Universidad de Valladolid

ESCUELA DE INGENIERÍA INFORMÁTICA

Trabajo Fin de Grado

GRADO EN INGENIERÍA INFORMÁTICA
Mención en Ingeniería del Software

Análisis y Experimentación Práctica de Frameworks Deep Learning Aplicados a la Astronomía

---

Autor:
Jorge Barrio Conde

Tutor:
Benjamín Sahelices Fernández

Cotutor:
Fernando Buitrago Alonso

A mi familia y amigos

Si se espera que una máquina sea infalible,
no se puede esperar que también sea inteligente.

Alan Turing.

Agradecimientos

En primer lugar, quiero agradecer a mis padres y a mi hermano por todo el apoyo que siempre me han brindado y por su inmensa paciencia en los días de mayor estrés. Por supuesto, agradecer al resto de mi familia por los valores y la educación que me han inculcado, en especial a los que ya no están para verlo.

A su vez, quiero dar las gracias a mis amigos y compañeros de universidad que han hecho de estos cuatro años tan duros, una etapa muy bonita y de crecimiento personal, de los que me llevo grandes recuerdos y vivencias tanto de los momentos de estudio como de los de desconexión y disfrute.

Por último, y no por eso menos importante, quiero agradecer a todas las personas que han estado involucradas en mi formación, en especial a Benjamín Sahelices y a Fernando Buitrago por su empeño y dedicación durante la tutela del proyecto, así como a Javier Rodriguez por sus aportes y conocimientos al inicio del mismo, haciendo de este Trabajo de Fin de Grado un gran proceso de aprendizaje.

Resumen

Con este Trabajo de Fin de Grado se busca indagar en los fundamentos del Aprendizaje Profundo y las redes neuronales para aplicarlos a problemas astronómicos relacionados con galaxias distantes.

El primer problema consistirá en una clasificación morfológica de galaxias en función del aspecto visual de las mismas. El segundo problema se centrará en la detección de sus bordes. Ambos contarán con una serie de experimentos en los que se evaluará y comparará la precisión de los distintos modelos en función de las diferentes modificaciones que se les aplique. Dichos modelos podrán ser tanto redes neuronales convolucionales propias como modelos pre-entrenados, con el propósito de explorar el potencial de la transferencia de aprendizaje.

Estos problemas se aplicarán sobre las galaxias distantes más masivas de tipo espiral que pertenecen al cartografiado *CANDELS*, uno de los proyectos más grandes del Telescopio Espacial Hubble y que manifiesta gran parte de las historia y evolución del universo.

La implementación de las soluciones será llevada a cabo utilizando el lenguaje de programación *Python* junto a la biblioteca de Aprendizaje Profundo *fastai* y *PyTorch*.

Palabras clave: Aprendizaje profundo, Redes Neuronales, Astroinformática, fastai, PyTorch, Astronomía, Hubble, CANDELS.

Abstract

The aim of this Bachelor's degree final thesis is to investigate the basics of Deep Learning and to dig into the neuronal networks in order to apply them to astronomical issues regarding distant galaxies.

The first point of concern will consist in the morphological classification of the galaxies according to their visual appearance. The second point of examination will focus on the detention of the galaxies' edges. Both of these points will rely on a series of experiments. Moreover, the accuracy of the different models will be assessed and compared according to the different modifications applied to those experiments. Such models may be either own convolutional neural networks or pretrained models to explore the potential of learning transfer.

These issues will apply upon the massive spiral-like faraway galaxies that belong to the cartographic *CANDELS*, which is one of the most renowned projects carried out by the Hubble Space Telescope and it manifests several aspects of the history and evolution of the universe.

The implementation of the solutions will be undertaken by *Python* language along with the Deep Learning *fastai* and *PyTorch* library.

Key words: Deep Learning, Neural Networks, Astroinformatics, fastai, PyTorch, Astronomy, Hubble, CANDELS.

Índice general

Lista de Figuras

Lista de Tablas

Capítulo 1

Introducción

El presente Trabajo de Fin de Grado se va a desarrollar dentro del ámbito de la astroinformática, un campo de estudio interdisciplinar que nace de la necesidad de utilizar los conocimientos de la informática y la computación sobre la inmensa cantidad de datos que maneja la astronomía [5].

Este proyecto consistirá en aprender e investigar los conocimientos clave del campo del *deep learning*, o aprendizaje profundo en español, para aplicarlos a problemas astronómicos como la clasificación morfológica de galaxias en función de su aspecto visual o la detección de sus bordes. Para estos problemas se utilizarán las observaciones realizadas por el Telescopio Espacial Hubble en su cartografiado CANDELS, el cual recoge algunas de las galaxias más lejanas que han sido captadas hasta el momento.

La Inteligencia Artificial es un campo de estudio muy extenso que engloba todo tipo de sistemas que hacen posible que máquinas imiten comportamientos y habilidades avanzadas atribuidas a los seres humanos.

Capacitar a las maquinas de habilidades humanas se puede lograr de múltiples formas. Un ejemplo de ello son los sistemas expertos. Estos sistemas son desarrollados para comportarse según unas reglas definidas como un conjunto de condicionales. De esta forma, a partir de unos datos de entrada, y en función de las reglas que se le haya programado, el sistema experto determinará la salida correspondiente. Este enfoque es bueno para tareas simples sin mucha variabilidad o complejidad de los datos. Si se piensa en una tarea como la de clasificar una imagen según si se trata de un perro o un gato, la infinidad de casos y estados posibles, hace que sea imposible programar todas esas reglas. Ahí es donde surge el campo del *Machine Learning* o aprendizaje automático en español, uno de los campos más importantes y extendidos dentro de la inteligencia artificial.

El *machine learning* se basa en que el sistema aprenda e infiera dichas reglas a partir de una aprendizaje o entrenamiento previo. Dentro de este campo se encuentra el *Deep Learning* o aprendizaje profundo en español [78]. El adjetivo "profundo" se le atribuye debido a que trabaja con redes neuronales profundas, es decir, redes con muchas capas de neuronas.

Pese a que gran parte de los fundamentos que sustentan el *deep learning* aparecieron hace décadas, no ha sido hasta los últimos años cunado ha experimentado su enorme crecimiento. Esto se debe a los grandes avances que ha experimenta el hardware, las investigaciones de nuevos algoritmos más eficientes y la exorbitante cantidad de datos que se tiene a disposición.

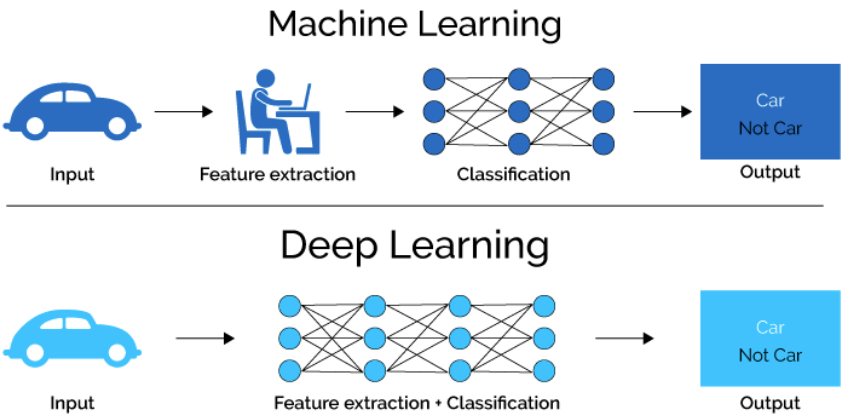


Figura 1.1: Comparativa de *Machine Learning* y *Deep Learning* [74]

La Figura 1.1 representa la diferencia sustancial entre *machine learning* y el *deep learning*. En el primero, a partir de un conjunto de datos, un profesional ha de extraer las "features" o características claves de estos datos y suministrárselas al modelo para que obtenga su predicción. Por el contrario, en el *deep learning* se omite ese paso intermedio, puesto que el propio modelo es el encargado de determinar cuáles son las características más importantes de los datos que se le está suministrando.

El *deep learning* y las redes neuronales prestan las mejores soluciones a muchos de los problemas actuales. Dentro de este campo se pueden encontrar diferentes áreas o disciplinas, como pueden ser:

- Visión por ordenador: área del *deep learning* que trabaja en base a la interpretación de imágenes o secuencias de estas.
- Generación de imágenes: es un área semejante al anterior, pero la misión de estos sistemas es generar nuevas imágenes a partir de una dada. Algunos de los usos de este área son proveer de una mayor resolución a una imagen o darle color a una en blanco y negro.
- Procesamiento del lenguaje natural (o Natural Language Processing, NLP): disciplina que procesa secuencias de texto o voz con el fin de interpretarlas e interaccionar con estas, tal y como lo haría un ser humano.
- Sistemas de recomendación: área muy extendida en aplicaciones web para ofrecer la mejor experiencia al usuario.

Todas estas disciplinas pueden ser aplicables a múltiples campos externos a la computación, como sería la medina para la detección enfermedades en base a anomalías en pruebas

médicas, el campo de la conducción autónoma para identificar elementos como coches, peatones o señales de tráfico, o el campo de la astronomía como será el caso de este proyecto [39, p. 3-5].

1.1. Motivación

La razón principal que me motivó a llevar a cabo este proyecto fue el amplio vacío de conocimiento que poseía en torno al campo de la inteligencia artificial y, más concretamente, al campo del aprendizaje automático. Es por ello, por lo que no quería finalizar mi grado en ingeniería informática sin adentrarme en esta disciplina que siempre llamó mi atención por su gran potencial y protagonismo que ha cobrado en estos últimos años.

A su vez, la astronomía siempre ha sido de gran interés para mí, por lo que poder aplicar todo este nuevo conocimiento a dicho campo, manejando imágenes de galaxias distantes capturadas por el Telescopio Espacial Hubble, propició la elección del desarrollo de este proyecto.

1.2. Objetivos del proyecto

Los objetivos principales en los que se basa este proyecto podrían resumirse en los cinco siguientes:

- Investigar y asimilar los conceptos y principios más importantes del *Deep Learning* y las redes neuronales.
- Investigar sobre las tecnologías existentes para desarrollar tareas de *Deep Learning*, elegir las más adecuadas y profundizar en ellas.
- Aprender a utilizar el software astronómico para procesar los datos del cartografiado CANDELS del Telescopio Espacial Hubble.
- Clasificar morfológicamente galaxias lejanas mediante técnicas de *deep learning* en base a su aspecto físico .
- Construir un modelo de *deep learning* capaz de detectar los bordes de galaxias lejanas.

1.3. Estructura del proyecto

El presente documento se ha estructurado en 8 capítulos y 2 apéndices. En adición a este primer capítulo de introducción, se encuentran los siguientes:

- **Capítulo 2: Planificación**. El segundo capítulo contendrá una breve explicación de la gestión y planificación que se ha seguido durante la evolución del desarrollo del proyecto.
- **Capítulo 3: Redes Neuronales**. A lo largo del tercer capítulo se explicará un marco general de los conocimientos teóricos relativos a las redes neuronales, poniendo un mayor énfasis en aquellos en los que se centrará el proyecto, como por ejemplo las redes neuronales convolucionales.
- **Capítulo 4: Astronomía**. El cuarto capítulo se centrará en contextualizar el proyecto desde un punto de vista astronómico, en el que se explicará tanto el origen como el significado de las imágenes de galaxias con las que se va a trabajar durante el proyecto.
- **Capítulo 5: Contexto tecnológico**. Este capítulo estará enfocado en la descripción de las tecnologías escogidas para la construcción de los modelos de *deep learning*, los softwares astronómicos utilizados para el tratamiento de las observaciones del telescopio Hubble y otras herramientas y servicios que han intervenido en el desarrollo de este proyecto.
- **Capítulo 6: Clasificación morfológica de galaxias**. El sexto capítulo estará dedicado a la resolución del problema de la clasificación morfológica de galaxias. En este se describirán los dos enfoques utilizados para abordar el problema (regresión y clasificación) y los diversos experimentos que han sido realizados para evaluar cómo afectan las distintas modificaciones sobre el rendimiento de los modelos, a fin de aprender y conseguir la mayor precisión posible. A lo largo del capítulo se describirán los conocimientos más importantes de fastai que han sido adquiridos.
- **Capítulo 7: Detección de bordes de galaxias**. El penúltimo capítulo estará enfocado a la resolución del problema de la detección de bordes de las galaxias. Para ello se utilizará un modelo de regresión encargado de predecir cinco valores que determinarán los bordes de las mismas. Al igual que el anterior capítulo, este contendrá una serie de pruebas para alcanzar la mejor precisión posible.
- **Capítulo 8: Conclusiones**. Para cerrar la memoria, este último capítulo resumirá las conclusiones obtenidas con el desarrollo del proyecto, otros posibles problemas astronómicos sobre los que se puede utilizar el *deep learning* y las líneas futuras por las que se puede continuar este trabajo.
- **Apéndice A: Manual de instalación de software utilizado en Ubuntu**. Este primer apéndice contendrá una guía rápida para instalar en Ubuntu el software que ha sido utilizado para desarrollar este proyecto.
- **Apéndice B: Contenidos del soporte digital**. Este último apéndice recogerá una breve descripción del material adicional que se incluye con el presente documento, así como su estructuración.

Capítulo 2

Planificación y gestión

2.1. Metodología utilizada

Las primeras etapas del proyecto han estado marcadas por una alta incertidumbre debido a la gran falta de conocimiento tanto de las tecnologías que se iban a utilizar como del campo de estudio sobre el que se iba a desarrollar el proyecto. Por ello, las primeras fases estuvieron desarrolladas bajo una metodología de entrega evolutiva o prototipado. Esta metodología se utiliza típicamente cuando no existe una claridad en la forma en la que se va a implementar el sistema final.

Desde un enfoque teórico, esta metodología permite comprobar rápidamente cualquier requisito o suposición sobre el sistema final, en base a la construcción de prototipos. Existen varios tipos de prototipos:

- Los *"Throw away" prototypes* o prototipos de "usar y tirar" son aquellos que se realizan con rapidez y sin necesidad de seguir los posibles estándares impuestos para el sistema final.
- Los *Evolutionary prototypes* o prototipos evolutivos, sin embargo, sí respetan los estándares. Esto se debe a que estos prototipos van a servir para evolucionar y alcanzar el producto final.

En este caso, los primeros prototipos eran de tipo "throw away" ya que su objetivo era puramente didáctico. El primero de ellos fue una red neuronal tradicional escrita en Python a partir de las explicaciones del libro de "Neural Networks and Deep Learning" de Michael A. Nielsen [64] , cuyo objetivo era clasificar las imágenes de dígitos manuscritos del famoso dataset de MNIST. El fin de este prototipo era entender los algoritmos que intervienen en el aprendizaje de una red e interiorizar qué es lo que constituye una red neuronal. El segundo prototipo fue una red neuronal convolucional construida en PyTorch, cuyo fin era realizar una clasificación binaria de imágenes entre gatos y perros.

Los siguientes prototipos que se desarrollaron fueron de tipo evolutivos ya que supusieron los primeros cimientos para los modelos que fueron desarrollados para resolver el problema de clasificación morfológica de galaxias.

Una vez se afianzaron los conocimientos de fastai y PyTorch, el desarrollo de los modelos siguió una metodología ágil tipo SCRUM. Las metodologías ágiles surgen para solventar los problemas e inconvenientes que ocasionan las metodologías estructuradas, como sería el exceso de documentación, la baja comunicación entre los miembros de los equipos o la tardanza a la hora de entregar productos a los usuarios. Este tipo de metodologías se sustentan bajo el *Manifiesto Ágil*. Algunas de las metodologías ágiles más conocidas son SCRUM, Altern o Extreme Programming (XP).

El origen de SCRUM nace de la necesidad de introducir rápidamente productos novedosos en el mercado. Su nombre proviene del término melé de rugby, el cual define una formación fija de jugadores que empujan hacia una misma dirección. En esta metodología, el trabajo se divide en *sprints* y su duración suele ser de una a cuatro semanas. Al principio de cada sprint se lleva a cabo una "review meeting", o reunión de planificación, en la que se establecen y priorizan los requisitos para ese sprint. Hasta que no se finalice el sprint actual no se debe modificar ninguno de los requisitos establecidos para ese sprint. Al finalizar un sprint se suele realizar una reunión de retrospección para evaluar el trabajo realizado. A lo largo de los sprints se realiza una "daily scrum" al día, que suele consistir en una reunión breve de unos 15 minutos en la que los miembros del equipo describen brevemente el trabajo realizado el día anterior y las tareas a las que se van a dedicar ese mismo día. Un inconveniente que se achaca a esta metodología, es la posible pérdida del punto de vista global.

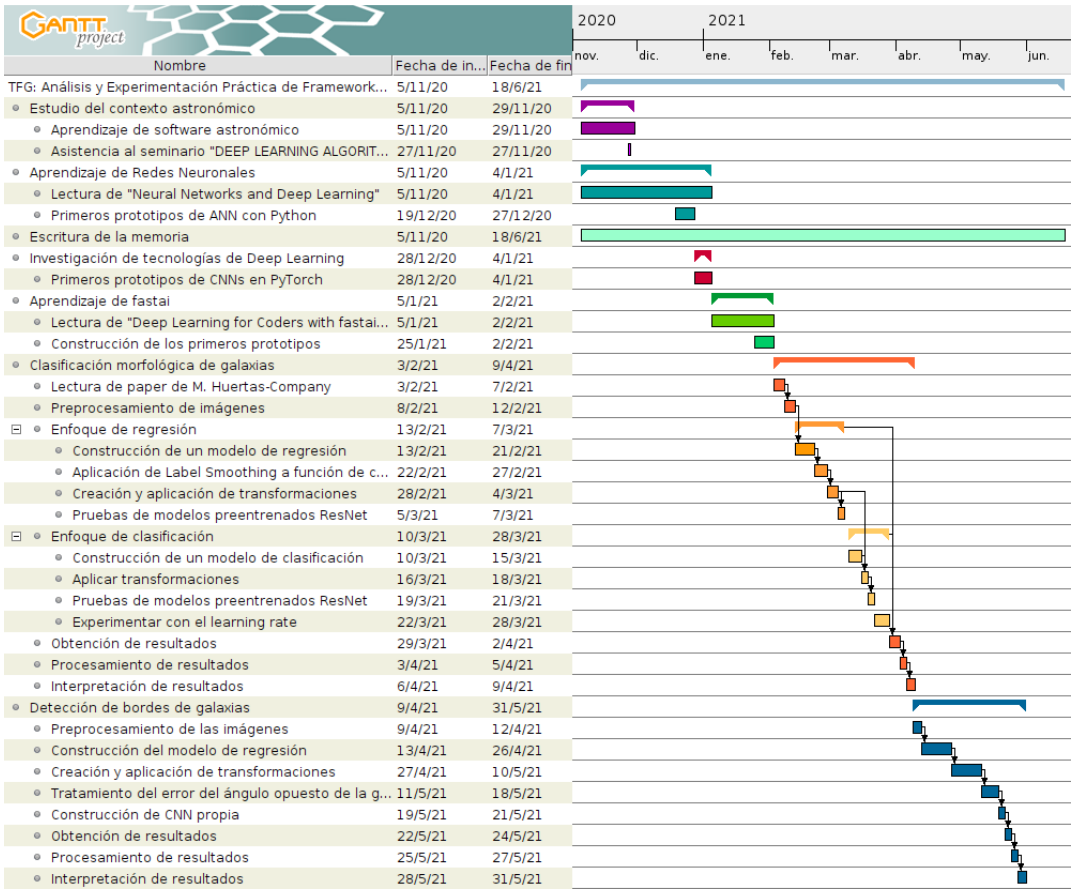
La metodología SCRUM define tres roles principales: el *Product Owner*, rol encargado de crear y priorizar las tareas para maximizar el valor del producto, el *SCRUM Master*, encargado de que se cumpla la metodología SCRUM, y el desarrollador, encargado de realizar el trabajo [42].

Acorde a las circunstancias, se adaptó dicha metodología a la forma de trabajar que se estaba siguiendo en el proyecto. El desarrollo se dividió en *sprints* de una semana. Cada semana se realizaba una reunión en la que, a modo de retrospectiva, se presentaba y evaluaba el trabajo realizado durante dicha semana. En estas, los tutores tomaban el rol de *product owner* y en caso de existir alguna desviación de los requisitos que se querían cumplir, se debatía sobre el tema y se tomaban decisiones que determinarían los siguientes *sprints*. Uniendo con esta reunión, se continuaba con una reunión de planificación en la que se decidían los alcances de la siguiente semana o sprint. Por razones de disponibilidad se omitieron las reuniones "daily scrum".

2.2. Seguimiento del proyecto

Una parte de la gestión de un proyecto es el seguimiento del progreso del mismo. Para ello se utilizan los diagramas de Gantt, los cuales recogen el tiempo requerido y el progreso de las diferentes tareas y grupo de tareas de un proyecto. A su vez, estos también representan las dependencias que puedan existir entre las distintas actividades. Un ejemplo de dependencia

sería que hasta que no se finalice el diseño de un módulo no se puede comenzar con su implementación.



Nombre	Fecha de in...	Fecha de fin
TFG: Análisis y Experimentación Práctica de Framework...	5/11/20	18/6/21
Estudio del contexto astronómico	5/11/20	29/11/20
Aprendizaje de software astronómico	5/11/20	29/11/20
Asistencia al seminario "DEEP LEARNING ALGORIT...	27/11/20	27/11/20
Aprendizaje de Redes Neuronales	5/11/20	4/1/21
Lectura de "Neural Networks and Deep Learning"	5/11/20	4/1/21
Primeros prototipos de ANN con Python	19/12/20	27/12/20
Escritura de la memoria	5/11/20	18/6/21
Investigación de tecnologías de Deep Learning	28/12/20	4/1/21
Primeros prototipos de CNNs en PyTorch	28/12/20	4/1/21
Aprendizaje de fastai	5/1/21	2/2/21
Lectura de "Deep Learning for Coders with fastai...	5/1/21	2/2/21
Construcción de los primeros prototipos	25/1/21	2/2/21
Clasificación morfológica de galaxias	3/2/21	9/4/21
Lectura de paper de M. Huertas-Company	3/2/21	7/2/21
Preprocesamiento de imágenes	8/2/21	12/2/21
Enfoque de regresión	13/2/21	7/3/21
Construcción de un modelo de regresión	13/2/21	21/2/21
Aplicación de Label Smoothing a función de c...	22/2/21	27/2/21
Creación y aplicación de transformaciones	28/2/21	4/3/21
Pruebas de modelos preentrenados ResNet	5/3/21	7/3/21
Enfoque de clasificación	10/3/21	28/3/21
Construcción de un modelo de clasificación	10/3/21	15/3/21
Aplicar transformaciones	16/3/21	18/3/21
Pruebas de modelos preentrenados ResNet	19/3/21	21/3/21
Experimentar con el learning rate	22/3/21	28/3/21
Obtención de resultados	29/3/21	2/4/21
Procesamiento de resultados	3/4/21	5/4/21
Interpretación de resultados	6/4/21	9/4/21
Detección de bordes de galaxias	9/4/21	31/5/21
Preprocesamiento de las imágenes	9/4/21	12/4/21
Construcción del modelo de regresión	13/4/21	26/4/21
Creación y aplicación de transformaciones	27/4/21	10/5/21
Tratamiento del error del ángulo opuesto de la g...	11/5/21	18/5/21
Construcción de CNN propia	19/5/21	21/5/21
Obtención de resultados	22/5/21	24/5/21
Procesamiento de resultados	25/5/21	27/5/21
Interpretación de resultados	28/5/21	31/5/21

Figura 2.1: Diagrama de Gantt del proyecto

La Figura 2.1 contiene el diagrama de Gantt del proyecto. Como puede observarse, las primeras semanas estaban centradas en el aprendizaje e investigación de los conocimientos relativos al *deep learning* y las redes neuronales mediante la lectura de libros. En paralelo y gracias a las sesiones explicativas del cotutor, se iba aprendiendo a utilizar los diversos softwares astronómicos para la visualización y manipulación de los datos astronómicos con los que se iban a trabajar. La duración de estas tareas se vio afectada por la alta carga de trabajo impuesta por las cinco asignaturas cursadas durante el primer cuatrimestre, lo que provocó la disminución del tiempo dedicado al proyecto. Tras realizar una replanificación, se recuperaron estas horas durante la época de exámenes (finales de diciembre e inicios de enero).

Durante este tiempo se asistió al seminario “Deep Learning Algorithms for Morphological Classification of Galaxies” impartido por Helena Domínguez Sánchez, puesto que el tema que se trataba en él, tocaba muy de cerca los objetivos de este proyecto.

Tras un intenso periodo de aprendizaje teórico, se pasó a una etapa de investigación de las tecnologías usadas para en el *deep learning*. En esta etapa, se dieron los primeros pasos en fastai y PyTorch, así como se leyó el libro "Deep Learning for Coders with Fastai and Pytorch:AI Applications Without a PhD" de Jeremy Howard y Sylvain Gugger [39].

A principios de febrero se comenzó con la parte práctica del proyecto, en la cual se comenzó abordando el problema de clasificación morfológica de galaxias. Hacia principios de abril, se comenzó el siguiente problema, la detección de los bordes de las galaxias.

Como puede verse representado en el diagrama de Gantt, desde el primer momento se estuvo escribiendo la memoria del proyecto para documentar todos los avances y descubrimientos que se iban consiguiendo.

2.3. Monitorización del progreso

Con respecto a la monitorización del proyecto por parte de los tutores, tal y como se ha comentado anteriormente, se realizaba una reunión semanal de aproximadamente una hora, en la que se trataban diversos temas, desde la exposición de las investigaciones y descubrimientos realizados durante la semana, hasta sesiones didácticas para aprender algunos de los conocimientos requeridos para el desarrollo del proyecto.

2.4. Herramientas para la gestión y planificación

Para hacer la planificación y gestión del proyecto más organizada y eficiente, se han utilizado diversas herramientas para poder seguir un flujo de trabajo óptimo.

Se ha utilizado la herramienta online Trello [4] para crear un tablero Kanvan para la organización del flujo de tareas del proyecto. Este tablero estaba compuesto por 4 columnas: "ToDo" (tareas pendientes), "In Progress" (tareas que se están realizando), "Blocked" (tareas bloqueadas que requieren consultas a los tutores o investigaciones más extensas) y "Done" (tareas finalizadas).

Para realizar el seguimiento del progreso del proyecto mediante el diagrama de Gantt expuesto anteriormente, se ha utilizado la herramienta gratuita de *GanttProject*, un software open source para la gestión de proyecto con soporte en Windows, Linux y MacOs [85].

Para almacenar y llevar un seguimiento de las las versiones de los distintos cuadernos que se iban creando y manipulando, se ha utilizado Google Drive, puesto que era la opción más cómoda para trabajar sobre Google Colab. En paralelo a esta plataforma, se creyó conveniente crear un repositorio de respaldo en GitHub para almacenar la evolución del proyecto.

Capítulo 3

Redes Neuronales

3.1. Introducción a las redes neuronales

Al hablar de redes neuronales artificiales se hace referencia a un modelo computacional basado en funcionamiento del cerebro humano. Para entender el funcionamiento de una red neuronal y sus capas, hay que comprender el funcionamiento del cerebro humano y su estructura. Para la tarea del reconocimiento visual, el cerebro realiza un procesamiento en cascada, de forma que cada hemisferio cuenta con una *córtex visual primario* (V_1), formada por millones de neuronas encargadas de identificar los elementos básicos. Las cortezas primarias se conectan con las siguientes cortezas (V_2) y estas con las siguientes (V_3, V_4, V_5). A medida que se avanza por estos niveles, el procesamiento se vuelve progresivamente más complejo.

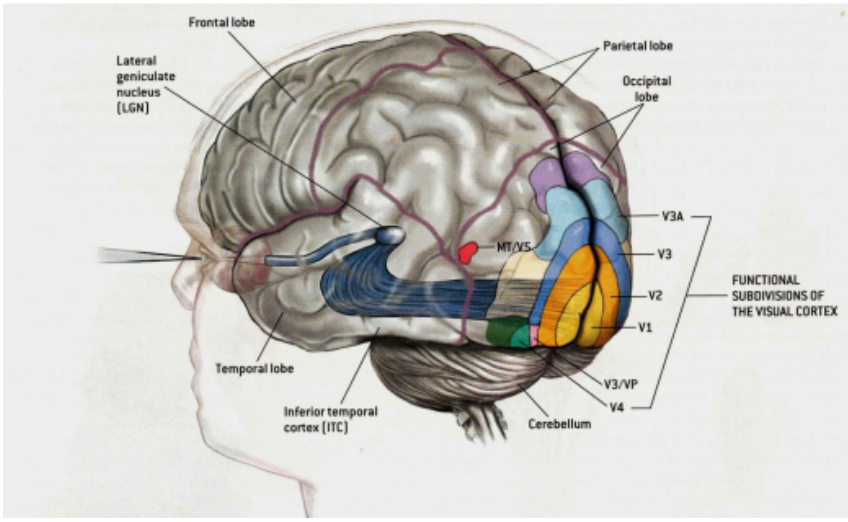


Figura 3.1: Córtex visual del cerebro humano [25]

La imagen anterior (ver Figura 3.1) representa la estructura en capas del córtex visual humano, la cual recibe la información de entrada a partir de los ojos.

Al igual que en el modelo biológico, una red neuronal artificial se estructura en capas formadas por neuronas, cuya función es inferir, mediante el aprendizaje, un conjunto de reglas que serían extremadamente difíciles de programar, y que permitiesen resolver un problema concreto [64, Cap. 1].

3.2. Neurona artificial

Antes de entrar a explicar los fundamentos de las redes neuronales, es necesario explicar cómo funcionan las unidades más básica que las conforman, es decir, cómo funciona una neurona artificial.

3.2.1. Perceptrón

En el año 1943, el psicólogo *Frank Rosenblatt* desarrolló el primer tipo de neurona artificial, denominada *Perceptrón*. Para ello se inspiró en el trabajo realizado por *Warren S. McCulloch* y *Walter Pitts*, en el que definió un modelo matemático capaz de emular el comportamiento de una neurona biológica [53]. La creación del perceptrón supuso un punto de partida hacía las redes neuronales artificiales.

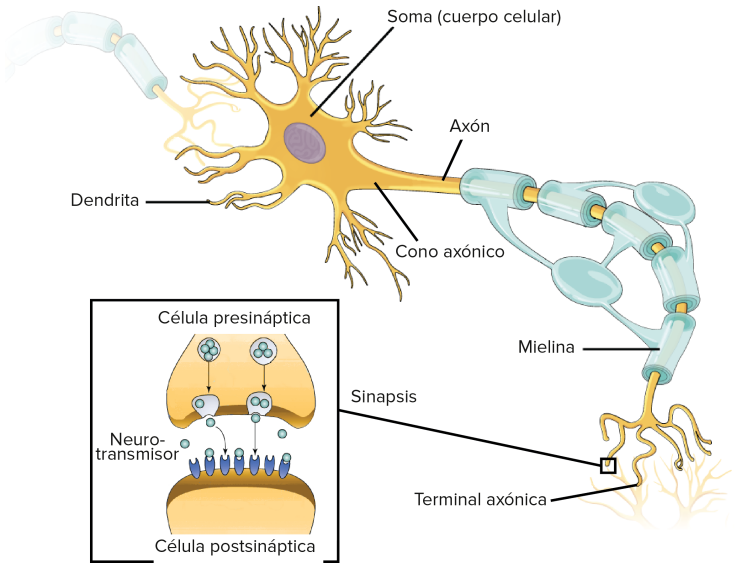


Figura 3.2: Partes y conexiones de una neurona biológica [2]

La Figura 3.2 ilustra las partes y conexiones de una neurona biológica. Estas neuronas poseen unas *dendritas*, cuya función es recibir impulsos eléctricos de otras neuronas. Estos

impulsos se procesan en el Soma, o cuerpo de la célula, y se emite un impulso a otras neuronas a través de los terminales del Axón. La neurona tan solo se excita y emite un impulso eléctrico si las señales recibidas superan un cierto umbral [53].

Al igual que el modelo biológico, la función del *perceptrón* es recibir unos entradas binarias, procesarlas y generar una salida también binaria, en base a un cierto umbral. Para ello, asigna un número de peso a cada entrada, lo que se traducirá en la importancia que se le va a aplicar a cada entrada. Entonces, calcula la suma ponderada de valores de las entradas con sus correspondientes pesos y comprueba si esta supera o no cierto umbral establecido.

$$y = \begin{cases} 0 & \text{si } \sum_{i=1}^{n} w_i x_i \leq umbral \\ 1 & \text{si } \sum_{i=1}^{n} w_i x_i > umbral \end{cases} \tag{3.1}$$

La ecuación anterior (ver Ecuación 3.1) refleja matemáticamente el funcionamiento del *perceptrón simple*, donde si la suma ponderada de las entradas (x_i) con sus correspondientes pesos (w_i) es menor o igual a 0, la neurona no se disparará y la salida será 0. Por el contrario, si la suma ponderada es mayor que cero, esta emitirá un salida con valor 1.

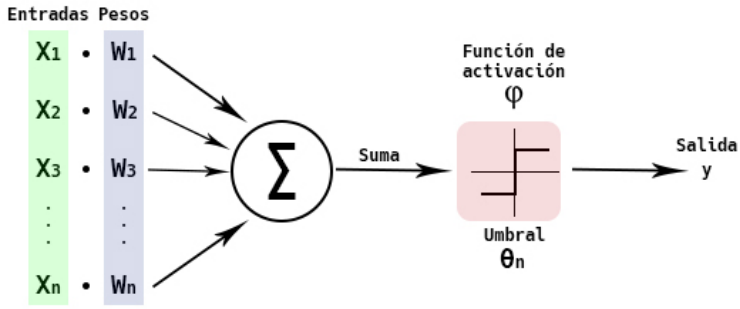


Figura 3.3: Modelo del funcionamiento del perceptrón simple [65]

La Figura 3.3 representa visualmente el funcionamiento del *perceptrón simple*, donde la salida de este, se obtiene al aplicar una función de activación escalonada sobre la suma ponderada de los pesos y las entradas.

Sin embargo, a fin de simplificar la notación, es muy común representar la fórmula del *perceptrón simple* como el producto escalar del vector entrada x $(x_1, x_2, ..., x_n)$ y el vector de pesos w $(w_1, w_2, ..., w_n)$ sumado al valor del sesgo (b), siendo este equivalente al valor en negativo del umbral (ver Ecuación 3.2).

$$y = \begin{cases} 0 & \text{si } w \cdot x + b \leq 0 \\ 1 & \text{si } w \cdot x + b > 0 \end{cases} \tag{3.2}$$

El sesgo o *bias* en inglés, representa la facilidad con la que el perceptrón se disparará, es decir, la facilidad con la que su *output* será 1. Por tanto, un valor de sesgo muy alto se traducirá en una facilidad alta de obtener una salida del perceptrón de valor 1.

Por todo ello, el *perceptrón simple* se trata de un modelo capaz de tomar decisiones binarias sencillas, en base a unos factores o evidencias que tienen distintas importancias en la deliberación del perceptrón [64, Cap. 1].

En el libro "Perceptrons" [55], escrito por Marvin Minsky y Seymour Papert, se demostró que usando varias capas de los perceptrones de Rosenblatt, estos podía aprender y modelar funciones matemáticas simples, como puertas lógicas. A este conjunto de capas de perceptrones, se le denomina *Multilayer Perceptron* (MLP) o perceptrón multicapa en español. Este modelo consiste en una red que conecta la entrada de unos perceptrones con la salida de otros [39, p. 6].

3.2.2. Neurona sigmoide

Para que una red neuronal aprenda a resolver un problema, es necesario realizar pequeñas correcciones graduales en sus neuronas, tanto en los pesos como en los sesgos, de forma que la salida de la red se aproxime al valor esperado y minimice el error entre el valor real y el esperado. En otras palabras, habría que hacer pequeñas modificaciones en los parámetros de las diferentes neuronas (Δw, Δb) que produjesen pequeños alteraciones en la salida ($\Delta output$) hasta conseguir el resultado que se desee.

En el caso de los perceptrones, tanto las entradas como la salidas son binarias. Esto significa que, una pequeña modificación en los pesos o los sesgos de un perceptrón de la red, podría provocar que su salida cambie completamente, de un valor 1 a un valor 0 o viceversa, y que esto alterase drásticamente el funcionamiento de la red neuronal.

Para solventar esta limitación de los perceptrones surgen las neuronas sigmoideas. Este tipo de neuronas es similar al perceptrón, pues somete a una función de activación el resultado de la suma ponderada de pesos y entrada, más el valor del sesgo. A diferencia del perceptrón, esta no trabaja con una función de activación escalonadas, sino con una suavización de la misma, denominada función sigmoide o logística ($\sigma(z)$). Esta función no lineal no discretiza sus entradas como si haría la función escalonada del perceptrón simple, permitiendo valores continuos entre el 0 y el 1, ambos no incluidos. Por ello, la salida de esta neurona se puede definir matemáticamente con la expresión $\sigma(w \cdot x + b)$.

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{3.3}$$

Para entender el funcionamiento de estas neuronas es necesario tener presente la función sigmoide (ver Ecuación 3.3 y Figura 3.4). Se puede apreciar como los valores más extremos se acercan a las asíntotas horizontales y, por tanto, resultan valores muy próximos a 0, en caso de valores negativos, o 1, en caso contrario. Por ejemplo, un valor de 10, dará como resultado $\sigma(10) = 0{,}999 \approx 1$. En contraposición, los valores más próximos al 0, resultan valores progresivos comprendidos entre el 0 y el 1. Por ejemplo, el valor -1, tendrá como resultado $\sigma(-1) = 0{,}269$.

Al conseguir suavizar la salida a valores comprendidos entre 0 y 1, es posible representar probabilidades y así abordar problemas más complejos [64, Cap. 1].

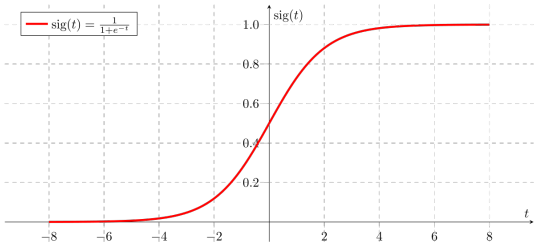


Figura 3.4: Representación de la función sigmoide

3.2.3. Otras funciones de activación

La única diferencia que separa al perceptrón simple de la neurona sigmoidea es la función de activación. Sin embargo, esta no es la única función que se le puede aplicar a la salida de una neurona. Entre las más conocidas se encuentran [39]:

- Función identidad: esta función se trata de una función lineal que no altera el resultado de la neurona, por lo que la salida de la función, es igual a la entrada de la misma. Puede ser útil para realizar regresiones lineales. La Ecuación 3.4 representa la expresión matemática de esta función.

$$f(x) = x \tag{3.4}$$

- Tangente hiperbólica: esta función es una versión de la función sigmoide reescalada al rango de (-1, 1) [39, p. 391]. Esta función se expresa matemáticamente de la siguiente forma:

$$tanh(x) = \frac{e^x + e^{-x}}{e^x - e^{-x}} = 2\sigma(2x) - 1 = \frac{2}{1 + e^{-2x}} - 1 \tag{3.5}$$

- ReLU: la *Rectified Linear Unit* o unidad lineal rectificada en español, es una función de activación que convierte cualquier valor negativo en 0. Para el resto de valores, esta función actúa como la función lineal identidad. La Ecuación 3.6 describe la expresión matemática de esta función.

$$ReLU(x) = max(0, x) = \begin{cases} 0 & \text{si } x < 0 \\ x & \text{si } x \geq 0 \end{cases} \tag{3.6}$$

  Existen algunas modificaciones de esta función, con el fin de solventar algún problema que esta función pueda provocar. Sin embargo, es una función de activación muy utilizada por su alto rendimiento.

3.3. Tipos de Redes Neuronales

A lo largo de esta sección se van a describir algunos de los tipo de redes neuronales más importantes que existe

3.3.1. Redes Neuronales Artificiales (ANN)

Las *Artificial Neural Networks* (ANN) o Redes Neuronales Artificiales en español, son un modelo de cómputo compuesto por neuronas conectadas y organizadas por capas, siendo la salida de una capa, la entrada de la siguiente. Se basan en grafos acíclico direccionales, es decir, no poseen ningún bucle, haciendo que la información se transmita desde la capa de entrada hasta la capa de salida sin que exista ningún tipo de retroceso. Por esta "alimentación hacia delante" en el procesamiento de la información, a estas redes también se las conoce como Feed-Forward Neural Networks (FNN).

Las capas que forman estas redes pueden clasificarse en tres tipos distintos en función del papel que tomen dentro de la red. Esta topología de capas se divide en:

- Capa de entrada: capa cuya función se basa en codificar los datos de entrada. Existe una capa de este por red y posee tantas neuronas como datos contenga la entrada.
- Capas ocultas: este tipo de capa se encarga de procesar los datos que han sido introducidos en la red y su entrada procede de la salida de la capa anterior.
- Capa de salida: esta capa está diseñada de tal forma que codifique una salida acorde al problema que se intenta resolver. Por tanto, el número de neuronas que formará esta capa, dependerá de cada problema.

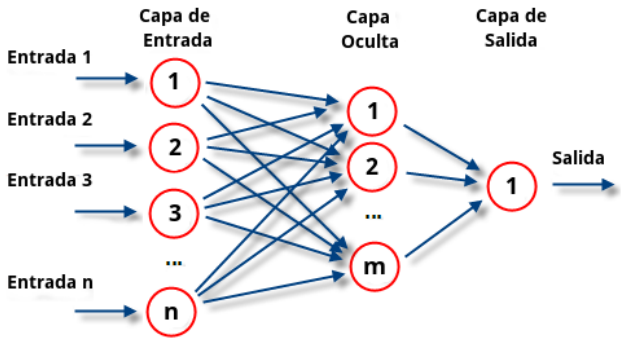


Figura 3.5: Perceptrón multicapa [96]

La Figura 3.5 representa un perceptrón multicapa formado por una capa de entrada que codifica el vector de entrada, una capa oculta que procesa dicha información y una última capa que codifica la salida de la red.

