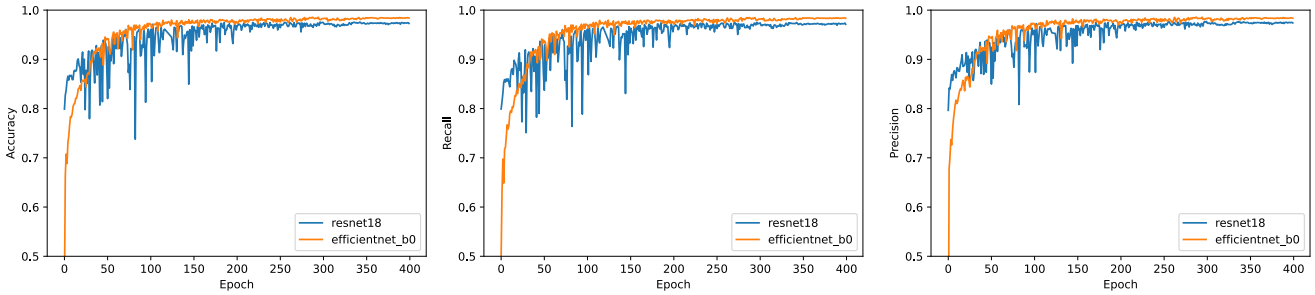


Figura A.9: Matrices de confusión sobre test para T04 y *clean dataset*

Apéndice B

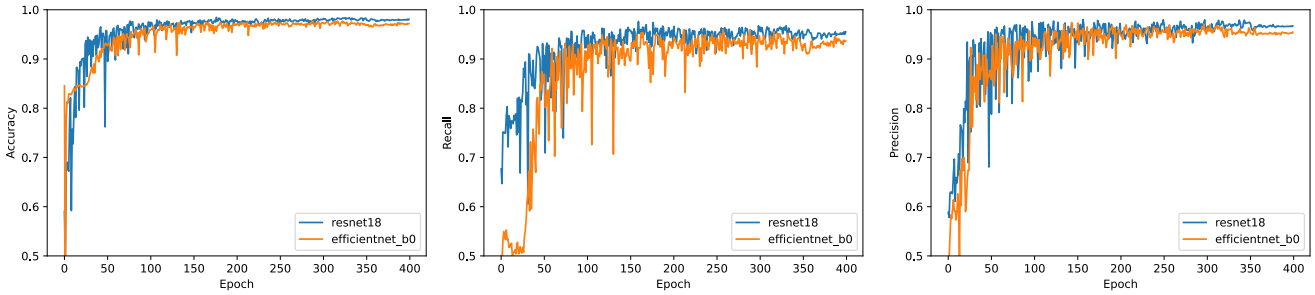
Gráficos de clasificación

B.1. Evolución de las métricas para *clean dataset*



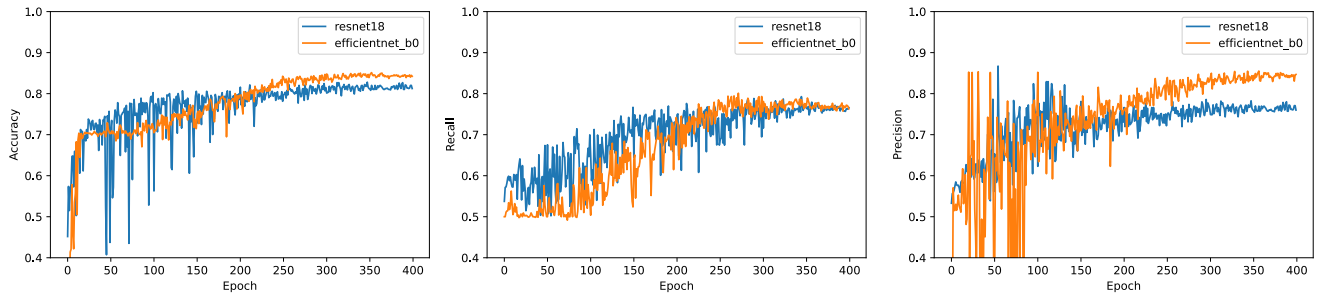
(a) *Accuracy* (b) *Recall* (c) *Precision*

Figura B.1: Evolución de varias métricas sobre el conjunto de entrenamiento y validación para T01 y *clean dataset*



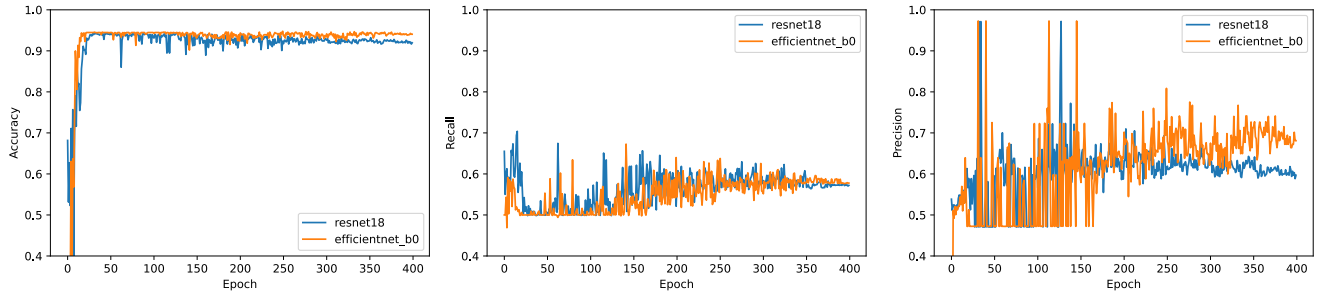
(a) *Accuracy* (b) *Recall* (c) *Precision*

Figura B.2: Evolución de varias métricas sobre el conjunto de entrenamiento y validación para T02 y *clean dataset*