Como ya se comentó en la sección 3.2.2, el aprendizaje de las redes neuronales se basa en ajustar los parámetros de sus neuronas (pesos y sesgos), a fin de conseguir el resultado que se espera. En las secciones posteriores se explicará más en detalle cómo se lleva a cabo este aprendizaje [39][64].

3.3.2. Redes Neuronales Recurrentes (RNN)

Por otro lado, existen las *Recurrent Neural Networks* (RNN) o redes neuronales recurrentes en español. Estas redes son modificaciones de las ANN tradicionales en las que se incluyen bucles en sus capas ocultas, haciendo que sus neuronas se activen temporalmente [39, p. 382]. Este tipo de redes están diseñadas para procesar secuencias de entrada de cualquier longitud, tratando estas como datos relacionados, en lugar de tratar las entradas de forma independiente como haría una ANN tradicional [26, Cap. 10].

Estas redes pueden ser muy útiles para realizar predicciones futuras en base al histórico de datos que se le haya suministrado a la red, semejante a lo que hace el cerebro humano [64].

3.3.3. Redes Neuronales Convolucionales (CNN)

Las *Convolutional Neural Networks* (CNN), o Redes Neuronales Convolucionales en español, son un tipo de redes neuronales cuyo primer diseño data del año 1989. Yann LeCun, su creador, buscaba una red neuronal capaz de detectar números manuscritos en cheques bancarios.

Este tipo de redes neuronales están enfocadas al procesamiento de datos tabulares o cuadriculados, siendo típicamente imágenes. Tanto su estructura como su funcionamiento están inspirados en el sistema de visión del modelo biológico del cerebro humano, que se explicó al inicio de este capítulo (ver Sección 3.1). La importancia principal de las CNN reside en su capacidad de detectar complejos patrones en grandes conjuntos de imágenes. Es por ello por lo que este tipo de red se considera el actual estado del arte en los modelos de visión por ordenador.

Al introducir una imagen en una red neuronal multicapa tradicional, se introducen los píxeles como variables independiente en forma de vector plano. Al no tener en cuenta la posición que posee cada píxel dentro de la imagen, cada píxel es independiente del valor de sus píxeles vecinos, lo que hace imposible detectar patrones visuales en las imágenes. Además, las redes neuronales tradicionales no son capaces de trabajar con imágenes a color de gran tamaño, debido al alto coste computacional que supondría tener un gran número de neuronas en la capa de entrada (alto x largo x 3 colores de rgb).

La *feature engineering*, o ingeniería de características en español, es una de las grandes ventajas atribuidas al deep learning. Una *feature* es una transformación que se realiza sobre los datos a fin de de facilitar el modelado. En una imagen, estas *features* podrían ser patrones visuales distintivos que permitan diferenciar los elementos de la misma. En el caso de los

números manuscritos, un ejemplo de *features* para el número "8" serían dos circunferencias unidas, una encima de otra.

El adjetivo convolucional que se le atribuye a este tipo de redes proviene de convolución, una operación matemática que se realiza en al menos una capa de la red y se basa en multiplicaciones y sumas de matrices.

Esta operación consiste en aplicar un kernel o filtro sobre una imagen. Un kernel es una matriz numérica de una cierta dimensión que según su configuración, se conseguirán resultados distintos al aplicarlo sobre la misma imagen de entrada, es decir, a partir de filtros diferentes se pueden detectar características diferentes. Un ejemplo muy típico es el desenfoque o la detección de bordes. Los resultados de una convolución se conocen comúnmente como mapas de características debido a que revela en que parte de la imagen de entrada se ha detectado una característica concreta. A su vez, también pueden ser nombrados como mapas de activación.

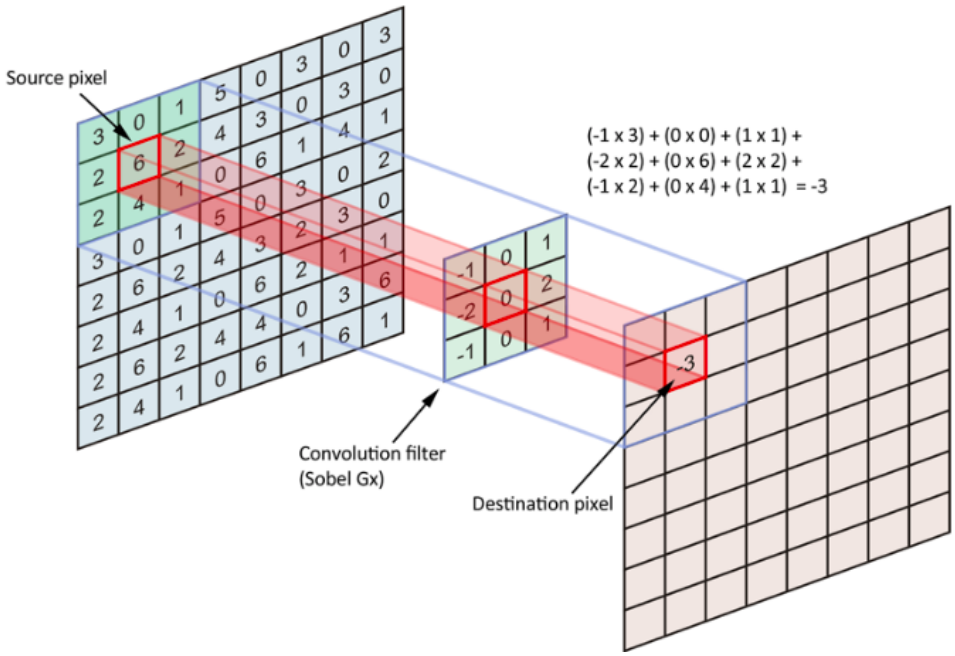


Figura 3.6: Convolución con un *kernel* Sobel G_x para detectar bordes verticales [16]

En la Figura 3.6 se puede ver gráficamente cómo funciona una convolución. En este caso se trata de un kernel de dimensiones 3x3 denominado *Sobel* G_x (matriz verde central), cuya función es encontrar los borde verticales de una imagen. Para ello, se multiplica cada elemento del kernel con su elemento correspondiente en un bloque de 3x3 de la imagen de entrada (matriz de color azul). La suma de los resultados de estas multiplicaciones forman el nuevo valor del mapa de características correspondiente a ese bloque 3x3. Esta operación se aplicará a cada bloque de la imagen de entrada, de forma que se obtengan todos los valores del mapa de características (matriz de color naranja).

-1	0	1
-2	0	2
-1	0	1

Tabla 3.1: Kernel Sobel G_x de dimensión 3x3

Si se analiza el kernel de la convolución de la Figura 3.6 (ver Tabla 3.1), los valores de la columna izquierda son los mismo que los de la derecha con signos invertidos. Esto provocará que si los valores de la columna izquierda y los de la derecha de un bloque de 3x3 de la imagen de entrada son iguales, se anularán y el resultado de la convolución para esa posición será cero, lo que significará que no hay ningún borde en dicha posición. Por el contrario si los valores de ambas columnas laterales son contrarios su activación valdrá más. De esta forma se consigue un filtro que se active ante diferencias de contrastes verticales.

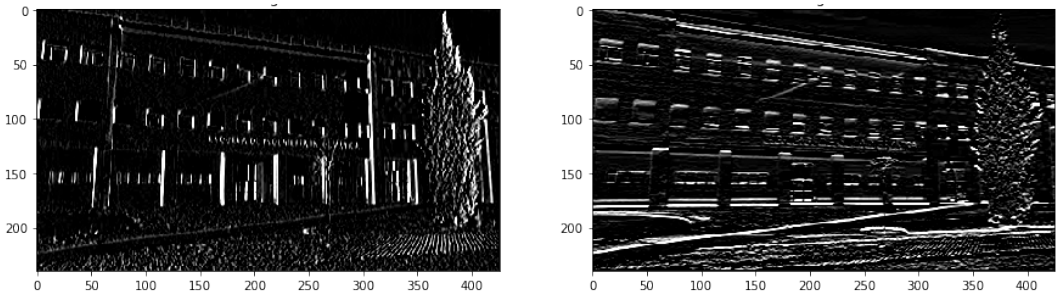


Figura 3.7: Filtro Sobel aplicado a una imagen de la Escuela de Ingeniería Informática

Fuente: elaboración propia.

La Figura 3.7 ilustra el resultado de un ejemplo donde se ha aplicado los filtros Sobel para la detección de bordes vertical (imagen de la izquierda) y horizontales (imagen situada a la derecha) sobre una imagen de la Escuela de Ingeniería Informática de la Universidad de Valladolid.

En este caso, al tratarse de un kernel 3x3, el resultado de la convolución será una imagen cuyas dimensiones son $H-2$ x $W-2$, siendo H y W el alto y el ancho de la imagen de entrada, respectivamente. Esto se debe al número de bloques 3x3 distintos que pueden existir en la imagen de entrada. Para no perder los píxeles exteriores, existe el padding o relleno. Esta técnica, permite que el mapa de activación de salida pueda ser del mismo tamaño (o mayor) que el de la imagen de entrada. Para ello se añade un marco exterior con píxeles adicionales, de forma que se puedan aplicar los kernels sobre los extremos. En este caso concreto, habría que añadir un padding de 2 píxeles para obtener una imagen resultante de tamaño HxW. De forma general, en filtros de dimensiones impares (los más comunes), el padding necesario para obtener las mismas dimensiones de salida que las de entrada es `ks//2`, siendo ks el tamaño del kernel y "//" la división descartando la parte decimal del cociente.

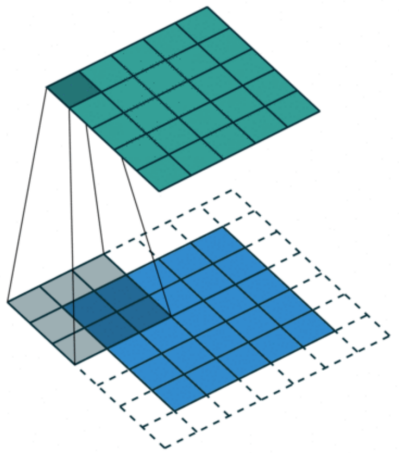


Figura 3.8: Padding en una matriz de entrada 5x5 con un kernel 3x3 para obtener una matriz de salida 5x5 [56]

La Figura 3.8 ilustra perfectamente cómo al aplicar un kernel de dimensiones 3x3 (matriz sombreada) sobre una entrada de tamaño 5x5 (matriz de color azul) y un padding de 1 (celdas blancas alrededor de la matriz de entrada), se consigue una salida con las mismas dimensiones que la entrada (matriz de color verde).

En la Figura 3.6 el kernel se desplaza píxel a píxel. Esto se puede modificar según su "stride". Se denomina "stride" al desplazamiento que se aplica al kernel sobre los píxeles de la imagen de entrada. Mientras que las convoluciones stride-1 permiten añadir capas convolucionales sin alterar el tamaño de la salida (junto a un padding de `ks//2`), las convoluciones stride-2 son útiles si se desea reducir el tamaño de la salida. En las CNN se busca añadir capas convolucionales con convoluciones stride-2, una seguida de otra, a fin de reducir el tamaño de imagen hasta un convertirlo en un vector.

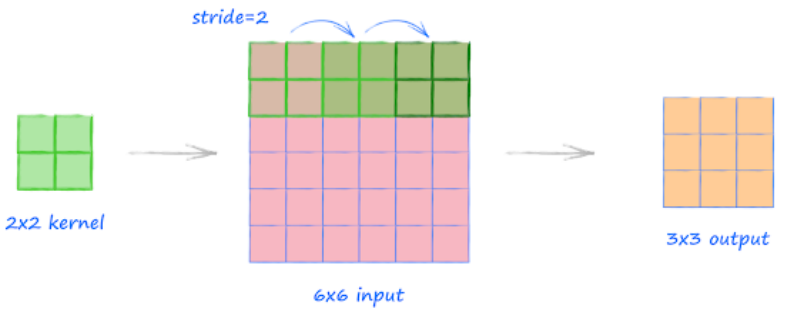


Figura 3.9: Convolución stride-2 con un kernel 2x2 sobre una matriz de entrada 6x6 [1]

Tal y como se puede observar en la Figura 3.9, el resultado de aplicar una convolución stride-2 con un kernel 3x3 sobre una matriz de entrada 6x6 resulta una matriz de inferiores dimensiones.

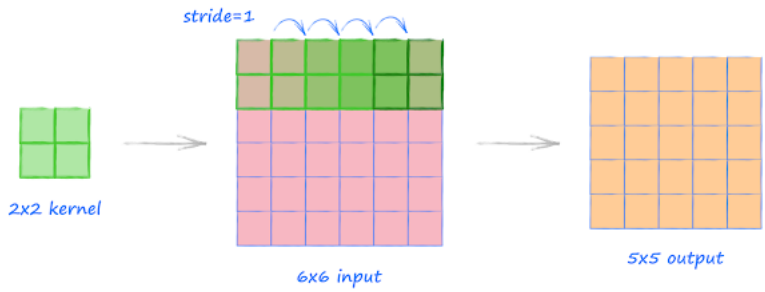


Figura 3.10: Convolución stride-1 con un kernel 2x2 sobre una matriz de entrada 6x6 [1]

Para comprender cómo afecta el stride sobre la convolución, en la Figura 3.10, se aplica una convolución stride-1 sobre el mismo ejemplo que en la imagen anterior. En este caso, a partir de la misma matriz 6x6, el resultado es una nueva matriz 5x5. Si se incluyese un padding de 1, la matriz de salida sería del mismo tamaño que la de entrada.

Para calcular el tamaño que tendrá el mapa de características de una convolución hay que aplicar la siguiente fórmula:

$$((s + 2 * padding - ks)//stride) + 1 \tag{3.7}$$

En la fórmula anterior, la "s" representa el tamaño de entrada, "ks" el tamaño del kernel, "padding" el padding que se le aplica a la convolución y "stride" el desplazamiento del kernel sobre los píxeles de la imagen de entrada [39, cap. 13].

Una vez comprendida la convolución, la operación más importante de este tipo de redes, se va a describir los tres tipos de capas que componen las Redes Neuronales Convolucionales. Según la función que empeñan en la red, las capas se pueden clasificar en los tres siguientes tipos [66] [26, Cap. 9]:

- **Capas convolucionales**: son aquellas capas que realizan la operación de convolución explicada anteriormente. Los resultados de las diferentes convoluciones de dicha capa, se pasan por una función de activación no lineal, siendo ReLU la función de activación más utilizada. Esta activación se aplica sobre cada píxel de la salida de la convolución.

  Al igual que en las redes neuronales tradicionales se ajustan los parámetros de sus neuronas durante el entrenamiento para obtener los resultados esperados, en las capas convolucionales de este tipo de red, los parámetros que se van a ajustar en el entrenamientos son los que determinarán la configuración de los distintos kernel, de forma que

la red aprenda a detectar las características y patrones más útiles para conseguir la mejor precisión de la red ante una tarea específica. Cada capa convolucional contiene un número de pesos equivalente a la multiplicación del número de canales de entrada por el numero de canales de salida, todo ello multiplicado por el número de valores que contiene el kernel (ks^2, siendo ks el tamaño del kernel). En cuanto a los sesgos o *bias*, existe un parámetro de sesgo por cada canal de salida. La suma del número de pesos y sesgos de todas las capas, conforman el número total de parámetros que la red deberá entrenar [39, Cap. 13].

- **Capas pooling**: estas capas suelen ir junto a las capas convolucionales y su función es aplicar una operación sobre el mapa de características de la capa convolucional para modificarlo. Hay dos operaciones típicas en este tipo de capas, que son: *max-pooling* y *average-pooling*. La primera de ellas, obtiene el valor más grande de cada bloque de un tamaño determinado (tamaño del *pool*) del mapa de activación. Esta operación reducirá el tamaño del mapa de características en un factor igual al tamaño del *pool*. Por el contrario, *average-pooling* se encarga de calcular el valor medio de dichos bloques. Ambos casos podrían verse como una forma de resumir el mapa de activación y reducir toda la información a lo más relevante [10].

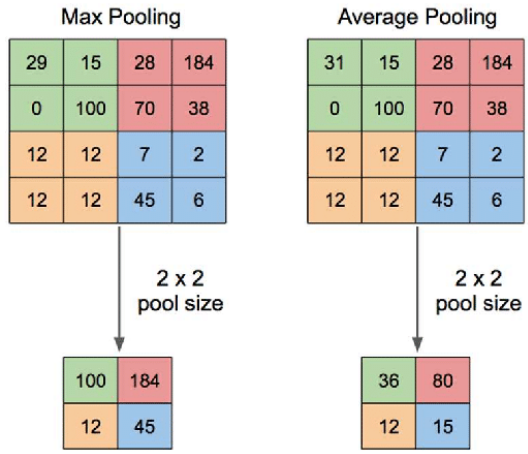


Figura 3.11: Operación de *max-pooling* frente a *average-pooling* [76]

La Figura 3.11 representa el resultado de aplicar por separado las dos operaciones *pooling* explicadas anteriormente sobre un mapa de características cualquiera.

- **Capas fully-connected**: estas capas conforman una red neuronal tradicional a la que se introducen las distintas activaciones obtenidas como resultado de haber aplicado todas las capas convolucionales de la red. El nombre de *fully-connected* proviene de su configuración, ya que todas las salidas de la capa anterior conforman las entradas de cada neurona de la capa actual. El conjunto de estas capas en una red neuronal convolucional desemboca en una última capa de salida cuyo número de neuronas dependerá del problema que se está resolviendo.

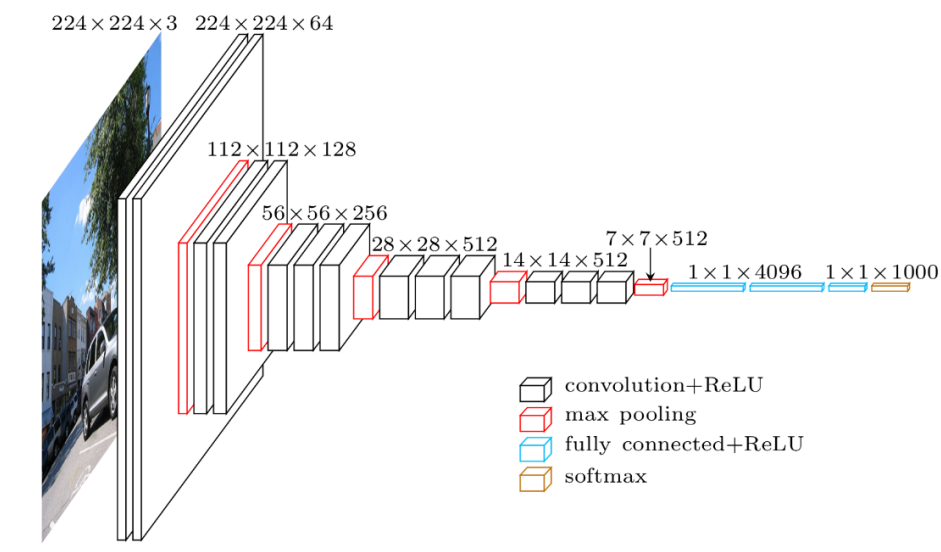


Figura 3.12: Ejemplo de red neuronal convolucional para clasificar fotografías [82]

En la figura anterior (ver Figura 3.12), se puede ver un ejemplo de cómo se aplican las diversas capa explicadas. Inicialmente, la imagen de entrada se codifica con tres canales correspondientes al los colores RGB. Cada canal de entrada posee unas dimensiones 224x224. A partir de esta entrada de dimensiones 224x224x3, se van aplicando diferentes capas convolucionales y capas *max-pooling*. Se puede observar como la arquitectura sigue una forma cónica. Esto refleja cómo la imagen de entrada se va reduciendo espacialmente en su alto y ancho, a medida que avanza por la red. Al contrario que la resolución, la profundidad va aumentando, es decir, aumenta la cantidad de mapas de características.

Al igual que se explicó en el modelo biológico de la visión humana, a medida que se avanza hacia lo profundo de la red neuronal convolucional, los patrones que detectan las capas convolucionales se vuelven progresivamente más abstractos. Las primeras capas codifican características generales, como la detección de bordes, y según se avanza en profundidad, las características se vuelven más específicas, como puede ser la detección de un ojo. Esta es la razón por la que las redes neuronales profundas son tan poderosas [39, Cap. 13].

En el momento en el que se han detectado todos los patrones o características necesarias, estos mapas de características pasarán a ser la entrada de una red neuronal multicapa tradicional para determinar la salida final de la red [26, Cap. 9].

Sin embargo, al final de la red, se ha colocado una nueva capa no mencionada antes, denominada *softmax*. El resultado de esta capa viene determinado por la función *softmax* que se aplica sobre la salida de cada neurona (z_j^L). Esta función de activación es similar a la función sigmoide que se explicó anteriormente.

$$a_j^L = \frac{e^{z_j^L}}{\sum_k e^{z_k^L}} \tag{3.8}$$

Poniendo atención a esta función (ver Ecuación 3.8), se puede observar cómo la acumulación de las salidas de esta capa, resultará un valor total de 1 y que todos los valores de este vector de activación son números positivos gracias a la función exponencial e^x a la que se somete z^L. En definitiva, el vector de activación de esta capa puede interpretarse como un distribución de probabilidades que la red asigna a cada posible salida, en función de la confianza o claridad de la predicción. Este comportamiento, hace que esta función sea muy útil en la ultima capa de una red neuronal destinada a tareas de clasificación [64, Cap. 3].

3.4. Aprendizaje de Redes Neuronales

3.4.1. Función de coste

El aprendizaje de una red neuronal reside en acortar la diferencia entre los resultados reales obtenidos por esta y los valores esperados, tal y como se comentó en la sección 3.2.2. Una función de coste, también conocida como función de pérdida o de objetivo, es una medida de rendimiento con la que el modelo pueda determinar el grado de inexactitud del aprendizaje, es decir, si la red neuronal devuelve un resultado muy próximo al valor esperado, la función de coste tomará valores muy cercanos a cero. De esta forma, el entrenamiento consistirá en minimizar el coste que se obtiene como resultado de esta función.

Existen multitud de funciones de costes a elegir según el problema que ha de resolver el modelo y el tipo de activación de su última capa. A continuación se van a enumerar algunas de las funciones de coste más relevantes y conocidas. Para problemas de regresión, es decir, problemas en los que el modelo posee una salida numérica, las funciones más típicas son las siguientes:

- Error cuadrático medio (o MSE, Mean Squared Error): Esta función de coste es la opción más típica en este tipo de problemas.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{3.9}$$

  La ecuación anterior (ver Ecuación 3.9) describe la expresión matemática del error cuadrático medio, en la que el coste se obtiene de la media de las diferencias elevadas al cuadrado de los valores esperados (y_i) y los valores predichos ($\hat{y}_i$).

  Elevar al cuadrado dicha diferencia provoca que los errores más grandes se tengan más en cuenta y se penalicen sobre el valor final del coste.

  Esta función de coste cuenta con una versión que minimiza ligeramente el efecto penalizador de MSE. Dicha función se conoce como el error logarítmico cuadrático medio o Mean Squared Logarithmic Error (MSLE) en inglés.

- Error absoluto medio (o MAE, Mean Absolute Error): esta función de coste se basa en la distancia media entre los valores reales y los predichos.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{3.10}$$

  La fórmula anterior (ver Ecuación 3.10) representa el error absoluto medio, donde $\hat{y}_i$ es el valor predicho, y_i el valor real y n el número de predicciones o de entradas.

  Si se compara esta función de coste con la anterior (MSE), tan solo distan en el tratamiento de la diferencia entre las predicciones y los valores reales, puesto que en este caso no se eleva al cuadrado. Esto provoca que MAE sea una función de coste más robusta y tolerable a valores atípicos.

Sin embargo, si el modelo que se desea entrenar está diseñado para resolver problemas de clasificación, la función de coste que se utiliza es la entropía cruzada (o Cross-Entropy Loss en inglés) o alguna versión de la misma [7]. Esta funciona bien en problemas de dos o más categorías, permitiendo a su vez, entrenamientos rápidos y confiables. Esta función impone una mayor penalización a aquellas predicciones cuya probabilidad es alta pero realmente son incorrectas, es decir, penaliza cuando el modelo está muy seguro de una predicción que es incorrecta [39].

$$CrossEntropy = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log\left(\hat{y}_i\right) + \left(1 - y_i\right)\log\left(1 - \hat{y}_i\right)\right] \tag{3.11}$$

En la Ecuación 3.11 se describe la expresión matemática de la función de coste de entropía cruzada [64].

3.4.2. Descenso del gradiente

El algoritmo del descenso del gradiente es un algoritmo de optimización de funciones cuyo objetivo es encontrar el conjunto de valores de entrada de una función, de forma que se minimice el valor de la salida de esta, es decir, que encuentre un mínimo de la función. Este algoritmo se utiliza en el campo del *machine learning* para poder llevar a cabo el aprendizaje de las redes neuronales. Este buscará, de forma iterativa, optimizar una determinada función de coste ajustando los parámetros de la red (pesos y sesgos), hasta encontrar aquellos que minimicen la salida de dicha función.

Michael Nielsen, en su libro *Neural networks and deep learning*, describe el algoritmo del descenso de gradiente con una analogía muy ilustrativa. El autor compara el algoritmo con el comportamiento que experimentaría una pelota que se deja en un lugar cualquiera de la ladera de un valle. Esta, sujeta a las leyes físicas, comenzará a rodar en dirección a la mayor inclinación hasta alcanzar el lugar cercano con menor altitud (mínimo local) [64, Cap. 1].

Para comprender el funcionamiento de este algoritmo es necesario entender el significado del gradiente y, por tanto, recordar qué representa una derivada. La derivada de una función ($\frac{df}{dx}$) en un punto a, representa la pendiente de la recta tangente a la función en dicho punto y, por consiguiente, la pendiente de la función en el punto a. La derivada simboliza el cómo afecta en la salida de la función, una pequeña modificación en la entrada. En el caso de funciones multivariables, este cálculo se puede realizar para cada una de las dimensiones de la función, en forma de derivadas parciales ($\frac{\partial f}{\partial z}$). De esta forma, es posible obtener las distintas pendientes de la función, relativas a cada variable [52, Cap. 7].

El gradiente de una función (∇f) es el vector que contiene el conjunto de todas las derivadas parciales de dicha función, ofreciendo la dirección de máxima pendiente ascendente en un punto específico, es decir, la dirección en la que el valor de la función experimenta el mayor crecimiento posible [69]. La Ecuación 3.12 representa el gradiente de una función de dos variables.

$$\nabla f(x, y) \equiv (\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y})^T \tag{3.12}$$

Una vez entendidos estos conceptos matemáticos, se va a describir el proceso iterativo que sigue el algoritmo del descenso del gradiente. En primer lugar, se establecen unos valores iniciales aleatorios que servirán como punto de partida. A partir de ellos se calcula valor del coste asociado a esos valores y el gradiente de la función de coste con respecto a estos valores iniciales. En base al vector gradiente, el algoritmo avanza en dirección opuesta a este, es decir, actualiza los parámetros de la red de forma que se reduzca lo máximo el coste. Una vez actualizados los valores, se repite el proceso hasta que la pérdida producida por los parámetros ajustados no varíe o lo haga mínimamente y de forma muy lenta. A este instante se le denomina convergencia [14].

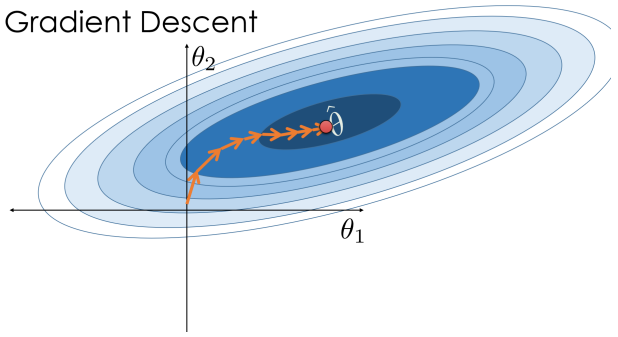


Figura 3.13: Proceso del descenso de gradiente hasta converger [70]

La Figura 3.13 representa este proceso iterativo de una forma gráfica y simple, donde la zona oscura representa un mínimo de la función de coste y el origen de coordenadas representa los valores iniciales de los dos parámetros de la función (θ_1 y θ_2), es decir, el punto de partida del algoritmo. A su vez, es posible apreciar cómo las primeras flechas son más largas que las últimas. Esto se debe a que el punto de partida está sobre un desnivel más pronunciado que

la zona en la que se encuentra el mínimo de la función, tal y como reflejan la cercanía de las curvas de desnivel. Una mayor cercanía de estas curvas de desnivel refleja una mayor inclinación.

Este algoritmo cuenta con un nuevo parámetro conocido como *learning rate* (α) o tasa de aprendizaje en español. Este valor, comúnmente comprendido entre 0 y 1, se encarga de escalar el gradiente, es decir, determinará la magnitud de las actualizaciones que se van a aplicar sobre los parámetros. Un learning rate pequeño va a suponer actualizaciones de parámetros muy pequeñas, lo que implicará un aprendizaje lento que requerirá de muchas épocas. El concepto de época hace referencia a una pasada del conjunto de datos completo con el que se está entrenando por la red. Un exceso de épocas en el entrenamiento supondrá, además de la alta demora en los tiempos requeridos para el aprendizaje, un potencial problema de sobreajuste que, tal y como se verá en secciones posteriores, hará que la precisión del modelo empeore. Sin embargo, un *learning rate* muy grande puede provocar que el modelo nunca converja, es decir, nunca alcance un mínimo relativo en la función de coste.

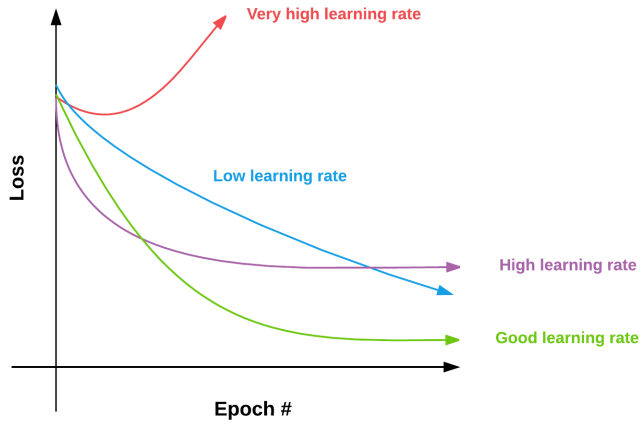


Figura 3.14: Coste simulado para los distintos tamaños de learning rate

La Figura 3.14 representa visualmente cómo afecta la elección del learning rate sobre las pérdidas de la función de coste a lo largo del tiempo, tal y como se ha explicado en el párrafo anterior.

La siguiente ecuación (ver Ecuación 3.13) describe la operación que realiza el algoritmo del descenso del gradiente para actualizar los parámetros de la función.

$$\theta := \theta - \alpha \nabla C \tag{3.13}$$

En dicha ecuación (ver Ecuación 3.13), los parámetros (θ) se actualizan en sentido contrario al gradiente de función de coste ($-\nabla C$) escalado según el *learning rate* (α) [64].

A fin de asimilar y explicar de una forma práctica y visual este algoritmo y cómo afecta en el éxito de su resultado la elección del learning rate, se ha realizado un estudio con una

función de coste cualquiera. La función de coste sobre la que se va a experimentar ha sido tomada de un ejemplo del artículo de la Wikipedia destinado a este algoritmo [94] y está definida a continuación (ver Ecuación 3.14).

$$F(x, y) = \sin\left(\frac{1}{2}x^2 - \frac{1}{4}y^2 + 3\right)\cos\left(2x + 1 - e^y\right) \tag{3.14}$$

En la siguiente figura (ver Figura 3.15), se ha representado una gráfica tridimensional de la función elegida para este experimento. Para ello, se ha utilizando el *toolkit* "mplot3d" de la biblioteca "Matplotlib" de Python.

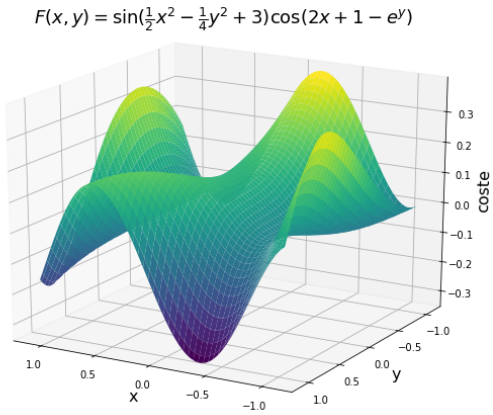


Figura 3.15: Representación tridimensional de la función de estudio

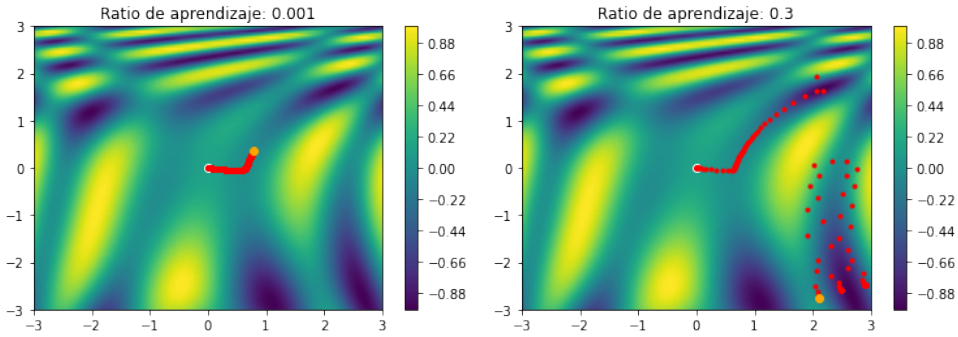
Fuente: elaboración propia a partir de una función de ejemplo que se describe en el artículo de "Gradient descent" de la *Wikipedia* [94].

Pese a que lo común es determinar el punto de partida de forma aleatoria, en este caso es necesario que sea invariante para poder observar cómo afectan los resultados del algoritmo partiendo de un mismo punto, únicamente alterando la tasa de aprendizaje. El punto de partida elegido manualmente es el $(0, 0)$.

Para el experimento se van a plantear los cuatro escenarios típicos que se dan en la elección del learning rate. Los dos primeros van a representar malas elecciones, comenzando por una tasa pequeña de valor 0,001 y siguiendo con una tasa grande de 0,3. Los dos casos restantes serán buenas elecciones, con un learning rate de 0,01 y uno algo más óptimo de 0,05.

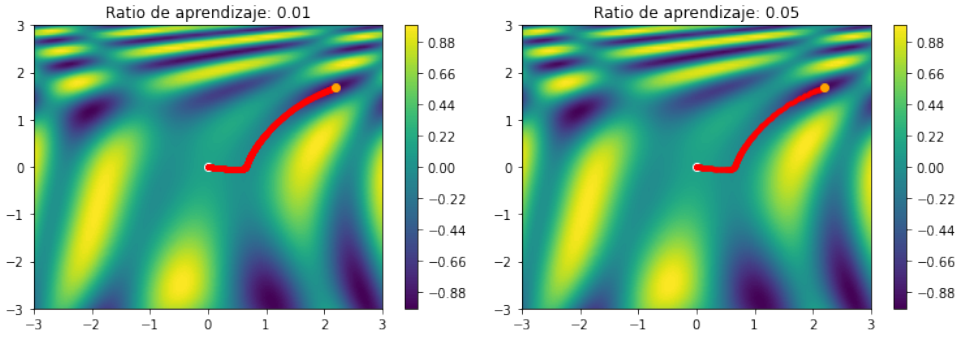
En la siguiente figura (ver Figura 3.16) se muestran los valores que se han ido actualizando con el algoritmo del descenso de gradiente en 10000 iteraciones para los cuatro distintos *learning rates*. Para una mejor interpretación, se han diferenciado en cada imagen los distintos valores con un código de colores. El punto de partida está representado por un punto blanco, los valores intermedios por puntos rojos y el valor final está ilustrado con un punto superpuesto naranja. Junto a cada imagen se ha colocado una barra de colores, la cual des-

cribe que los colores más claros hacen referencia a los valores de pérdida más altos y, por el contrario, los colores más oscuros a los más bajos.



(a) Learning rate pequeño

(b) Learning rate grande



(c) Learning rate bueno

(d) Learning rate óptimo

Figura 3.16: Evolución de los parámetros al aplicar el descenso de gradiente con cuatro *learning rates* distintos

En el primer caso, la ilustración (ver Figura 3.16a) muestra como los valores siguen perfectamente el algoritmo del descenso de gradiente en busca de un mínimo relativo. Al tratarse de un valor excesivamente pequeño, el proceso finaliza a medio camino, siendo insuficientes el número de iteraciones de algoritmo para alcanzar la convergencia en el mínimo.

En segundo lugar estaría el caso de un learning rate grande. Como se puede observar en la segunda imagen (ver Figura 3.16b), inicialmente los valores de los parámetros se actualizan dirigiéndose hacia un mínimo relativo. Pese a que el learning rate es un valor relativamente alto, las actualizaciones en los primeros tramos son bastante pequeñas. Esto se puede deber a la pequeña inclinación que experimenta la función en dicha zona. A continuación, los valores acaban alcanzando una zona próxima a un mínimo local. En ese momento, los valores se actualizan buscando la convergencia. Sin embargo, la actualización aplicada es ligeramente más grande de lo debido, alcanzando los valores a un punto de la función muy cercano pero con un alto coste. La gran pendiente que existe en este punto provocada por su alta proximidad al mínimo local, sumado al alto *learning rate* establecido, provoca que los valores

de la función den un salto muy grande en dirección al mínimo, lo que hace que se desvíen a una nueva zona ajena al mínimo relativo actual. En esta nueva zona, el optimizador sigue actualizando los valores en busca de otro mínimo pero, como puede verse, el punto final queda próximo al nuevo mínimo pero sin llegar a converger en él, tal y como era de esperar.

Por último, las figuras inferior izquierda (ver Figura 3.16c) e inferior derecha (ver Figura 3.16d), muestran un recorrido aparentemente idéntico, donde ambas tasas de aprendizaje cumplen con su cometido, converger en un mínimo de la función, acorde al número de iteraciones establecidas.

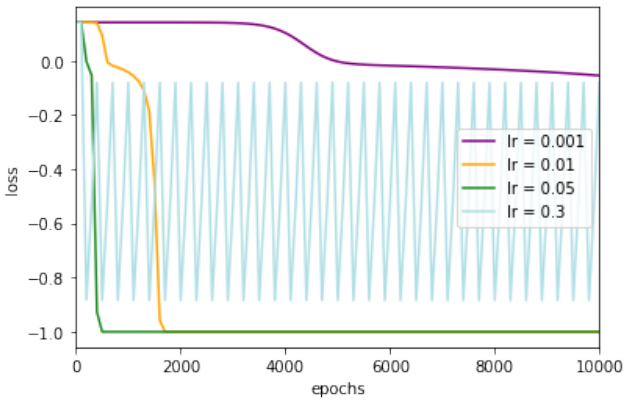


Figura 3.17: Costes de cada *learning rate* a lo largo de las iteraciones

Analizando los valores de pérdida que se van obteniendo a lo largo de todas las iteraciones para los diferentes casos (ver Figura 3.17), se puede observar claramente la diferencia ente los dos casos en los que algoritmo converge. Pese a que con un learning rate de 0,01 se consigue alcanzar el mínimo local sin problemas, una tasa más afinada como es 0,05 consigue la convergencia con mucha más rapidez, lo que supone un gran ahorro en el tiempo de computación, haciendo de este último caso, el más óptimo.

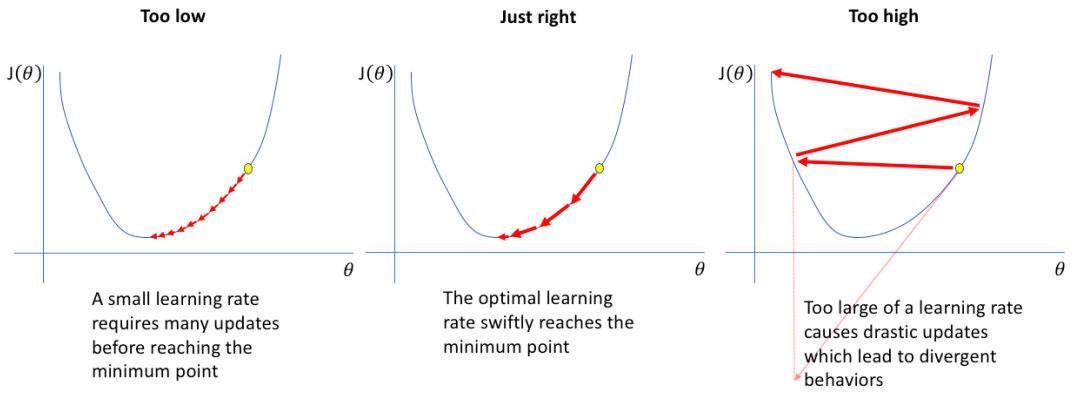
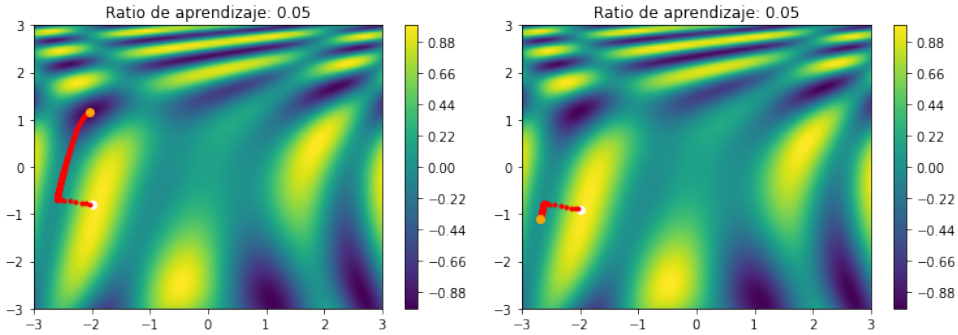


Figura 3.18: Efecto de la magnitud del *learning rate* sobre el descenso de gradiente [44]

La Figura 3.18 refleja y explica a la perfección lo ocurrido en cada caso de prueba, especialmente la divergencia experimentada en el caso del learning rate alto.

Concluyendo este breve experimento, todo lo expuesto demuestra cuán importante es la elección de un buen learning rate que se adapte a la función de coste utilizada y al número de épocas establecidas en el entrenamiento, sobre el éxito y la eficiencia del aprendizaje de un modelo.

Sin embargo, determinar un buen learning rate no lo es todo. Si el descenso de gradiente parte del punto $(-2, -0{,}8)$ y con un learning rate de 0,05 (valor descrito como óptimo en los casos anteriores), el proceso del descenso de gradiente deriva los valores de los parámetros a un mínimo relativo donde la función adopta valores pequeños de pérdida, de forma que optimiza bien la función (ver Figura 3.19a). Por el contrario, si se parte de un punto ligeramente más próximo al eje de abscisas, como es el punto $(-2, -0{,}9)$, el punto final que alcanza el descenso de gradiente es un mínimo también pero, en este caso, mucho más superficial y, por ello, bastante menos óptimo que el anterior (ver Figura 3.19b).



(a) Mínimo relativo profundo (b) Mínimo relativo superficial

Figura 3.19: Evolución de los parámetros al aplicar el descenso de gradiente partiendo de distintos puntos

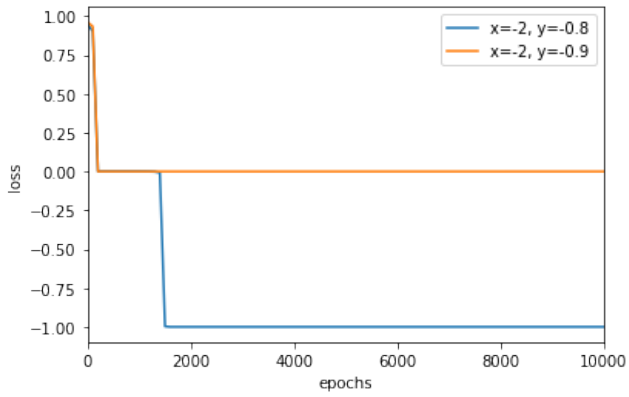


Figura 3.20: Costes obtenidos por los correspondientes puntos de partida

Lo que se intenta explicar con este último ejemplo es que pese a que la elección de learning rate es aparentemente buena, el punto de partida, a su vez, determina el resultado final según los mínimos locales que le rodean. Por tanto, este factor influye en el grado de optimización de la función conseguido por el algoritmo. Es por esta razón por la que se parte de valores aleatorios.

3.4.3. Algoritmo de Backpropagation

Una vez entendido el algoritmo del Descenso del Gradiente, se presenta el algoritmo de diferenciación conocido como *Backpropagation* (propagación hacia atrás de errores o retropropagación en español). Pese a que el algoritmo de *backpropagation* ya se introdujo en los años 70, no fue hasta 1986 que se publicó el artículo "Learning representations by back-propagating errors" [79] de David Rumelhart, Geoffrey Hinton y Ronald Williams donde se popularizó dicho algoritmo. En este artículo se compara la eficiencia del aprendizaje utilizando este nuevo enfoque frente a las técnicas anteriores sobre distintas redes neuronales. Esta comparativa reflejó el gran potencial y relevancia de este algoritmo para abordar problemas que hasta el momento eran inalcanzables [64, Cap. 2].

El algoritmo del Backpropagation trabaja junto al del descenso del gradiente. Como ya se ha explicado en la sección anterior, el descenso del gradiente se encarga de optimizar el coste de una función en base al vector gradiente. Por ello, el objetivo del algoritmo de Backpropagation es calcular el conjunto de derivadas parciales correspondientes a cada parámetro de la red con respecto al coste o pérdida la misma, es decir, se busca la responsabilidad o repercusión que ha tenido cada neurona sobre el coste final de la red. De esta forma, dicho algoritmo consigue construir el vector gradiente que será utilizado por el descenso del gradiente para optimizar dicho coste de la red.

Este algoritmo se divide en dos etapas: Feedforward o pasada hacia delante y Backward o pasada hacia atrás. La primera de ellas consiste en calcular los distintos resultados de cada neurona a partir de una entrada, a medida que se avanza por la red. Una vez se calcule la activación final de la red y se calcule el error de esta, se lleva a cabo la propagación del error hacia atrás.

Pese a la complejidad matemática que pueda suponer este algoritmo, se va a intentar explicar dichos cálculos de la forma más simple posible sobre una red neuronal tradicional.

Tal y como se ha explicado antes, tras calcular todas sumas ponderadas y las activaciones de cada neurona, se procede al cálculo de las derivadas parciales de cada capa. Para obtener las derivadas parciales del coste con respecto a los pesos ($\frac{\partial C}{\partial w^L}$) y los sesgos ($\frac{\partial C}{\partial b^L}$) de la última capa ($L$), se va a partir del coste de esta última capa.

$$C(a(z^L)) = C(a(w^L \cdot a^{L-1} + b^L)) \tag{3.15}$$

La expresión anterior (ver Ecuación 3.15) describe el coste de la última capa, donde "C" representa la función de coste, "a" la función de activación y "z^L" la suma ponderada de

cada neurona de la última capa. Esta suma ponderada es el resultado de multiplicar los pesos (w^L) por cada una de sus entradas, las cuales equivalen a las activaciones de la capa anterior (a^{L-1}), más sus sesgos (b^L). Para calcular estas derivadas parciales a partir de dicha composición de funciones, se ha de utilizar la *Chain rule* o regla de la cadena en español, de forma que estas derivadas parciales quedan de la siguiente forma:

$$\frac{\partial C}{\partial w^L} = \frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L} \cdot \frac{\partial z^L}{\partial w^L} = \delta^L \cdot \frac{\partial z^L}{\partial w^L} = \delta^L \cdot a^{L-1} \tag{3.16}$$

$$\frac{\partial C}{\partial b^L} = \frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L} \cdot \frac{\partial z^L}{\partial b^L} = \delta^L \cdot \frac{\partial z^L}{\partial b^L} = \delta^L \cdot 1 \tag{3.17}$$

El símbolo delta (δ_L) que aparece en las dos ecuaciones anteriores (Ecuación 3.16 y 3.17), representa lo que se denomina como error imputado a la neurona ($\frac{\partial C}{\partial z^L}$), es decir, el impacto que tiene cada neurona sobre el resultado final. Este error de la última capa se calcula como la multiplicación de la derivada de la función de coste multiplicado por la derivada de la función de activación que se esté aplicando en las neuronas de esa capa, tal y como se describe en la Ecuación 3.18. Un valor alto representará que una variación en el resultado de la neurona (suma ponderada) provoca un efectos sobre la salida de la red.

$$\delta^L = \frac{\partial C}{\partial z^L} = \frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L} \tag{3.18}$$

Por último en la Ecuación 3.17, la derivada parcial de z^L respecto de b^L resulta 1 ya que se trata de un término independiente. Sin embargo, en la Ecuación 3.16, la derivada parcial de z^L respecto de w^L es igual a las entradas de las neuronas de dicha capa o, en otras palabras, a las activaciones de la capa anterior (a^{L-1}).

Una vez calculadas las derivadas parciales de los parámetros de la capa final, se retrocede a la capa anterior ($L-1$). Esta vez habrá que aplicar la regla de la cadena a la siguiente expresión:

$$C(a^L(w^L a^{L-1}(w^{L-1}a^{L-2} + b^{L-1}) + b^L)) \tag{3.19}$$

Para dicha expresión (ver Ecuación 3.19), las derivadas parciales de los parámetros de esta capa quedan de la siguiente forma:

$$\frac{\partial C}{\partial w^{L-1}} = (\frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L}) \cdot \frac{\partial z^L}{\partial a^{L-1}} \cdot (\frac{\partial a^{L-1}}{\partial z^{L-1}}) \cdot (\frac{\partial z^{L-1}}{\partial w^{L-1}}) \tag{3.20}$$

$$\frac{\partial C}{\partial b^{L-1}} = (\frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L}) \cdot \frac{\partial z^L}{\partial a^{L-1}} \cdot (\frac{\partial a^{L-1}}{\partial z^{L-1}}) \cdot (\frac{\partial z^{L-1}}{\partial b^{L-1}}) \tag{3.21}$$

Teniendo en cuenta que la expresión "$\frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L}$" es el error de la última capa (δ^L), que "$\frac{\partial z^{L-1}}{\partial b^{L-1}}$" equivale a 1 y "$\frac{\partial z^{L-1}}{\partial b^{L-1}}$" es igual a las activaciones de la capa anterior (a^{L-2}), al igual que en la capa anterior, y que "$\frac{\partial a^{L-1}}{\partial z^{L-1}}$" representa la derivada de la función de activación, tan solo queda una derivada parcial sin calcular. Esta última derivada parcial de z^L respecto de a^{L-1}, representa cómo afecta a las sumas ponderadas de las neuronas de la última capa la variación de las activaciones de la capa actual. El resultado de esta derivada parcial equivale al conjunto de pesos de la última capa (w^L). De esta forma se está retropropagando el error hacia la capa anterior en función de los pesos o ponderaciones de cada entrada. Teniendo esto en cuenta, las expresiones anteriores se pueden reducir a las siguientes:

$$\frac{\partial C}{\partial w^{L-1}} = \delta^L \cdot w^L \cdot \frac{\partial a^{L-1}}{\partial z^{L-1}} \cdot a^{L-2} = \delta^{L-1} \cdot a^{L-2} \tag{3.22}$$

$$\frac{\partial C}{\partial b^{L-1}} = \delta^L \cdot w^L \cdot \frac{\partial a^{L-1}}{\partial z^{L-1}} \cdot 1 = \delta^{L-1} \cdot 1 \tag{3.23}$$

Como se puede observar en ambas expresiones (ver Ecuación 3.22 y 3.23), la expresión final incluye el error imputado a las neuronas de esta capa (δ^{L-1}). Esto se debe a que la expresión "$\delta^{L-1} \cdot w^L \cdot \frac{\partial a^{L-1}}{\partial z^{L-1}}$" equivale a la derivada parcial del coste con respecto a las sumas ponderadas de la capa actual ($\frac{\partial C}{\partial z^{L-1}}$).

A partir de este punto, para calcular las derivadas parciales de los parámetros de una capa intermedia i (numerando estas capas en orden ascendente de 1 a L), tan solo se requiere calcular el error imputado a cada neurona de la capa actual (δ^i), retropropagando el error de la capa anterior (δ^{i+1}), de la siguiente forma (ver Ecuación 3.24):

$$\delta^i = w^{i+1} \cdot \delta^{i+1} \cdot \frac{\partial a^i}{\partial z^i} \tag{3.24}$$

Finalmente, con dicho error se pueden calculan las correspondientes derivadas parciales de los parámetros de la capa actual (ver Ecuación 3.25 y 3.26) [26, Cap. 6][93].

$$\frac{\partial C}{\partial w^i} = \delta^i \cdot a^{i-1} \tag{3.25}$$

$$\frac{\partial C}{\partial b^i} = \delta^i \tag{3.26}$$

3.4.4. Descenso de Gradiente Estocástico (SGD)

Como se ha visto en la experimentación del algoritmo del gradiente descendente (ver Sección 3.4.2), si se aplica dicho algoritmo sobre una función no convexa, es muy probable que no se alcance un mínimo absoluto, estancándose en un mínimo local.

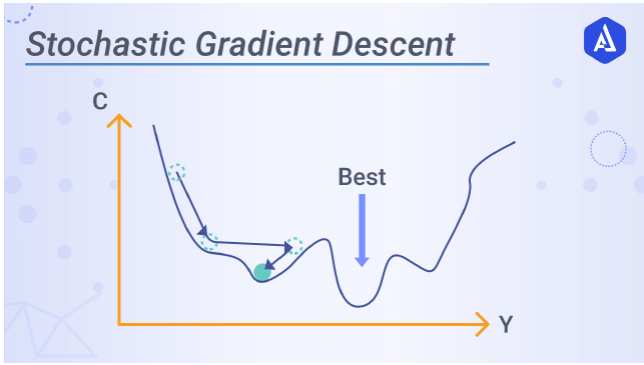


Figura 3.21: Descenso de gradiente sobre una función no convexa [3]

Tal y como se puede observar en la Figura 3.21, ante un ejemplo de función no convexa de una variable, el algoritmo del descenso del gradiente converge en un mínimo local sin alcanzar el mínimo global que se encuentra a su lado y así poder optimizar al máximo la función de coste. Para solventar estos inconvenientes, se creó el Stochastic Gradient Descent (SGD) o descenso de gradiente estocástico.

El algoritmo del gradiente descendente es un proceso iterativo el cual actualiza los parámetros una vez se hallan procesado todas las entradas. A este conjunto de entradas que se procesan para hallar el gradiente se denomina *batch* o lote en español. Este proceso se detiene cuando la función converge en un mínimo. Por el contrario, el SGD ofrece un nuevo enfoque que se basa en establecer un tamaño de *batch* de uno, es decir, calcula la pérdida obtenida para una única entrada evaluando una determinada función de coste sobre la salida de la red con respecto al valor esperado y, en base al vector gradiente, actualiza los parámetros de la red. Esto da lugar a iteraciones del algoritmo mucho más rápidas y que requieran mucha menos ocupación de la memoria. El adjetivo estocástico procede de la elección aleatoria de cada entrada que se procesa por la red en cada iteración [13].

Este comportamiento, implicará una mayor fluctuación en la actualización de los valores de los parámetros de la red, ya que al iterar el descenso del gradiente sobre cada entrada, los valores que estas entradas provocarán sobre las salidas de la red, serán muy poco uniformes entre ellas, lo que derivará en pérdidas muy distintas. Esta fluctuación en la actualización de los parámetros, hace que sea más probable alcanzar mínimos más óptimos. En contraposición, esto hace que el entrenamiento sea muy inconsistente.

Para suplir esta inconsistencia surge una versión intermedia entre el gradiente descendente y SGD, en la que en lugar de aplicar el descenso del gradiente sobre cada entrada o el conjunto completo de datos, se aplica sobre un subconjunto de este, denominado mini-batches o mini-

lotes en español. A esta técnica se la denomina Descenso de gradiente estocástico en mini-lotes [39, p. 471].

Un mayor tamaño del batch, hará que el entrenamiento sea más estable ya que al contar con más datos, el gradiente posee una mayor precisión. Sin embargo, esto también implica un menor número de batches por cada época, lo que se refleja en menos actualizaciones o ajustes de pesos de la red [39, Cap. 13].

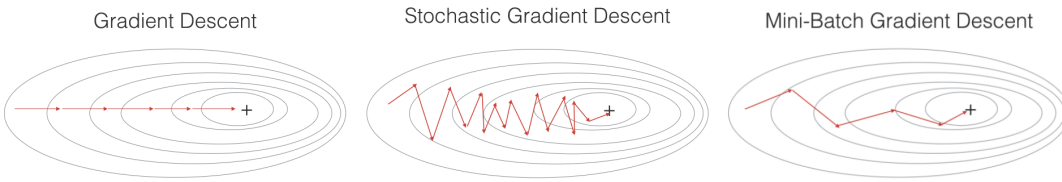


Figura 3.22: Comparativa de *Gradient Descent*, SGD y SGD con *mini-batches*

Fuente: edición propia a partir de una imagen del curso *"Improving deepneural networks: Hyperparameter tuning, regularization and optimization"* [63].

La Figura 3.22 compara el algoritmo del descenso del gradiente con su versión estocástica y la versión de mini-batches, pudiendo observar las distintas fructuraciones.

Sin embargo, el Descenso de Gradiente Estocástico de mini-batches no es la única variación de SGD que existe . Entre las distintas versiones de SGD destacan algunas de las siguientes:

- **Momentum** [87]: esta técnica nace para solventar el problema de oscilación que puede verse implicado en algunos casos en los que se aplica SGD. Lo que busca es aplicar un *momentum* o impulso a la dirección de la actualización de parámetros. Para ello, en lugar de actualizar los parámetros usando el gradiente, se utiliza una media móvil, calculada a partir de las medias de los parámetros y del gradiente. En esta interviene un parámetro que intensifica dicho *momentum*. Esta técnica puede llegar a acelerar el entrenamiento y alcanzar mejores precisiones [39, p. 474].

- **RMSProp**: esta variante de SGD fue creada por Geoff Hinton como parte del contenido de su curso "Neural Networks for Machine Learning" de Coursera. Lo que propone esta versión es utilizar un learning rate adaptativo, es decir, otorgar un learning rate a cada parámetro que es controlado por el learning rate global. El objetivo de esta modificación es acelerar el aprendizaje de la red estableciendo learning rates más altos en parámetros que necesiten mucho ajuste y tasas más bajas para aquellos parámetros que son buenos [39, p. 477].

- **Adam** (Adaptive Moment Estimation) [48]: esta versión se basa en una mezcla de SGD con momentum y RMSProp[39, p. 479].

3.4.5. Overfitting y Underfitting

El overfitting o sobreajuste en español es un problema que afecta al aprendizaje de modelos de *machine learning* y hace mención a la incapacidad de generalizar el aprendizaje, es decir, las redes neuronales ajustan sus parámetros para predecir exitosamente un conjunto de datos que ya conoce (conjunto de entrenamiento) pero son incapaces de generalizar este conocimiento a datos ajenos a este conjunto, obteniendo pérdidas o costes mayores. Este problema sería análogo a un estudiante que memoriza un modelo de examen en lugar de aprender y adquirir los conocimientos de una materia.

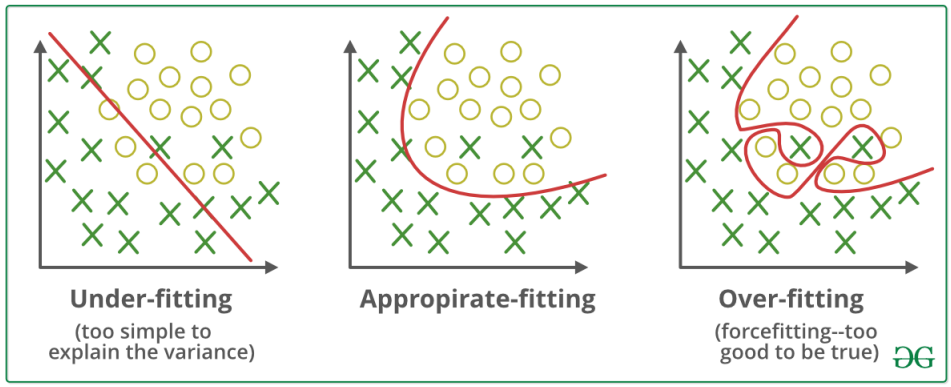


Figura 3.23: Comparativa de un modelo con *underfitting*, ajuste apropiado y *overfitting* sobre un problema de clasificación [24]

La Figura 3.23 modela de una forma gráfica el problema del ajuste de un modelo. En dicho problema, el modelo ha de aprender a clasificar en dos categorías las distintas entradas que se le suministran. La gráfica de la izquierda muestra un modelo con *underfitting* o infrajuste en español. Su frontera de decisión es lineal y pese a que quizás las predicciones no sean del todo malas, esta no se ajusta en absoluto al problema, por lo que obtendrá precisiones bajas. Por el contrario, la gráfica de la derecha muestra un modelo sobreajustado, el cual ofrece una frontera de decisión muy flexible y extremadamente adaptada al conjunto de datos que se le muestra. Por ello, el modelo predecirá exitosamente todos estos casos. Sin embargo, si se suministran datos nuevos al modelo, la precisión del mismo caerá debido a que dicha frontera de decisión no es generalizable y tan solo vale para ese conjunto de datos específico. Por último, la gráfica central refleja un modelo bien ajustado, cuya solución es capaz de modelar la clasificación del problema, aprendiendo a clasificar los diversos datos en ambas categorías pero sin ajustarse en exceso a los datos que se le muestran, de forma que, ante nuevas muestras, este modelo seguirá devolviendo buenas predicciones.

Pese a que el ejemplo expuesto se trata de un problema de clasificación logística, el overfitting puede darse en multitud de modelos, siendo este, un problema muy común en muchos de los campos del *machine learning*, especialmente en el *deep learning*, ya que este conlleva modelos con una gran cantidad de parámetros, lo que los hace tener más riesgo de overfitting.

Este problema puede surgir por la descompensación entre el tamaño del conjunto de datos de entrenamiento y el número de épocas establecidas para dicho entrenamiento. Si el conjunto de datos es limitado en cuanto al número de muestras y se establece un número de épocas grande, esto provocará un potencial sobreajuste del modelo.

Para poder detectar si el modelo está experimentando un overfitting, existe una técnica conocida con el nombre de *hold-out*. Esta consiste en dividir el conjunto de datos en dos subconjuntos, uno de entrenamiento y otro de validación. El primer conjunto contendrá el conjunto de datos que se le mostrará al modelo para que aprenda. Por el contrario, el conjunto de validación contendrá un subconjunto de datos con los que el modelo no ha sido entrenado, es decir, datos que el modelo no conoce. Gracias a este conjunto de datos se podrá evaluar la precisión real del modelo y su capacidad de generalización. Por lo general, el conjunto de entrenamiento es más grande que el de validación [64, Cap. 3].

A partir de estos dos conjuntos de datos se obtienen y comparan las pérdidas obtenidas por ambos. Si el coste del conjunto de validación, tras estar mejorando, se aplana y el del conjunto de entrenamiento continua disminuyendo, el modelo estaría comenzando a experimentar un potencial overfitting. Es muy probable que a continuación, tanto el coste del conjunto de validación como la precisión del modelo comiencen a empeorar, mientras que las pérdidas obtenidas con el conjunto de entrenamiento continúen disminuyendo, tal y como muestra la gráfica de la Figura 3.24.

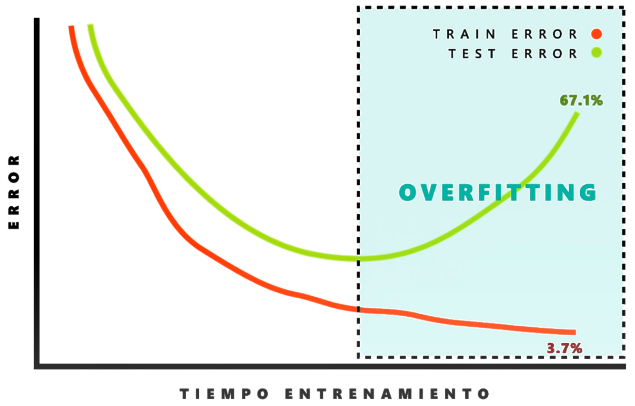


Figura 3.24: Comparativa del coste del conjunto de entrenamiento frente al del conjunto de prueba en un modelo que experimenta overfitting [93]

Un detalle que hay que tener en cuenta a la hora de aplicar esta técnica es separar los datos de una forma equivalente. Si esta división no se cuida, es posible que el conjunto de validación esté compuesto por casos atípicos, de un mismo tipo o incluyo que exista alguna dependencia con los datos del conjunto de entrenamiento. Esto haría desacreditar la detección del overfitting y los resultados que se obtuviesen de la red, derivando en conclusiones imprecisas y de cuestionable veracidad [93].

Una vez detectado el overfitting, hay que solventarlo. Existe una técnica conocida como parada temprana con dicho propósito. Esta consiste en evaluar el coste del conjunto de datos de validación y si se detecta que deja de mejorar, se detiene el entrenamiento [64, Cap. 3].

En la parte práctica del proyecto se experimentará y tratará más a fondo el problema del *overfitting* y cómo afecta a la precisión del modelo.

3.4.6. Regularización

La parada temprana es una técnica que ayuda a prevenir al modelo de un overfitting, sin embargo, esto hace que la precisión del mismo se vea limitada. Por ello, existen las técnicas de regularización, un conjunto de prácticas destinadas a reducir el overfitting e incrementar la precisión del modelo. Estas técnicas deberían usarse solo si se detecta un overfitting ya que podrían no ser necesarias y empeoren la precisión del modelo [39, p. 30].

A continuación se van a explicar algunas de las técnicas de regularización más comúnmente utilizadas [64, Cap. 3]:

- **Regularización L2**: esta regularización, también conocida como reducción de peso o *weight decay* en inglés, es el tipo de regularización más utilizada y consiste en extender la función de coste con un nuevo elemento denominado término de regularización [39, Cap. 8].

  El fin del término de regularización es penalizar los pesos grandes e incentivar a la red a aprender pesos pequeños. Esto no significa que no pueda aprender pesos grandes ya que si dichos pesos minimizan notoriamente la función de coste base (C), se tolerarán. Este balance entre pesos pequeños y minimización de la función de coste se verá influenciado por el valor del parámetro de regularización (λ). Un valor λ grande, agravará la penalización de los pesos elevados.

  $$C_{L2} = C + \frac{\lambda}{2n} \sum_{w} w^2 \tag{3.27}$$

  En la ecuación 3.27, se describe de forma matemática la regularización L2. En esta, se extiende la función de coste con la suma cuadrática de todos los pesos de la red neuronal. Esta suma cuadrática se encuentra escalada en base al cociente del parámetro de regularización (λ) y el doble del número de muestras del conjunto de datos (n).

  El motivo principal de esta regularización reside en que al reducir el valor de los pesos, se reducirá el efecto que tenga la alteración de sus entradas y, así, conseguir una mejor generalización del conocimiento, haciendo que la red se centre en patrones generales más que en peculiaridades de los datos [64, Cap. 3].

- **Regularización L1**: esta regularización es muy similar a la L2, ya que fuerza a que la red opte por pesos más pequeños.

  $$C_{L1} = C + \frac{\lambda}{n} \sum_{w} |w| \tag{3.28}$$

  La ecuación anterior (ver Ecuación 3.28) describe la extensión de la función de coste para esta regularización. La principal diferencia respecto a la regularización L2 es cambiar

la suma cuadrática de los pesos por una suma de los pesos en valor absoluto. Teniendo esto en cuenta, se puede deducir que a valores de w más grandes, la penalización que supone la regularización L2 será mayor que la impuesta por la L1. Sin embargo, para pesos pequeños, la penalización será mayor con la regularización L1. Además de esta modificación, en este caso, el parámetro de regularización (λ) lo divide el número de muestras del conjunto de datos (n), en lugar del doble de este número [64, Cap. 3].

- **Dropout**: esta técnica de regularización fue descrita en 2012 en el paper "Improving Neural Networks by Preventing Co-Adaptation of Feature Detectors" [35]. A diferencia de la regularización L1 y L2, esta consiste en alterar la red en sí misma. Esta regularización se basa en desactivar de forma aleatoria y temporal un determinado porcentaje de neuronas de las capas ocultas de la red durante el entrenamiento. Tras actualizar los parámetros de la red, se reactivan las neuronas desactivadas y se desactivan otras al azar. Lo que se consigue con esto es simular distintas redes neuronales que han podido sufrir distintos sobre-ajustes pero que al ponerlas a trabajar juntas, estos sobre-ajustes se compensan de forma que se vea reducido, otorgando una mayor robustez al modelo [64, Cap. 3][39, Cap. 12].

- **Batch normalization**: la normalización del batch fue descrita en el paper "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift" [43] y lo que hace es normalizar las activaciones de las neuronas añadiendo un paso antes de la función de activación. Esto deriva en entrenamientos más rápido y mayor tolerancia a learning rates más altos [39, Cap. 13].

- **Data augmentation**: el aumento de datos es una técnica de regularización que se refiere a aplicar variaciones aleatorias sobre los datos de entrada, de forma que parezcan entradas diferentes aún tratándose de las mismas muestras. Las rotaciones, deformaciones de perspectiva, volteos y variaciones del brillo o contraste, son algunos ejemplos de data augmentation que se aplican a imágenes en problemas de visión por ordenador. Con esta técnica se consigue simular un aumento del tamaño del dataset [39].

3.4.7. Tipos de aprendizaje automático

Existen tres principales paradigmas de aprendizaje en los que se agrupan los distintos algoritmos de *machine learning* en función de la forma en la que se procese la información de entrada y salida, y cómo se genere el conocimiento a partir de estas. Estos son [12]:

- **Aprendizaje supervisado**: este tipo de aprendizaje consiste en suministrar a la red datos de entrada que se encuentran etiquetados y la red, tendrá que predecir la etiqueta correspondiente. La finalidad del aprendizaje supervisado es conseguir un modelo que, una vez haya aprendido con datos etiquetados, pueda predecir satisfactoriamente la etiqueta que le correspondería a una nueva entrada nunca antes vista. Este tipo de aprendizaje suele verse en problemas de regresión, cuyas etiquetas son de tipo numérico, o clasificación, donde las etiquetas corresponden a distintas categorías. Este paradigma es el que se usará para el desarrollo del proyecto.

- **Aprendizaje no supervisado**: este paradigma tiene un gran potencial dentro del machine learning y es aquel que se utiliza cuando no se dispone de unos datos etiquetados. Al no contar con la salida que se quiere conseguir y solo contar con los propios datos, este tipo de aprendizaje se basa en la exploración de la estructura de los datos a fin de detectar correlaciones o patrones que puedan agruparlos. Uno de los problemas más reconocidos sobre este paradigma es el de *clustering*, problema que consiste en inquirir agrupaciones en los datos en base a similitudes y correlaciones. La ventaja principal de este tipo de aprendizaje frente al anterior es la facilidad con la que se pueden obtener los datos, puesto que para el primer tipo de aprendizaje habría que etiquetar de forma manual cada una de las múltiples muestras que se va a suministrar al modelo.
- **Aprendizaje reforzado**: a este tipo de aprendizaje pertenecen algoritmos cuyo fin es el de mejorar la precisión del modelo en base a una monitorización de cómo afectan las acciones que el modelo haya decidido tomar en función de las observaciones de su entorno. Esta forma de interaccionar con el entorno, hace que se denomine como un tipo de aprendizaje de "prueba-error". Estos algoritmos no se consideran no supervisados ya que se conoce cuál es el feedback. A su vez, no se consideran supervisados porque se fundamentan sobre el feedback generado como respuesta de cada elección tomada y no sobre un conjunto de datos etiquetados.

3.4.8. Transfer Learning

El *transfer learning* o transferencia de aprendizaje en español, consiste en utilizar como punto de partida una red neuronal que ha sido previamente entrenada con muchos datos para resolver una tarea distinta pero relacionada a la que se desea resolver. Para que el *transfer learning* sea efectivo, es necesario que el modelo base no haya sido entrenado para una tarea muy específica, de forma que se pueda extrapolar a más problemas.

Es una técnica muy popular para generar modelos de visión por computadora y de NLP puesto que permite obtener mejores rendimientos reduciendo considerablemente el tiempo de entrenamiento y, por tanto, la carga computacional asociada a dicho entrenamiento.

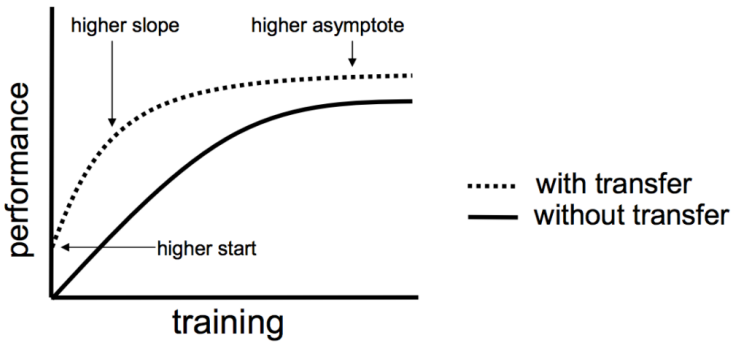


Figura 3.25: Precisión del aprendizaje de un modelo pre-entrenado utilizando *transfer learning* frente a un entrenamiento desde cero [6]

Tal y como se ilustra en la Figura 3.25, cuando se utiliza transfer learning, el modelo no parte de cero, por lo que el rendimiento inicial, antes de reajustar el modelo para la nueva tarea, es mayor. A su vez, tanto la tasa de mejora como la convergencia del modelo serán, por lo general, más altas.

Aprovechar el conocimiento ya adquirido por una red requiere el uso total o parcial de la misma. En la mayoría de casos, es necesario adaptar las ultimas capas de estos modelos para que su salida se adapte a la requerida en el nuevo problema que se desea abordar. Esto implicará generar nuevos pesos aleatorios que han de ser entrenados. Para poder entrenar estos nuevos pesos sin alterar los pesos ya entrenados, existe una técnica que consiste en congelar esas primeras capas ya entrenadas, para así, entrenar tan solo los nuevos pesos. Una vez ya ajustados los nuevos pesos, se descongelan las capas y se entrena el modelo completo a fin de refinarlo para el problema en cuestión [39, p. 207-208].

Capítulo 4

Astronomía

4.1. Telescopio Espacial Hubble

El Telescopio Espacial Hubble, también conocido por las siglas HST de su nombre en inglés (Hubble Space Telescope), fue enviado al espacio en el año 1990 a bordo del transbordador espacial Discovery, lanzado desde el centro espacial Kennedy en Florida, Estados Unidos. Este forma parte del programa de Grandes Observatorios de la NASA (National Aeronautics and Space Administration), junto a otros tres telescopios espaciales que están especializados en distintas franjas del espectro electromagnético. Se le bautizó con el nombre de Hubble por el astrónomo pionero Edwin Powell Hubble. Este telescopio espacial se encuentra orbitando en la órbita terrestre baja (en torno a 570 km de altura sobre la superficie terrestre), fuera de las distorsiones atmosféricas, contaminación lumínica y factores atmosféricos, gracias a lo que posee una visión muy nítida del universo. Su aplicación va desde la observación de los planetas vecinos del sistema solar (como el descubrimiento de lunas orbitando alrededor de Plutón o la observación de un cometa colisionando en Júpiter), hasta las galaxias y estrellas más lejanas que se han podido divisar hasta el momento [59].

Las cámaras del Hubble trabajan con distintos filtros. Éstos son delgados materiales transparentes cuya función es permitir filtrar las distintas longitudes de onda electromagnética (luz) para seleccionar la que se desee captar. Su funcionamiento se basa en absorber todas las longitudes de onda ajenas a la que se quiere observar, de forma que esta sea la única que pase el filtro e incida en el detector. De esta forma, las cámaras del Hubble poseen un conjunto de filtros que cambian de forma mecánica en función de la longitud de onda que se quiera que alcance el detector.

Las imágenes que se muestran al público son el resultado de la técnica RGB. Esta consiste en combinar la observación realizada por un filtro que actúe de rojo (aunque no sea el color rojo, se escoge el que tenga mayor longitud de onda), uno que actúe de verde (filtro de longitud de onda intermedia) y, finalmente, uno que actúa de azul (filtro con una longitud de onda baja).

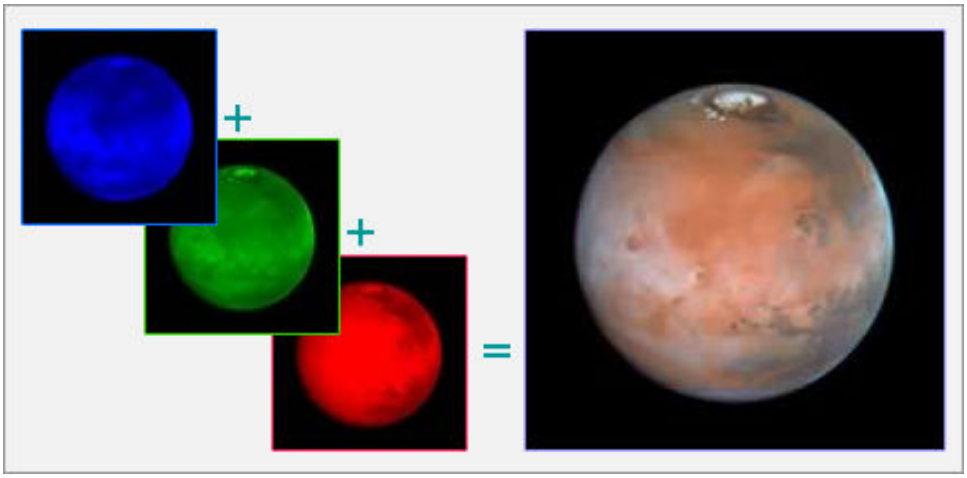


Figura 4.1: Combinación de imágenes de Marte de los filtros azul, verde y rojo [61]

La Figura 4.1 representa la combinación de las imágenes captadas por tres filtros correspondientes a los colores rojo (R), verde (G) y azul (B) para obtener una imagen en color de Marte.

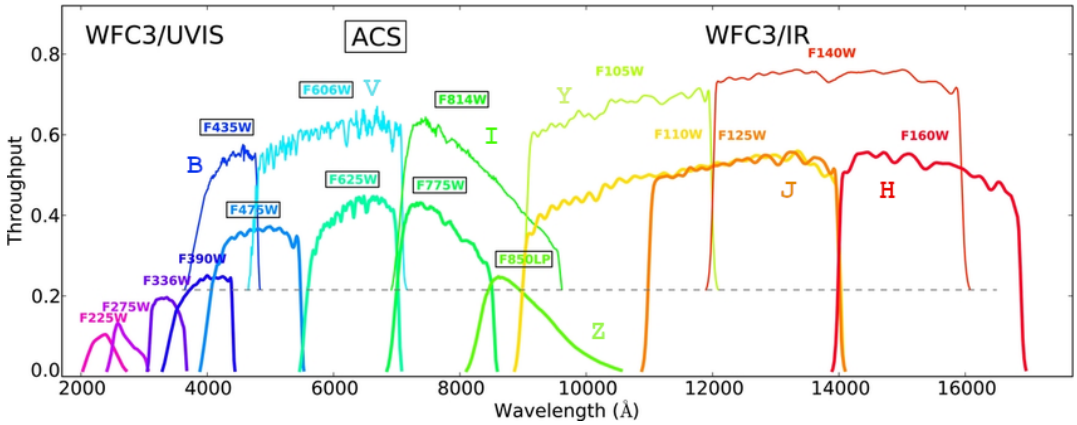


Figura 4.2: Filtros del Telescopio Espacial Hubble

Fuente: Imagen editada a partir de "The Cluster Lensing and Supernova Survey with Hubble: An Overview" de ResearchGate [54].

La Figura 4.2 representa los diversos filtros sobre los que trabaja el Telescopio Espacial Hubble. El eje horizontal representa las diversas longitudes de onda (λ). La notación que viene sobre cada filtro representa el ancho de banda (o Full Width Half Maximum, FWHM) y representa el rango de longitudes de onda que permite pasar el filtro. El eje vertical representa el *throughput* o rendimiento en español. Este valor representa el porcentaje de luz que atraviesa el filtro respecto al total de luz incidente. Lo idóneo sería un rendimiento del 100 % [61].

Las siglas ACS y WFC3 que aparecen sobre los filtros, en la parte superior de la imagen anterior (ver Figura 4.2), hacen referencia al sistema de cámaras del Telescopio Espacial Hubble que abarca las diferentes longitudes de ondas. Este cuenta dos sistemas de cámaras principales destinados a la captura de imágenes del espacio. Por un lado, se encuentra el "Advanced Camera for Surveys" (ACS), instalado en el telescopio en 2002 y cuya función es la de capturar imágenes en el rango de longitudes de onda correspondientes a la luz visibles, aunque es capaz de alcanzar la luz infrarroja y ultravioleta cercana. El ACS posee tres cámaras denominadas canales, cuyo fin es capturar diferentes tipos de imágenes. Actualmente solo dos de los canales están operativos, después de que en enero de 2007 se produjese un fallo electromagnético, dejando dañados los dos canales más utilizados, uno de los cueles fue reparado posteriormente en 2009. Por otro lado, se encuentra el "Wide Field Camera 3" (WFC3). Este sistema, instalado en 2009 a fin de complementar el sistema ACS, es capaz de capturar imágenes, no solo en el espectro visible, sino también en el rango del espectro electromagnético correspondiente a la luz ultravioleta e infrarroja, donde desempeña mayormente su función [60].

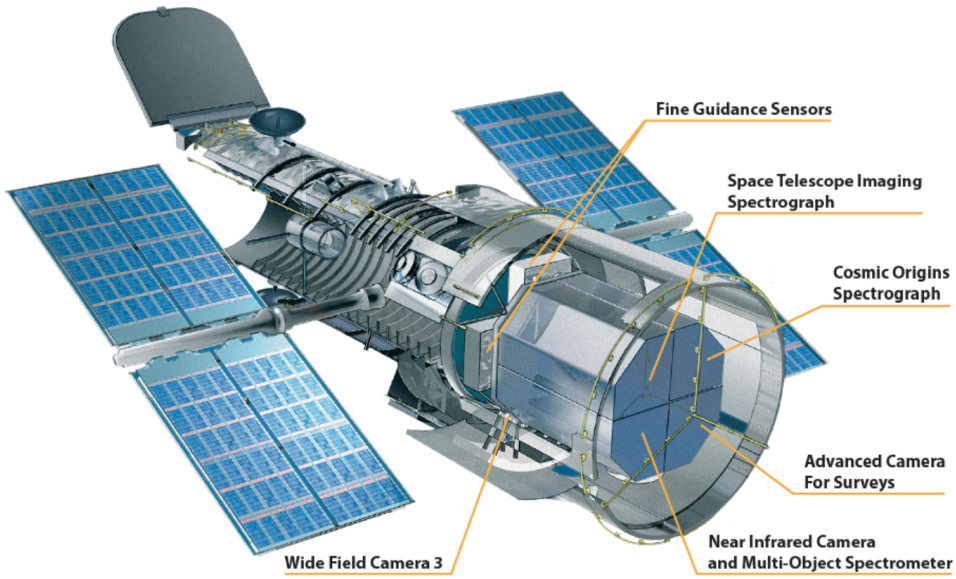


Figura 4.3: Instrumentos de captación fotográfica del Telescopio Espacial Hubble [60]

La Figura 4.3 muestra cómo están distribuidas las dos cámaras descritas, junto al resto de instrumentos del Hubble. Los dos paneles solares laterales permiten al Hubble cargar seis baterías que posibilitan su auto-abastecimiento. Tiene unas dimensiones de 13,2 metros de largo y un diámetro de 4,2 metros, dentro del cual contiene un espejo de 2,4 metros [18].