(a) *Accuracy* (b) *Recall* (c) *Precision*

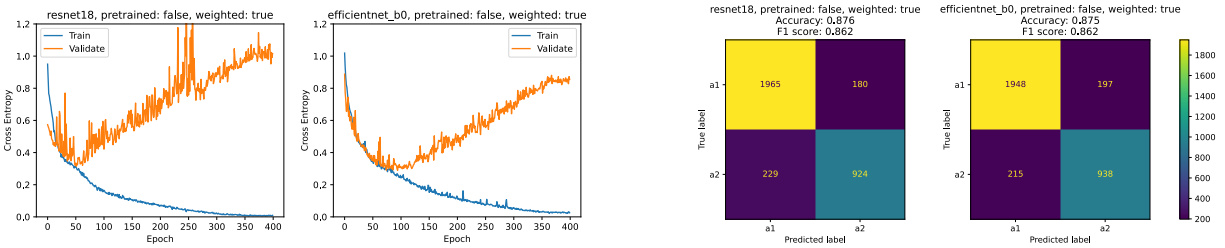
Figura B.3: Evolución de varias métricas sobre el conjunto de entrenamiento y validación para T03 y *clean dataset*



(a) *Accuracy* (b) *Recall* (c) *Precision*

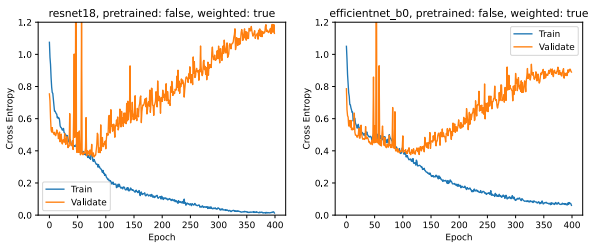
Figura B.4: Evolución de varias métricas sobre el conjunto de entrenamiento y validación para T04 y *clean dataset*

B.2. Evolución de la función de pérdida y matrices de confusión para *all galaxies dataset*

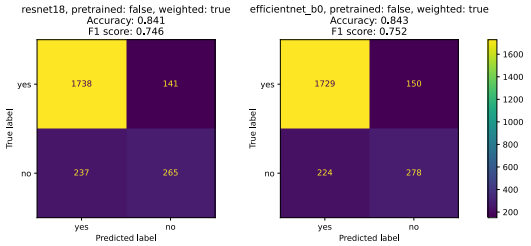


(a) Función de pérdida (b) Matriz de confusión con *accuracy* y *F1 score*

Figura B.5: Tarea T01 con *all galaxies dataset* realizando clasificación con los modelos seleccionados. Evolución de la función de pérdida en entrenamiento y validación y matriz de confusión con *accuracy* y *F1 score* sobre test

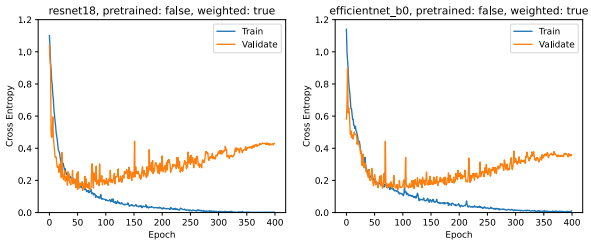


(a) Función de pérdida

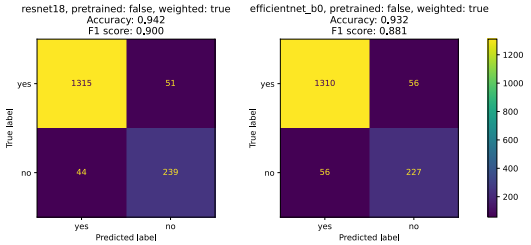


(b) Matriz de confusión con *accuracy* y *F1 score*

Figura B.6: Tarea T12 con *all galaxies dataset* realizando clasificación con los modelos seleccionados. Evolución de la función de pérdida en entrenamiento y validación y matriz de confusión con *accuracy* y *F1 score* sobre test

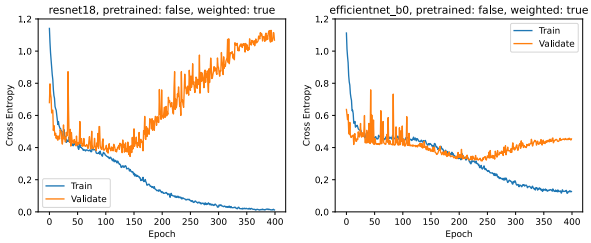


(a) Función de pérdida

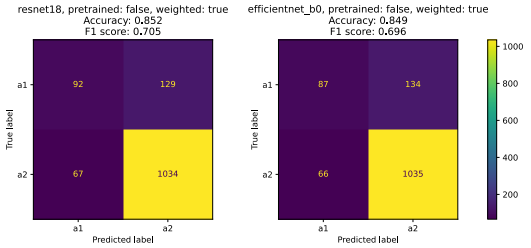


(b) Matriz de confusión con *accuracy* y *F1 score*

Figura B.7: Tarea T02 con *all galaxies dataset* realizando clasificación con los modelos seleccionados. Evolución de la función de pérdida en entrenamiento y validación y matriz de confusión con *accuracy* y *F1 score* sobre test

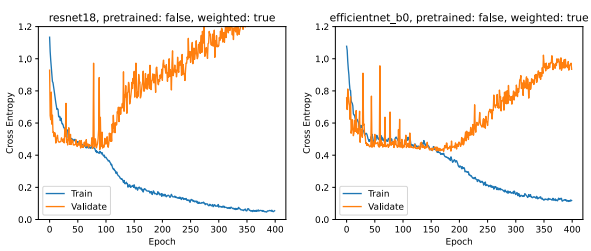


(a) Función de pérdida

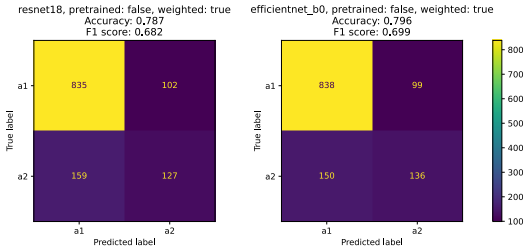


(b) Matriz de confusión con *accuracy* y *F1 score*

Figura B.8: Tarea T03 con *all galaxies dataset* realizando clasificación con los modelos seleccionados. Evolución de la función de pérdida en entrenamiento y validación y matriz de confusión con *accuracy* y *F1 score* sobre test