A través de cinco misiones, se ha ido mejorando el telescopio Hubble gracias a nuevo instrumental y recambio de piezas, lo que ha hecho que el telescopio siga operativo en la actualidad tras más de 30 años desde su lanzamiento. Durante toda su historia, el Hubble ha llevado a cabo en torno a un millón y medio de observaciones, estimadas en más de 150 TB de información. Dichas observaciones han derivado en más de 18 mil *papers* con descubrimientos.

Por todo ello, es innegable que el Telescopio Espacial Hubble ha supuesto un punto clave en el campo de la astrofísica, convirtiéndose en uno de los proyectos más importantes y productivos de toda la ciencia [18][59].

Figura 4.4: Nebulosa del Águila captada por el HST en luz visible e infrarroja [59]

4.2. Cartografiado CANDELS

CANDELS es el proyecto más grande del Telescopio Espacial Hubble. Sus siglas provienen de *"Cosmic Assembly Near-infrared Deep Extragalactic Legacy Survey"* [49] o Estudio del Legado Extragaláctico Profundo del Infrarrojo Cercano de la Asamblea Cósmica en español. Dicho proyecto consistió en la observación del universo temprano con el fin de analizar la evolución de los agujeros negros y las galaxias. Para ello se utilizaron las dos cámaras descritas anteriormente (ACS y WFC3) para captar las galaxias de cinco regiones del cielo: GOOD-N, GOOD-S, EGS, COSMOS y UDS. Dichas observaciones tardaron un total aproximado de 1450 horas [91].

La Figura 4.5 representa el campo ultraprofundo obtenido por el cartografiado CANDELS. Pese a que pueda parecer no tan impresionante como la fotografía de los *Pilares de la Creación* de la Figura 4.4, realmente lo que representa si lo es. Este cartografiado contiene aproximadamente 30 mil galaxias, de las cuales, las más lejanas se encuentran a $13Gyr$ (gigayear), aproximadamente, siendo equivalente $1Gyr$ a 1000 millones de años. Teniendo en cuenta que la luz emitida por las galaxias ha tardado todos esos años en llegar hasta el Hubble y que la edad del universo se estima en unos $13{,}8Gyr$, este cartografiado representa cómo ha evolucionado el universo a lo largo de gran parte de su historia, permitiendo analizar las alteraciones aspectuales y estructurales que han experimentado las galaxias desde los tiempos más tempranos del universo [19].

Figura 4.5: Observaciones del cartografiado *CANDELS* [91]

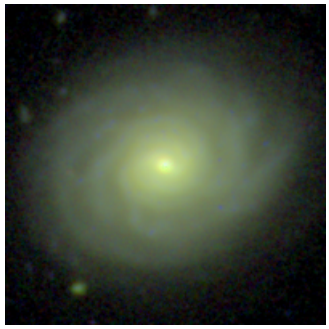
Las galaxias con colores más rojizos son las más antiguas y lejanas, puesto que según se expande el universo, la longitud de onda de la luz que emitieron también lo hace, experimentando un estiramiento mediante el efecto llamado desplazamiento al rojo. A su vez, en dicho cartografiado se pueden encontrar puntos cuya luz se capta con forma de cruz. Estos puntos brillantes son estrellas de nuestra propia galaxia que se posicionan entre las galaxias lejanas y el Telescopio Espacial Hubble. En la Figura 4.5 se puede ver un ejemplo de estrella en la esquina superior izquierda [91].

La NASA publicó un vídeo en el que hacía un recorrido tridimensional por el campo UDS (Ultra Deep Survey) del cartografiado CANDELS [62].

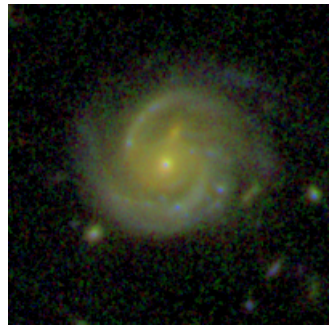
La Figura 4.6 recoge una serie de galaxias pertenecientes al cartografiado CANDELS, entre las que se encuentra una galaxia muy lejana (ver Figura 4.6c) y una galaxia en la que una estrella interfiere y no deja captar toda su luz (ver Figura 4.6f). Será este cartografiado el que se utilizará durante el desarrollo del proyecto.

4.3. Almacenamiento de datos

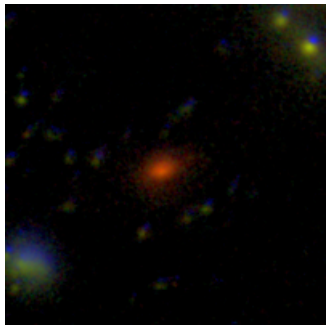
Típicamente, el almacenamiento, procesamiento e intercambio de datos astronómicos como espectros electromagnéticos o imágenes (como es este caso), se realiza mediante ficheros con un formato estándar abierto, denominado *FITS*. Dichas siglas provienen de *Flexible Image Transport System* o sistema flexible de transporte de imágenes en español. Los ficheros *.fits* están compuesto de una serie de HDU (Header Data Units), formados a su vez por una cabecera ASCII, legible para las personas, y un cuerpo de datos en binario [28].



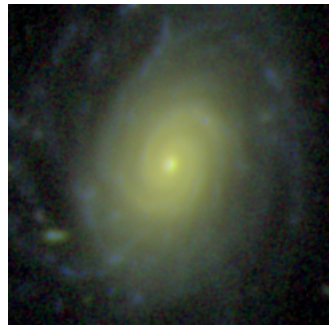
(a) Galaxia 4600 del campo GOOD-S



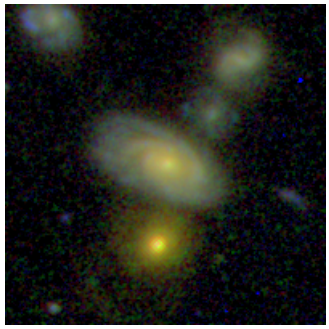
(b) Galaxia 13677 del campo EGS



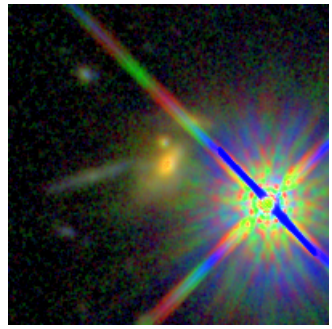
(c) Galaxia 8396 del campo GOODS-S



(d) Galaxia 23475 del campo GOODS-N



(e) Galaxia 4064 del campo GOODS-N



(f) Galaxia 16401 del campo UDS

Figura 4.6: Ejemplos de galaxias del cartografiado CANDELS

Fuente: elaboración propia.

Capítulo 5

Contexto tecnológico

5.1. Redes Neuronales

5.1.1. Lenguaje de programación

Existen multitud de lenguajes de programación utilizados en el campo del *machine learning* como R o Julia. Sin embargo, Python es sin lugar a duda el lenguaje más comúnmente utilizado y extendido en el campo del *deep learning*.

Python es un lenguaje open source interpretado de alto nivel y multiparadigma. En 1991, *Guido van Rossum*, lanzó su primera versión como un sucesor del lenguaje ABC. Se caracteriza principalmente por su sencillez y facilidad de aprendizaje. Gracias a la gran cantidad de módulos y bibliotecas con las que cuenta, el uso de Python se extiende a multitud de disciplinas y áreas de desarrollo. Todas estas características hacen de python un lenguaje muy presente en una infinidad de sistemas, desde los más pequeños hasta los más grandes como Youtube o Google [72]. Python cuenta con múltiples bibliotecas de diversas índoles. Para este proyecto se van a utilizar principalmente scipy, astropy, numpy, Pandas y Matplotlib, entre otras.

Tan solo con el lenguaje de programación, es posible crear redes neuronales y entrenarlas, tal y como se hacía con los primeros modelos. Esto supone tener que implementar los diversos algoritmos de aprendizaje y optimización, las propias redes neuronales, las funciones de activación y coste, sus derivadas, etc., lo que lo convierte en un proceso muy lento y tedioso. Sin embargo, trabajar bajo este enfoque puede resultar interesante para comprender el funcionamiento de una red neuronal y los algoritmos que intervienen en su aprendizaje.

A lo largo de los últimos años, a fin de agilizar este proceso, surgieron las bibliotecas de diferenciación automática. Estas bibliotecas añaden un nivel de abstracción que hace más manejable y sencillo el desarrollo y entrenamiento de redes neuronales. Estas bibliotecas consisten básicamente en automatizar los algoritmos de entrenamiento y optimización, reali-

zando los cálculos necesarios para entrenar cualquier modelo con el que se esté trabajando en base a su grafo computacional. Entre las bibliotecas de diferenciación automática de python destacan PyTorch y TensorFlow [93].

5.1.2. PyTorch

PyTorch es una biblioteca open source de diferenciación automática desarrollada por el *Facebook's AI Research lab* (FAIR) que está dedicada al aprendizaje automático. PyTorch destaca por su accesibilidad y facilidad de uso a través de la sencillez, tanto de su sintaxis como de su depuración. A su vez, cuenta con una API optimizada.

PyTorch trabaja con tensores, un tipo de datos que consiste en una matriz multidimensionales que alberga números, vectores o matrices, semejantes a las matrices de la biblioteca NumPy. A diferencia de NumPy, sus tensores cuentan con operaciones matemáticas optimizadas y aceleradas a través de las unidades de procesamiento gráfico o GPU. Esta aceleración, produce aumentos de velocidades de procesamiento en un rango de 50 veces superiores que utilizando la unidad central de procesamiento o CPU [86].

Todo ello hace de esta biblioteca de Python, una gran opción para introducirse en el aprendizaje automático, así como desarrollar proyectos profesionales de alto nivel. Sin embargo, PyTorch cuenta con multitud de alternativas. Su claro rival es Tensorflow, biblioteca desarrollada por Google.

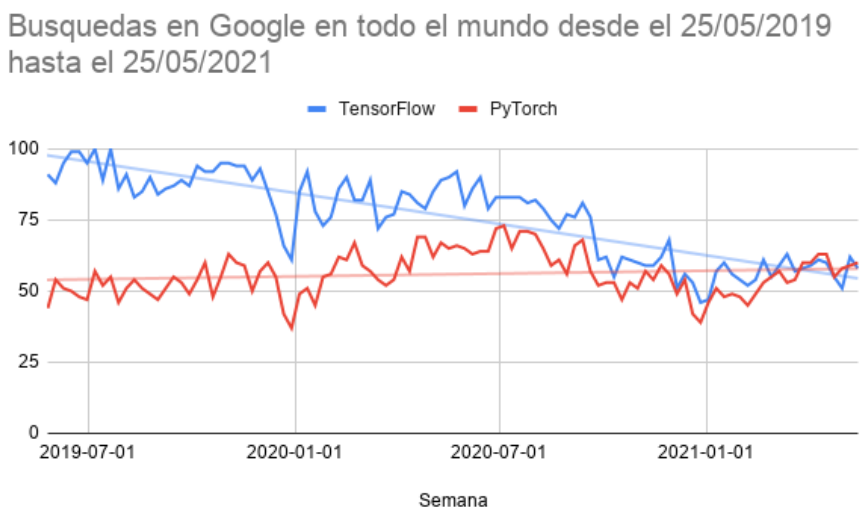


Figura 5.1: Búsquedas en Google de TensorFlow frente a PyTorch en todo el mundo desde el 25/05/2019 2019 hasta el 25/05/2021

Fuente: elaboración propia a partir los datos de Google Trends [27].

Analizando la popularidad en todo el mundo de ambas tecnologías según muestra Google Trends para los dos últimos años (ver Figura 5.1), se puede ver cómo la popularidad de TensorFlow era extremadamente alta pero ha ido aminorando con el tiempo. Sin embargo, PyTorch ha experimentado una tendencia positiva, ganando poco a poco más popularidad. El eje vertical de dicho gráfico representa la popularidad, donde un valor 100, refleja la máxima popularidad [27].

5.1.3. fastai

La razón principal y de mayor peso por la que se decidió desarrollar este proyecto con PyTorch, además de por la ventajas vistas anteriormente, es por fastai, una biblioteca de *deep learning* que aplica un nivel de abstracción sobre PyTorch. Esta biblioteca fue lanzada en 2018 por el grupo de investigación *fast.ai*, fundado sin ánimo de lucro en 2016 por Jeremy Howard (ex presidente de Kaggle) y la doctora en matemáticas Raquel Thomas [22].

Esta biblioteca ofrece componentes de distintos niveles de abstracción, permitiendo obtener resultados de una forma muy rápida y sencilla gracias a los componentes de más alto nivel o desarrollar opciones más personalizadas a través de la combinación de los componentes de menor nivel [20].

El objetivo principal de esta biblioteca es la accesibilidad, es decir, permitir la realización de proyectos de *deep learning* sin la necesidad de unos conocimientos muy amplios sobre dicha disciplina. Por ello, esta biblioteca destaca principalmente por su sencillez, productividad, rendimiento y flexibilidad.

El grupo *fast.ai* ofrece un curso masivo gratuito online (Massive Open Online Course, MOOC) llamado "Practical Deep Learning for Coders" (o aprendizaje profundo práctico para desarrolladores en español) [38] que está impartido por Jeremy Howard y Sylvain Gugger y cuyo único requisito previo es conocer Python [20]. Este curso, dividido en dos partes de siete lecciones, se basa en el contenido del libro de fastai o "fastbook" [37]. Este libro es un libro interactivo que está dividido en jupyter notebooks en el que el lector puede ejecutar las diversas celdas de código según va aprendiendo. Este libro se encuentra en *github* y es de acceso gratuito. Los autores también ofrecen una versión del libro con formato de documento. El eslogan del curso es "AI Applications Without a PhD", lo que se traduce como la posibilidad que ofrece fastai para crear aplicaciones de inteligencia artificial sin la necesidad de tener un nivel de doctorado. Además de esto, fastai cuenta con una extensa documentación y una comunidad activa muy grande.

Para el desarrollo del proyecto se va a utilizar la versión 2 del software, una versión donde se reescribe por completo su versión antecesora, haciéndola una tecnología más rápida, fácil y flexible [22].

5.2. Astroinformática

Para la visualización, manipulación y procesamiento de los datos astronómicos del Hubble, se ha requerido el uso de software especializado. A lo largo de esta sección se describirán los programas astroinformáticos que se han utilizado durante el desarrollo del proyecto.

5.2.1. SAOImage DS9

SAOImage DS9 es un software destinado a la Visualización de imágenes astronómicas, permitiendo el soporte de imágenes con formato FITS y tablas binarias, mapas de color, manipulación de regiones, etc. Las primeras versiones de este, datan del año 1999 y ha ido ganando popularidad con los años. DS9 posee un GUI (Graphical User Interface o interfaz gráfica de usuario) con una amplia variedad de funcionalidades disponibles [9].

5.2.2. SExtractor

Source-Extractor, comúnmente conocido como SExtractor, es un software astronómico encargado de generar un catálogo con los objetos detectados en una imagen astronómica. Está diseñado principalmente para reducir el volumen de datos a estudiar de galaxias sobre imágenes a gran escala, sin embargo, también puede trabajar bien sobre campos de estrellas no especialmente densos. Estos catálogos de salida son flexibles y se puede especificar que parámetros se desean obtener de cada objeto. SExtractor soporta FITS de múltiples extensiones, permitiendo trabajar con imágenes muy grandes. Todo ello optimizado para trabajar a velocidades altas [81].

5.2.3. Topcat

Topcat es un software astronómico dedicado a la visualización y procesamiento de datos tabulares de una forma gráfica e interactiva. Pese a que está especialmente diseñado para datos astronómicos, su uso se puede extender a datos normales, siendo especialmente útil y rápido para hacer gráficas de tablas de grandes dimensiones. Entre los formatos de ficheros astronómicos soportados está FITS, VOTable (Virtual Observatory Table) y CDF, aunque permite incluir más. Al seguir los estándares del Virtual Observatory (VO) o Observatorio Virtual en español, tiene una gran compatibilidad con cualquier servicio, herramientas y datos que pertenezcan o sigan los estándares del Observatorio virtual [88].

5.2.4. Aladin

Aladin es un software semejante a SAOImage DS9, ya que está diseñado para la visualización de imágenes astronómicas. Este software se define como un atlas de cielo interactivo.

Este software también sigue los estándares del Virtual Observatory, lo que le otorga una gran compatibilidad con otras herramientas, pudiendo conectarse con *Topcat*.

Cuenta una versión de navegador llamada *Aladin Lite* la cual permite una visualización simple de las regiones del cielo. A su vez, cuenta con una versión de escritorio mucho más completa, llamada *Aladin Desktop*, que fue creada por el *Centre de Données astronomiques de Strasbourg* en 1999 utilizando el lenguaje de programación Java. Esta versión permite al usuario acceder e interactuar tanto con datos que este posea en local como datos recogidos en gran parte de los servidores astronómicos de todo el mundo. [11].

5.3. Otras tecnologías

5.3.1. Sistema operativo

En cuanto al sistema operativo, hay diversas opciones a elegir. Para el desarrollo del proyecto se ha escogido Linux como sistema operativo.

Linux es una sistema operativo open source creado por Linus Torvalds en 1991 como una alternativa gratuita de sistema operativo. En la actualidad, linux se ha convertido en el sistema operativo más utilizado en servidores y supercomputadoras, que cuenta con una gran comunidad [30].

Por su gratuidad, rendimiento, seguridad y filosofía open source, se ha convertido en mi sistema operativo de trabajo principal y, por consiguiente, el sistema más indicado para desarrollar el proyecto. Para ser más exactos, se va a utilizar la distribución de Ubuntu en su versión número 20.

5.3.2. Jupyter Notebook

Jupyter Notebook está desarrollado por el Jupyter Project, una organización sin ánimo de lucro creada en 2014 que desarrolla software de código abierto, estándares y servicios.

Este software consiste en una aplicación web cliente-servidor basada en un documento interactivo, el cual permite incluir código, texto formateado a través de Markdown, ecuaciones en notación de LaTex, ficheros multimedia como imágenes o vídeos, entre otros. Estos cuadernos permiten multitud de lenguajes entre los que se encuentran Python y R. Estos pueden ser una forma muy buena de compartir código entre varias personas debido a su gran facilidad de compartición. Las ejecuciones de los distintos fragmentos de código pueden generar salidas interactivas [71]. Los cuadernos Jupyter pueden servir como un registro del cómputo completo de una sesión, que contienen el desarrollo, la documentación, ejecución de código y los resultados de las ejecuciones. Estos cuadernos se guardan en ficheros con extensión `.ipdb` y cuyo contenido tiene un formato JSON. El entorno de Jupyter, por lo general, se despliega en la máquina local, pudiendo acceder al servicio en un navegador a través de la dirección y puerto local en el que se haya desplegado el servidor (normalmente está en

`http://127.0.0.1:8888`). Sin embargo, es posible desplegar el servidor en una máquina remota y acceder de tal forma a dicho entorno. Un jupyter notebook abierto tiene creada una sesión interactiva en un kernel, donde se ejecutará el código que desee el usuario. Esta sesión durará hasta que el usuario detenga dicho kernel [89].

Jupyter ha supuesto un gran impacto en muchas disciplinas, tanto a nivel académico como a nivel industrial, siendo el software más ampliamente utilizado en el campo del deep learning. Por todo ello, fue galardonado con el ACM Software System Award, el mayor reconocimiento para un software [39, p. 15].

5.3.3. Google Colaboratory

Google Colaboratory, también conocido simplemente como Google Colab, es una plataforma online gratuita, producto de Google Research, que permite escribir jupyter notebooks y ejecutarlos sobre recursos computacionales alojados. Los notebooks se conectan a máquinas virtuales cuyo tiempo de vida es de un máximo de 12 horas. También se desconectarán si, durante un tiempo prolongado, el usuario permanece inactivo.

Los notebooks se almacenarán en el Google Drive del usuario, permitiendo también cargarlos desde GitHub. Estos pueden ser compartidos con otros usuario, al igual que cualquier servicio de ofimática de Google.

A su vez, en el apartado de entorno de ejecución, te permite cambiar el tipo de aceleración por hardware, ofreciendo la ejecución sólo en CPU (sin aceleración), acelerada mediante GPU o mediante TPU.

Puede darse el caso de que el usuario intente conectarse a un entorno con GPU o TPU pero la plataforma indique la imposibilidad temporal por motivos de limitación de uso. Google Colab prioriza el uso de este hardware para ciertos usuarios frente a otros usuario cuyas ejecuciones son más costosas y de larga duración o el uso sobre dichos recursos ha sido mayor recientemente. Así mismo, ofrecen una versión pro de pago para estos usuarios que requieren un mayor poder de computo de una forma continuada.

Google colab cuenta con varios tipos de GPU que varían con el tiempo. Estas son: K80, T4 y P100 de la marca NVIDIA. Sin embargo, no es posible elegir a que tipo de gráfica conectarte.

Google Colab especifica la prohibición con la denegación del servicio al uso indebido de la plataforma para minería de criptomonedas [75].

Para desarrollar el proyecto se ha contado con un ordenador cuya tarjeta gráfica era de tan solo 2GB, la cual se quedaba corta para las necesidades del proyecto. Por esta razón se ha utilizado Google Colab para el desarrollo del proyecto.

Capítulo 6

Clasificación morfológica de galaxias

6.1. Introducción al problema

Una vez asimilados los conocimientos teóricos más importantes del deep learning y haber entendido el contexto astronómico en el que se centrará el proyecto, se va a abordar el primer problema, a fin de interiorizar todos estos conocimiento, en base a ponerlos en práctica, y ganar destreza en PyTorch y fastai, las tecnologías escogidas para llevar a cabo este proyecto.

Dicho problema consistirá en realizar una clasificación morfológica de galaxias pertenecientes a los cinco campos CANDELS. Una clasificación morfológica consiste en separar las distintas galaxias en función de su apariencia visual. Existen varios sistemas de clasificación, destacando entre todos ellos, el propuesto por *Edwin Hubble*. Este sistema, comúnmente conocido como "Hubble Fork", divide las galaxias según su aspecto visual en tres diferentes morfologías: galaxias elípticas (E), galaxias espirales (SB) y galaxias lenticulares (S0) [19]. Se puede observar este esquema de clasificación en la Figura 6.1.

Como breve mención al estado del arte, en 2020 se publicó el paper 'Morpheus: A Deep Learning Framework For Pixel-Level Analysis of Astronomical Image Data" [32]. En este, sus autores, Ryan Hausen y Brant Robertson, presentaron Morpheus, un modelo de red neuronal desarrollado en Tensorflow, capaz de clasificar morfológicamente galaxias de los cinco campos CANDELS, permitiendo a su vez, generar mapas de segmentación [31].

El desarrollo de este problema va a estar inspirado en el paper de Huertas-Company et al. (2015) titulado "A catalog of visual-like morphologies in the 5 CANDELS fields using deep-learning" [41]. En este artículo, se busca categorizar morfológicamente las galaxias de los cinco campos CANDELS mediante técnicas de deep learning. Haciendo uso de una red neuronal convolucional, se intenta reproducir la clasificación morfológica visual descrita en el paper "Candels visual classifications: Scheme, data release, and first results" [47], en la que

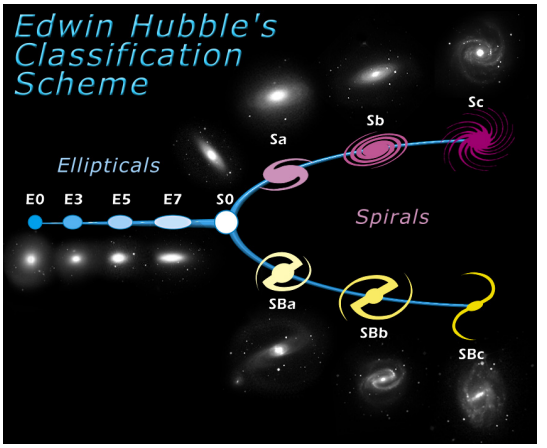


Figura 6.1: Esquema de clasificación propuesto por Edwin Hubble [95]

65 expertos (entre 3 y 5 por cada galaxia) analizaron visualmente y de forma independiente todas las galaxias correspondientes al campo GOOD-South, a fin de determinar una serie de *flags* referentes a si la galaxia en cuestión contenía un disco o un esferoide, presentaba alguna irregularidad, se trataba de una fuente puntual o era inclasificable. A partir de estos *flags*, se obtenía las fracciones de expertos que determinaron una característica específica en una cierta galaxia. Estas fracciones se denominarán como $f_{spheroid}$, f_{disk}, f_{irr}, f_{PS} y f_{Unc}. Los *flags*, salvo el de "inclasificable", no son excluyentes, lo que significa que la suma de las fracciones no necesariamente resultará 1 [41].

Al final de la sección "3.1 Convolutional Neural Network (ConvNet) configuration" del paper de Huertas-Company et al. (2015), tras hacer una breve introducción a las Redes Neuronales Convoluciones, se comenta cómo a principio de ese mismo año (2015), se aplicaron por primera vez este tipo de redes para la clasificación morfológica de galaxias en el *Galaxy Zoo Challenge* promovido por Kaggle, una comunidad de profesionales del machine learning y científicos de datos que pone a disposición de estos, multitud de conjuntos de datos y código, incluso permitiéndoles el acceso gratuito a GPUs y a competiciones y desafíos en linea [45].

Esta competición arrojó resultados muy prometedores sobre el potencial de la Redes Neuronales Convolucionales sobre el problema de clasificación morfológica de galaxias, gracias a que el ganador alcanzó un RMS (Root Mean Square o raiz de la media cuadrática en español) muy bueno. Pese a que el problema de Huertas-Company et al. (2015) trabajase sobre galaxias del cartografiado CANDELS, y no sobre SDSS (Sloan Digital Sky Survey) como hacía el ganador del reto, la similitud de ambos problemas y los prometedores resultados, hicieron que el paper de Huertas-Company et al. (2015) replicase la configuración de la CNN utilizada por el ganador del reto, que se encuentra descrita en el paper "Rotation-invariant convolutional neural networks for galaxy morphology prediction" [15].

La Figura 6.2 representa un modelo arquitectónico de la red neuronal convolucional que se utiliza en el paper de Huertas-Company et al. (2015). Esta está compuesta de 4 capas convolucionales seguidas de 3 capas *fully-connected*, siendo la última de ellas una capa de salida para la regresión de las cinco fracciones ($f_{spheroid}$, f_{disk}, f_{irr}, f_{PS} y f_{Unc}).

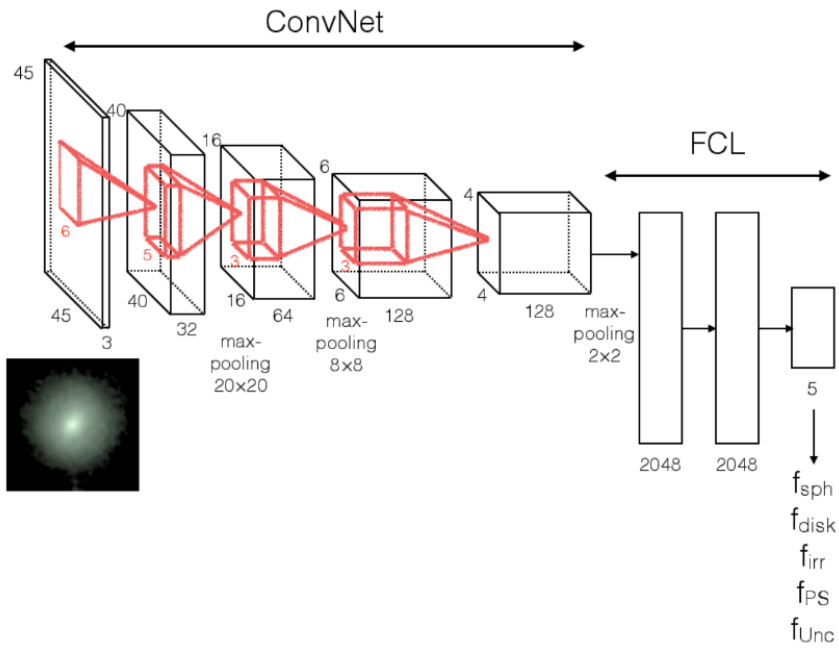


Figura 6.2: Arquitectura de la CNN utilizada por Huertas-Company et al. (2015) [41]

En la sección 6 del paper de Huertas-Company et al. (2015), se explica cómo la interpretación de las fracciones sobre los que se ha realizado la regresión depende en gran medida de los propósitos científicos y de las propiedades de las galaxias que se quieran resaltar para determinar cual es la morfología de cada galaxia. Sin embargo, proponen una posible clasificación en 5 morfologías, en base a unos umbrales establecidos para las diferentes fracciones (ver Fragmento de código 6.1). Estas 5 morfologías son: esferoide (`SPH`), disco (`DISK`), disco-esferoide (`DISKSPH`), disco-irregular (`DISKIRR`), irregular (`IRR`) o ninguna (`NONE`) [41].

```
if f_sph > 0.667 and f_disk < 0.667 and f_irr < 0.1 then
    morphology := "SPH"
else if f_sph < 0.667 and f_disk > 0.667 and f_irr < 0.1 then
    morphology := "DISK"
else if f_sph > 0.667 and f_disk > 0.667 and f_irr < 0.1 then
    morphology := "DISKSPH"
else if f_sph < 0.667 and f_disk > 0.667 and f_irr > 0.1 then
    morphology := "DISKIRR"
else if f_sph < 0.667 and f_disk < 0.667 and f_irr > 0.1 then
    morphology := "IRR"
else
    morphology := "NONE"
```

Listing 6.1: Pseudocódigo de la clasificación propuesta por Huertas-Company et al. (2015) [41]

Teniendo esto en cuenta, este problema podría ser resuelto por dos lineas o enfoques de trabajo diferentes. La primera de ellas consistiría en desarrollar un modelo de regresión capaz de predecir las fracciones de expertos que determinarían cada característica de cada galaxia ($f_{spheroid}$, f_{disk}, f_{irr}, f_{PS} y f_{Unc}). Por otro lado, el segundo enfoque se centraría en buscar un modelo de clasificación cuya función fuese la de predecir qué categoría morfológica de las propuestas en el paper, le corresponde a cada galaxia según su aspecto visual 6.1).

6.2. El conjunto de datos

Para el desarrollo de este problema se ha contado con un conjunto de datos bastante completo. Este cuenta con 3460 muestras de galaxias captadas por el Telescopio Espacial Hubble en los cinco campos del cartografiado *CANDELS*. De las 3460 galaxias, 950 pertenecen al campo *COSMOS*, 684 al campo *EGS*, 795 al campo *GOODS-North*, 445 al campo *GOODS-South* y las 586 restantes pertenecen al campo *UDS*.

Para escoger las galaxias con las que se iba a trabajar, se han optado por aquellas galaxias cuya morfología fuese de clase espiral, según la clasificación "Hubble Fork" antes mencionada. A su vez, de todas las galaxias espirales, se han escogido sólo las más masivas, cuya masa estelar fuese mayor a 10 mil millones de masas solares. Estas galaxias sobre las que se va a trabajar, se encuentra a una distancia comprendida entre los 1,9 Gyr y 7,8 Gyr, siendo $1Gyr$ equivalente a 1000 millones de años, tal y como se explicó en el Capítulo 4.

Cada muestra de galaxia está compuesta por 4 ficheros `.fits` con imágenes de 200x200 píxeles correspondientes a 4 filtros del Telescopio Espacial Hubble. Estos filtros son: "V" (F606W), "I" (F814W), "J" (F125W) y "H" (F160W), en orden de más azul (menor longitud de onda) a más rojo (mayor longitud de onda) en el espectro electromagnético, tal y como se describió en la Figura 4.2 del capítulo correspondiente al marco teórico astronómico.

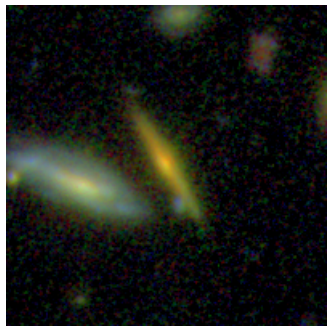
Para obtener estos ficheros *fits*, se han procesado con SExtractor cada fichero *fits* correspondientes a los filtros "V", "I", "J" y "H" de cada campo CANDELS. El resultado que se obtiene con SExtractor, es un fichero `.cat` con toda la información relevante de cada galaxia, a partir del cual, se puede conocer la posición central de cada galaxias, de forma que se aplique un recorte de tamaño 200x200 píxeles sobre los *fits* de cada filtro y cada campo, haciendo coincidir el centro del recorte con el centro de cada galaxia que se quiere clasificar.

Es importante destacar que las imágenes del *dataset* no contienen los elementos de forma aislada, es decir, se pueden encontrar más de una galaxia en la misma imagen. Sin embargo, tan sólo la galaxia central de la imagen es aquella que se quiere analizar. Por esta razón, por cada filtro existe un nuevo fichero `.fits` relativo a su correspondiente máscara. Esta máscara permitirá distinguir qué píxeles de la imagen aportan información sobre la galaxia que se está analizando, haciendo posible descartar todo el ruido externo a esta. Los píxeles que ofrecen información pertenecen tanto a la propia galaxia como a partes de sus galaxias vecinas. Para obtener estas máscaras, SExtractor genera un mapa de segmentación por cada filtro y cada campo, sobre los que se aplica un recorte de 200x200 centrado en cada galaxia que se quiera clasificar.

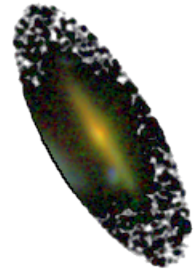
6.3. Preprocesamiento de los datos

Como parte del preprocesamiento del dataset para este problema, se han convertido todas las muestras a imágenes RGB almacenadas en ficheros con extensión `.png`. Para ello se ha utilizado un programa escrito en python, nombrado como "creating_rgb_images_lupton.py". Primeramente, lee y procesa el fichero `.cat` que contiene toda la información relevante de cada galaxia. Con estos datos, obtiene la categoría morfológica de cada una de ellas, según se describió en el Fragmento de código 6.1.

Seguidamente, compone los tres canales RGB (Red, Green y Blue) a partir del contenido de los ficheros *fits* de cada filtro, haciendo coincidir el filtro "H" con el canal rojo (R), el filtro "J" con el verde (G) y el filtro "I" con el azul (B). A cada canal RGB, se le aplica una máscara. Para ello, se lee el contenido de los ficheros *fits* correspondientes a las máscaras de cada filtro. Ya cargadas, se sobrescribe cada canal RGB, multiplicándolo por su máscara correspondiente. En este caso, indiferentemente de si fuera el canal rojo (R), verde (G) o azul (B), se ha aplicado siempre la máscara del filtro "H", es decir, del canal rojo. La razón de esto se debe a que hay casos en los que no existe el fichero de máscara para los otros filtros. Gracias a esto se consigue una nueva imagen enmascarada donde, en la medida de lo posible, se ha descartado toda información externa a la galaxia que pueda despistar a la red neuronal y entorpecer su aprendizaje, de forma que focalice su atención en los píxeles correspondientes a la galaxia en cuestión.



(a) `DISK_649_goodsn.png` sin máscara



(b) `DISK_649_goodsn.png` con máscara

Figura 6.3: Enmascaramiento de la imagen de una galaxia

Fuente: elaboración propia.

La Figura 6.3 muestra el resultado de enmascarar una galaxia, comparando una imagen original de una galaxia transformada a RGB frente a su correspondiente imagen enmascarada. En esta nueva imagen se puede ver cómo, en la medida de lo posible, solo se mantienen los píxeles que aportan información sobre esa galaxia específica.

Finalmente, al crear las imágenes RGB se reescalan los diversos canales. Esta escala viene especificada en una lista de tres elementos, nombrada como *scale*, cuyos valores se han establecidos a 1,0 para el canal azul (B), 0,7 para el verde (G) y 0,7 para el rojo (R). Esto se hace para darle más peso al canal azul para que se vean mejor los brazos de la galaxia.

El resultado final de todo este preprocesamiento da lugar a 3460 imágenes RGB del mismo tamaño que los ficheros `.fits` originales (200x200 píxeles), que están almacenadas como ficheros `.png`. En el nombre de estos ficheros `.png` viene codificada la información principal de cada muestra, de forma que se puedan auto identificar. Dichos nombres siguen la siguiente estructura: <`morfología`>_<`identificador_de_galaxia`>_<`campo_candels`>`.png`. Por ello, de la figura anterior (ver Figura 6.3), se puede saber tan solo con su nombre, que se trata de la galaxia 649 del campo *GOODS-N* y su clase morfológica según la interpretación del paper de Huertas-Company et al. (2015) es *DISK*.

6.4. Descripción de los experimentos

Para ambos enfoques se van a realizar una serie de experimentos en los que, a partir de un primer sistema base de partida, se irán haciendo diferentes pruebas analizando distintas modificaciones sobre la construcción del modelo y su entrenamiento, a fin de ir mejorando la precisión de este. Durante cada experimento, solo se modificará lo que se quiera estudiar, manteniendo estáticas todas las posible variación que afecten al modelo. De esta forma todos los casos de cada experimento trabajarán sobre el mismo escenario. Además, para realizar los experimentos y poder obtener resultados replicables, se ha buscado que todo factor externo a la experimentación sea lo más invariante posible. Para ello, se aplica una semilla aleatoria que, como se explicará más adelante, proveerá la misma secuencia aleatoria para los distintos casos de prueba, de forma que estos trabajen sobre los mismos conjuntos de entrenamiento y validación.

Al final de cada experimento, una vez obtenidas las conclusiones y evaluadas las mejoras, se explicará qué cambios se mantendrán para el sistema del siguiente experimento, con el objetivo de finalizar con la mayor precisión posible.

6.5. Enfoque de regresión

6.5.1. Implementación de la solución

Puesto que el desarrollo del proyecto se va a realizar utilizando Google Colab, lo primero que será necesario hacer es instalar las dependencias correspondientes. Pese a que Google Colab cuenta con *fastai*, será necesario actualizar la biblioteca de su versión 1 (1.0.61) a su versión 2 (2.3.1) e instalar la biblioteca *fastbook*. Para ello, se ha ejecuta la siguiente celda de código (ver Fragmento de código 6.2).

```
!pip install -Uqq fastbook
import fastbook
fastbook.setup_book()
```

Listing 6.2: Instalación de fastai y fastbook en Google Colab [37]

El signo de exclamación del código anterior (ver Fragmento de código 6.2) sirve para ejecutar funciones del sistema. Es muy probable que la ejecución de dicha celda requiera al usuario introducir un código de autorización que se obtiene a partir de un enlace que se muestra en el propio resultado de la celda.

Una vez instaladas las dependencias, se importan los módulos necesarios de las mismas. Tal y como se puede ver en el siguiente código (ver Fragmento de código 6.3), para este proyecto se han necesitados todos los módulos de fastbook y los relativos a la visión por ordenador de fastai. Las versiones que se han utilizado de estas bibliotecas son: `fastai 2.3.1` y `fastbook 0.0.16`.

```
from fastai.vision.all import *
from fastbook import *
```

Listing 6.3: Importación de módulos de fastbook y fastai [37]

Seguidamente, se han declarado los *paths* o rutas con las que se van a trabajar a lo largo del notebook. En este caso se han declarado tres *paths*. El primero de ellos indica la ubicación del fichero `.cat` que posee toda la información referente a cada galaxia (`cat_file_path`). El segundo especifica la ruta del directorio que contiene las imágenes del dataset (`png_files_path`). Por último, se especifica dónde se quiere localizar el fichero `.csv` con los resultados de cada época del entrenamiento (`csv_results_path`). De esta forma, se consigue un código más legible y fácil de reubicar, ya que tan solo hay un punto en todo el código en el que habría que modificar las rutas.