(a) Función de pérdida



(b) Matriz de confusión con *accuracy* y *F1 score*

Figura B.9: Tarea T04 con *all galaxies dataset* realizando clasificación con los modelos seleccionados. Evolución de la función de pérdida en entrenamiento y validación y matriz de confusión con *accuracy* y *F1 score* sobre test

Apéndice C

Gráficos de clasificación ponderada

C.1. Evolución de la función de pérdida

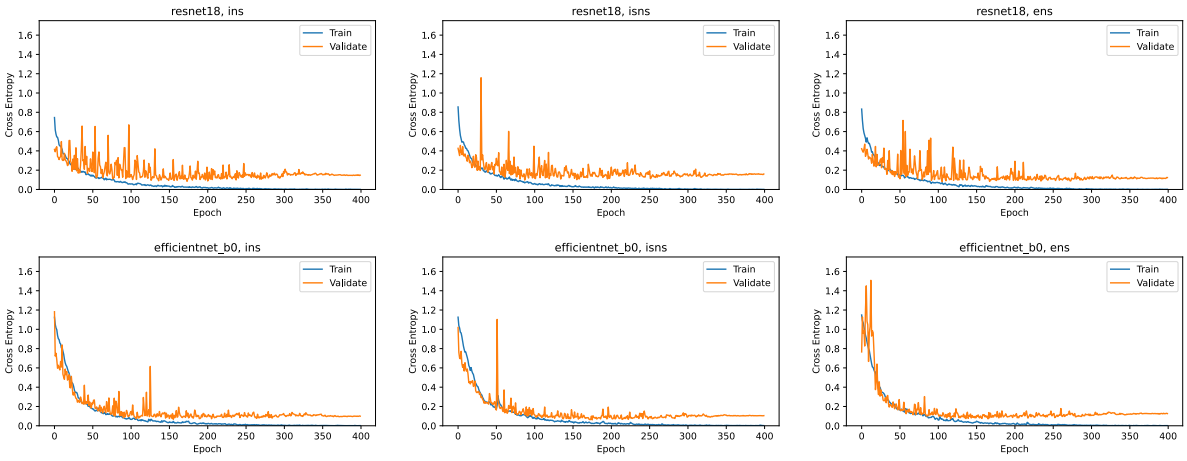


Figura C.1: Evolución de la función de pérdida de entrenamiento y validación con T01 y *clean dataset*

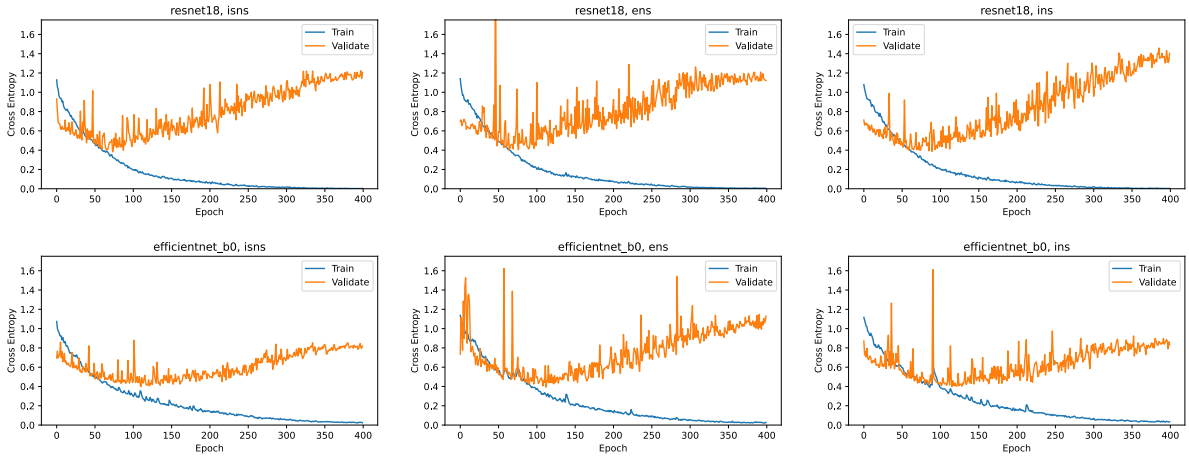


Figura C.2: Evolución de la función de pérdida de entrenamiento y validación con T12 y *clean dataset*

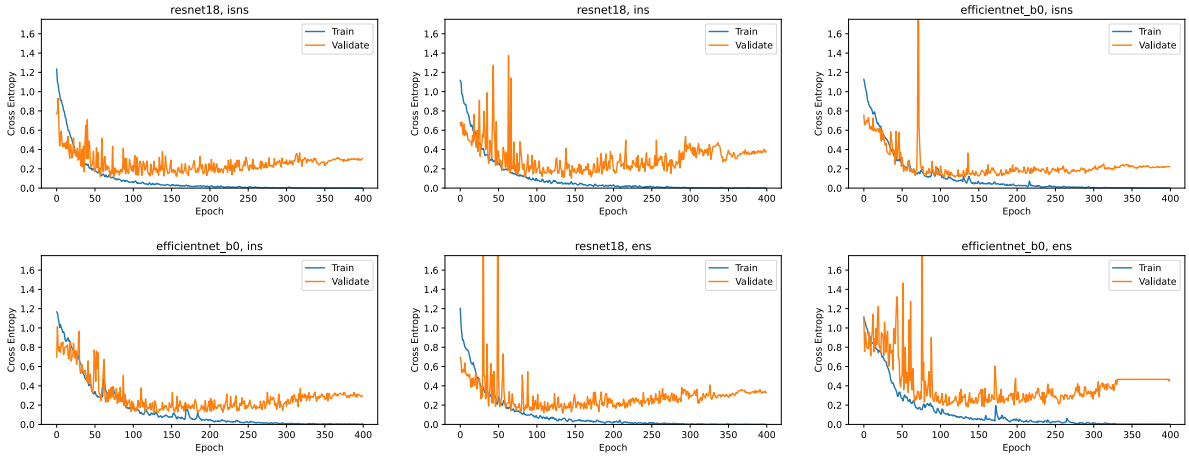


Figura C.3: Evolución de la función de pérdida de entrenamiento y validación con T02 y *clean dataset*

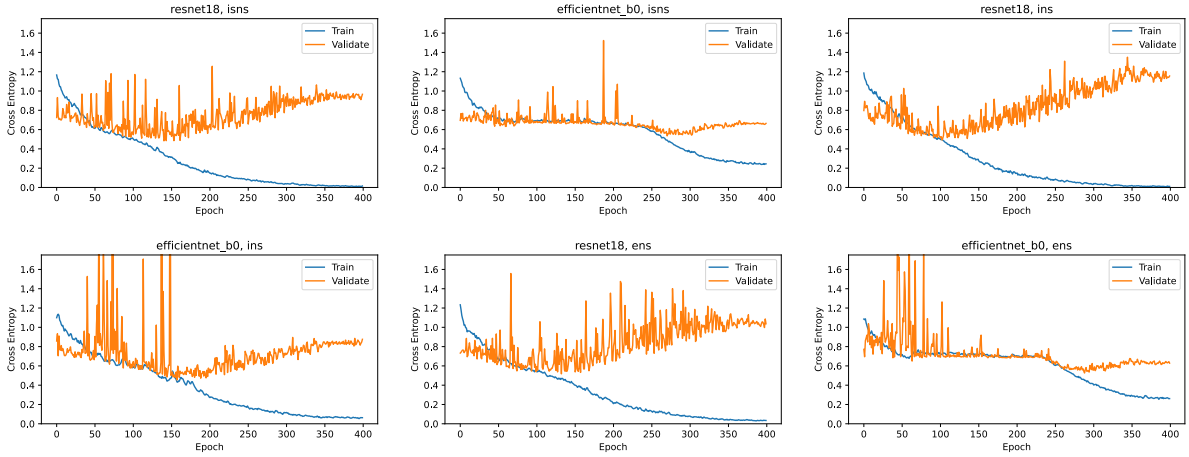


Figura C.4: Evolución de la función de pérdida de entrenamiento y validación con T03 y *clean dataset*

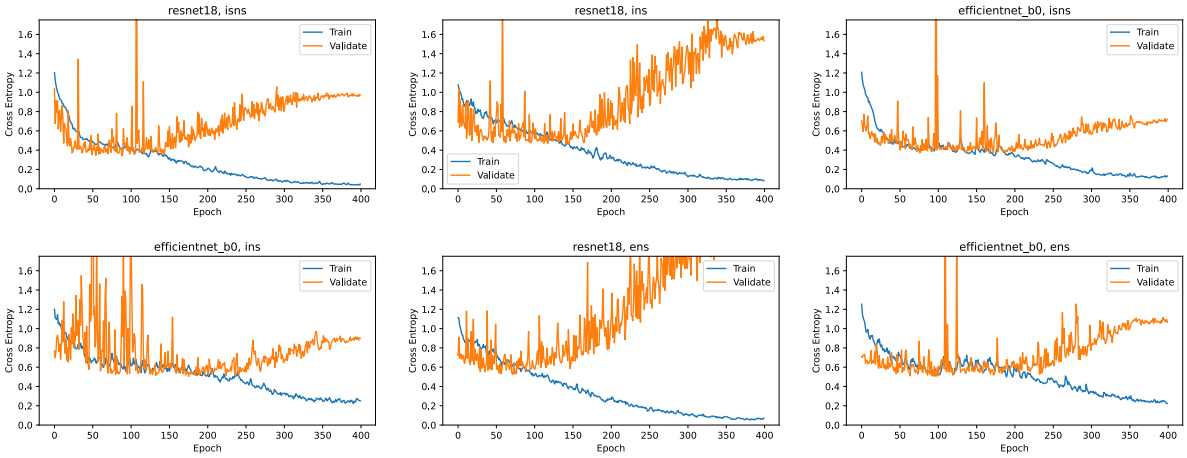


Figura C.5: Evolución de la función de pérdida de entrenamiento y validación con T04 y *clean dataset*