A continuación, se lee y procesa el fichero `.cat` que contiene toda la información de las galaxias. A partir de estos datos se crea un diccionario bautizado como `galaxies_fs`, cuyas claves son los nombres de los ficheros de las imágenes y sus valores son arrays compuestos por los 5 números de la regresión en punto flotante, tal y como se describe en el Fragmento de código 6.4. Un diccionario es una estructura de datos de *Python* compuesta por un conjunto de claves-valor en el que cada clave tiene tan solo un único valor. A diferencia de otras estructuras de datos que se indexan por un número asociado a su posición, esta se indexa por las claves [72].

```
{ morph_id_field.png : [f_sph, f_disk, f_irr, f_ps, f_unc] }
```

Listing 6.4: Estructura del diccionario que posee la información necesaria para la regresión

Posteriormente, se crea el objeto `DataBlock` y se definen todas las funciones necesarias para su instanciación. Un `DataBlock` busca encapsular, a modo de bloque o contenedor, la información necesaria para construir objetos `Datasets` o `DataLoaders` de una forma rápida. La información que recoge el objeto `DataBlock` va referida a la forma de obtener los datos de entrada, su naturaleza, cómo están etiquetados, cómo van a ser divididos entre el conjunto de entrenamiento y el de validación o qué transformaciones se van a aplicar, bien a cada elemento o cada batch, entre otras configuraciones [20].

```
blocks = DataBlock(
    blocks=(ImageBlock, RegressionBlock),
    get_items=get_image_files,
    get_y=get_y,
    splitter=RandomSplitter(valid_pct=0.2, seed=42),
    batch_tfms=tfms
)
```

Listing 6.5: DataBlock creado para los experimentos de regresión

Tal y como se puede observar en el código anterior (ver Fragmento de código 6.5), lo primero que se especifica en el *DataBlock* es el argumento `blocks`, una tupla que contiene el tipo de lo que llaman variables independientes y dependientes. Las variables independientes, también conocidas con la letra "x", son aquellas que se van a usar para llevar a cabo las predicciones, sin embargo, las dependientes, denominadas también por la letra "y", son el objetivo de dichas predicciones. Para definir estos tipos de variables se utilizan objetos de tipo `TransformBlock`. A continuación, se van a enumerar y explicar los tres tipos básicos de `TransformBlock`, seguidos de otros cinco tipos con los que cuenta la aplicación de visión por ordenador de fastai [20]:

- CategoryBlock: bloque que representa las categorías de una clasificación de una única etiqueta por *input* de datos.
- MultiCategoryBlock: bloque para las categorías de las clasificaciones multicategoría, es decir, más de una etiquetas por *input*.
- RegressionBlock: bloque diseñado para los números en punto flotante de una regresión.
- ImageBlock: tipo de bloque creado para representar imágenes.
- MaskBlock: bloque ideado para máscaras de segmentación. Un problema de segmentación es aquel cuyo fin es predecir la clase correspondiente de cada píxel de una imagen de entrada.
- PointBlock: tipo de `TransformBlock` para problemas de predicción de puntos situados en una imagen. Transforman los puntos en valores entre -1 y 1 que representan coordenadas sobre la imagen.
- BBoxBlock: bloque que representa cuadros delimitadores en una imagen. Un ejemplo de aplicación es una tarea de detección de objetos a partir de una imagen.
- BBoxLblBlock: lo único que lo diferencia de un `BBoxBlock` es que en este caso los cuadros están etiquetados.

Teniendo esto en cuenta y puesto que las variables independientes representan imágenes, se definirán como `ImageBlock`. Por otro lado, las variables dependientes al tratarse de números en punto flotante sobre los que se desea hacer una regresión, se definirán como `RegressionBlock`.

Seguido a esto, se define la forma en la que se obtienen tanto las variables independientes como las dependientes. Para ello es necesario especificar unos *getters* que trabajarán sobre una lista de elementos. Para especificar la forma de obtención de esta lista, existe el argumento `get_items`. Para este problema se ha utilizado la función de fastai "get_image_files", encargada de retornar una lista con el *path* de cada una de las imágenes existentes en directorio que se le especifique.

Una vez obtenida la lista de elementos con los que se va a trabajar, se pueden definir los *getters* que determinarán cómo obtener ambas variables de la lista. Para obtener las variables independientes, existe el argumento `get_x`. En este caso, como la lista contiene la ruta de cada imagen, no es necesario especificar una función para `get_x`. En cuanto a las variables dependientes, el *DataBlock* posee un argumento `get_y` para ello. En este caso, se ha definido una función `get_y(source)`, encargada de procesar la ruta de cada imagen y consultar los valores de la regresión en el diccionario `galaxies_fs`, creado anteriormente [20][21].

Lo siguiente que se especifica es la forma en la que se va a dividir el conjunto total de datos en los datasets de entrenamiento y validación. Para ello se ha indicado en el argumento "splitter" el uso de la función `RandomSplitter`, encargada de separar aleatoriamente todos los elementos en los dos datasets, en base al porcentaje de datos que se le diga que se desea destinar al conjunto de validación (un 20 % en este caso). A su vez, se ha especificado un segundo argumento denominado *seed*. Este permite establecer a la generación aleatoria, una semilla o "seed" en inglés, de forma que la secuencia aleatoria que se genere durante la ejecución del notebook sea siempre la misma. Gracias a esto, se consigue esa invariabilidad que se comentaba anteriormente. Algo que se detectó durante el desarrollo de la solución es la necesidad de importar la biblioteca `fastbook` para que los resultados puedan ser replicables, es decir, que se obtengan los mismos valores aunque se reinicie el entorno de ejecución.

Finalmente se introduce en el argumento `batch_tfms`, las transformaciones que se desean aplicar a cada batch. Alternativamente a `batch_tfms`, existe `item_tfms` para aplicar transformaciones sobre cada entrada. La diferencia entre ambos argumentos es que `item_tfms` realiza las transformaciones necesarias previas a crear los *batches*. Al aplicar estas transformaciones antes de cargar las imágenes, estas se realizarán en la CPU. Por el contrario, `batch_tfms` realiza las transformaciones sobre cada batch completo tras ser cargado. Por tanto, estas transformaciones se realizan en la GPU, aprovechando la aceleración que supone su poder de cómputo. Lógicamente, será más conveniente aplicar las transformaciones sobre la GPU. Si se ejecuta `DataBlock.item_tfms`, se puede observar que pese a que no se ha incluido ninguna transformación en `item_tfms`, devuelve una transformación llamada `ToTensor`. Esta transformación es necesaria para transformar los datos de entrada (imágenes y listas de valores de regresión en este caso) a tensores. Igualmente, si se ejecuta `DataBlock.batch_tfms`, devolverá una transformación llamada `IntToFloatTensor`, precediendo a las transformaciones que se añadan en dicho argumento del DataBlock. La transformación `IntToFloatTensor` se encarga de transformar los los valores tipo `int` de los tensores a tipo `float` puesto que las transformaciones de datos en fastai requieren tensores punto flotantes en la GPU [20].

A partir del DataBlock instanciado, se genera un objeto DataLoaders, encargado de preparar los datos para entrenar la red. Este objeto no es más que un contenedor de dos objetos DataLoader, uno para el conjunto de entrenamiento y otro para el conjunto de validación. Estos objetos de tipo DataLoader, se encargan de iterar sobre un `Dataset` concreto,

suministrando *mini-batches* en forma de tuplas, formadas por un conjunto de variables independientes y dependientes [39, p. 222]. Para instanciarlo hay que especificar un campo `source` o fuente. En este caso, se ha indicado la ruta en la que se encuentran las imágenes y, gracias a este campo, el argumento `get_items` del *DataBlock* que se explicó anteriormente, podrá obtener la lista de imágenes de dicho directorio.

Por último, se crea el Learner, un objeto encargado de agrupar el modelo, los `DataLoaders` y la función de pérdida para llevar a cabo el entrenamiento. En este caso se va a usar `cnn_learner`, un método definido en el módulo de visión de fastai y que está enfocado al transfer learning.

```
learn = cnn_learner(
    dls, resnetXX,
    metrics = rmse,
    loss_func = loss_func,
    cbs = [
        ShowGraphCallback(),
        CSVLogger(fname=csv_results_path, append=true)
    ]
)
```

Listing 6.6: Creación del Learner con la función `cnn_learner`

En el fragmento de código 6.6 se ve como para crear el objeto Learner, se le pasa a la función `cnn_learner` el objeto DataLoaders y el modelo preentrenado que se quiere utilizar. Entre otras muchas configuraciones, este método permite seleccionar el optimizador que se desee aplicar al entrenamiento. Si no se especifica ninguno, se establecerá por defecto "Adam" [20]. Para obtener una gráfica de las pérdidas durante el entrenamiento, se ha utilizado un *callback* de fastai llamado `ShowGraphCallback()`. Esta función muestra una gráfica cuyo eje vertical representa el valor del coste y su eje horizontal, simboliza el número de *batches* que pasaron por el *learner*. Puesto que hablar de número de *batches* es algo complejo y poco intuitivo, se ha decidido añadir un nuevo *callback* al argumento `cbs`, conocido como `CSVLogger`. Este permite exportar los datos del entrenamiento a un fichero *.csv* que, utilizando una hoja de cálculo como *Excel*, permite obtener gráficas con mayor flexibilidad. Estas gráficas serán las que se expondrán a lo largo del documento [20].

En cuanto al funcionamiento de `cnn_learner()`, al ejecutar dicha función con un modelo preentrenado, fastai realiza la descarga de los pesos preentrenados. Una vez descargados, como ya se explicó en la sección teórica de transfer learning (ver Sección 3.4.8), para aplicar esta técnica, hay que remplazar la última capa de la red puesto que está preparada para categorizar *ImageNet*. Para determinar el punto en el que realizar dicho corte para remplazar las sucesivas capas, existe una función llamada `model_meta` que indica dónde finaliza el cuerpo de la red y dónde comienza la cabeza. Se denomina cabeza de la red a la parte que es específica de una tarea en concreto y, por lo general, no se puede generalizar a otros problemas. El retorno de esta función será un *json* con tres claves: "cut", "split" y "stats", siendo el primero el que indica el punto de corte. En este caso, para el modelo preentrenado *ResNet34*, el punto de corte se encuentra en "−2". A partir de este punto, se crea la nueva

cabeza utilizando la función `create_head`. En dicha función se puede especificar el número de capas lineales que se quieren añadir al final, cuanto *dropout* usar después de cada una de ellas y que tipo de *pooling* usar. Sin embargo, fastai, por defecto, aplica tanto *average pooling* como *max pooling*, agrupadas en una capa conocida como *AdaptiveConcatPool2d*. La razón por la que fastai combina estos dos tipos de pooling, se debe a una pequeña mejora que se obtiene sobre utilizar solo *average pooling*. En cuanto al número de capas lineales adicionales, fastai opta por introducir dos capas en lugar de una (enfoque típico) debido a que descubrieron que dos capas lineales adicionales hacían que el transfer learning fuese más sencillo y rápido.

Otra función que se usa en visión por ordenador es `unet_learner`, destinada a tareas de segmentación, donde la salida es una nueva imagen que posee una etiqueta predicha para cada píxel. No solo se usa para tareas de segmentación, sino para cualquier tarea cuyo resultado sea una nueva imagen, como por ejemplo convertir una imagen de blanco y negro a color o mejorar la resolución de una imagen [39, Cap. 15].

En cuanto al entrenamiento, fastai cuenta con diversas funciones para entrenar. De entre todas ellas, se ha decidido utilizar la función `fine_tune`. Esta función, a diferencia de otras como `fit`, está diseñada de tal forma que si se aplica sobre un modelo preentrando, mantendrá y mejorará estos parámetros previamente entrenados. Para ello, entrena una primera época congelando las capas ya entrenadas, de forma que solo se ajusten los parámetros aleatorios de las nuevas capas creadas por el `cnn_learner` al conjunto de datos del problema. Una vez finalizada esta primera época, descongela por completo la red y comienza a entrenar toda la red con el número de épocas seleccionado. Pese a que este es el comportamiento por defecto, posee cierta flexibilidad que permite personalizarlo.

Si se quisiera entrenar una red neuronal desde cero, lo más común es utilizar la función de fastai `fit_one_cycle`. Sin embargo, `fine_tune` también se puede utilizar en dichos casos, pudiendo llegar a obtener mejores resultados en función del conjunto de datos con el que se esté trabajando [39, p. 46-47].

En `fine_tune` se puede fijar el learning rate base, entre otros parámetros. Respecto a este learning rate, hay que saber que `fine_tune` implementa una política de un ciclo. En 2018, Leslie Smith, publicó "Super-Convergence: Very Fast Training of Neural Networks Using Large Learning Rates" [84]. En dicho artículo planteó un entrenamiento en dos fases conocido como *entrenamiento de un ciclo*. En la primera fase, a modo de calentamiento, el learning rate se va incrementando gradualmente desde un valor mínimo a uno máximo. Por el contrario, la segunda fase es la acción inversa, se reduce el learning rate desde el valor máximo hasta devolverlo a su valor inicial, a modo de enfriamiento. La razón en la que se sustenta esta teoría es que en las primeras épocas del entrenamiento, la red posee unos pesos que, muy probablemente, no se ajusten muy bien al problema que se está resolviendo. Por ello, entrenar estas primeras épocas con un learning rate alto, podría provocar una divergencia instantánea. Por otro lado, tampoco es conveniente procesar las últimas épocas del entrenamiento con un learning rate alto, ya que se podría estar saltando un mínimo de la función de coste. Sin embargo, durante el resto del entrenamiento, el learning rate puede ser mayor. Esto provocaría que el entrenamiento se agilizase y que se saltasen los mínimos locales menos profundo, de forma que la red se sobreajustase menos. Esta política consigue entrenar con learning rates más altos que cualquier otra metodología, permitiendo

conseguir modelos más precisos en menores tiempos. Leslie denomina a esta característica como "superconvergencia" [39, p. 430-431].

6.5.2. Métrica y función de coste

En cuanto a la elección de la función de coste, esta ha de hacerse teniendo en cuenta el problema que se esté abordando. Una vez más, fastai cuenta con una funcionalidad que automatizará esta elección, determinando la mejor opción según la especificación del DataBlock. Sin embargo, es posible escoger manualmente cuál es la función de coste que se desea utilizar. En este caso, al tratarse de una regresión, fastai determina que la mejor opción es el error cuadrático medio. Más concretamente, fastai usará por defecto la función `MSELossFlat`, una versión de `nn.MSELoss` que aplana tanto la entrada como el objetivo [20]. Para comprobar la elección tomada por fastai, una vez que se crean los DataLoaders a partir del DataBlock, se ejecuta la función `DataLoaders.loss_func` [20]. La respuesta que devuelve en esta ocasión es "FlattenedLoss of MSELoss()". Siguiendo estas indicaciones, inicialmente se va a utilizar la función `MSELossFlat` como función de coste.

Al igual que existe una función de coste para que el ordenador pueda evaluar el rendimiento del modelo, buscando optimizar dicha función, existen las métricas, una medida para la interpretación humana con la que se puede evaluar la precisión que tiene el modelo entrenado. En problemas de regresión es muy típico encontrar métricas como la raíz del error cuadrático medio (RMSE, Root Mean Squared Error) o el error absoluto medio porcentual (MAPE, Mean Absolute Percentage Error) [80].

Finalmente, la métrica escogida para este problema ha sido RMSE. Además de su popularidad, la razón principal de dicha elección es que tanto el paper de Huertas-Company et al. (2015) como el de Dieleman et al. (2015) trabajan y expresan los resultados con dicha métrica. La Ecuación 6.1 describe la expresión matemática de esta métrica.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2} \tag{6.1}$$

6.5.3. Sistema base

A fin de experimentar y probar todo el potencial del transfer learning, se ha decidido optar por un modelo preentrenado de ResNet como sistema base de partida. Pese a que existan otras opciones de modelos preentrenados como los modelos DenseNet, los modelos de ResNet son los que mejores precisiones obtienen. Estas redes han sido entrenadas previamente con ImageNet, una base de datos con multitud de imágenes cuyo acceso es libre bajo uso no comercial. ImageNet ha resultado clave en muchos de los avances de la investigación del deep learning en el campo de la visión por ordenador.[50].

Existen cinco variantes de ResNet: 18, 34, 50, 101 y 152. Estos números hacen referencia al número de capas de cada variante [39, p. 30]. Teniendo en cuenta que a medida que se

incrementa la profundidad de la red, el coste computacional requerido a la GPU también incrementará debido al aumento de parámetros a entrenar, se ha escogido la versión 34 de ResNet, una versión intermedia que ofrece buenos resultados y sin suponer una gran carga computacional.

A modo de breve explicación, la arquitectura ResNet consiste en una arquitectura de redes neuronales convolucionales que fue introducida en 2015 a través del artículo "Deep Residual Learning for Image Recognition" [33] cuya autoría pertenece a Kaiming He, Xiangyu Zhang, Shaoqing Ren y Jian Sun, cuatro investigadores de la compañía Microsoft. Su nombre procede de "Residual Network" o red residual en español. En la actualidad, la arquitectura ResNet se ha convertido en la más utilizada para problemas de visión por ordenador [39, Cap. 14].

Realizando algunas pruebas previas, se ha determinado un número de 30 épocas para los entrenamiento. La razón principal es que a partir de la decimoquinta época, las pérdidas y la métrica se reduce de forma casi inapreciable.

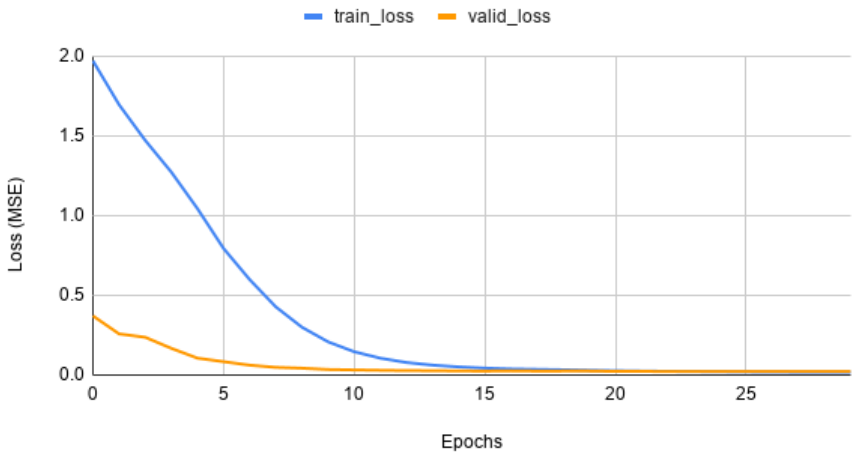


Figura 6.4: Costes del conjunto de entrenamiento y validación del sistema base

La imagen anterior (ver Figura 6.4), muestra las pérdidas obtenidas con la función `MSELossFlat` para el conjunto de entrenamiento y el de validación durante las 30 épocas del entrenamiento del sistema base. Esta gráfica muestra cómo en la primera mitad del entrenamiento, los pesos mejoran a buen ritmo. Al sobrepasar la época número 15, los pesos continúan descendiendo pero con de forma casi imperceptible. La métrica final obtenida con este modelo es de un RMSE de $0,149$.

6.5.4. Función de coste RMSE con Label Smoothing

Para observar cómo afecta la elección de la función de coste sobre el entrenamiento del modelo, se va a entrenar de igual forma el sistema base de antes con la función *RMSE* como función de coste.

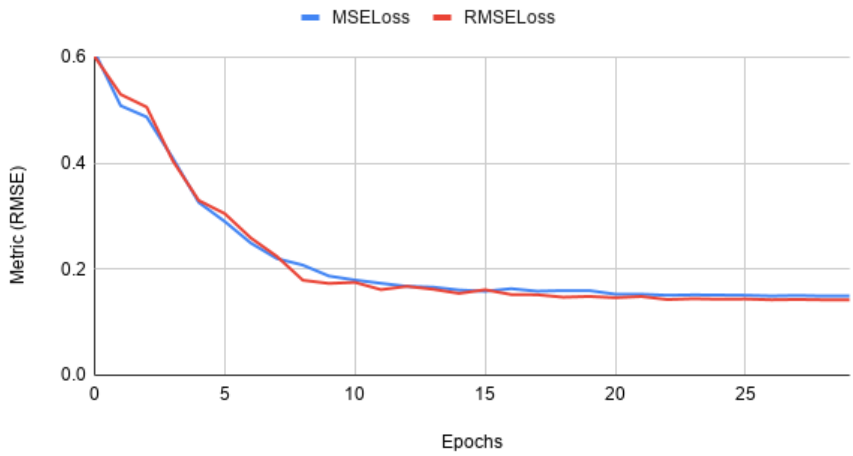


Figura 6.5: Precisión RMSE del modelo según la función de coste escogida

Según se puede ver en la imagen anterior (ver Figura 6.5), la diferencia entre las métricas obtenidas por ambas funciones de coste es mínima. Sin embargo, `RMSELoss` obtiene resultados ligeramente superiores, reduciendo la métrica de un RMSE final de 0,149 a 0,142.

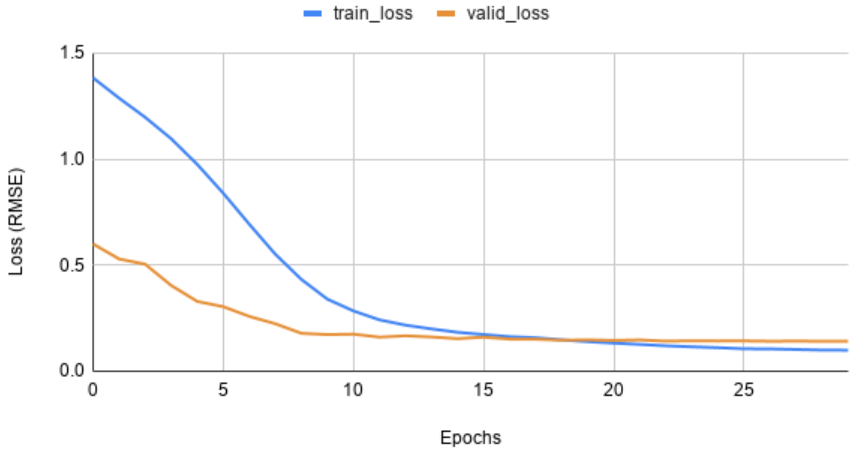


Figura 6.6: Pérdidas del conjunto de entrenamiento y validación con `RMSELoss`

Algo que sí se puede distinguir entre ambos casos, es una ligera superposición de la pérdida del conjunto de validación sobre el de entrenamiento, que surge en la época número 19 al utilizar como función de coste RMSE (ver Figura 6.6). Sin embargo, pese a que esto podría reflejar un potencial *overfitting*, como se ha visto en la métrica, esto no afecta en absoluto a la precisión final del modelo.

Debido a que que no hay una diferencia notoria, se continuará utilizando `RMSELoss`. Sin embargo, no se va a utilizar la función tal cual se encuentra, sino una modificación de esta. Como ya se comentó en la presentación del problema (ver Sección 6.1), los valores que se busca

que el modelo prediga, proceden de decisiones y evaluaciones subjetivas. Por esta razón y a fin de evitar que el modelo se sobreajuste a dichos valores, se va a aplicar un ruido aleatorio de un 10 %, es decir, una variación de ±0,05 a los valores objetivo. Para ello se ha creado la función `rmse_label_smoothing(preds, targs)`, encargada de aplicar dicha variación sobre los valores "targets" u objetivos en español (ver Fragmento de código 6.7).

```
def rmse_label_smoothing(preds, targs):
    targs += (torch.rand(targs.size(1),device='cuda')-0.5)*0.1
    return ((preds-targs)**2).mean().sqrt()  # rmse
```

Listing 6.7: Función de coste RMSE con label smoothing

Se ha ejecutado nuevamente el entrenamiento con esta nueva función. Puesto que los resultados obtenidos son extremadamente semejantes a los obtenidos con `RMSELoss`, se ha decidido no incluir ningún gráfico de los resultados de esta configuración.

6.5.5. Indicar el rango de las predicciones al modelo

Investigando en la documentación y en el libro de fastai, se encontró una forma para indicar al modelo cuál va a ser el rango de valores de las predicciones. El *cnn_learner* posee un argumento llamado `y_range` destinado a esto. Para ello, añade a la última capa un *SigmoidRange*, un módulo sigmoide que contiene el rango especificado. Para comprobar esto, se puede ejecutar sobre el *Learner* la función `summary()` y observar la última capa de la red [20]. Puesto que los valores sobre los que se está realizando la regresión son fracciones comprendidas entre el 0 y el 1, se ha entrenado con `y_range=(0,1)`.

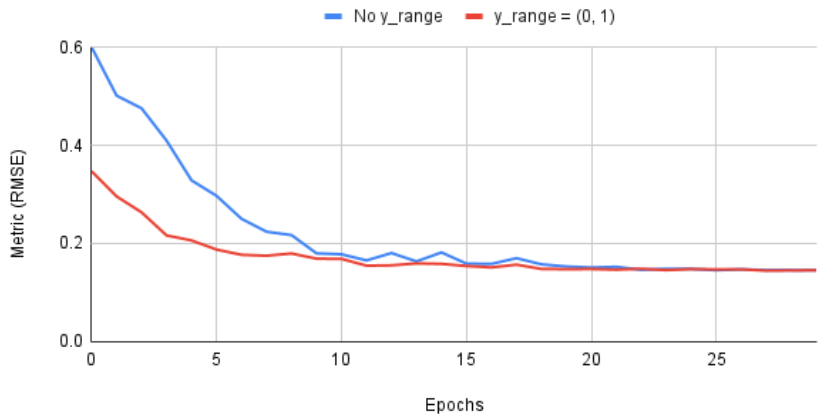


Figura 6.7: RMSE obtenido en función de la especificación de `y_range`

Como se puede observar en la Figura 6.7, esta técnica ayuda a conseguir rápidamente una buena precisión. Sin embargo, hacia la época número 9, ambas métricas comienzan a

estabilizarse, llegando a un decrecimiento casi inapreciable. Finalmente ambos casos alcanzan un RMSE de 0,145. Por ello, el uso de `y_range` podría ser una muy buena opción en caso de que los medios con los que se cuente para entrenar sean insuficientes para soportar mucha carga computacional, de forma que con pocas épocas se consiga una buena precisión del modelo.

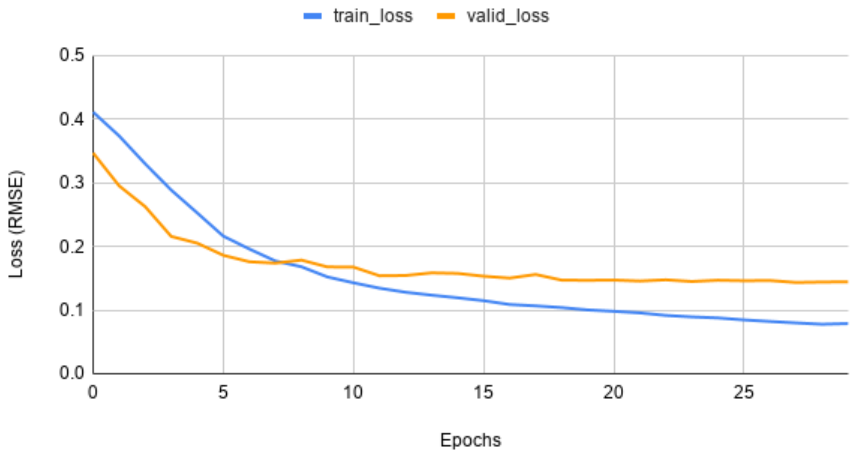


Figura 6.8: Pérdidas de entrenamiento y validación obtenidas al especificar `y_range=(0,1)`

Analizando las pérdidas obtenidas por el conjunto de entrenamiento y el de validación al especificar el `y_range` (ver Figura 6.8), se puede detectar un pequeño overfitting que arranca en la época número 7. Sin embargo, como se ha visto, esto no repercute negativamente sobre la precisión del modelo.

Puesto que esta modificación no supone ninguna mejora sobre la precisión del modelo en entrenamientos largos y añade un ligero y temprano overfitting, se ha decidido no continuar con esta técnica en los siguientes casos.

6.5.6. Transformaciones de datos

Incrementar el número de imágenes que se suministra al modelo durante el entrenamiento ayudaría a que su precisión y generalización mejorase. Una técnica que se utiliza para simular este incremento es el de *Data augmentation*, tal y como se vio en la Sección 3.4.6.

La biblioteca fastai proporciona una función denominada `aug_transforms` que suministra un conjunto estándar de variaciones aplicables con éxito a imágenes normales [39, p. 74]. Pese a ello, a la hora de elegir las transformaciones que se van a aplicar, hay que tener en cuenta la naturaleza de las imágenes con las que se está trabajando. En tan solo 4000 píxeles se codifican imágenes de galaxias distantes. Esto quiere decir que cada píxel posee un gran valor por la inmensa cantidad de información que representa. No es posible trabajar de igual forma con un dataset como MNIST, que representa números manuscritos, que con un dataset

de imágenes de galaxias lejanas. Por esta y otras razones astronómicas, las transformaciones que se pueden aplicar para la clasificación morfológica, son muy limitadas. En base a lo explicado en la sección "3.3. Pre-processing" del paper de Huertas-Company et al. (2015), se van a aplicar básicamente tres transformaciones [41].

La primera transformación que se va a aplicar es una normalización de los datos de entrada. La normalización se aplica para que los píxeles de todas las imágenes tengan una distribución similar para favorecer la convergencia del entrenamiento.

La segunda transformación a la que se someterán las imágenes es una rotación. Esta se va a aplicar en múltiplos de 90°. Para aplicar dicha transformación, se ha utilizado la clase de fastai `Rotate`. Sin embargo, para forzar esta rotación en múltiplos 90°, se ha definido una función nombrada como `random_rotation()` (ver Fragmento de código 6.8), que se introducirá en el campo `draw` de dicha transformación. A su vez, para obligar a que siempre se aplique esta transformación, se ha especificado que la probabilidad de su aplicación sea del 100 %. Para ello se ha establecido un valor de 1,0 en el campo `p` de `Rotate` [20].

```
def random_rotation(x):
    '''
    Return a float tensor equal in size to batch with random
    rotations multiples of 90.

    x: full batch tensor
    '''
    size = x.size(0)
    result = torch.zeros(([size]), dtype=float, device='cuda')

    for i in range(size):
        degree = torch.rand(1)
        if (degree < 0.25):
            result[i] = 0.0
        elif (degree < 0.5):
            result[i] = 90.0
        elif (degree < 0.75):
            result[i] = 180.0
        else:
            result[i] = 270.0

    return result
```

Listing 6.8: Función definida para realizar las transformaciones de rotación

Tras la rotación, se va a aplicar un ruido Gaussiano aleatorio a cada imagen. El ruido que se aplica es muy pequeño, de forma que no afecta visualmente al aspecto de las galaxias pero altera levemente los píxeles de estas, haciendo que para la red sean imágenes diferentes [41]. Para aplicar esta transformación se ha creado una clase llamada `GaussianNoise` que extiende de la clase de fastai `Transform` (ver Fragmento de código 6.9).

```
class GaussianNoise(Transform):
    order = 100 #after normalize

    def encodes(self, x:TensorImage):
        noise = torch.normal(
            mean=0.0,
            std=0.005,
            size=x.shape,
            device='cuda'
        )
        return x + noise
```

Listing 6.9: Transformación definida para aplicar ruido Gaussiano

Para entender cómo se implementa una transformación propia, es necesario comprender cómo funciona la clase `Transform`. Dicha clase posee un constructor `__init__` que se puede redefinir para conseguir una mayor personalización de la nueva transformación. En este caso, no se ha visto necesario redefinirlo. A su vez, posee una función `setups` que permite realizar operaciones o cálculos iniciales que puedan ser necesarios para la posterior transformación. El punto más importante reside en la función `encodes`, encargada de aplicar la transformación sobre los *inputs*. Su definición permite especificar el tipo de entrada que ha de admitir. En este caso, se ha exigido que sea una entrada de tipo `TensorImage` para que la transformación se aplique únicamente a las imágenes. La definición de este modelo se basa en generar una matriz de ruidos que se van a aplicar a cada píxel de la imagen. Por último existe una función `decodes`, destinada a realizar el proceso contrario del `encodes`, a fin de revertir la transformación. En este caso, como no ha sido necesaria, no se ha implementado. En adición a todo esto, se puede especificar otros comportamiento de la transformación a partir de unos atributos. Entre estos atributos se encuentra `order`, el cual permite especificar el orden en el que se debe aplicar la transformación frente a otras transformaciones de fastai [20].

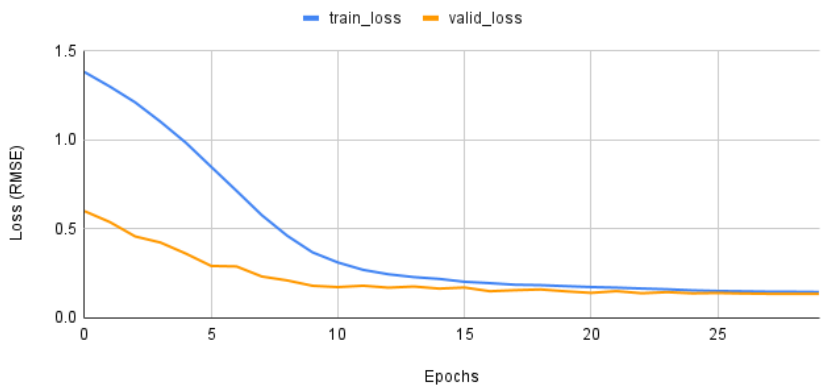


Figura 6.9: Pérdidas del conjunto de entrenamiento y validación obtenidas al aplicar transformaciones sobre las imágenes

Según se puede observar en la gráfica de pérdidas (ver Figura 6.9), al incorporar esta lista de transformaciones al argumento `batch_tfms` del DataBlock, en el orden descrito, el overfitting que se obtuvo en la Figura 6.6 al comenzar a utilizar la función RMSE como función de coste (ver Figura 6.6) ha desaparecido por completo. En todo momento la pérdida de validación es inferior a la de entrenamiento. Es posible esperar que la métrica haya mejorado.

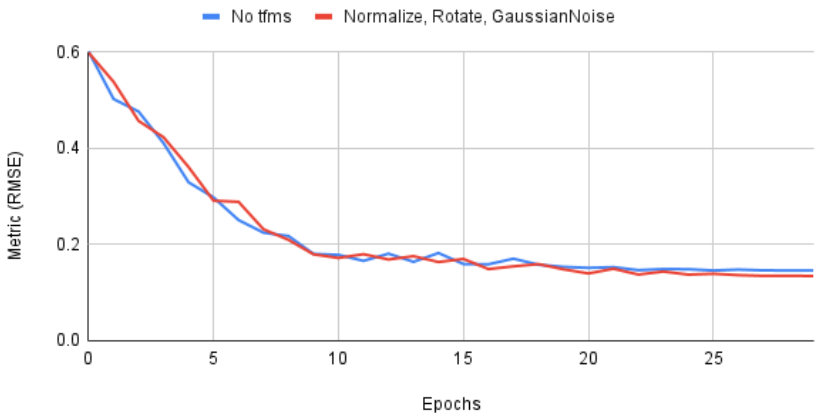


Figura 6.10: RMSE obtenido en función de si se han aplicado o no las transformaciones

Comparando la métrica obtenida mediante el uso de las transformaciones frente al mismo caso sin aplicarlas (ver Figura 6.10), ambas precisiones van muy a la par durante casi todo el entrenamiento. No es hasta la época 20 que comienza a definirse la nueva métrica como ligeramente mejor. Al final del entrenamiento, el modelo que usó las transformaciones obtuvo un RMSE de 0,134 frente al 0,145 obtenido con el modelo ligeramente sobreajustado sin transformaciones.

6.5.7. Análisis de distintas arquitecturas ResNet

Como se ha comentado anteriormente, ResNet cuenta con 5 variantes según la profundidad de su arquitectura. Una arquitectura más profunda se traduce en un mayor número de parámetros en el modelo, lo que suele derivar en una mejor optimización de las pérdidas del conjunto de entrenamiento. Sin embargo y en contraposición, al contar con más parámetros a ajustar, hay una mayor probabilidad de overfitting. A su vez, un mayor número de capas va a suponer, como es lógico, una mayor carga computacional sobre la GPU, exigiendo más memoria. Existen algunas formas de reducir la carga computacional si la GPU no es suficiente, como reducir el tamaño del *batch* al crear los *DataLoaders* o reducir la precisión de los números punto flotantes a media precisión gracias a ejecutar la función `to_fp16()` sobre el *Learner* [39, p. 213-215]. Para comprobar cómo afecta la profundidad de la red sobre la precisión de los modelos, se han probado las diversas variedades de ResNet, partiendo del caso anterior en el que se han aplicado las transformaciones.

Figura 6.11: RMSE obtenido con cada variante ResNet

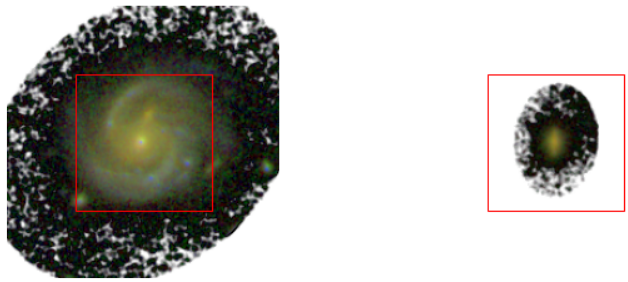
Observando los valores de RMSE obtenidos como métrica (ver Figura 6.11), según se va incrementando el número de capas de la arquitectura, la precisión aumenta. La ResNet18 obtuvo un RMSE final de 0,1340, muy semejante al obtenido con la ResNet34, cuyo valor es de 0,1335. En el caso de la ResNet50, se ha obtenido una mejora de la precisión algo más notoria, alcanzando un RMSE de 0,126. A partir de aquí, el incremento de capas hace que la precisión se descontrole. Es posible que esto se debe a la excesiva complejidad que se le está añadiendo, haciendo que el modelo sea ineficiente e impreciso.

Si se comparan estas precisiones con la descrita en el paper de Huertas-Company et al. (2015) [41, Sec. 3.3.], utilizando las versiones de ResNet 18 y 34, los valores RMSE obtenidos son ligeramente inferiores al RMSE aproximado de 1,3 que se describe en dicho paper. Sin embargo, el modelo de ResNet50, al alcanzar la cifra de 0,126, supera ligeramente dicha precisión. Es necesario remarcar que esta comparativa es meramente informativa ya que se están trabajando con distinto número de muestras y distintos conjuntos de entrenamiento y validación. Sin embargo, esta sirve para poder evaluar el rendimiento del modelo actual, pudiendo concluir que las precisiones obtenidas con la ResNet50 son buenas y cumplen bien con su labor de predecir las fracciones de expertos que determinarían una cierta característica de la galaxia.

Haciendo referencia a la carga computacional que han supuesto los entrenamientos de cada red, la ResNet18 ha requerido de 6 minutos y 31 segundos para computar las 30 épocas. La ResNet34, ha necesitado 10 minutos y 25 segundos para realizar su entrenamiento. Finalmente, la ResNet50 ha tardado 16 minutos y 26 segundos en completar todo el entrenamiento. Pese a que estos valores son totalmente relativos y dependientes del hardware utilizado para el entrenamiento, sirven para observar la relación entre el incremento de capas y el incremento temporal.

6.5.8. Reducción de imágenes a 100x100 píxeles

Analizando las imágenes con las que se está trabajando, se llegó a la conclusión de que en muchas de ellas, su galaxia ocupaba muy poco espacio en la imagen, dando lugar a un gran marco exterior sin información. Esto, a su vez, provocaba que los entrenamiento fueran más costosos a nivel computacional. Por estas razones, se ha decido probar a reducir el tamaño de las imágenes a una cuarta parte. Para ello, se ha recortado de cada lado de la imagen, 50 píxeles, lo que deriva en imágenes con tamaños de 100x100 píxeles.



(a) `DISK_13677_egs` (b) `DISKIRR_10411_goodsn`

Figura 6.12: Recorte de 100x100 píxeles sobre una imagen con mucha información frente a otra con pocos píxeles de información

Tal y como se puede ver en la Figura 6.12, en el caso de la galaxia 13677 del campo EGS (ver Figura 6.12a), el recorte hace que se pierda información relevante de la galaxia. Sin embargo, en la imagen de la derecha (ver Figura 6.12b), no se pierde nada de información con el recorte. Este segundo caso se repite en multitud de muestras.

Estas nuevas imágenes, al ser mucho más livianas que las anteriores, deberían agilizar el proceso de aprendizaje. Lo que se pretende con esta experimentación es comprobar cuál es deterioro de la precisión del modelo que supondrá esta reducción. Para comprobar este comportamiento, se ha partido del mejor escenario del experimento anterior, es decir, una ResNet50 que aplica las transformaciones descritas.

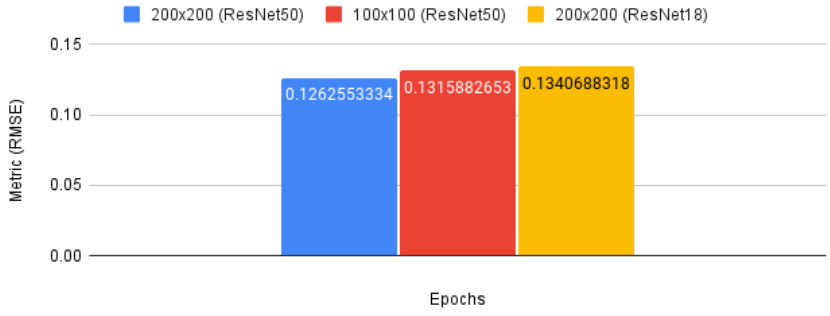


Figura 6.13: Métrica RMSE obtenida según el tamaño de las imágenes

La Figura 6.13 representa las métricas obtenidas con el mismo modelo de ResNet50 variando el tamaño de las imágenes de entrada. Adicionalmente se ha añadido la precisión de la ResNet18 sobre imágenes de 200x200 píxeles. La métrica es inferior con la reducción, obteniendo un RMSE de 0,132 frente al 0,126 de su rival. Sin embargo, esta precisión supera en 0,002 a la obtenida con la ResNet18 sobre imágenes de 200x200 píxeles (RMSE de 0,134).

En cuanto a los tiempos requeridos para el entrenamiento, la versión de 100x100 píxeles ha sido mucho más eficiente, tardando un total de 6 minutos y 2 segundos en completar el entrenamiento. Esta marca, supera incluso a la ResNet18 con imágenes de 200x200 (6 minutos y 31 segundos).

6.6. Enfoque de clasificación de categoría morfológica

Como se describió en la introducción del problema, el paper ofrece una interpretación de las fracciones de expertos que determinaron una característica específica ($f_{spheroid}$, f_{disk}, f_{irr}, f_{PS} y f_{Unc}) para separar las galaxias en seis categorías morfológicas distintas (`SPH`, `DISK`, `DISKSPH`, `DISKIRR`, `IRR` y `NONE`).