C.2. Matrices de confusión

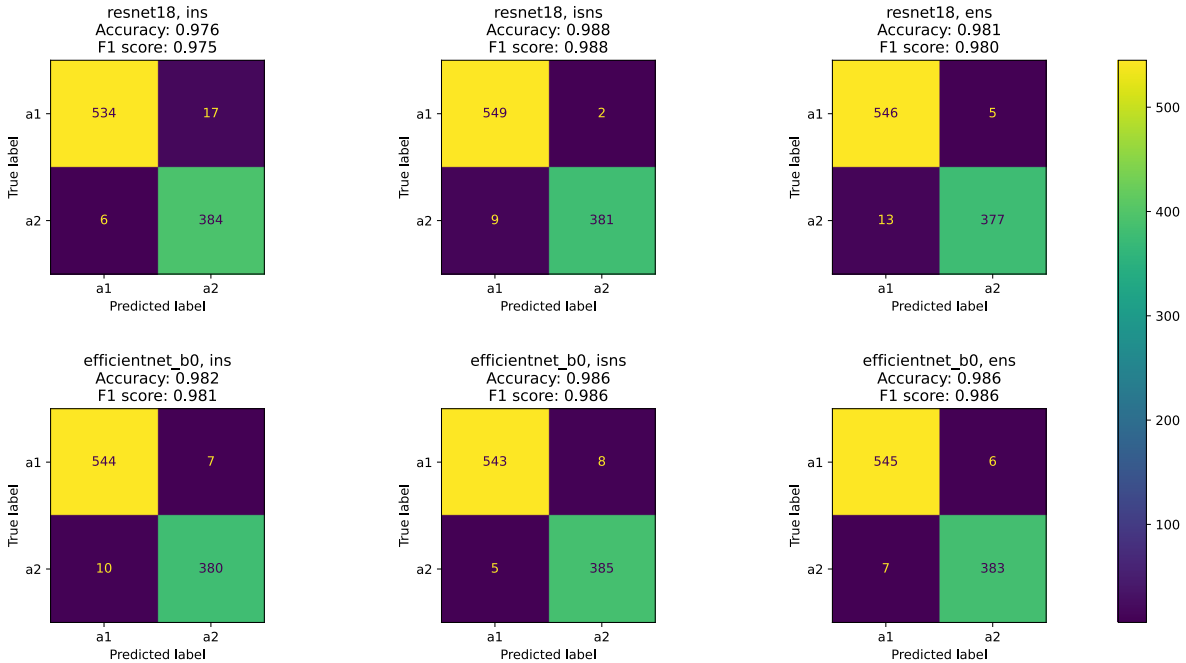


Figura C.6: Matrices de confusión sobre test para T01 y *clean dataset*

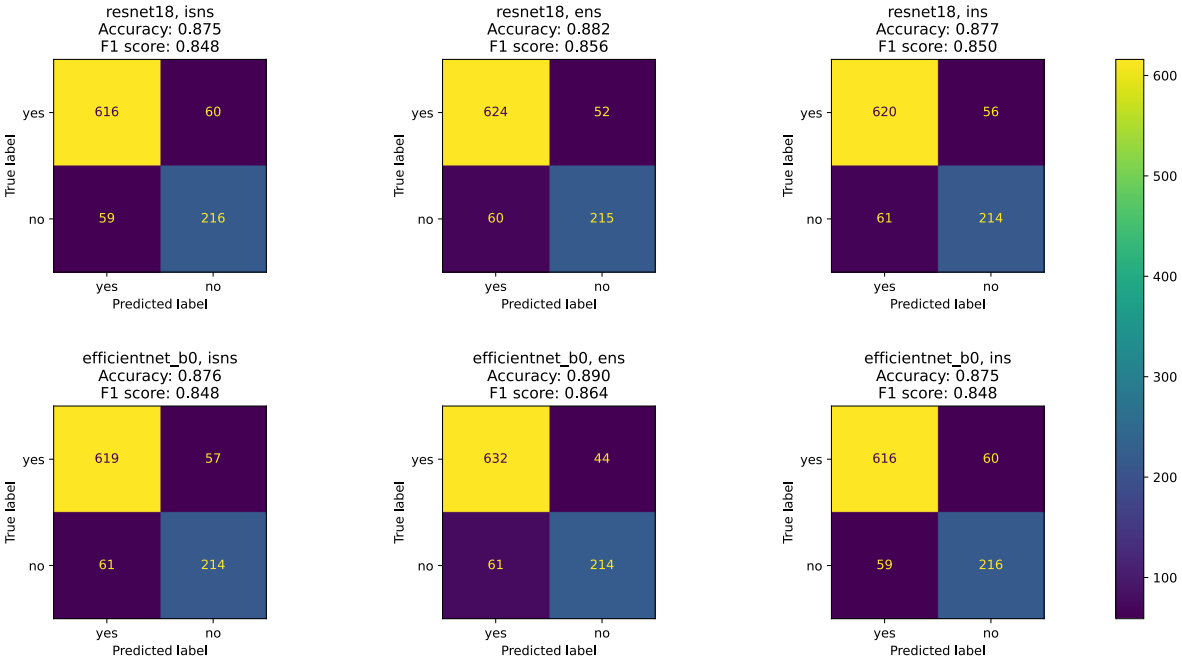


Figura C.7: Matrices de confusión sobre test para T12 y *clean dataset*

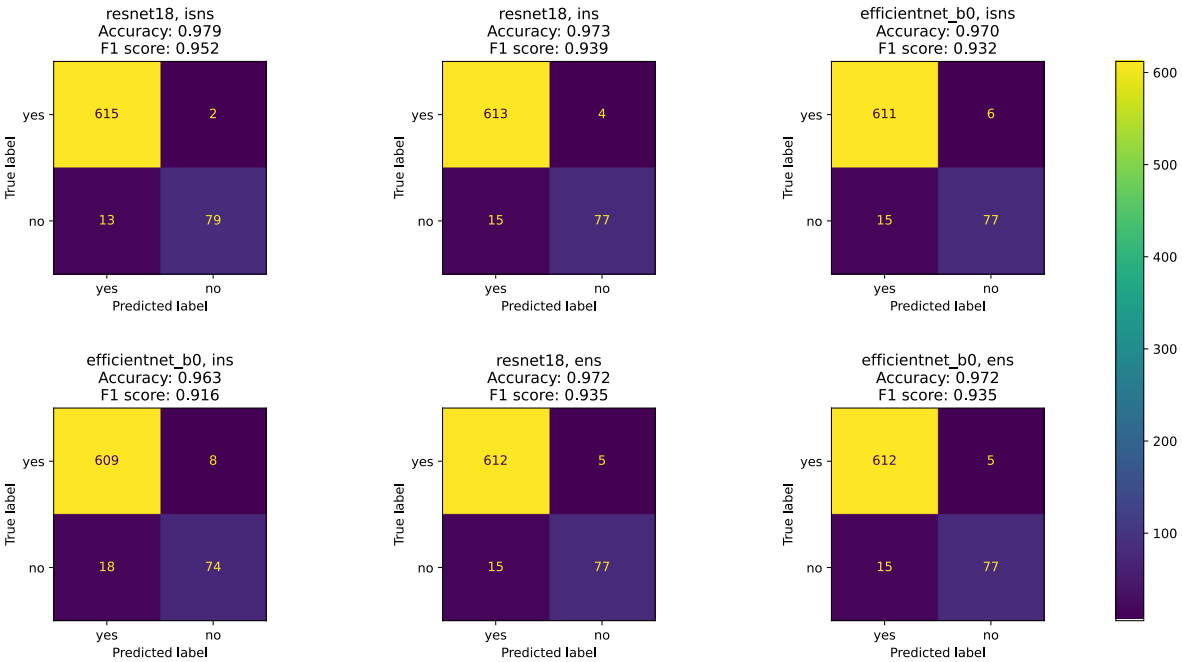