Al trabajar con imágenes enmascaradas en las que tan solo se desea obtener la clasificación morfológicamente de la galaxia central de la imagen, este enfoque se va a abordar como un problema de clasificación de única etiqueta.

6.6.1. Implementación de la solución

La implementación de esta solución ha sido muy semejante a la de regresión. A diferencia del enfoque anterior, en este caso no se requiere leer ningún fichero con información externa acerca de las galaxias. La única información que se necesita conocer es la categoría morfológica de cada galaxia. Como ya se explicó en la sección del conjunto de datos (ver Sección 6.2), en el propio nombre de la imagen viene codificada la morfología, el identificador de galaxia y el campo *CANDELS* al que pertenece. Realizando las modificaciones convenientes, se han redefinido el `DataBlock` para la tarea de clasificación (ver Fragmento de código 6.10).

```
blocks = DataBlock(
    blocks = (ImageBlock, CategoryBlock),
    get_items = get_image_files,
    splitter = RandomSplitter(valid_pct=0.2, seed=42),
    get_y = Pipeline([
        attrgetter("name"),
        RegexLabeller(pat = r'^(.*)_\d+_.*.png')
    ]),
    batch_tfms = tfms
)
```

Listing 6.10: DataBlock para la clasificación morfológica de galaxias

Lo primero que se indica, igual que en la regresión, es la naturaleza de las variables independientes y dependientes. Las variables independientes continúan siendo imágenes, por lo que se le asigna la clase "ImageBlock". Sin embargo, las variables dependientes han cambiado a etiquetas únicas, representadas por la clase "CategoryBlock".

Teniendo en cuenta que los nombres de los ficheros `.png` siguen la siguiente estructura: "`<morfología><identificador_galaxia><campo_candels>.png`", el primer campo del nombre codifica la categoría de la clasificación. Por ello, se ha especificado en el argumento `get_y` del DataBlock, que se obtengan las categorías a parir de los nombres de la imágenes, utilizando la expresión regular "`^(.*)_\d+_.*.png`". Una expresión regular es un patrón que busca hacer coincidencias con combinaciones de caracteres [17][58].

El símbolo '`^`' indica el inicio de la línea. El punto coincide con cualquier carácter excepto el carácter '`\n`' equivalente al salto de línea. El asterisco hace que la expresión regular resultante corresponda con 0 o más repeticiones de la anterior. El '`\d`' hace referencia a cualquier número comprendido entre el 0 y el 9. El símbolo de suma es similar al asterisco pero en este caso se exige al menos una repetición, es decir, una multiplicidad de 1 a muchos. Por último, tanto el carácter "_" como la cadena "`.png`" hacen referencia a sus propios valores *ascii*. La parte de la expresión regular que se encuentra entre paréntesis es la que corresponderá con la categoría de la clasificación [72]. Existen otras muchas alternativas para proporcionar la variable "y" al modelo. Un ejemplo sería la función "`parent_label`" que determinaría la categoría a partir del nombre del directorio en el que se encuentren las distintas imágenes, por lo que sería necesario separar las imágenes en subdirectorios diferentes en función de su morfología [20]. El resto del DataBlock es exactamente igual que el descrito en regresión (ver Sección 6.5.1), por lo que se va a omitir su descripción.

Para asegurar que la distribución de las imágenes en los dos datasets (entrenamiento y validación) sea homogénea y no haya ninguna descompensación, se ha implementado una función llamada `datasets_info(dls)`, cuyo fin es mostrar la información de estos conjuntos de datos por separado.

Categoría	Train	Valid	Suma
DISK	731 (26,4 %)	170 (24,6 %)	901
DISKIRR	287 (10,4 %)	62 (9,0 %)	349
DISKSPH	795 (28,7 %)	181 (26,2 %)	976
IRR	70 (2,5 %)	30 (4,3 %)	100
NONE	131 (4,7 %)	32 (4,6 %)	163
SPH	754 (27,2 %)	217 (31,4 %)	971
Total	2768 (100 %)	692 (100 %)	3460

Tabla 6.1: Distribución de los conjuntos de datos de entrenamiento y validación según las 6 diferentes categorías

Como puede observarse en la Tabla 6.1, la división de los dos *datasets* por cada categorías es muy homogénea. Ambos conjuntos de datos toman aproximadamente la misma proporción de imágenes por cada morfología.

6.6.2. Función de coste y métrica

Tal y como se comentó en la sección de regresión, fastai determinará la mejor función de coste acorde al problema que se esté resolviendo. En el caso de las clasificaciones, dependerá de si se trata de una clasificación de etiqueta única o múltiples etiquetas. Para el primer tipo, lo recomendado es utilizar la función de PyTorch *nn.CrossEntropyLoss* (función de pérdida de entropía cruzada). Por el contrario, para una clasificación de múltiples etiquetas convendría optar por *nn.BCEWithLogitsLoss* (función de pérdida de entropía cruzada binaria con activación sigmoide [73]) [39, p. 237].

Para analizar la precisión del modelo, se ha decidido escoger una métrica sumamente intuitiva como es la de *accuracy* o precisión en español. Esta métrica representa del 0 al 1 el número de aciertos sobre el número total de casos, es decir, representa el porcentaje de galaxias del conjunto de validación que el modelo ha acertado su morfología. En este caso, al tratarse de 6 categorías, una predicción totalmente aleatoria equivaldría a un *accuracy* de 1/6, es decir, 0,167. Una vez sobrepasado dicho umbral, el modelo estaría aprendiendo.

6.6.3. Sistema base

El sistema base del que se va a partir es el mismo que en el caso de regresión. El modelo que se va a utilizar va a ser una *ResNet* en su versión 34 y se va a comenzar sin aplicar ninguna transformación sobre el *batch*. En cuanto al número de épocas, en este caso, tras realizar diversas pruebas, se ha escogido 20 épocas para realizar el entrenamiento, manteniendo el learning rate base que `fine_tune` tiene establecido por defecto.

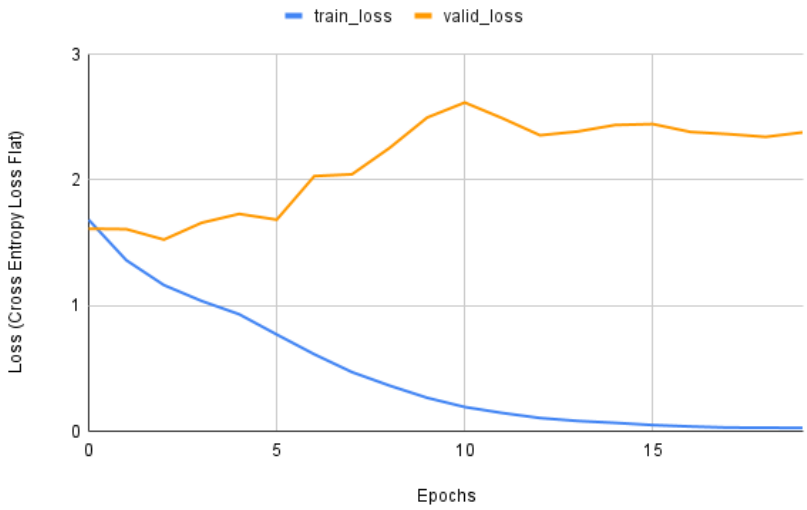


Figura 6.14: Pérdidas del conjunto de entrenamiento y validación obtenidas con el sistema base

La gráfica anterior (ver Figura 6.14) muestra las pérdidas obtenidas con sistema base, donde se puede observar cómo el coste del conjunto de entrenamiento desciende durante todo el entrenamiento, mientras que el de validación, tras comenzar descendiendo muy lentamente durante las tres primeras épocas, empieza a ascender, es decir, empieza a empeorar. En ese mismo instante, el modelo comienza a sobreajustarse.

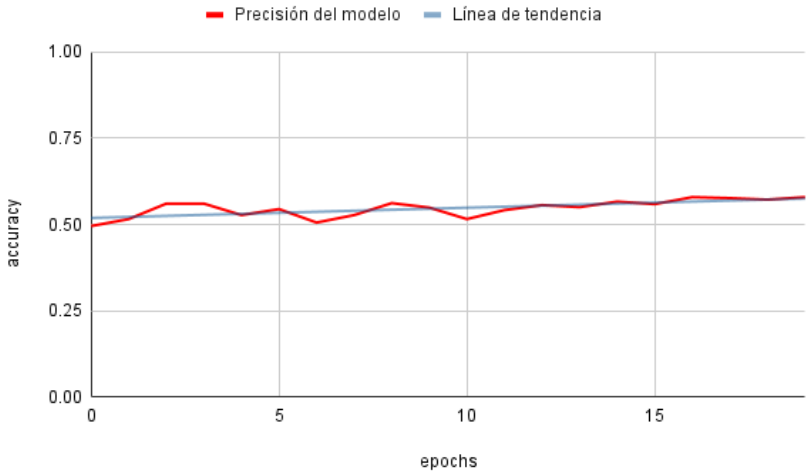


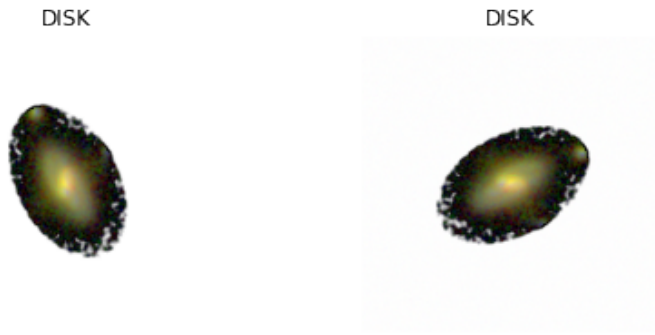
Figura 6.15: Precisión obtenida con el modelo base de clasificación

Sin embargo, según revela la línea de tendencia de la gráfica de la Figura 6.15, la precisión del modelo sigue mejorando durante todo el entrenamiento, pese a que lo haga muy lentamente.

Este caso refleja un modelo que se está confiando, es decir, cada vez está más seguro de las predicciones que hace. Esto no es alarmante puesto que lo que importa realmente es la precisión del mismo. Seguido a esto, es común que la métrica pueda empeorar, por lo que incrementar en exceso el número de épocas del entrenamiento podría provocar un deterioro de la precisión del modelo. En ese caso, la red estaría memorizando el conjunto de entrenamiento, lo que hace perder al modelo su capacidad de generalizar el conocimiento [39, p. 212].

6.6.4. Transformaciones de datos

Para solventar el sobreajuste que se produce en el sistema base, se van a aplicar las mismas transformaciones que fueron descritas en el apartado de regresión (ver Sección 6.5.6). Para comprobar que estas transformaciones se están realizando con éxito, se ha ejecutado la función `show_batch()` antes y después de aplicar las transformaciones. Como el *DataBlock* posee una semilla aleatoria específica, la imagen que devolverá dicha función, corresponde a la misma muestra. De esta forma se puede observar si realmente se han ejecutado las transformaciones.



(a) Sin transformaciones (b) Con transformaciones

Figura 6.16: Resultado de `show_batch()`, antes y después de aplicar las transformaciones

Tal y como se puede ver en la Figura 6.16, se ha aplicado una rotación de 90° en sentido horario sobre la imagen. Como ya se comentó y se puede comprobar, el ruido Gaussiano no es perceptible a simple vista.

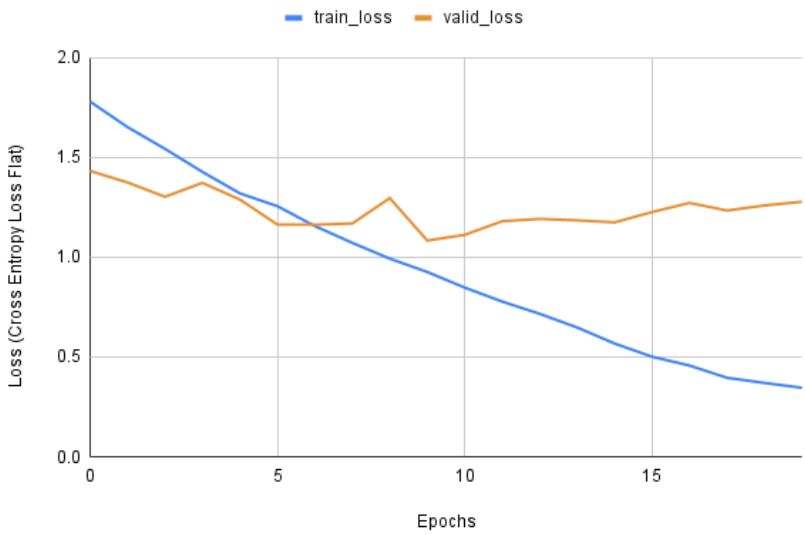


Figura 6.17: Pérdidas del conjunto de entrenamiento y validación obtenidas al aplicar las transformaciones descritas en cada batch

En esta ocasión, como se puede observar en la Figura 6.17, al aplicar las transformaciones, el overfitting se ha reducido considerablemente, apareciendo en la séptima época y sin experimentar apenas crecimiento en la pérdida obtenida con el conjunto de validación, pasando de 1,16 en la séptima época a 1,28 en la última.

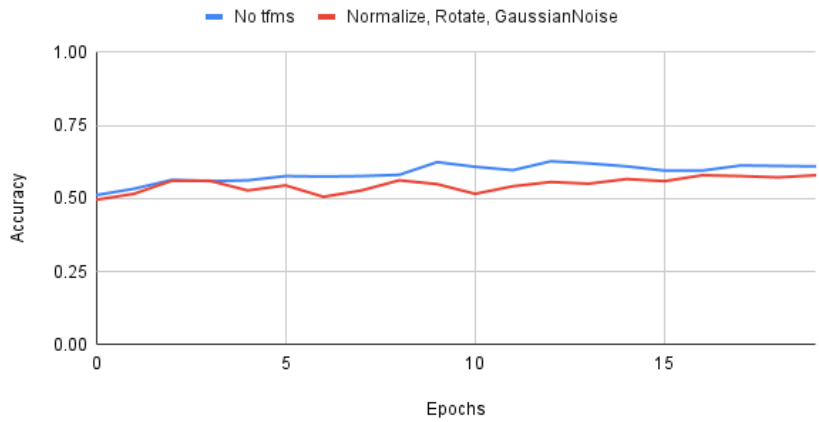


Figura 6.18: Precisión obtenida al aplicar transformaciones frente a la del sistema base

Poniendo atención a las métricas, siendo esto lo de mayor interés (ver Figura 6.18), se puede observar una pequeña mejora de un 2 % aproximadamente, en la precisión del modelo tras aplicar las transformaciones, pasando de un 57, 9 % de acierto a un 61 %. Pese a que la mejora no es muy grande, es lo suficiente para decidir mantener estas transformaciones a lo largo de los siguientes experimentos.

6.6.5. Análisis de distintas arquitecturas ResNet

La existencia constante del overfitting hace cuestionar cómo afectará la variación de la profundidad de la arquitectura sobre la precisión del modelo. En esta ocasión, se va a probar tanto la ResNet18 como la ResNet50 y se van a comparar con la ResNet34 del apartado anterior. Es de suponer que el overfitting sea mayor según se incremente el número de capas.

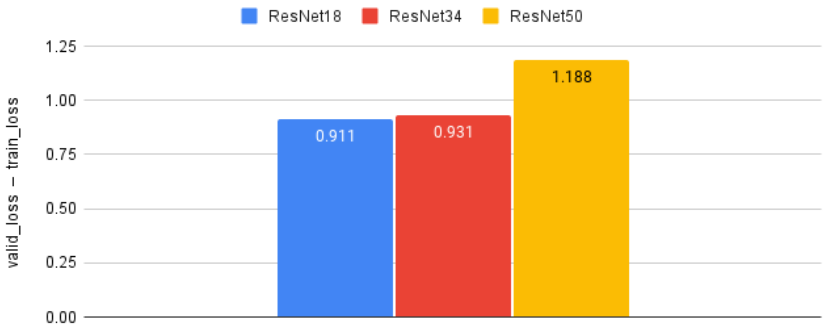


Figura 6.19: Diferencias entre las pérdidas de entrenamiento y validación en la última época según la arquitectura ResNet

Analizando la diferencia entre las pérdida obtenidas por el conjunto de entrenamiento y el de validación en la última capa para cada modelo ResNet (ver Figura 6.19), a medida que se amplía el número de capas de la red, la diferencia entre ambas pérdidas aumenta. La ResNet18 ha obtenido una pérdida final en el conjunto de entrenamiento de 0,37 y 1,28 en el de validación. La versión 34 ha obtenido 0,346 de pérdida en el conjunto de entrenamiento y 1,28 en el de validación. Por último, la versión 50, ha obtenido una pérdida de 0,15 en el conjunto de entrenamiento y 1,35 en el de validación. Como se puede observar, el incremento de la profundidad de la red ha supuesto una disminución progresiva de los costes del conjunto de entrenamiento y un aumento de las pérdidas del conjunto de validación.

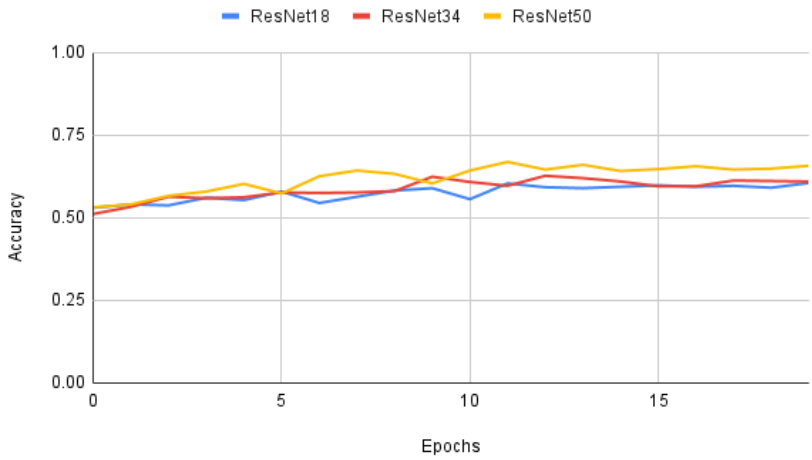


Figura 6.20: Accuracy obtenido por las diferentes arquitecturas de ResNet

Observando la gráfica de las distintas precisiones (ver Figura 6.20), estas mejoran acorde al incremento de profundidad de la red. Pese a que finalmente la precisión de la ResNet18 (60,55 %) sea muy próxima a la ResNet34 (60,98 %), durante todo el entrenamiento, se observa una pequeña pero clara superioridad de la ResNet34. Finalmente, la mejor precisión obtenida es de un 65,75 %, correspondiente a la ResNet50.

6.6.6. Modificación del learning rate y el número de épocas

Los casos de este último experimento partirán del mejor caso obtenido hasta el momento, una red ResNet50 que aplica las transformaciones descritas anteriormente.

Uno de los factores más importantes a tener en cuenta durante la fase de entrenamiento del modelo es el de elegir un *learning rate* óptimo. En 2015, Leslie Smith ideó un buscador de *learning rates* que consistía en simular un entrenamiento, comenzando con un *learning rate* muy pequeño aplicado al primer *mini-batch*. Tras observar las pérdidas obtenidas, este hiperparámetro se incrementa en cierto porcentaje y se aplica sobre un nuevo *mini-batch*. Este proceso se itera hasta que las pérdidas dejan de mejorar y comienzan a empeorar. En ese momento es donde se ha de elegir una tasa de aprendizaje anterior a dicho punto [83].

Existen ciertas teorías o reglas para determinar una buena tasa de aprendizaje. Una de ellas consiste en tomar la tasa de aprendizaje correspondiente a la mínima pérdida obtenida pero con un orden de magnitud inferior, es decir, escoger la división de aquella tasa que obtiene la mínima pérdida entre 10. La otra opción que se plantea es escoger el *learning rate* en el que la pérdida se reduce con mayor rapidez, es decir, aquella tasa en la que la gráfica de las pérdidas obtenga su mayor inclinación [39, Cap. 5].

En fastai existe una función denominada `lr_find()` que realiza la búsqueda del *learning rate* para una configuración de *Learner* específica. Esta función es bastante flexible, permitiendo configurar el learning rate de partida (`start_lr`) y el de finalización (`end_lr`), así como detener la búsqueda si diverge.

La ejecución de dicha función mostrará una gráfico en escala logarítmica que representa el incremento del *learning rate* frente al correspondiente valor de pérdida obtenido. A su vez, si el argumento `suggestions` está activado (por defecto toma el valor de `True`), también devolverá dos números correspondientes a las dos reglas explicadas anteriormente. La primera de ellas tomará el seudónimo de `lr_min` y la segunda el de `lr_steep` [20].

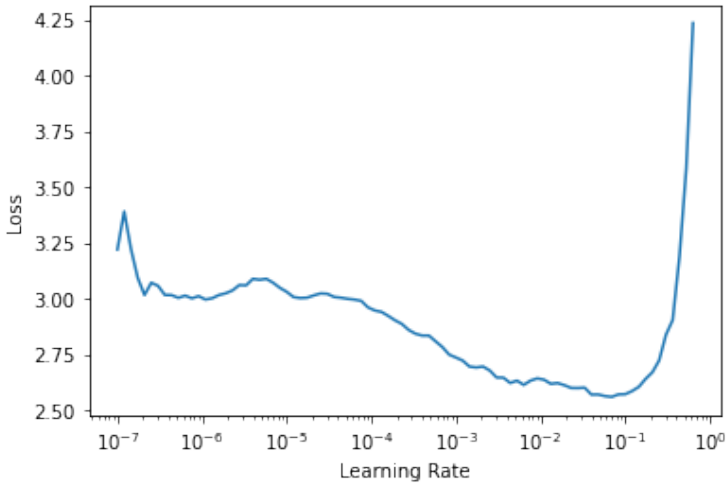


Figura 6.21: Gráfica obtenida de la función `lr_find()`

Observando la gráfica obtenida (ver Figura 6.21), se puede ver que desde un *learning rate* algo inferior a 10^{-6} hasta el valor 10^{-4}, aproximadamente, las pérdidas no experimentan ninguna tendencia a descender. Esto demuestra que las tasas comprendidas entre esos dos valores son tan pequeñas que el modelo no llega a aprender. A partir de ese punto, las pérdidas obtenidas comienzan a descender hasta un valor aproximado de 10^{-1}, donde se alcanza el mínimo de la función y comienzan a ascender.

Teniendo todo esto en cuenta, se van a probar los dos valores recomendados por el `lr_find`, es decir, se va a experimentar la precisión del modelo tras un entrenamiento de 20 épocas con un *learning rate* de 6,92e-03 (`lr_min`) frente al mismo entrenamiento con una tasa de aprendizaje de 6,92e-04 (`lr_steep`).

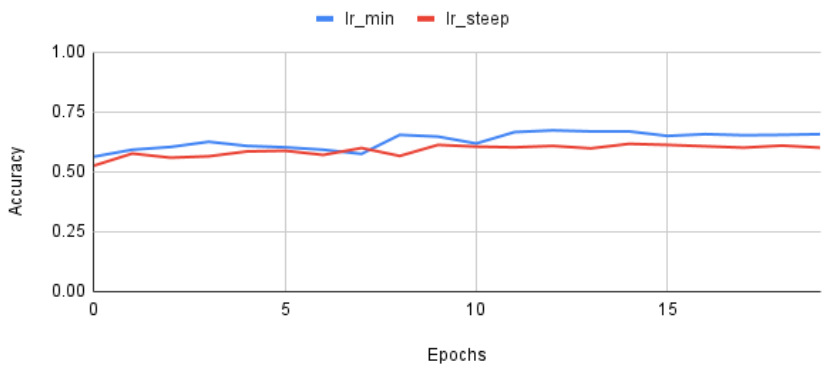


Figura 6.22: Accuracy obtenido con `lr_min` y `lr_steep`

Comparando la evolución de ambas precisiones a lo largo de todo el entrenamiento (ver Figura 6.22), se puede ver cómo la precisión de los dos modelos se mantiene bastante estática en las 10 últimas épocas, con una tendencia muy aplanada. Durante todo el entrenamiento se puede observar cómo la precisión obtenida con el *learning rate* más alto, es decir, el `lr_min` es siempre superior a la de su rival, alcanzando un éxito de 65,61 % frente al 59,97 del `lr_steep`.

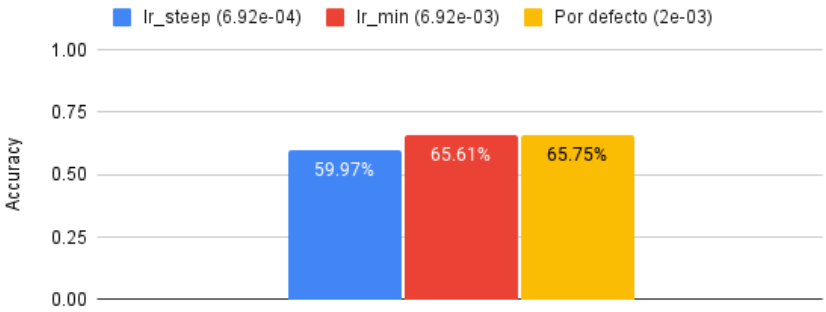


Figura 6.23: Precisiones obtenidas al utilizar `lr_min`, `lr_steep` y el learning rate por defecto de `fine_tune`

Si se comparan ambas precisiones con la obtenida sin especificar un valor base de *learning rate*, es decir, manteniendo el que `fine_tune` establece por defecto (ver Figura 6.23), pese a que las elecciones de *learning rates* hayan sido buenas, la precisión del modelo en el que no se especifica un l*earning rate* concreto es algo superior. Sin embargo, la diferencia entre la precisión obtenida con el `lr_min` y el *learning rate* por defecto es casi inapreciable. Esto se debe a que el *learning rate* que `fine_tune` establece por defecto es de 0,002, un valor sumamente próximo al de `lr_min` [20].

Teniendo en cuenta que los dos *learning rates* más altos obtienen las mejores precisiones, se puede deducir una relación entre el incremento del *learning rate* y la mejora de la precisión. Como ya se explicó en el Capítulo 3, un *learning rate* muy grande, provocará que el

aprendizaje no converja y, por tanto, la precisión se desplome. Por ello, se ha optado por probar un nuevo valor ligeramente superior a los anteriores, para comprobar si mejora la precisión del modelo. El *learning rate* escogido es 0,01.

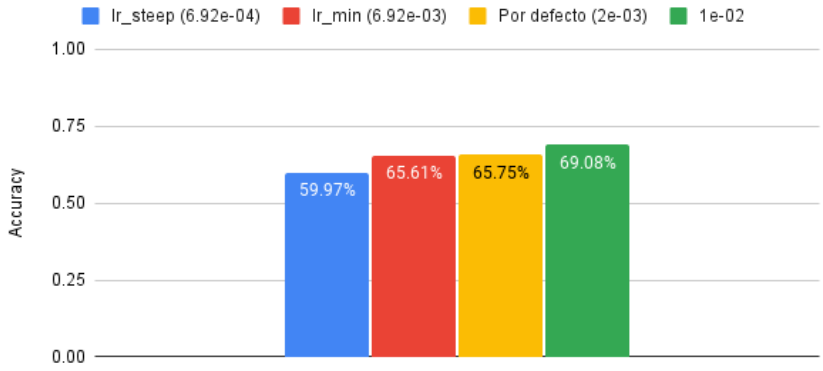


Figura 6.24: Accuracy obtenido usando los distintos learning rates

Volviendo a comparar todos los *learning rates* probados (ver Figura 6.24), se puede observar cómo la precisión asociada a este nuevo *learning rate* de 0,001 ha superado todas las anteriores. Con este nuevo *learning rate* se ha alcanzado el 69,08 % de precisión, casi un 10 % más de la precisión obtenida con el `lr_steep`.

Analizando la progresión que sigue la precisión a lo largo del entrenamiento, se puede ver que en la época decimoquinta se obtiene una precisión de 70,38 %, valor ligeramente superior al de la última época. Para buscar mejorar todavía más esa precisión, se va a probar a modificar el número de épocas a entrenar. Reduciendo el entrenamiento a 15 épocas, se obtuvo una precisión algo superior, sin embargo, aún había épocas con mejor precisión que la final, como por ejemplo la décima época, con una precisión del 70,95 %. De la misma forma que antes, se redujo de nuevo el entrenamiento a 10 épocas [39, p. 212-213].

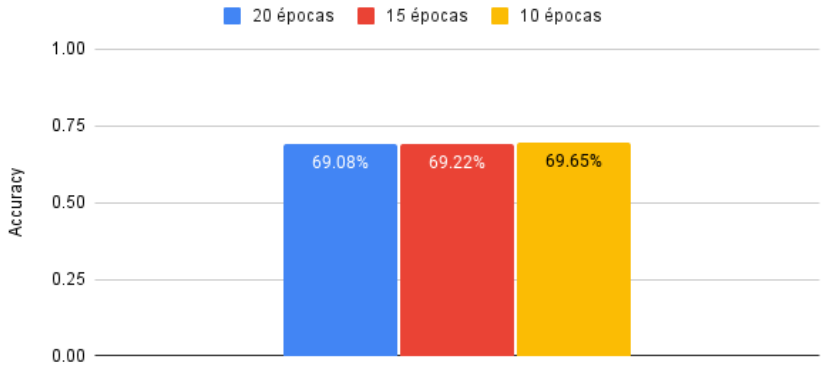


Figura 6.25: Precisión obtenida según el learning rate utilizado

Tal y como se puede observar en la Figura 6.25, a medida que se iba recortando el entrenamiento según en qué época se obtenía la mejor precisión, la precisión final del modelo mejoraba ligeramente. Esto se debe al comportamiento de `fine_tune` que, como ya se explicó anteriormente, reduce el learning rate en las últimas épocas, pudiendo alcanzar los mínimos.

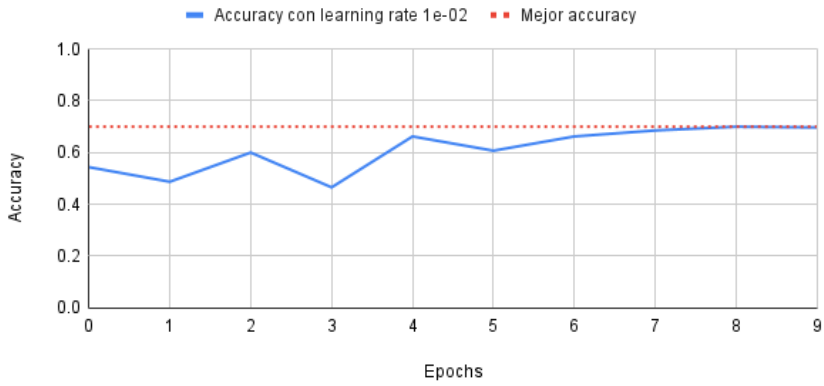


Figura 6.26: Evolución de la precisión obtenida con un learning rate de 1e-02 entrenando durante 10 épocas

Observando la precisión de este último entrenamiento (ver Figura 6.26), su última época recibe la mayor precisión de todo ese entrenamiento (a excepción de la época anterior que le supera por tan solo 0,002), convirtiéndose en la mejor marca hasta el momento.

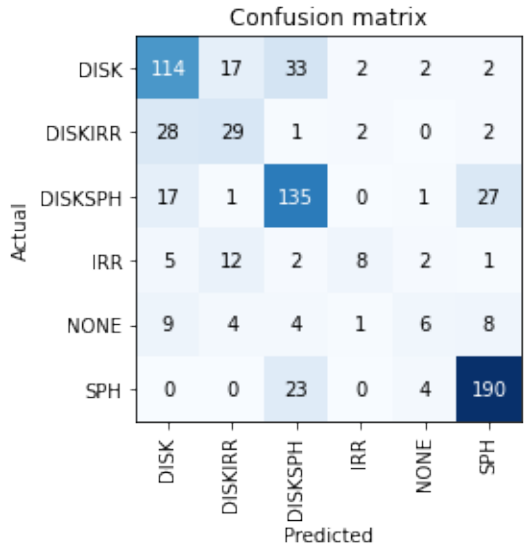


Figura 6.27: Matriz de confusión obtenida con el modelo entrenado durante 10 épocas

La Figura 6.27 muestra la matriz de confusión para este último y mejor caso. Una matriz de confusión es una matriz cuadrada cuyo lado tiene tantos elementos como número de categorías tenga la clasificación, en la que cada fila representa todas las muestras del *dataset* por categoría y las columnas representan las muestras que el modelo predice por cada categoría. Por tanto, la diagonal de dicha matriz representa todas las muestras que han sido clasificadas correctamente por el modelo. Toda aquella celda externa a la diagonal, son clasificaciones erróneas. Gracias a esta matriz es posible detectar desde errores en el dataset, como el mal etiquetados de los datos, hasta errores en el modelo, como la forma en la que se tratan los datos u otros factores [39, Cap. 2].

La biblioteca fastai hace especialmente visual la representación de esta matriz utilizando mayores intensidades de color para las celdas con las cifras más altas. De esta forma, con un simple vistazo, si la zonas externas a la diagonal tienen los colores más claros y la diagonal los más oscuros, el modelo estaría haciendo buenas predicciones.

Para obtener la matriz de confusión hay que generar un `ClassificationInterpretation`, clase que contiene métodos para la interpretación de modelos de clasificación [20]. A partir de este objeto, se obtiene la matriz de confusión, según puede verse en el Fragmento de código 6.11.

```
interp = ClassificationInterpretation.from_learner(learn)
interp.plot_confusion_matrix()
```

Listing 6.11: Obtención de la matriz de confusión [39, p. 76]

En este caso concreto, las filas representan las imágenes del conjunto de validación que son de tipo `DISK`, `DISKIRR`, `DISKSPH`, `IRR`, `NONE` y `SPH`, respectivamente. En cuanto a las columnas, estas representan las predicciones que el modelo ha realizado pensando que se trataba de una morfología de tipo `DISK`, `DISKIRR`, `DISKSPH`, `IRR`, `NONE` o `SPH`, en dicho orden.

A partir de esta matriz, se puede obtener el porcentaje de acierto del modelo por cada morfología. Por tanto, la categoría `DISK` ha obtenido un 67,06 % de acierto, las imágenes `DISKIRR` un 46,77 %, la categoría `DISKSPH` ha conseguido la segunda mejor precisión con un 74,59 % de acierto, las imágenes de tipo `IRR` un 26,67 %, las `NONE` han obtenido la peor precisión con tan solo un 18,75 % de acierto y, por último, la categoría `SPH` ha alcanzado la mejor precisión con un éxito del 87,56 %. Las peores precisiones están asociadas a las morfologías con menor número de muestras en el conjunto de datos.

Analizando las confusiones del modelo, se puede deducir que estas están relacionadas con la semejanza de morfologías, es decir, existirá una mayor confusión entre morfologías que comparten características (por ejemplo `DISKSPH` comparte características con `DISK` y `SPH`).

En el caso de las imágenes de tipo `DISK`, en muchas ocasiones, el modelo las confunde con galaxias de morfología `DISKIRR` o `DISKSPH`. Casi el 30 % de las predicciones totales, indican que las imágenes son de ambos tipos, lo que equivale a un 89,29 % de las predicciones erróneas (un 30 % para `DISKIRR` y un 59 % para `DISKSPH`).

En cuanto a la morfología DISKIRR, la mayor confusión del modelo se ubica en la morfología DISK, consiguiendo casi el mismo porcentaje que las predicciones correctas de esta categoría (45,16 %). Por el contrario, el modelo apenas confunde esta categoría con su otra categoría semejante, IRR (tan solo un corresponde a un 3,2 % de las predicciones).

En el caso de la morfología DISKSPH, casi el 15 % de las predicciones de dichas imágenes, indican que son de tipo SPH y el 9,39 %, creen que se trata de una morfología DISK. Ambos porcentajes reflejan una baja confusión. Esto equivale a un 59 % de las predicciones erróneas, aproximadamente.

La morfología IRR posee una gran confusión, focalizada en la morfología DISKIRR. El modelo predice el 40 % de las ocasiones que se trata de una morfología DISKIRR.

Sin embargo, la categoría NONE presenta una confusión más general, es decir, el modelo confunde esta morfología con cualquier otra de forma casi indistinta. La categoría que más predicciones recibe para esta morfología es DISK, seguida de SPH.

Por último y pese a la gran similitud que pueda existir, tan solo el 11 % de las predicciones realizadas sobre galaxias de tipo SPH, indican que la categoría correcta es DISKSPH, reflejando una confusión muy baja.

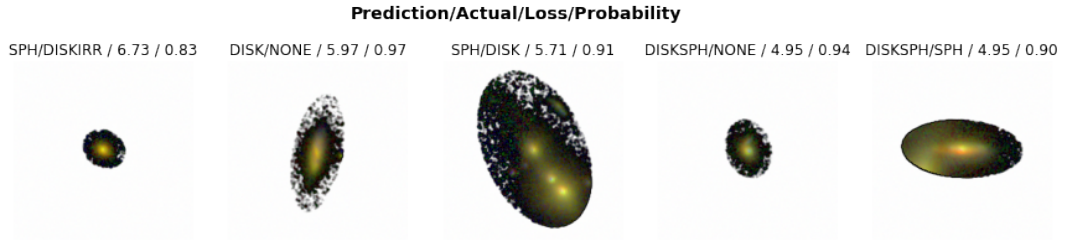


Figura 6.28: Peores pérdidas de validación para el modelo entrenado con 10 épocas

La Figura 6.28 muestra la salida de la función `plot_top_losses()` perteneciente a la clase `ClassificationInterpretation` [20]. Esta representa las imágenes del conjunto de validación que han obtenido las pérdidas más altas. Hay que tener en cuenta que una pérdida grande puede implicar que el modelo está haciendo malas predicciones. Esta pérdida aumentará si, además de hacer malas predicciones, tiene una alta confianza sobre ellas. Por otro lado, también puede significar que el modelo no posee una gran confianza en sus predicciones pese a que estas sean correctas. El primer caso de ellos, el modelo predice que se trata de una galaxia de morfología SPH con una confianza de un 83 %.

Esta función puede ser de gran ayuda para depurar los *datasets*, analizando las imágenes que han obtenido las peores pérdidas, ya que si se obtiene una alta confianza en una predicción errónea, puede significar que realmente el modelo está en lo correcto y la imagen en cuestión está mal etiquetada [39, p. 76-78].

6.7. Conclusiones

Se ha creído conveniente, a modo de conclusión del capítulo, analizar y comparar la precisión de ambos enfoques para el mismo problema, categorizar morfológicamente las galaxias.

Para ello se va a utilizar la última configuración del entrenamiento de clasificación, es decir, una ResNet50 que aplica transformaciones sobre las imágenes de entrada y que se entrena a un *learning rate* de 0,01 durante las épocas que mejor le convengan, siendo en este caso 30 épocas. De esta forma se va a probar cómo el enfoque de regresión se puede extrapolar a una clasificación categórica.

Para evaluar la precisión del enfoque de regresión para esta tarea, se ha definido una nueva métrica denominada `morph_accuracy`, en la que a partir de los 5 valores predichos por el modelo, se obtiene su categoría morfológica esperada, utilizando la interpretación de Huertas-Company et al. (2015) (ver Fragmento de código 6.1), y se compara con la categoría morfológica real que se obtiene a partir de los valores objetivo de la regresión. De esta forma se consigue una métrica equivalente a la que se utiliza en el enfoque de clasificación, basada en el porcentaje de casos exitosos sobre el número de casos totales.

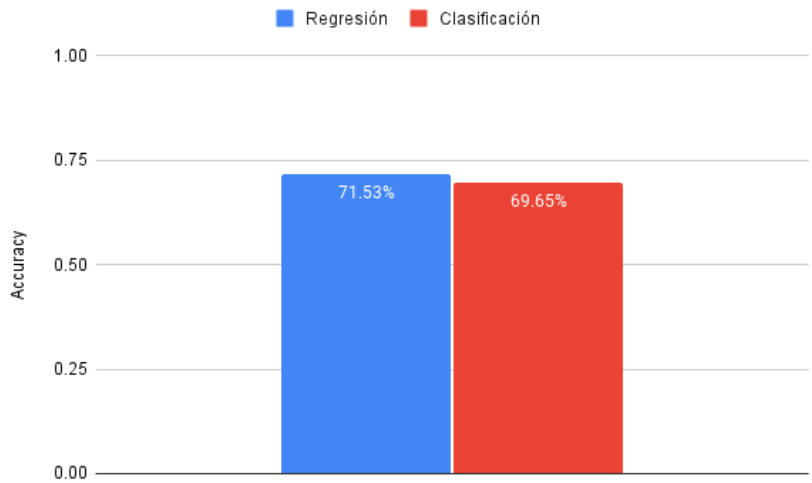


Figura 6.29: Comparativa de la precisión del modelo de regresión frente al de clasificación

La Figura 6.29 compara los resultados obtenidos por el enfoque de regresión frente al modelo equivalente del enfoque de clasificación. Puede verse que bajo las mismas condiciones, el enfoque de regresión se posiciona ligeramente por encima, obteniendo una precisión del 71,53 % frente al 69,65 % obtenido con la clasificación.

Podría pensarse que no se encuentran en las misma condiciones pues el enfoque de clasificación se entrena con tan solo 10 épocas frente a las 30 épocas de regresión. Sin embargo, haciendo pruebas de incremento del número de épocas para este segundo enfoque, la preci-

sión final del modelo es inferior a la obtenida con 10 épocas. Esta fluctúa en todo momento en torno a un 69 %.

Ante esa pequeña diferencia, la conclusión a la que se puede llegar es que ambos enfoques tienen un rendimiento muy semejante en la clasificación morfológica de galaxias, obteniendo precisiones en torno al 70 % de éxito.

Este último modelo de regresión es el que mejor precisión ha obtenido. Su RMSE es de 0,117, que si se compara con el modelo de regresión basado en una ResNet50 que se vio en la Sección 6.5.7, este ha mejorado en casi 0,01, al entrenarlo bajo un learning rate de 1e-02. Por otro lado, si se compara esta métrica con la descrita en el paper de Huertas-Company et al. (2015), el RMSE del modelo actual mejora en 0,013 la métrica del paper, aproximadamente. De nuevo, se remarca que esta segunda comparación es informativa y no se puede concluir nada más allá de la semejanza de ambas métricas ya que no se está trabajando bajo los mismos conjuntos de datos.

Capítulo 7

Detección de bordes de galaxias

7.1. Introducción al problema

A lo largo de este capítulo se va a buscar una solución para un problema de detección de los bordes de las galaxias a partir de las observaciones del Telescopio Espacial Hubble. La detección del borde de las galaxias no es una tarea trivial, requiere de la interpretación de un experto de la materia para establecer dichos bordes o límites.

Para la resolución de este problema surgen dos caminos: realizar una segmentación de las imágenes sin enmascarar, de forma que la red genere un mapa de segmentación detectando y distinguiendo la galaxia central de sus galaxias vecinas, o realizar una regresión en la que la red prediga los parámetros que puedan trazar dichos bordes. A fin de especializarse y ganar una mayor destreza, se ha decidido optar por el camino de la regresión.

7.2. El conjunto de datos

En este caso, se va a utilizar un subconjunto del *dataset* del capítulo anterior. A modo de recordatorio, se está trabajando con galaxias distantes pertenecientes a los cinco campos del cartografiado *CANDELS*, realizado por el Telescopio Espacial Hubble.

Para este problema se van a analizar muestras fotográficas de galaxias que contengan un disco. Teniendo en cuenta la categorización morfológica propuesta en el paper de Huertas-Company et al. (2015) (ver Fragmento de código 6.1), estas imágenes pertenecerán a galaxias con morfologías de tipo `DISK`, `DISKIRR` y `DISKSPH`.

En esta ocasión, el conjunto de datos lo forman 1053 muestras de galaxias, cada una acompañada de una entrada en un fichero *csv*, en el que se especifican cinco valores necesarios para establecer los límites de la galaxia y, por tanto, determinar sus bordes. Este fichero

está dividido en seis columnas, de las cuales, la primera está etiquetada como `gal_id` y contiene los nombres de los ficheros de las imágenes. Estos nombres siguen una estructura de "`<morfologia>_<id_galaxia>_<campo_candels>`". La segunda columna, etiquetada como `q`, contiene la relación entre los semiejes de la galaxia, es decir, la división del semieje menor entre el semieje mayor. La tercera columna, nombrada como `pa`, corresponde al ángulo entre el eje de ordenadas ("y") y el semieje mayor de la galaxia. El rango de este ángulo oscila entre $-90°$ y $90°$. La cuarta columna, `r_edge`, corresponde a la longitud en píxeles del semieje mayor, por lo que, a partir de este valor y de su `q` correspondiente, se puede determinar la longitud del semieje menor. Las dos últimas columnas, etiquetadas como `xcH` e `ycH`, corresponden a la posición del centro de la galaxia, tanto en el eje "x" como en el "y", respectivamente.

Al igual que antes, por cada galaxia se cuenta con cuatro ficheros *fits* correspondientes a los filtros "V" (F606W), "I" (F814W), "J" (F125W) y "H" (F160W) que, a su vez, poseen un fichero de máscara correspondiente. Todos ellos poseen una dimensión de 200x200 píxeles.

7.3. Preprocesamiento de datos

El tratamiento que se le ha aplicado a las imágenes es el mismo que el aplicado en el capítulo anterior (ver Sección 6.3). Los ficheros *fits* correspondientes a los filtros "H", "J" e "I", han sido combinados para formar imágenes RGB. Al tratarse de una regresión sobre cada galaxia individual, las imágenes se han enmascarado a fin de descartar la mayor cantidad de información posible que sea ajena a la galaxia central (la que se desea analizar), para que la red se focalice en el objeto central y su aprendizaje no se vea entorpecido por información irrelevante. Tan solo hizo falta adaptar los ficheros utilizados para el preprocesamiento del capítulo anterior, especificando el directorio en el que se encontraban dichas imágenes y sus máscaras. Por todo ello, las imágenes utilizadas para este problema tienen el mismo aspecto que las imágenes utilizadas en el capítulo anterior (ver Figura 6.3), tratándose de imágenes RGB de tamaño 200x200 píxeles con galaxias enmascaradas.

7.4. Implementación de la solución

En cuanto a la estructura del *notebook* de la solución, se ha dividido en siete puntos y un primer punto inicial de instalación de fastbook y fastai, tal y como se explicó en el capítulo anterior (ver Fragmento de código 6.2). En el siguiente punto se establece el entorno, importando los módulos necesarios de fastai y fastook (ver Fragmento de código 6.3) y declarando los *paths* de los *inputs* (imágenes y *csv*) y los outputs (*csv* con los resultados del entrenamiento).

A continuación, en el punto número dos, utilizando *pandas*, una biblioteca de python dedicada al análisis y procesamiento de datos [68], se lee el *csv* con la información antes mencionada, necesaria para realizar la regresión. Estos valores del *csv* se han procesado para obtener los cinco valores sobre los que se ha decidido hacer la regresión: `pa` (ángulo de la galaxia), `xcH` (centro respecto al eje "x"), `ycH` (centro respecto al eje "y"), `majorAxis`

(longitud del semieje mayor) y `minorAxis` (longitud del semieje menor). Se ha optado por utilizar el semieje menor en lugar de la relación entre ejes (`q`) por la escala de dicha relación ya que sus valores están comprendidos entre el 0 y el 1, escala muy diferente al resto de valores. Esto podría entorpecer el rendimiento de la función de coste al detectar una muy baja diferencia entre los valores esperados de `q` y los predichos, aunque se tratase de su diferencia máxima ($MAE = 1$).

En el tercer punto se han definido las funciones propias, entre las que se encuentran los *getters* que se usarán en el *DataBlock* (`get_x` y `get_y`), la función de pérdida MAE con *label smoothing* y una función para obtener las métricas MAE para todo el conjunto de validación divididas por cada valor de la regresión, a fin de poder observar cuáles son los valores que peor se estiman. A esta última función se la ha denominado `show_results_stats()`. Además de estas funciones, se han creado algunas más para la visualización de los bordes de las galaxias.

Una tarea sumamente importante es la de visualizar los datos para comprobar que estos son correctos y que su interpretación, así como su comprensión, es adecuada. Para poder visualizar estos datos, se ha definido una función bautizada como `show_edge(x, y)`, donde, a partir de una imagen y sus cinco valores de regresión (ángulo, centro de la galaxia en el eje "x" e "y", semieje mayor y semieje menor, en dicho orden), se representa, utilizando la biblioteca *Matplotlib*, dicha imagen junto a una elipse en color rojo superpuesta sobre ella, representando el borde que se desea aprender a predecir. Bajo la imagen se ha colocado un texto que incluye estos 5 valores etiquetados.

Una vez definida esta función, se pensó que podría ser una buena idea definir una función alternativa a la función de fastai `show_results()`, donde, a partir de un objeto *Learner* instanciado y un *DataLoaders*, se pudiera mostrar tantas predicciones de las imágenes del conjunto de validación como se desease. Para ello se pensó en algo semejante a la función anterior, pero en este caso con dos subfiguras, una con el borde real y otra con el borde predicho por la red. Bajo esta segunda subfigura, junto a los valores etiquetados se incluye cuál ha sido el error MAE de cada valor en concreto. Sobre las dos subfiguras, y a modo de título, se ha representado cuál es el MAE total obtenido con la predicción de dicha galaxia. Esta función se ha definido como "`show_edge_results(learn, dls, n_samples=5, random=True)`". Como se puede ver en el último argumento, se permite elegir si las predicción que se muestran son de imágenes escogidas aleatoriamente o son las primeras del conjunto de validación. Esto podría ser útil para visualizar cómo mejora la precisión de la red sobre unos casos concretos.

Por último, se han definido dos variantes de la función anterior, una para obtener las mejores predicciones y otra para las peores, de forma que se puedan identificar correlaciones entre ambos grupos y así obtener conclusiones relativas a la precisión del modelo. Para ello, se obtienen los valores de las métricas por cada imagen del conjunto de validación, se ordenan en sentido ascendente o descendente según se desee y se representan las primeras predicciones de la lista (tantas como se quiera).

En el punto número cuatro, se construye el `DataBlock` y se muestra información relevante de este. Para su definición, se tuvo que decidir sobre que acompañaría al `ImageBlock` en el campo `blocks`. Para realizar esta regresión se valoraron múltiples opciones como separar los valores en `RegressionBlock` distintos para poder definir un rango de "y" personalizado para

cada uno. También se pensó en establecer cinco `PointBlock` con los que se pudiera trazar la elipse correspondiente al borde de la galaxia. Finalmente, tras investigar sobre el tema y realizar varias pruebas, se optó por un único `RegressionBlock` para los cinco valores de la regresión. Además de los `blocks`, se especificó el campo `splitter`, `batch_tfms` y `get_y` de igual forma que se hizo en la regresión del capítulo anterior (ver Fragmento de código 6.5). Algo que ha cambiado respecto al `DataBlock` del capítulo anterior, es la forma en la que se obtienen las imágenes. En este caso no se va a utilizar el argumento `get_items` puesto que la lista de todos los elementos procede del contenido del fichero *csv* mencionado anteriormente. Para obtener las imágenes a partir de los elementos de dicha lista, se ha especificado en el atributo `get_x` una función encargada de concatenar la ruta del directorio que contiene todas las imágenes con el nombre del fichero de la imagen de cada galaxia, más su extensión ("`.png`").

En el quinto punto se instancia el objeto `DataLoaders` y muestra información relativa a este. A su vez, se ejecuta un `show_batch` para comprobar que las entradas de datos son correctas. Para instanciar el `DataLoaders` se introduce como argumento la lista con toda la información que ha sido obtenida a partir del *csv*. Esta lista será utilizada por los *getters* del `DataBlock` para obtener tanto las imágenes como los valores de la regresión.

En el penúltimo punto, es donde se escoge la red que se va a utilizar para el entrenamiento, se declara el `Learner` y se pone a entrenar. Tras el entrenamiento se muestran los resultados para su posterior análisis.

Por último, al final de cada *notebook* hay una sección dedicada a *tests*, cuyo fin es comprobar que todas las mejoras y funciones que se vayan añadiendo funcionan tal y como deben hacerlo.

7.5. Función de coste y métrica

En este caso, la métrica que se ha escogido para medir la precisión del modelo es el Mean Absolute Error (MAE) o error absoluto medio en español (ver Ecuación 3.10). La razón principal por la que se ha escogido esta métrica ha sido su alta facilidad de interpretación, ya que al tratarse de la media de todas las diferencias entre los objetivos y las predicciones, resulta mucho más visual que otras métricas donde dicha diferencia experimenta otras operaciones, como por ejemplo RMSE. Por otro lado, y al contrario que en capítulo anterior, esta vez no se ha tomado ningún modelo existente como referencia que trabaje sobre la misma tarea, por lo que no hace falta estar sujeto a ninguna métrica en concreto ya que no se va a realizar ninguna comparación con otro modelo.

En cuanto a la función de coste, teniendo en cuenta las diversas opciones de funciones de coste aplicables a un problema de regresión que se explicaron en la Sección 3.4.1, se ha optado por una función MAE con *label smoothing* (aplicar pequeñas variaciones aleatorias sobre los valores objetivos). La razón de esta elección está basada en la distinta naturaleza y escala de los diversos valores de la regresión puesto que se ha preferido no penalizar aún más las diferencias más grandes entre los objetivos y las predicciones. El *label smoothing* se aplica sobre la función de pérdida porque los valores sobre los que se está haciendo la regresión no

son valores inmutables y fijos. Como ya se comentó anteriormente, estos valores proceden de una interpretación subjetiva de expertos de la materia, por lo que no poseen la verdad absoluta y pueden estar sujetos a variaciones.

7.6. Sistema base

Para establecer un sistema base del que partir, se ha seguido un enfoque muy parecido al del capítulo anterior. Se ha optado por un modelo preentrenado de ResNet, más concretamente la versión 34. Inicialmente, no se ha incluido ninguna transformación en el campo `batch_tfms` del DataBlock.

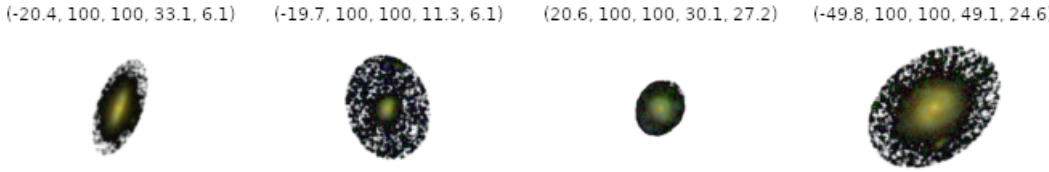


Figura 7.1: Resultado de `show_batch` sobre el sistema base

La Figura 7.1 muestra la salida de la función `show_batch()` a fin de verificar que tanto las imágenes como los valores de la regresión sobre las que va a trabajar la red son los correctos.

En cuanto al entrenamiento, se ha determinado un número de 50 épocas para su ejecución, ya que pruebas anteriores reflejaron la capacidad del modelo para seguir aprendiendo durante más épocas y que cada época tan solo requería una media de 6 segundo para computarse.

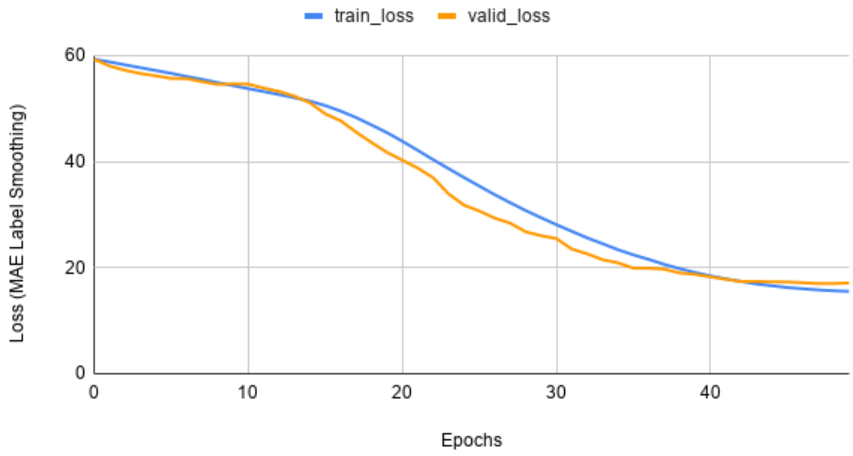


Figura 7.2: Pérdidas del conjunto de entrenamiento y validación del sistema base

Analizando las curvas de pérdida, tanto del conjunto de entrenamiento como del conjunto de validación (ver Figura 7.2), se puede observar cómo arranca con valores de MAE relativamente altos, en torno a 60. Al alcanzar la época número 15, ambas pendientes se pronuncian, de forma que se acelera el entrenamiento. Hacia la época 40, la curva vuelve a aplanarse hasta el final del entrenamiento. Esta baja inclinación de las curvas, hace que el entrenamiento sea muy lento y que la red no aprenda lo suficiente. En cuanto a la precisión, el modelo ha obtenido un MAE final de 17,12, aproximadamente. Se trata de una precisión ciertamente alta que tiene mucho que mejorar.

Gracias a la función `show_results_stats()` que se definió anteriormente, se puede conocer el error medio para los cinco distintos valores de la regresión.

Error medio por cada valor	
PA	12,85
xcH	17,71
ycH	20,41
majorAxis	16,78
minorAxis	13,52

Tabla 7.1: MAE de cada valor de las regresión del sistema base

Para este caso base (ver Tabla 7.1), el peor valor que la red predice es el del centro de la galaxia en el eje "y", con un error medio de 20,41, seguido del valor del centro de la galaxia en el eje "x". Sin embargo, las mejores predicciones se atribuyen al ángulo, con un MAE medio de 12,85.

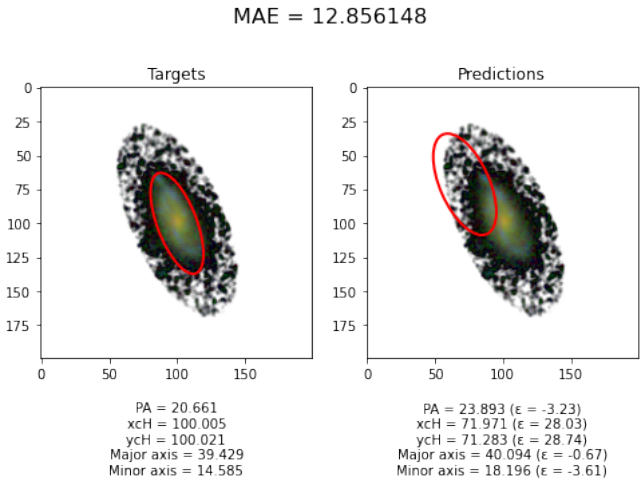


Figura 7.3: Resultado de `show_edge_results()` para el sistema base

En la Figura 7.3, se puede observar y confirmar, gracias a la función `show_edge_results()`, que realmente estima relativamente bien tanto el ángulo como el tamaño de los semiejes, sin embargo no consigue buenas predicciones en los valores relativos al centro de la galaxia, cuyos errores medios rondan valores de MAE de 28.

7.7. Modificación del learning rate

Para poder inclinar la curva del entrenamiento y así agilizar el aprendizaje, de forma que disminuya sus pérdidas y alcance mejores precisiones, se ha pensado en modificar el valor del hiperparámetro de *learning rate* que se le da a la función `fine_tune()` a *learning rate* base. Por defecto, el *learning rate* base está definido como 0,002 [20]. Para determinar el nuevo *learning rate*, se ha ejecutado la función `lr_find()` sobre el Learner del sistema base.

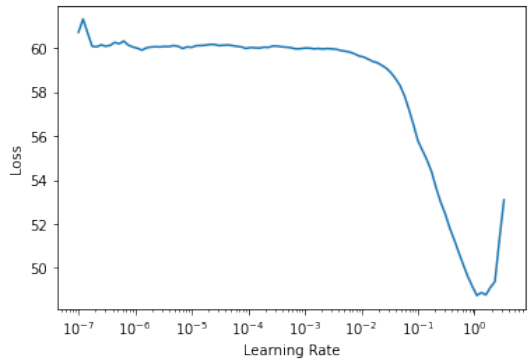


Figura 7.4: Resultado de la función `lr_find()` sobre el sistema base

La Figura 7.4 representa la gráfica que muestra la función `lr_find()`. Como se puede ver, hasta valores de learning rate de 10^{-2} la pérdida se encuentra estancada y no experimenta ninguna disminución. A partir de este valor, los valores de pérdida comienzan a descender hasta el learning rate de 10^{0} o 1. La función indica que el `lr_min` o valor de un orden inferior al valor de pérdida mínima, se encuentra en 1,10e-01 y el `lr_steep` o punto de mayor inclinación se ubica en 8,32e-02. A partir de estos valores, se ha optado por un *learning rate* de 1e-01 (0,1), un valor comprendido entre los dos valores que se recomiendan como *learning rates*.

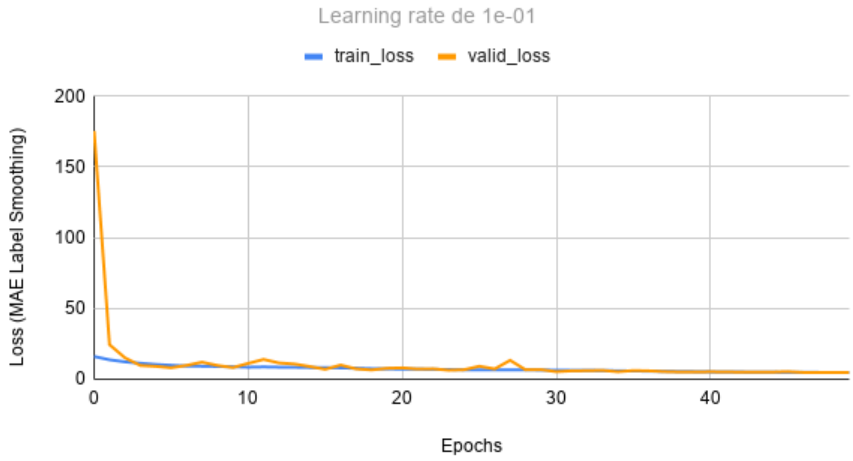


Figura 7.5: Pérdidas de entrenamiento y validación con el learning rate de 1e-01

Como se puede observar en la Figura 7.5, pese a que la primera época obtenga una pérdida para el conjunto de validación mucho más alta que el caso anterior, inmediatamente después, el entrenamiento se dirige hacia pérdidas mucho más bajas que el caso anterior. En tan solo 3 épocas se obtiene una precisión mejor que la obtenida en la última época del caso anterior, obteniendo un MAE de 15,18 frente al 17,12 del caso anterior. A partir de esta época, la precisión se posiciona en torno a 9, llegando alcanzar un MAE final aproximado de 4,82. Es necesario comentar que un *learning rate* tan alto también hace que las pérdidas obtenidas fluctúen más durante el entrenamiento que lo que se puede ver en el caso anterior.

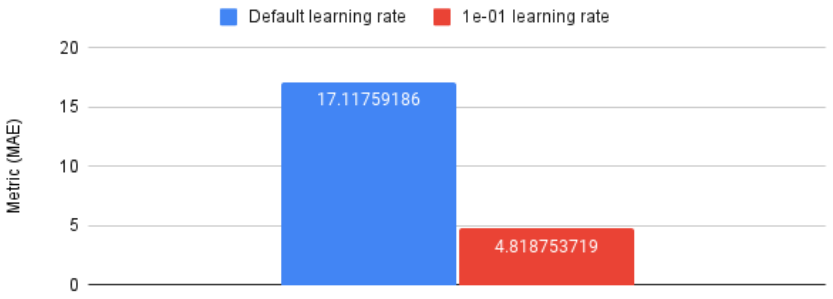


Figura 7.6: Métrica MAE del sistema base en función el learning rate

La Figura 7.6 hace una comparativa entre la precisión del modelo actual con un *learning rate* elegido a mano frente al caso anterior donde se ha dejado el *learning rate* por defecto. Como puede observarse, la diferencia es extremadamente grande.

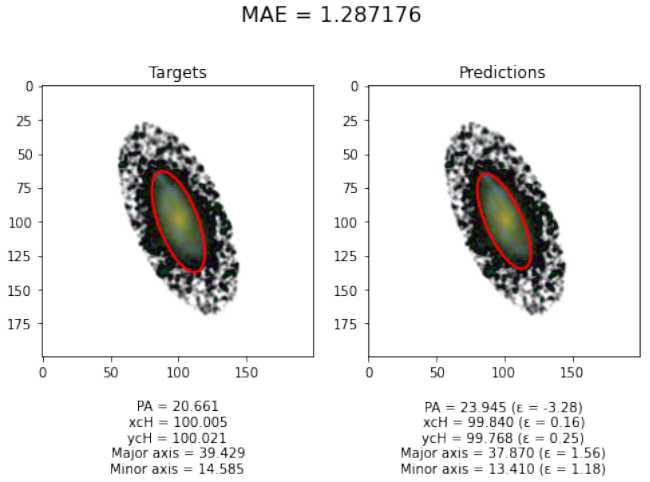


Figura 7.7: Resultado de `show_edge_results()` al aplicar un learning rate de 0,1

La imagen anterior (ver Figura 7.7) demuestra cómo ha mejorado la precisión de una forma visual, pudiendo comparar sobre una misma galaxia las predicción del modelo actual con respecto al del caso anterior (ver Figura 7.3). En esta ocasión puede verse una gran mejora en la predicción del centro de la galaxia.

Error medio por cada valor	
PA	1,31
xcH	0,91
ycH	1,49
majorAxis	2,31
minorAxis	2,29

Tabla 7.2: MAE de cada valor de la regresión al aplicar un learning rate de 0,1

Observando los errores relativos a cada valor (ver Tabla 7.2), se puede corroborar que los valores de la posición central de la galaxia han mejorado mucho, pasando de valores de MAE aproximados de 28 a valores en torno a 1.

Viendo la gran mejora que ha experimentado el modelo al utilizar dicho *learning rate*, se va a mantener dicho hiperparámetro para el resto de los experimentos, a fin de buscar la mejor precisión posible.

7.8. Transformaciones de datos

En cuanto al *data augmentation*, para este problema no se puede utilizar la transformación de la rotación tal y como se utilizó en el problema de clasificación morfológica ya que uno de los valores de la regresión representa el ángulo entre el eje "y" y el semieje mayor de la galaxia. Por tanto, rotar las imágenes haría que este ángulo se vea alterado y no coincida con el ángulo que se le ha suministrado a la red neuronal como objetivo de la regresión.

Una de las tareas que automatiza fastai es alterar los valores de las variables dependientes (variable "y") acorde a las transformaciones que se apliquen. Sin embargo, esto tan solo funciona en el caso de tratarse de `TransformBlock` de tipo `PointBlock`, `BBoxBlock`, `BBoxLblBlock` o `MaskBlock` [20]. En este caso, al tratarse de una regresión en la que fastai no posee información alguna sobre la naturaleza de cada valor del `RegressionBlock`, no puede determinar que al rotar la imagen debe alterar cierto valor de la regresión. Por ello, hay que abarcar este problema de forma manual [39, Cap. 6].

Sin embargo, la transformación del ruido Gaussiano se puede utilizar tal y como se definió en el capítulo anterior (ver Fragmento de código 6.9) ya que no afecta a ningún valor de la regresión. Esta transformación solo aplica una cierta alteración aleatoria sobre cada píxel de la imagen, de forma que la red crea que es una imagen totalmente nueva y aprenda en lugar de memorizar la combinación de los valores de los píxeles. A su vez, también es posible aplicar la transformación de la normalización de la entrada.

En primer lugar se va a probar cómo mejora el rendimiento del modelo aplicando estas dos transformaciones sobre cada *batch*. Para ello, se ha especificado en el argumento `batch_tfms` una lista con la transformación *Normalize* y *GaussianNoise*, en dicho orden. Con esta configuración se han conseguido los siguientes resultados.

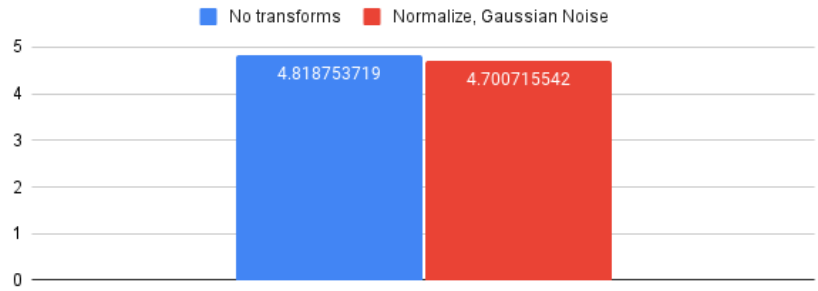


Figura 7.8: MAE en función de si se aplica o no `GaussianNoise` y `Normalize`

Tal y como se puede observar en el gráfico anterior (ver Figura 7.8), al aplicar ambas transformaciones, la precisión mejora ligeramente, pasando de un MAE de 4,82 a 4,7.

Además de estas, se ha pensado en aplicar otras dos transformaciones que podrían mejorar la capacidad de aprendizaje de la red. Una de ellas se encargará de aplicar un movimiento a la galaxia de forma que los valores correspondientes a su centro varíen en un cierto rango de píxeles y no sean en la mayoría de casos valores iguales a 100, ya que por lo general, las galaxias están ubicadas en el centro de la imagen 200x200. De esta forma se conseguiría evitar que la red se sobreajuste a valores de 100 en los valores correspondientes a la posición central de la galaxia. La otra transformación contemplada es la de aplicar a las imágenes una rotación en ángulos múltiplos de 90°, tal y como se hacía en el capítulo anterior. En esta ocasión, hay que redefinir la forma en la que se aplica dicha transformación para que modifique los valores objetivo de la regresión correspondientes a los ángulos, acorde a la rotación que se le aplique a cada galaxia. Para construir estas dos nuevas transformaciones, ha sido necesario implementar dos clases que heredan de la clase `Transform` de fastai que se explicó en la Sección 6.5.6 del capitulo anterior.

La clase que se ha implementado para la transformación del movimiento aleatorio de la imagen se ha nombrado como `MovementTfm`.

```
class MovementTfm(Transform):
    split_idx  =  0
    order = 102 #after normalize

    def __init__(self, min:int, max:int):
        super()
        self.min = min
        self.max = max

    def encodes(self, x:Tensor):
        if isinstance(x, TensorImage):
            """
            TensorImages
            """
```

```
            self.movements = torch.randint(
                low=self.min, high=self.max,
                size=(2, x.shape[0]),
                dtype=torch.int, device='cuda'
            )

            for img_idx in range(x.shape[0]):
                for ch_idx in range(x.shape[1]):
                    y_movement = self.movements[0][img_idx]
                    x_movement = self.movements[1][img_idx]

                    x[img_idx][ch_idx] = torch.roll(
                        x[img_idx][ch_idx],
                        shifts=(y_movement, x_movement),
                        dims=(0,1)
                    )

                    #Remove pixels that fall out of the tensor
                    if(x_movement >= 0):
                        x[img_idx][ch_idx][:,0:x_movement] = 0
                    else:
                        x[img_idx][ch_idx][:,x_movement:] = 0

                    if(y_movement >= 0):
                        x[img_idx][ch_idx][0:y_movement] = 0
                    else:
                        x[img_idx][ch_idx][y_movement:] = 0
        else:
            """
            Regression values
            """
            for img_idx in range(x.shape[0]):
                x[img_idx][1] += self.movements[0][img_idx] #X
                x[img_idx][2] += self.movements[1][img_idx] #Y

        return x
```

Listing 7.1: Transformación definida para aplicar movimiento aleatorio a las imágenes

En el Fragmento de código 7.1, se puede ver cómo se ha realizado esta implementación. A fin de conseguir una cierta flexibilidad de ajuste a la hora de instanciar la transformación, se ha redefinido el constructor `__init__` con dos nuevos argumentos que permiten especificar el valor mínimo y máximo de píxeles que se desea mover la imagen tanto en eje X como en el eje Y. La función `encodes` se ha definido de tal forma que el tipo de los *inputs* sea de tipo `Tensor`. Para distinguir entre las variables independientes (`ImageBlock`) y las dependientes (`RegressionBlock`), se comprueba si el tensor es instancia de la clase `TensorImage`. En caso afirmativo, se trataría del conjunto de imágenes correspondiente a un batch específico. En

caso contrario, la entrada correspondería al conjunto de tensores con los 5 valores objetivo de la regresión. Al determinar que se trata de un conjunto de tensores `TensorImage`, se genera una sucesión de números aleatorios correspondientes a la cantidad de píxeles que se van a desplazar cada una de las imágenes del *batch*, tanto en eje X como en el Y, dentro del rango de valores que se ha especificado en el constructor al instanciar la transformación. Para simular el desplazamiento, se ha hecho uso de la función `torch.roll()`, la cual permite desplazar tantas filas y columnas como se desee, así como el sentido en el que debe hacer, es decir, en sentido ascendente o descendente en el caso de las filas o desplazamientos hacia la izquierda o la derecha en cuanto a las columnas [73]. Los valores que sobrepasan los límites del tensor, se introducen en el extremo contrario. Esto es algo perjudicial para lo que se quiere conseguir puesto que podría haber píxeles correspondientes a la galaxia que apareciesen sin ningún sentido en el lado contrario de la imagen, lo que entorpecería el aprendizaje. Por ello, se han sustituido todos aquellos valores que se reinsertan en el extremo contrario del tensor por valores igual a cero. Tras aplicar las transformaciones sobre los `TensorImage`, se volverá a hacer una llamada a la función `encodes`, esta vez con los tensores de los valores objetivos de la regresión. De acuerdo a los valores generados de forma aleatoria, se editan los valores correspondientes a la posición central de la galaxia (`xcH` y `ycH`), haciendo corresponder estos con el nuevo centro de la galaxia.

Además del atributo `order` que se ha definido a 101, existe `split_idx`. En este caso se ha definido un valor de 0, lo que significa que solo debe aplicarse esta transformación sobre el conjunto de entrenamiento. En caso de indicar un 1 en este campo, las transformaciones solo se aplicarán en el conjunto de validación. Si no se especifica nada, se aplicarán sobre ambos conjuntos [20]. La razón por la que se ha especificado que las transformaciones no se apliquen en el conjunto de validación es para poder comparar realmente la efectividad del uso de cada transformación sobre exactamente las mismas imágenes de validación, sin aplicarles ninguna modificación.

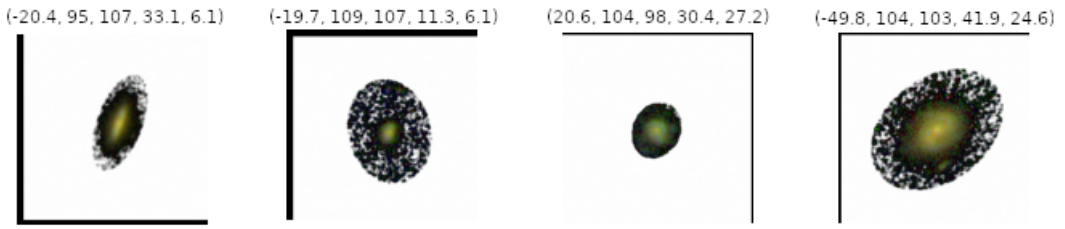


Figura 7.9: Resultado de `show_batch` aplicando la transformación `MovementTfm`

La Figura 7.9 muestra el resultado de ejecutar la función `show_batch()` sobre el objeto `DataLoaders` donde se aplica la transformación `MovementTfm`. Como puede verse en la primera imagen, el `xcH` toma un valor de 107, es decir, está desplazado la imagen hacia la derecha en 7 píxeles respecto al punto central de la imagen (100). Las franjas negras que aparecen en el lado izquierdo de la imagen corresponden a este desplazamiento horizontal. De igual forma se desplazan verticalmente. Es necesario comentar que el origen de coordenadas de las imágenes está localizado en la esquina superior izquierda, por lo que un valor de `ycH` inferior a 100, como es del primer caso, supondrá un desplazamiento vertical hacia la parte superior de la imagen y, por tanto, la aparición de las franjas negras en la parte inferior de la misma.

Para la transformación de las rotaciones aleatorias en múltiplos de 90° se ha creado la clase `RotateTfm` con el siguiente código:

```
class RotateTfm(Transform):
    split_idx  =  0 # Only apply to the training set
    order = 101

    def encodes(self, x:Tensor):
        if isinstance(x, TensorImage):
            """
            TensorImages
            """
            # Generate random values between 0 and 3
            # 0: 0.0, 1: 90.0, 2: 180.0, 3: 270.0
            self.rotation = torch.randint(
                low=0, high=4, size=(x.shape[0],),
                dtype=torch.int, device='cuda'
            )

            for img_idx in range(x.shape[0]):
                for ch_idx in range(x.shape[1]):
                    for r in range(self.rotation[img_idx]):
                        x[img_idx][ch_idx] = torch.rot90(
                            x[img_idx][ch_idx], 1, [0, 1]
                        )
        else:
            """
            Regression values
            """
            for img_idx in range(x.shape[0]):
                if self.rotation[img_idx]%2 != 0:
                    x[img_idx][0] = (
                        x[img_idx][0] + 90
                        if x[img_idx][0] < 0
                        else x[img_idx][0] - 90
                    )
        return x
```

Listing 7.2: Transformación definida para aplicar rotación aleatorio múltiplo de 90°

Definir esta transformación ha sido un proceso muy semejante al anterior. En este caso, en lugar de generar aleatoriamente los píxeles que se va a desplazar la imagen en el eje X y en el eje Y, se van a generar aleatoriamente números enteros comprendidos entre el 0 y el 3, de forma que el 0 corresponda a una rotación de 0°, 1 a una de 90°, 2 a 180° y 3 a 270° de rotación. Para realizar las rotaciones sobre las imágenes, se ha utilizado la función `torch.rot90()` sobre cada canal RGB. Esta función aplica rotaciones de 90° en sentido antihorario [73], por lo que se ejecutará sobre un mismo canal, tantas veces como sea necesario para alcanzar el ángulo correspondiente.

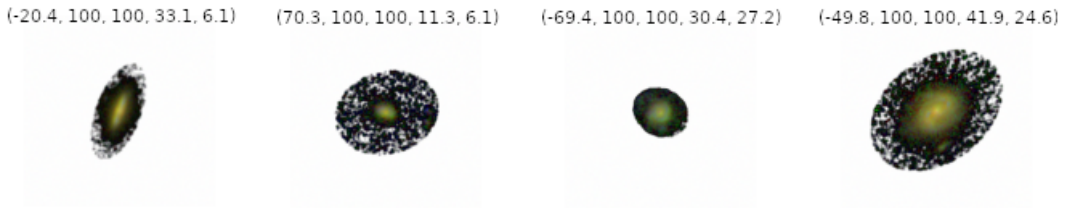


Figura 7.10: Resultado de `show_batch` aplicando la transformación `RotateTfm`

En la Figura 7.10 se muestra la salida de la función `show_batch` sobre el `DataLoaders` que cuenta con esta nueva transformación. Si se compara con la ejecución del `show_batch()` del sistema base (ver Figura 7.1), se puede observar cómo las imágenes han sufrido una rotación en sentido antihorario de 180°, 90°, 270° y 0°, respectivamente.

Junto a estas transformaciones, se han incluido unas pruebas en la sección siete de los notebooks para comprobar que estas transformaciones solo se apliquen sobre el conjunto de datos de entrenamiento, dejando el conjunto de validación sin transformar. De esta forma se comprueba que el atributo `split_idx = 0`, definido en ambas clases, funciona correctamente. Para poder comprobarlo, se ha definido una función encargada de obtener del `DataLoader` de validación todas las imágenes y sus valores de regresión, de forma que si se estuviese aplicando algún tipo de transformación sobre estas, sería apreciable en las muestras cargadas en el `DataLoader`.

Se ha decidido incluir esta nueva función en el `show_edge_results` para obtener tanto las imágenes y sus valores de regresión como sus predicciones asociadas. De esta forma, si posteriormente se decide aplicar transformaciones sobre el conjunto de validación, la función `show_edge_results` contemplaría esta posibilidad, representado las imágenes acorde a las transformaciones aplicadas.

Para comprobar el comportamiento del modelo aplicando las distintas transformaciones, se va a entrenar por separado la transformación `RotateTfm` y `MovementTfm`, ambas junto al ruido Gaussiando y la normalización. Estas a su vez, se van a comprar con el rendimiento que obtiene el modelo al aplicar las cuatro transformaciones juntas en el siguiente orden: `Normalize`, `GaussianNoise`, `RotateTfm` y `MovementTfm`.

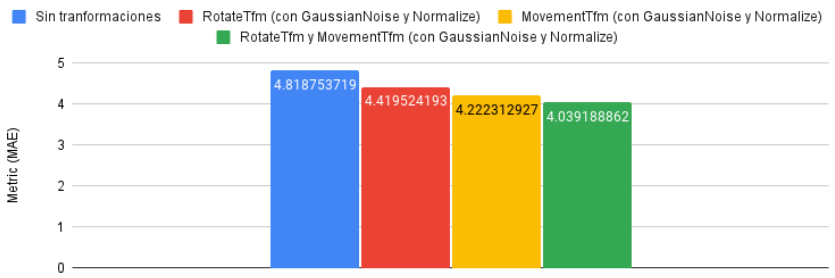


Figura 7.11: Métricas MAE según las transformaciones aplicadas

Analizando el gráfico de barras obtenido a partir de los diversos entrenamientos (ver Figura 7.11), se puede observar cómo al aplicar la transformación `RotateTfm` junto al ruido Gaussiano y la normalización de la entrada, la precisión del modelo mejora en casi 0,3 respecto al uso único del ruido Gaussiano y la normalización. Sin embargo, la transformación `MovementTfm`, alcanza una mejora de MAE de casi 0,5, con respecto al uso único del ruido Gaussiano y normalización. Observando la precisión alcanzada con el uso de las cuatro transformaciones juntas, la precisión mejora todavía más, como era de esperar. En este caso se ha alcanzado un valor de MAE de tan solo 4,04.

Analizando los tiempos necesitados para llevar a cabo el entrenamiento, el caso en el que se aplicó la transformación de rotación requirió de 7 segundos por época. El entrenamiento en el que se aplicó la transformación del movimiento hizo una media aproximada de 7,94 segundos por época. Aplicar las cuatro transformaciones ha implicado un entrenamiento en el que cada época requirió de 8,12 segundos de media, aproximadamente.

7.9. Equivalencia del ángulo opuesto de la galaxia

Pese a que la precisión del modelo ha alcanzado un valor de MAE muy bajo, si se pone atención a los peores casos, se detectarán una serie de anomalías.

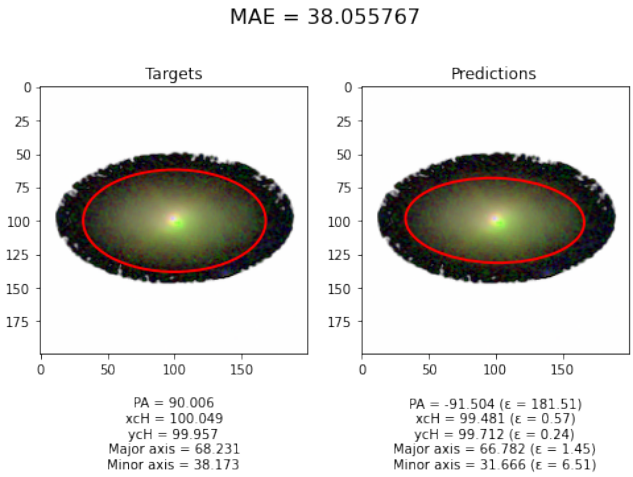


Figura 7.12: Peor predicción del modelo

La Figura 7.12 muestra el peor resultado obtenido por el modelo anterior en el que se aplican las tres transformaciones descritas. A simple vista, la elipse predicha parece extremadamente similar a la esperada. Pese al gran parecido, se le atribuye un MAE de 38,01 a las predicciones realizadas por el modelo para este caso en concreto. Si se observan los errores obtenidos por cada valor de la regresión, se verá que para los valores relativos al centro de la galaxia (`xcH` y `ycH`) poseen un error que casi no supera el medio píxel y para los valores relativos a la longitud de los semiejes (`majorAxis` y `minorAxis`), pese a ser errores

algo mayores, estos son minúsculos. Sin embargo, el primer valor (`pa`), relativo al ángulo de la galaxia tiene asociado un error de 181,5. Esta anomalía se debe a que la red neuronal está prediciendo el ángulo opuesto al dado, es decir, el ángulo $PA \pm 180°$, siendo este igual de válido para el objetivo del problema.