Figura C.8: Matrices de confusión sobre test para T02 y *clean dataset*

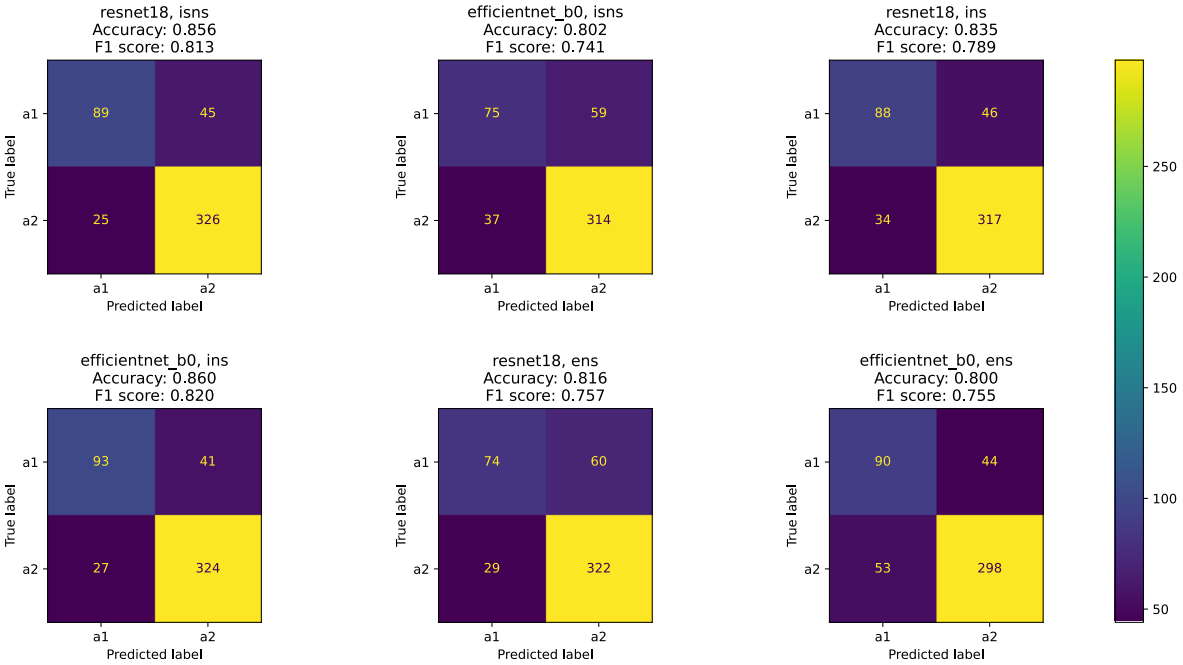


Figura C.9: Matrices de confusión sobre test para T03 y *clean dataset*

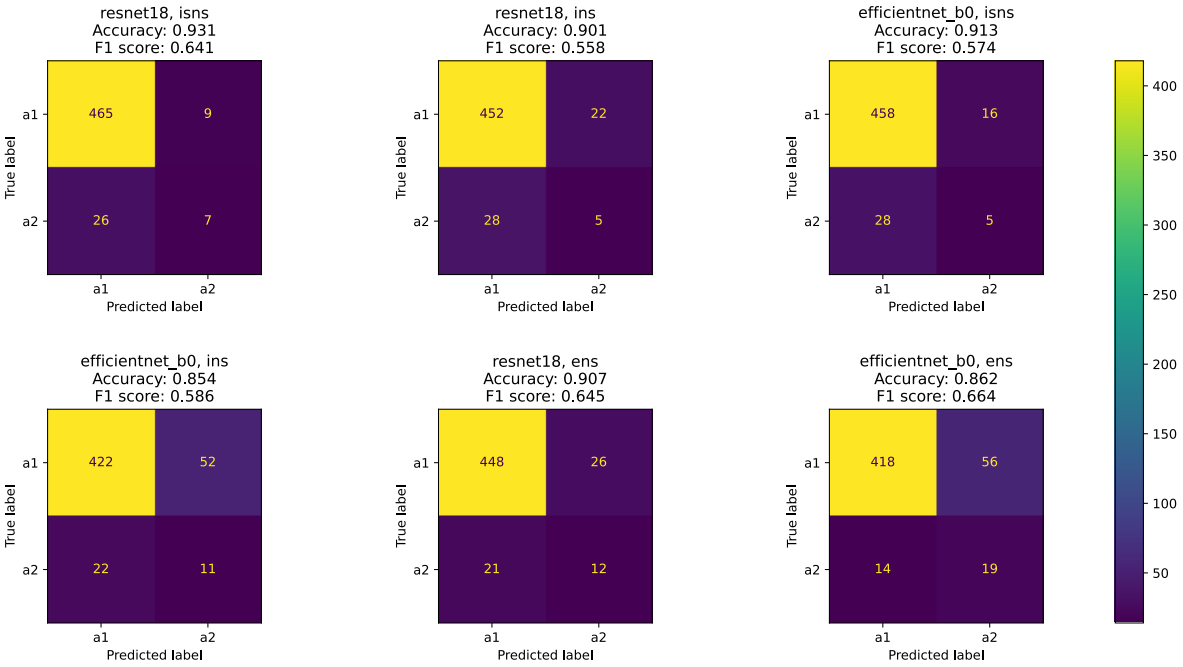
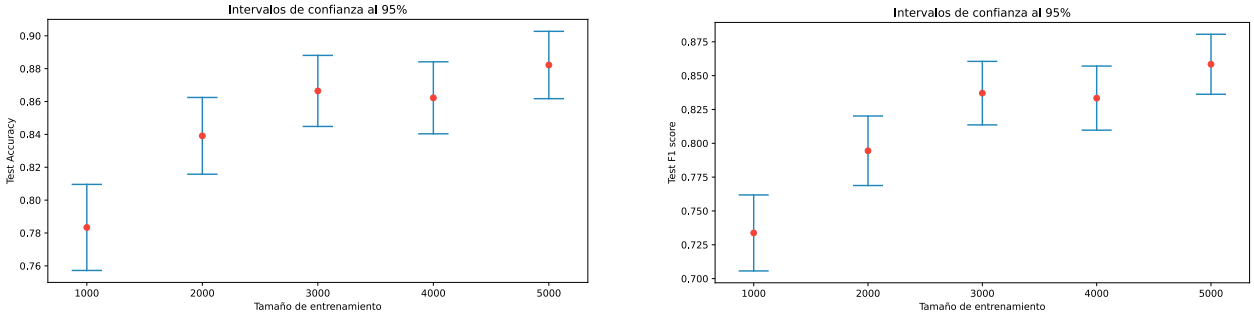


Figura C.10: Matrices de confusión sobre test para T04 y *clean dataset*

Apéndice D

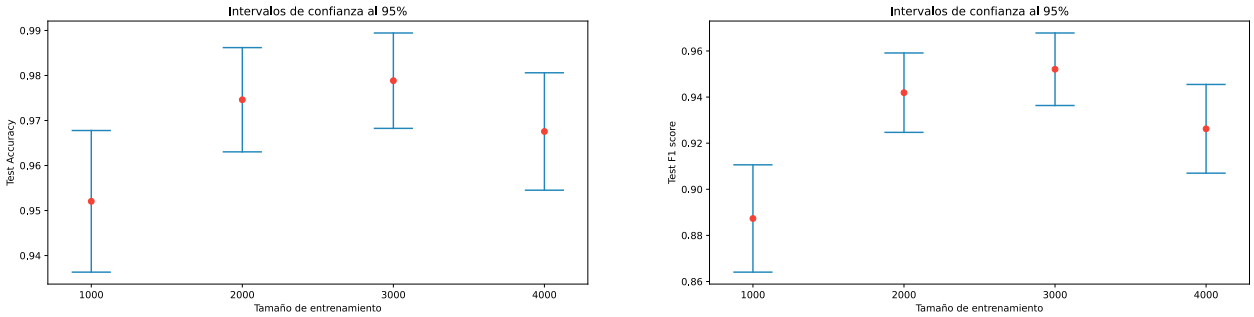
Gráficos de la cuestión 2

D.1. Intervalos de confianza para *clean dataset*



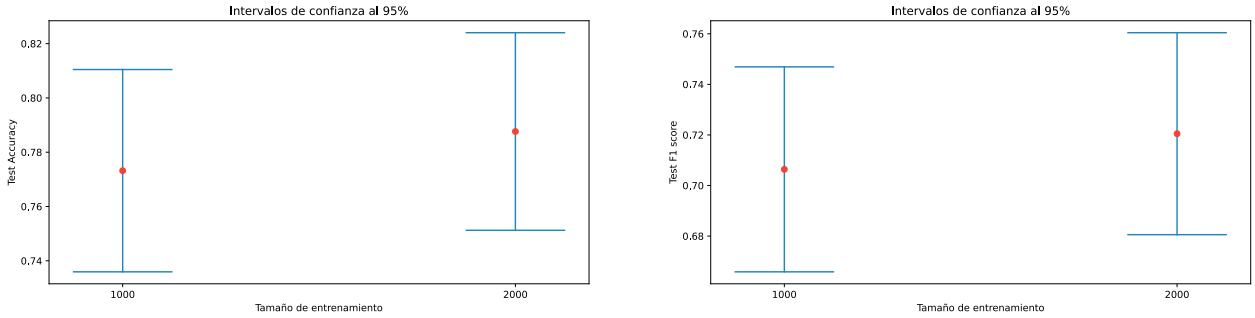
(a) *Accuracy* (b) *F1 score*

Figura D.1: Comparación de las métricas de test en la tarea T12 con *clean dataset* según el número de instancias de entrenamiento



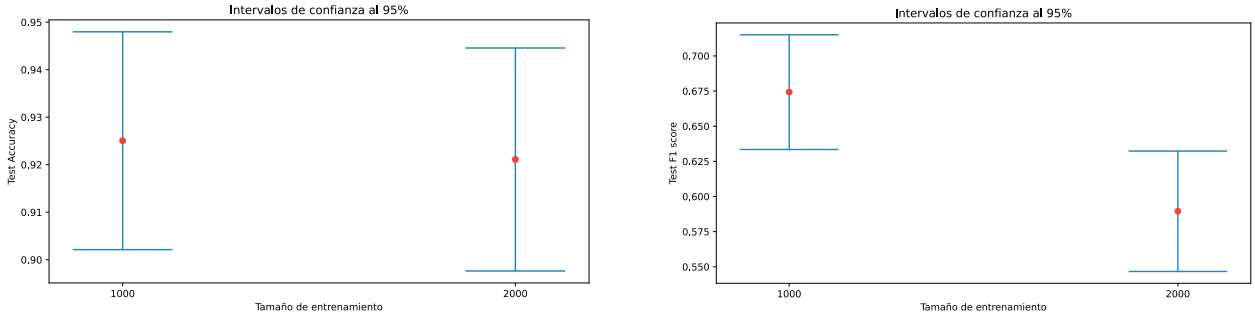
(a) *Accuracy* (b) *F1 score*

Figura D.2: Comparación de las métricas de test en la tarea T02 con *clean dataset* según el número de instancias de entrenamiento



(a) *Accuracy* (b) *F1 score*

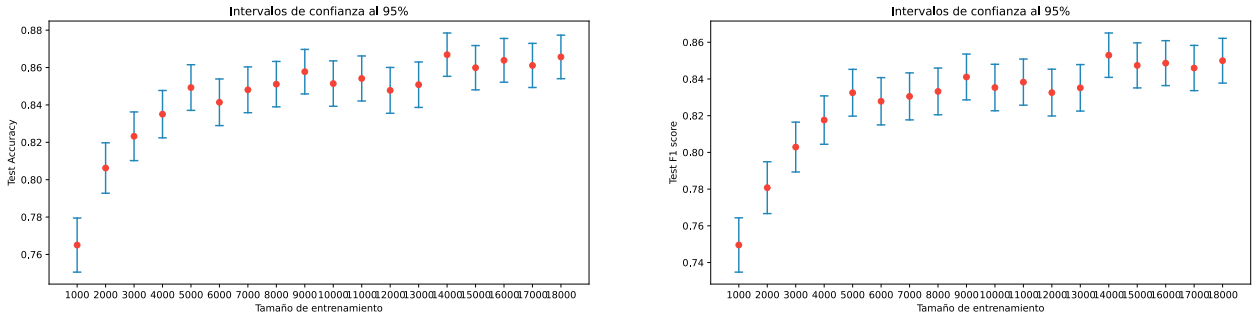
Figura D.3: Comparación de las métricas de test en la tarea T03 con *clean dataset* según el número de instancias de entrenamiento



(a) *Accuracy* (b) *F1 score*

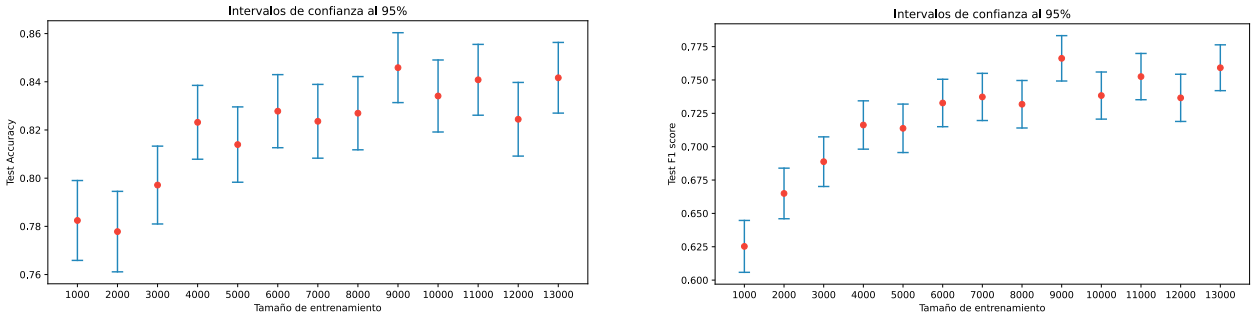
Figura D.4: Comparación de las métricas de test en la tarea T04 con *clean dataset* según el número de instancias de entrenamiento

D.2. Intervalos de confianza para *all galaxies dataset*



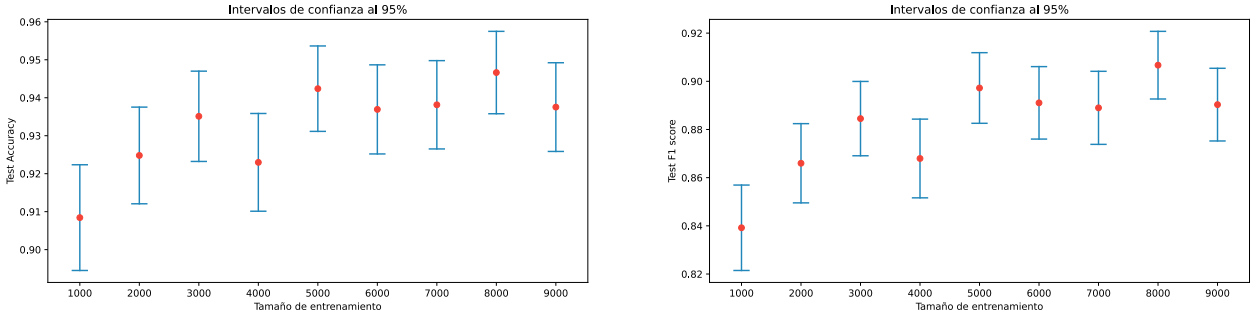
(a) *Accuracy* (b) *F1 score*

Figura D.5: Comparación de las métricas de test en la tarea T01 con *all galaxies dataset* según el número de instancias de entrenamiento



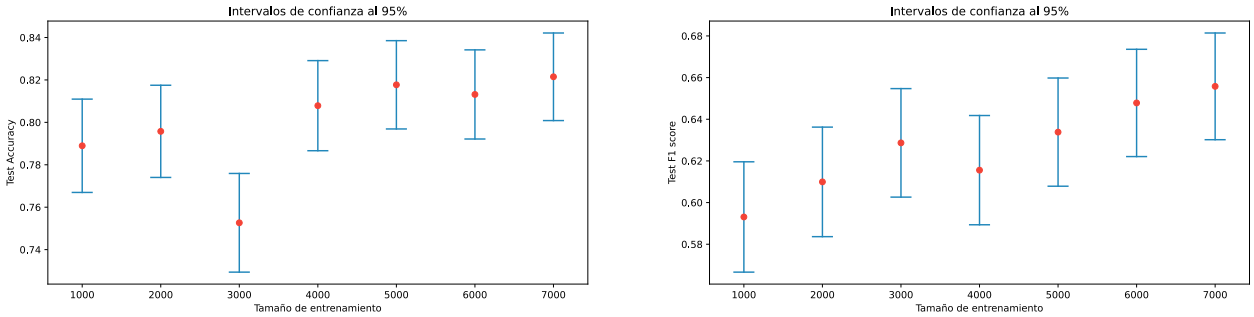
(a) *Accuracy* (b) *F1 score*

Figura D.6: Comparación de las métricas de test en la tarea T12 con *all galaxies dataset* según el número de instancias de entrenamiento



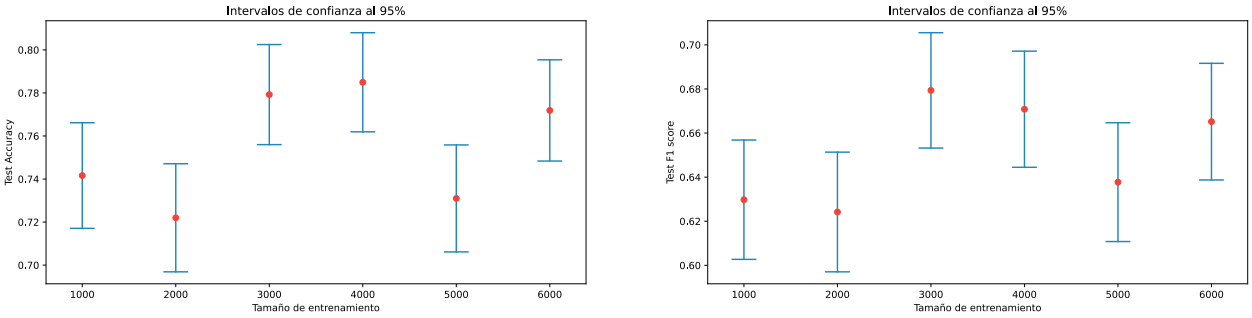
(a) *Accuracy* (b) *F1 score*

Figura D.7: Comparación de las métricas de test en la tarea T02 con *all galaxies dataset* según el número de instancias de entrenamiento



(a) *Accuracy* (b) *F1 score*

Figura D.8: Comparación de las métricas de test en la tarea T03 con *all galaxies dataset* según el número de instancias de entrenamiento



(a) *Accuracy* (b) *F1 score*

Figura D.9: Comparación de las métricas de test en la tarea T04 con *all galaxies dataset* según el número de instancias de entrenamiento