Para solucionar estos casos particulares, se ha modificado tanto la métrica como la función de coste para que los ángulos opuestos, al tratarse de ángulos equivalentes a los dados, posean errores y pérdidas nulas. De esta forma, se fuerza a que el error máximo del ángulo de la galaxia sea de tan solo 90°.

La implementación de esta modificación se basa en calcular la diferencia que existe entre el ángulo predicho y los ángulos válidos, es decir, PA, $PA + 180°$ y $PA - 180°$. Por último, a partir del resultado de estas tres diferencias, se conserva la menor de ellas utilizando la función de PyTorch `torch.min(tensor, tensor)` [73].

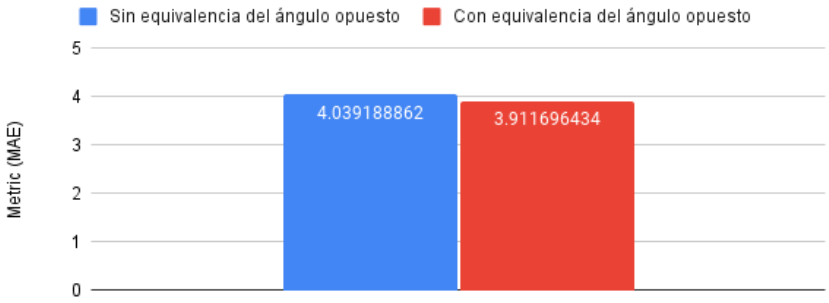


Figura 7.13: Métrica MAE según si se acepta como válido el ángulo opuesto

Como puede verse en el gráfico de barras de la Figura 7.14, al aplicar esta nueva métrica y función de pérdida, la precisión del modelo ha conseguido bajar de 4, obteniendo un MAE final de 3,91.

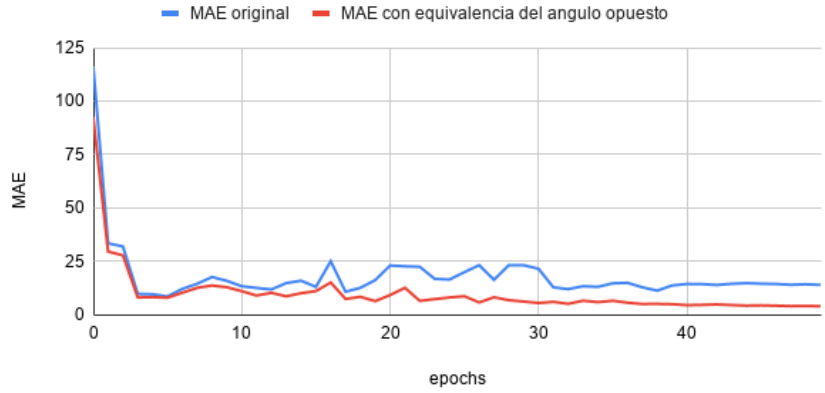


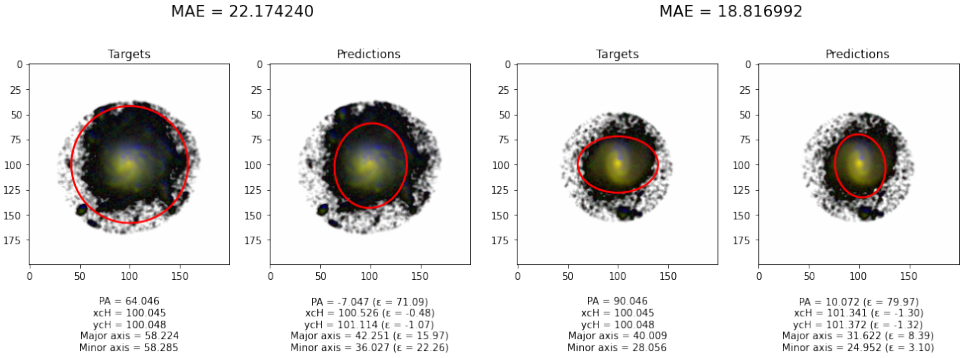
Figura 7.14: Métrica MAE original frente a la nueva modificación

Según puede verse en la Figura 7.14, el valor del MAE original (13,88) dista bastante del valor del MAE en el que se contempla la equivalencia del ángulo opuesto (3,91). Esto se debe a que la nueva función de coste ha dejado de castigar al modelo por predecir el ángulo opuesto, de forma que la red se ha ajustado a esta nueva condición.

Error medio por cada valor	
PA	1,14
xcH	0,90
ycH	1,51
majorAxis	3,20
minorAxis	1,72

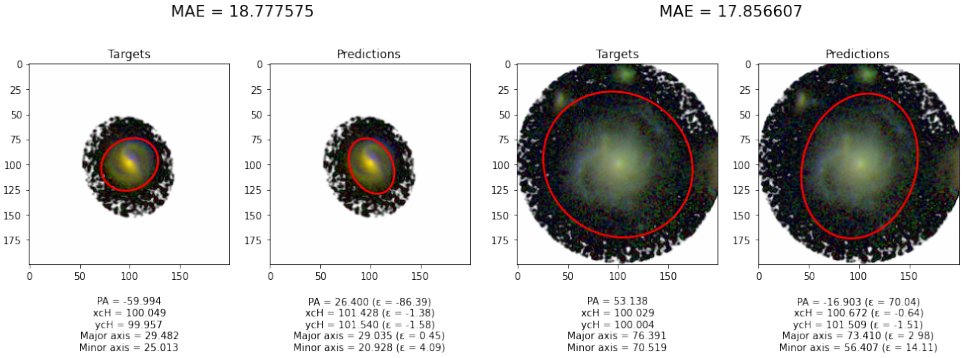
Tabla 7.3: MAE de cada valor al aceptar los ángulos opuestos como válidos

Poniéndole atención a la respuesta que devuelve la función `show_results_stats()` (ver Tabla 7.3), el error medio que se atribuye a los ángulos ha bajado de un 1,53, obtenido en el caso anterior, a un 1,14 actual, casi 0,4 menos.



(a) Primera peor predicción

(b) Segunda peor predicción

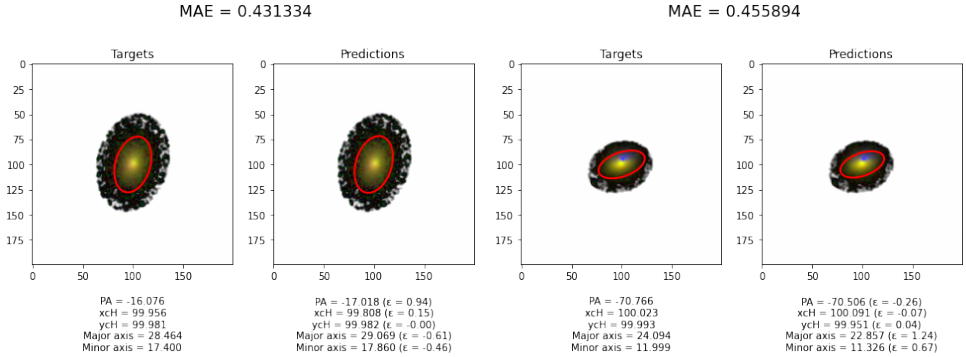


(c) Tercera peor predicción

(d) Cuarta peor predicción

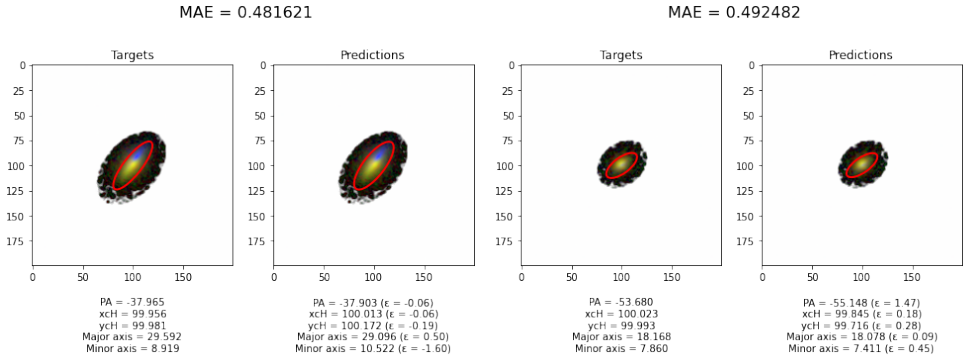
Figura 7.15: Cuatro peores predicciones del modelo

Prestando atención a los nuevos peores resultados obtenidos con el modelo actual, se puede observar cómo ya no se da ninguna anomalía en torno al ángulo de la galaxia. En base a estos casos, se puede llegar a la conclusión de que el modelo predice peor las galaxias con formas más circulares. Es lógico pensar que las galaxias con formas más redondeadas, es decir, aquellas cuya relación entre ejes es próxima a uno y por tanto la diferencia entre el semieje mayor y el menor es muy pequeña, es muy complejo determinar sus ángulos al no poseer una forma elíptica muy definida. Podría ser una buena opción, buscar cómo penalizar las predicciones de los ángulos en función de la relación entre los semiejes, es decir, penalizar en menor medida las predicciones de los ángulos de galaxias cuyas formas sean más circulares.



(a) Primera mejor predicción

(b) Segunda mejor predicción



(c) Tercera mejor predicción

(d) Cuarta mejor predicción

Figura 7.16: Cuatro mejores predicciones del modelo

Por el contrario, observando los mejores resultados (ver Figura 7.16) se puede corroborar que con galaxias con formas elípticas bien definidas, la red predice realmente bien sus bordes, consiguiendo en estas los valores de MAE más bajos.

7.10. Redes neuronales convolucionales propias

En el capítulo anterior se hicieron pruebas con las distintas versiones de arquitecturas de redes ResNet. En este caso se ha querido implementar una red neuronal convolucional propia, a fin de aprender cómo se define una arquitectura de este tipo y así asimilar de una forma práctica y visual el funcionamiento de este tipo de redes.

Partiendo de todos los conocimientos explicados en la sección de redes neuronales convolucionales (ver Sección 3.3.3), al crear una red de este tipo, hay que tener en cuenta que al aplicar convoluciones stride-2, es muy común aumentar la cantidad de características para no reducir la capacidad de la capa, ya que se estaría reduciendo la cantidad de activaciones del mapa de características. Si se mantuviese constante el número de canales de salida en todas las capas convolucionales, el número de cálculos que ha de realizar la red se iría reduciendo según se avance en profundidad por la misma, ya que el número de mapas de características sería siempre el mismo pero sus dimensiones se irían reduciendo. Teniendo esto en cuenta y que a medida que se avanza hacia las capas más profundas, las características y patrones que se obtienen son de mayor valor semántico, es muy importante incrementar el número de canales de salida según se profundice en la red.

Para entender esto, existe el *receptive field* o campo receptivo en español. Este término hace referencia al área de píxeles de una imagen que participa en la activación de una capa. Podría entenderse como el número de píxeles de la imagen de entrada que representa cada píxel de la capa actual.

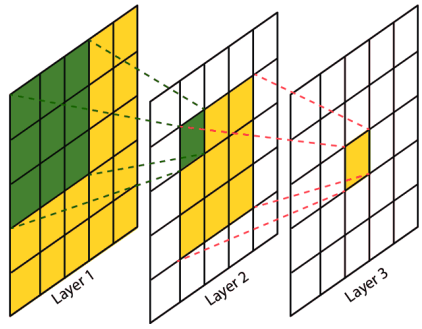


Figura 7.17: Campo receptivo de una CNN [77]

En la imagen anterior (ver Figura 7.17), entendiendo que las matrices equivalen a la salida de cada capa que se indica bajo ellas y que la capa uno muestra la imagen original de entrada, la capa número dos, tendría un campo receptivo de 3x3, ya que para obtener cada píxel del mapa de características de dicha capa, intervienen bloques de 3x3 píxeles de la imagen de entrada. En la capa número tres, su único píxel se calcula a partir del bloque 3x3 de la capa anterior, que se obtiene a partir de todos los píxeles de la imagen de entrada. Por esta razón, la última capa tendrá un campo receptivo de 5x5.

Por tanto, a mayor campo receptivo, mayor número de píxeles de la imagen de entrada entran en juego a la hora de calcular un solo píxel de la capa actual. Por ello, es sumamente

importante corresponder a tal nivel de complejidad que representa cada píxel con más cálculos. Podría verse como versiones o patrones distintos que se deben obtener del mismo grupo de píxeles de la imagen original para obtener más información [39, Cap. 13].

Siguiendo las explicaciones del capítulo dedicado a las Redes Neuronales Convolucionales del libro de fastai [39, Cap. 13], se va a estudiar el comportamiento de cuatro arquitecturas distintas de redes neuronales convolucionales. Todas ellas van a utilizar el mismo bloque de convolución, aquello que se aplicará en cada convolución. Dicho bloque está definido en la función "`conv(ni,nf,ks=3,act=True)`", tal y como puede verse en el Fragmento de código 7.3. El parámetro `ni` hace referencia al número de canales de entrada, `nf` al número de mapas de características de salida, `ks` al tamaño del kernel que se va a aplicar en la convolución y `act` representa un booleano que determinará si se aplica una activación ReLU a la salida de la convolución o no.

```
def conv(ni, nf, ks=3, act=True):
    layers = [nn.Conv2d(ni, nf, stride=2, kernel_size=ks,
   padding=ks//2)]
    if act: layers.append(nn.ReLU())
    layers.append(nn.BatchNorm2d(nf))
    return nn.Sequential(*layers)
```

Listing 7.3: Función `conv` que define lo que se usará en cada convolución [39]

Según se describe en el Fragmento de código 7.3, el bloque convolucional definido está compuesto por una capa convolucional `Conv2d` con stride-2 y padding de ks//2, siendo ks el tamaño del kernel, sucedida por una función de activación ReLU (salvo que se indique lo contrario) y una normalización de batch.

La primera arquitectura de red que se va a probar, está formada por la concatenación de ocho bloques de convolución, de forma que los mapas de características se vean reducidos a una dimensión de 1x1 para las entradas de 200x200 píxeles. Si se ajusta esta última capa para que su tensor de salida tenga dimensiones de BSx5x1x1 (siendo BS el tamaño del batch), utilizando el módulo `Flatten()`, se redimensiona ese tensor, de forma que la red emita un tensor de salida de tamaño BSx5.

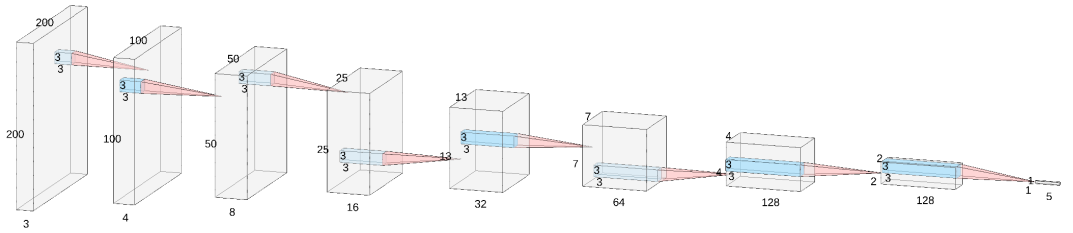


Figura 7.18: Red neuronal convolucional solo con capas convolucionales

Fuente: elaboración propia a través de `http://alexlenail.me/NN-SVG`.

La segunda arquitectura que se va a probar está basada en la red del paper de Huertas-Company et al. (2015), que fue descrita en la Sección 6.1 del capítulo anterior. Esta red utiliza seis bloques de convolución que reducen el tamaño de los mapas de características hasta un tamaño 4x4. Entonces, aplanando el tensor con `Flatten()`, lo introducen a una red *fully-connected* compuesta por dos capas de 2048 neuronas que finalmente desembocan en una tercera capa de salida con cinco neuronas, tal y como se muestra en la Figura 7.19.

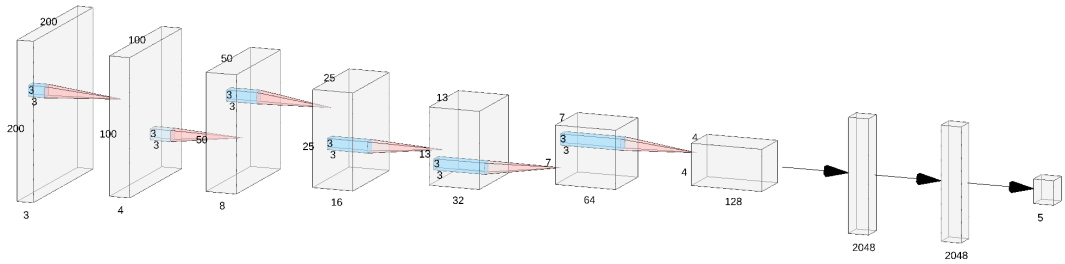


Figura 7.19: Red neuronal convolucional estilo Huertas-Company et al. (2015) [41]

Fuente: elaboración propia a través de `http://alexlenail.me/NN-SVG`.

La tercera red consistirá en una combinación de las dos primera, es decir, se va a reducir los mapas de características hasta un tamaño de 1x1 y, entonces, tras aplicar `Flatten()`, se introducirá el tensor a una red *fully-connected*, compuesta por dos capas de 128 neuronas que desembocarán en una última capa de salida con cinco neuronas, tal y como se ha representado en la Figura 7.19. Reducir los mapas de características hasta una dimensión de 1x1 implica el incremento en dos nuevas capas de convolución con respecto a la arquitectura anterior. Resumiendo, esta arquitectura está compuesta por 8 bloques convolucionales y 3 capas *fully-connected*. De esta forma, se experimentará sobre cómo afecta la profundidad de la parte convolucional de la red sobre su rendimiento.

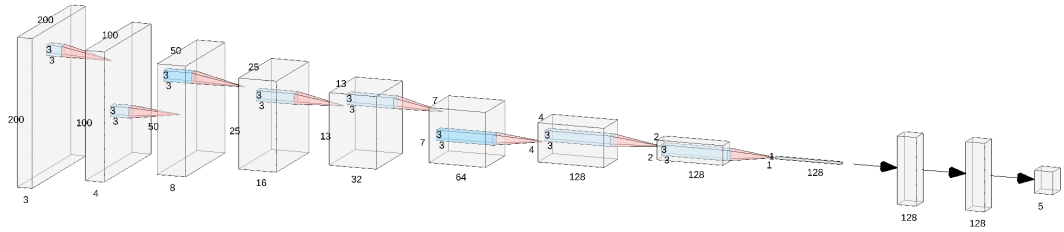


Figura 7.20: CNN con ocho capas convolucionales y tres *fully-connected*

Fuente: elaboración propia a través de `http://alexlenail.me/NN-SVG`.

Por último, para probar cómo afecta la profundidad de la parte *fully-connected*, se ha creado una nueva arquitectura que incrementa la parte *fully-connected* de la red anterior con una capa de 128 neuronas adicional. El siguiente código (ver Fragmento de código 7.4) muestra cómo se ha definido dicha arquitectura utilizando el bloque convolucional descrito anteriormente. Se puede observar cómo el código incluye unos comentarios después de añadir

cada bloque convolucional. Estos comentarios reflejan el tamaño de los mapas de características que se obtendrán como resultado de aplicar dicha capa, partiendo de imágenes de tamaño 200x200 (tamaño de las imágenes con las que se está trabajando en este proyecto). Para determinar estos tamaños, tan solo ha sido necesario aplicar la fórmula descrita en la sección de redes convolucionales del marco teórico de redes neuronales (ver Ecuación 3.7).

```
def simple_cnn():
    return sequential(              # <- 200x200
        conv(3 ,4),                 #100x100
        conv(4 ,8),                 #50x50
        conv(8 ,16),                #25x25
        conv(16,32),                #13x13
        conv(32,64),                #7x7
        conv(64,128),               #4x4
        conv(128,128),              #2x2
        conv(128,128),              #1x1
        Flatten(),
        nn.Linear(128,128),
        nn.Linear(128,128),
        nn.Linear(128,128),
        nn.Linear(128, 5)
    )
```

Listing 7.4: CNN con ocho capas convolucionales y cuatro *fully-connected*

Para comparar estas arquitecturas, se ha partido del sistema anterior, en el que se aplicaban las transformaciones descritas y se permitía el ángulo opuesto como ángulo válido. Bajo este escenario, se ha realizado sobre cada arquitectura un entrenamiento de 50 épocas con un learning rate de 0,1, como se lleva haciendo hasta el momento. Complementariamente, para comparar los resultados con un modelo ResNet, se ha realizado este entrenamiento sobre una resnet18.

La Figura 7.21 representa los valores de MAE obtenidos con las diversas arquitecturas de redes convolucionales, entrenadas bajo las mismas condiciones. La disposición de las redes sigue un orden de izquierda a derecha, siendo la arquitectura más a la izquierda la primera red que fue descrita anteriormente.

Como puede observarse, la primera red, la cual no poseía una parte *fully-connected*, ha obtenido los peores resultados con diferencia. El MAE obtenido por esta red ha sido de 43,58. La segunda arquitectura, la cual estaba inspirada en la red descrita en el paper de Huertas-Company et al. (2015), ha obtenido un MAE de 8,56. Esto demuestra la mejora que experimenta la red al incluir una una parte *fully-connected*. La tercera arquitectura, que en comparación con la anterior, se incrementa en dos el número de bloques convolucionales, ha mejorado la precisión, obteniendo un MAE de 5,08. Por último, la arquitectura que incrementa en una capa la parte *fully-connected* de la red anterior, mejora aún más la precisión, consiguiendo un MAE de tan solo 4,95. La precisión de esta última arquitectura está próxima a la del modelo preentrenado resnet18, cuyo MAE es de 4,21.

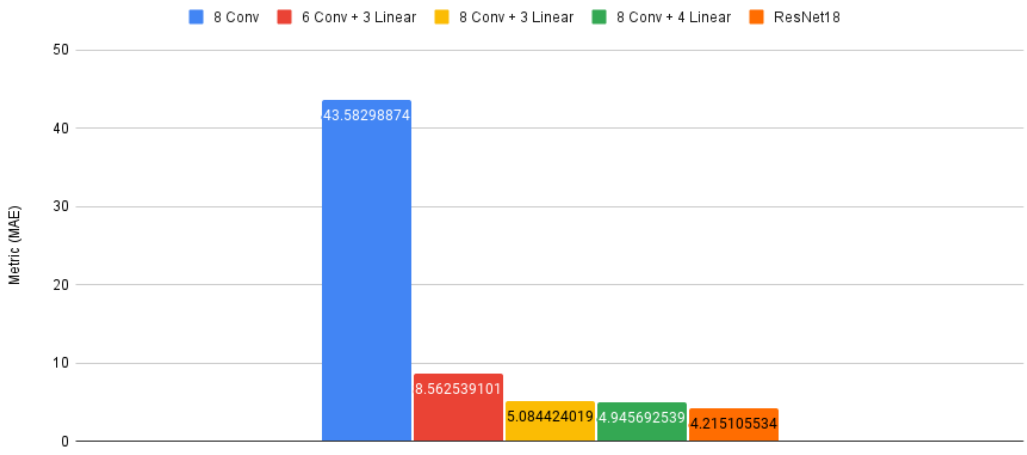


Figura 7.21: Métricas MAE de las distintas arquitecturas CNN

La conclusión a la que se puede llegar con esta experiencia es que una mayor profundidad de la red, por lo general, suele conseguir mejores precisiones.

Capítulo 8

Conclusiones

El desarrollo de este Trabajo de Fin de Grado ha supuesto un arduo camino de aprendizaje e investigación acerca del campo del *deep learning*. Sin embargo, este ha aportado mucho a mi conocimiento, siendo la primera experiencia en esta disciplina. Gracias al desarrollo del mismo, he podido comprender el gran alcance, potencial y utilidad del deep learning sobre muchas disciplinas y, especialmente, en la rama de visión por ordenador.

Desde un punto de vista tecnológico, la elección de usar fastai junto a PyTorch ha resultado una muy buena decisión debido principalmente a su accesibilidad y productividad, permitiendo a usuarios de toda clase, desde aquellos ajenos al mundo de las redes neuronales hasta los más experimentados, construir y entrenar modelos de *deep learning*, con muy pocas líneas de código, para resolver tareas específicas, consiguiendo muy buenos resultados.

El servicio de Google Colab ha resultado imprescindible para poder llevar a cabo este proyecto, consiguiendo un poder de cómputo alojado relativamente elevado de forma gratuita, que un hardware cotidiano no puede alcanzar.

Desde un punto de vista más técnico, los distintos errores derivados de la inexperiencia, han demostrado la gran importancia que tiene realizar un buen pre-procesamiento sobre las imágenes y una buena distribución de estas en los conjuntos de entrenamiento y validación, para poder obtener resultados robustos de los que poder sacar conclusiones veraces y lo más fieles a la realidad.

La técnica de regularización de *data augmentation* ha resultado sumamente útil para evitar que los modelos caigan en un overfitting y que su capacidad de aprendizaje aumente, alcanzando mejores precisiones. También se ha apreciado cómo al aplicar una política de entrenamiento de un ciclo, tal y como hace por defecto la función `fine_tune` de fastai, las precisiones que se consiguen son superiores y en un menor tiempo de entrenamiento. Además, se ha observado la importancia de escoger unos hiperparámetros, como el *learning rate* y el número de épocas, que sean adecuados.

En cuanto a las redes neuronales utilizadas, se ha podido llegar a la conclusión de que usar técnicas de *transfer learning* para aprovechar los modelos pre-entrenadas, son una gran opción para alcanzar mejores precisiones y en menores tiempos. Concretamente, los mejores resultados obtenidos a lo largo del proyecto han sido con la versión 50 de ResNet. Gracias a estos modelos pre-entrenados con el dataset ImageNet, es posible conseguir modelos de visión por ordenador muy preciso, aún contando con un dataset muy escaso.

Referido a las soluciones propuestas para los problemas, en la clasificación morfológica, utilizando el modelo de regresión, se obtuvo un mejor caso de RMSE 0,117. Por otro lado, el modelo de clasificación obtuvo un mejor caso de un 70 % de aciertos, aproximadamente. En la detección de bordes de galaxias, a partir de un modelo de regresión, se obtuvo un mejor caso de MAE 3,91. Teniendo en cuenta la baja resolución de las imágenes por la desmesurada distancia a la que se encuentran muchas de las galaxias y las distintas limitaciones encontradas en cada problema (por ejemplo la detención del ángulo de una galaxia redonda), los resultados revelan que las soluciones propuestas son bastante precisas y realizan suficientemente bien sus tareas.

Finalmente, con este trabajo se ha conseguido abordar y cumplir con éxito todos los objetivos propuestos y enumerados en la introducción del mismo.

8.1. Otras aplicaciones del Deep Learning en el campo de la astronomía

Existen multitud de problemas del campo de la astronomía que pueden ser abordados con técnicas de *Deep Learning*. Durante una de las reuniones, se describió un problema muy típico que consistía en la clasificación de los cuerpos celestes entre estrellas y galaxias.

Hay dos características claves que pueden ayudar a determinar si un elemento de una imagen es una estrella o una galaxia. Las estrellas de una imagen destacan por un brillo alto. En contraposición, las galaxias corresponden a elementos con altos valores en la longitud del semieje mayor.

La Figura 8.1 contiene una gráfica con todos los elementos del campo COSMOS del cartografiado CANDELS, cuyo eje horizontal representa la magnitud de apertura fija circular y el eje vertical la longitud del semieje mayor. Para obtener estos datos se ha utilizado SExtractor, en cuyo resultado viene descrita la apertura fija circular como `MAG_APER` y la longitud del semieje mayor como `A_IMAGE` [81]. Como puede analizarse en la gráfica, salen dos colas de puntos, una horizontal y otra vertical. Teniendo en cuenta que una menor apertura fija circular representa un cuerpo muy brillante, los puntos de la cola horizontal muy probablemente correspondan a estrellas. Por el contrario, los puntos de la cola vertical corresponderán a galaxias, tal y como se ha ilustrado sobre la imagen. Tras analizar estos dos factores, queda una nube de puntos indefinida. Esta nube corresponde a todos aquellos cuerpos que, debido a su distancia con respecto al Hubble, no pueden ser clasificados con tanta facilidad y es donde las técnicas de aprendizaje automático podrían realizar la clasificación que el sistema experto basado en estas normas no es capaz.

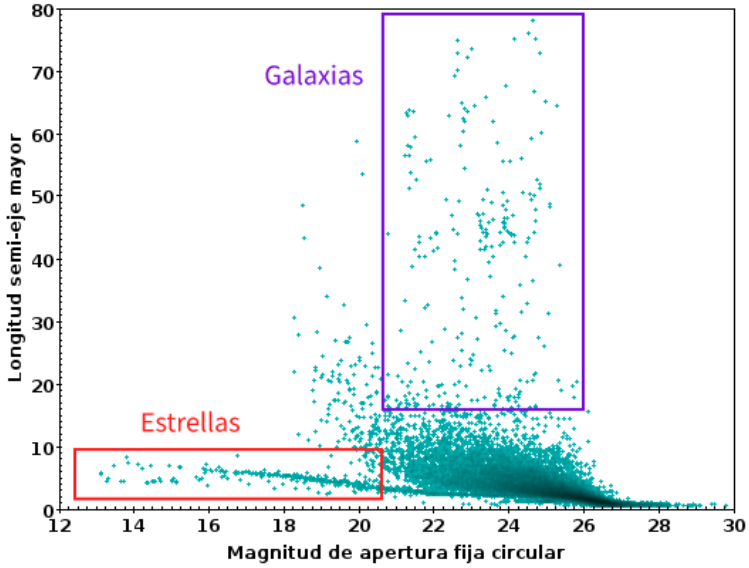


Figura 8.1: Estrellas frente a galaxias de manera analítica

Fuente: elaboración propia utilizando Topcat con los datos obtenidos con SExtractor para un campo COSMOS del cartografiado CANDELS.

8.2. Líneas de trabajo futuro

En cuanto a las líneas sobre las que se puede continuar este trabajo, en el problema de detección de bordes se podría modificar la función de coste y la métrica, de forma que a mayor relación entre los semiejes de la galaxia ($\frac{semieje\_menor}{semieje\_mayor}$), el coste del error de la predicción del ángulo de la galaxia se aminore, sin que esto despiste el aprendizaje de la red para el resto de valores de la regresión. De esta forma se conseguiría afinar la precisión a la hora de detectar los bordes de las galaxias cuya forma sea más circular, siendo estas los peores casos obtenidos por el modelo.

A su vez, pueden probarse otros planteamientos para abordar este último problema. Un ejemplo sería desarrollar un modelo que en lugar de predecir cinco valores, tenga que predecir cinco puntos en la imagen. Estas coordenadas de la imagen, representadas en fastai con la clase *PointBlock*, podrían ser cuatro puntos para los extremos de los ejes de las galaxias y un quinto punto para el centro de las mismas. De esta forma, el error del ángulo, no entorpecería el entrenamiento como si ocurre en el caso de la regresión.

La Figura 8.2 representa con cruces blancas los cinco puntos de la imagen que podrían determinar el borde de una galaxia.

También podría ser interesante probar este mismo problema, enfocándolo como una tarea de segmentación, en la que, partiendo de un conjunto de imágenes de galaxias sin enmascarar, se desarrollase un modelo capaz de generar un mapa de segmentación que detectase la galaxia central y sus vecinas, pudiendo hacer una distinción entre ambas.

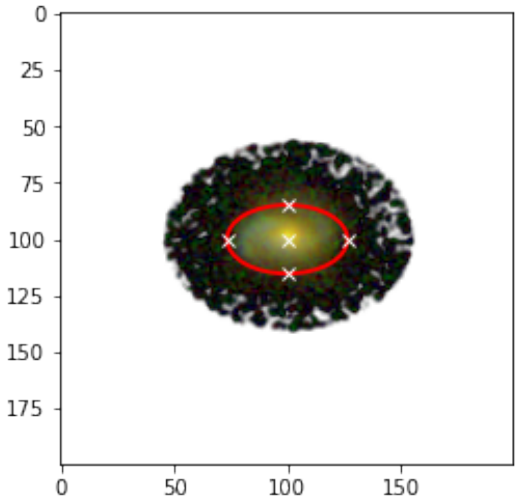


Figura 8.2: Puntos de una galaxia que podrían determinar su borde

Fuente: elaboración propia utilizando Matplotlib.

Apéndice A

Manual de instalación de software utilizado en Ubuntu

Para instalar el software **SAOImage DS9** en Ubuntu, se puede utilizar el sistema de gestión de paquetes del sistema operativo (APT, *Advanced Packaging Tool*). Para ello, basta con ejecutar el siguiente código en la terminal:

```
sudo apt update
sudo apt install saods9
```

Listing A.1: Instalación de SAOImage DS9 en Ubuntu

Como se ve en el fragmento de código A.1, antes de ejecutar el `apt install` se hace un `apt update`. Esto es necesario para actualizar la lista de paquetes que están disponibles, bien para su instalación o su actualización [29, Cap. 2].

La instalación de **Source Extractor** sigue el mismo procedimiento que la instalación de *SAOImage DS9*. Para ello, se va a ejecutar el siguiente código:

```
sudo apt update
sudo apt install sextractor
```

Listing A.2: Instalación de SExtractor en Ubuntu

Hay diversas formas de utilizar **Aladin** en el equipo local. En este caso, la opción optada ha sido la de descargar y ejecutar el fichero `.jar` que se puede encontrar en el siguiente enlace: `https://aladin.u-strasbg.fr/java/nph-aladin.pl?frame=downloading`. La ejecución de este fichero se realiza con el siguiente código:

```
java -jar Aladin.jar
```

Listing A.3: Ejecución de Aladin en Ubuntu

En el enlace anterior se pueden encontrar las instrucciones de instalación para otros sistemas operativos distintos a Linux.

El software **Topcat** se puede instalar utilizando el *apt* de Ubuntu, como los dos primeros softwares. Para ello habrá que ejecutar el siguiente código sobre la terminal:

```
sudo apt update
sudo apt install topcat
```

Listing A.4: Instalación de Topcat en Ubuntu

Para instalar las diversas bibliotecas externas de *Python* hay que utilizar uno de sus instaladores de paquetes. Principalmente existen `pip` y `conda`. Para este proyecto se ha usado `pip` en su versión número tres por estar familiarizado con su uso. Para instalar los *jupyter notebooks* con `pip`, habría que ejecutar el siguiente código en la terminal [71]:

```
pip3 install jupyter

jupyter-notebook    # Launch the Jupyter Notebook App
```

Listing A.5: Instalación y ejecución de Jupyter Notebook

De la misma forma que se han instalado los *jupyter notebooks* se podría instalar el resto de bibliotecas de *Python* que se han utilizado (Pandas, astropy, scipy, etc.). Por lo general, la mayoría de bibliotecas de *Python* se encuentran *PyPI*, un repositorio oficial de bibliotecas de terceros [23].

Apéndice B

Contenidos del soporte digital

Junto al presente documento se incluye un conjunto de material digital. Estos forman parte de todas las experimentaciones realizadas para llevar a cabo el proyecto y están estructurados en los siguientes tres directorios:

- `galaxy_edge_detection`: directorio con el material correspondiente al problema de detección de los bordes de las galaxias. Dentro de `src/regression` se pueden encontrar trece *jupyter notebooks*. En el nombre de los diversos *notebooks* se incluye el número de experimento y el número de caso de prueba correspondiente. A su vez, se incluye parte de la configuración del modelo y del entrenamiento para poder saber, de un simple vistazo, qué es lo que diferencia a ese cuaderno del resto. En el directorio `results` están todos los ficheros `.csv` con los resultados obtenidos de la ejecución de los entrenamientos de los *notebooks*. El nombre de estos ficheros siguen una estructura muy semejante a la de sus *notebooks*, incluyendo el número de experimento y caso de prueba al que pertenecen. Junto a los ficheros `.csv` se incluye un documento PDF con el resumen de los resultados y las gráficas obtenidas que se han presentado a lo largo del documento.
- `galaxy_morphological_classification`: este segundo directorio contiene todos los ficheros correspondientes al problema de clasificación morfológica de galaxias. Dentro del directorio `src/regression` se encuentran los diversos *notebooks* del enfoque de regresión junto a un subdirectorio nombrado como `results`, que contiene todos los ficheros `.csv` y el documento PDF del resumen de los resultados y las gráficas obtenidas a partir de estos. De igual forma el directorio `src/classification` contiene los cuadernos `.ipynb` junto al subdirectorio `results`, el cual contiene los resultados y el resumen de los mismos. Los nombres de los distintos cuadernos y sus resultados siguen la misma estructura que la descrita en el punto anterior.
- `gradient_descent_experiment`: contiene un único *jupyter notebook* con el experimento del algoritmo del descenso del gradiente descrito en la sección 3.4.2 del cuarto capítulo.

La estructura de los diversos *notebooks* depende de cada problema. Dicha estructura está descrita en las secciones correspondientes de este documento (secciones de implementación de

la solución). Sin embargo, todos ellos siguen una organización muy semejante y comparten que al inicio de los mismos se incluyen unas celdas descriptivas en las que se resume el experimento que se está realizando y sus datos de interés, como la configuración del modelo o parámetros del entrenamiento. Todo el texto y código de los *notebooks* están escritos en inglés, siendo esta una buena praxis de desarrollo para conseguir un código lo más universal posible.

Bibliografía

[1] Calculating the output size of convolutions and transpose convolutions. *Make your own neural network*, 2020. http://makeyourownneuralnetwork.blogspot.com/2020/02/calculating-output-size-of-convolutions.html. Accedido: 12/04/2021.

[2] Khan Academy. Función y estructura de la neurona. https://es.khanacademy.org/science/biology/human-biology/neuron-nervous-system/a/overview-of-neuron-structure-and-function. Accedido: 02/11/2020.

[3] Akira.AI. Stochastic gradient descent. https://www.akira.ai/glossary/stochastic-gradient-descent/. Accedido: 17/05/2021.

[4] Atlassian. Trello. https://trello.com/. Accedido: 05/06/2021.

[5] Kirk D. Borne. Astroinformatics: data-oriented astronomy research and education. *Earth Sci Inform*, 3, May 2010.

[6] Jason Brownlee. A gentle introduction to transfer learning for deep learning, 2017. https://machinelearningmastery.com/transfer-learning-for-deep-learning/. Accedido: 13/01/2021.

[7] Jason Brownlee. How to choose loss functions when training deep learning neural networks. 2019. https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/. Accedido: 20/01/2021.

[8] The Center for Astrophysics — Harvard & Smithsonian. *SAOImageDS9 Users Manual*. https://ds9.si.edu/doc/user/index.html. Accedido: 13/12/2020.

[9] The Center for Astrophysics — Harvard & Smithsonian. *SAOImageDS9 website*. https://sites.google.com/cfa.harvard.edu/saoimageds9. Accedido: 14/03/2021.

[10] Dan Ciresan, Ueli Meier, Jonathan Masci, Luca Maria Gambardella, and Jürgen Schmidhuber. Flexible, high performance convolutional neural networks for image classification. pages 1237–1242, 07 2011.

[11] Centre de Données astronomiques de Strasbourg. Aladin sky atlas. https://aladin.u-strasbg.fr/. Accedido: 17/03/2021.

[12] Paloma Recuero de los Santos. Tipos de aprendizaje en machine learning: supervisado y no supervisado. *Telefónica - Think Big*, Nov. 2017. https://empresas.blogthinkbig.com/que-algoritmo-elegir-en-ml-aprendizaje/. Accedido: 24/02/2021.

[13] Google Developers. Reducción de la pérdida: Descenso de gradiente estocástico. https://developers.google.com/machine-learning/crash-course/reducing-loss/stochastic-gradient-descent. Accedido: 28/12/2020.

[14] Google Developers. Reducción de la pérdida: Descenso de gradientes. https://developers.google.com/machine-learning/crash-course/reducing-loss/gradient-descent?hl=es-419. Accedido: 27/12/2020.

[15] Sander Dieleman, Kyle W. Willett, and Joni Dambre. Rotation-invariant convolutional neural networks for galaxy morphology prediction. *Monthly Notices of the Royal Astronomical Society*, 450(2):1441–1459, Apr 2015.

[16] Schuchen Du. Understanding deep self-attention mechanism in convolution neural networks, 2020. https://medium.com/ai-salon/understanding-deep-self-attention-mechanism-in-convolution-neural-networks-e8f9c01cb251. Accedido: 13/12/2020.

[17] María Luisa González Díaz. Léxico y sintaxis de los lenguajes de programación. In *Lenguajes de Programación, Grado de Ingeniería Informática*, chapter 2. 2020.

[18] ESA. Esa/hubble. https://esahubble.org/. Accedido: 17/05/2021.

[19] ESA. Hubble explores the origins of modern galaxies, 2013. https://esahubble.org/news/heic1315/. Acceido: 18/05/2021.

[20] Fastai. *Fastai documentation*. https://docs.fast.ai/. Accedido: 15/02/2021.

[21] fast.ai. Forums for fast.ai deep learning courses. https://forums.fast.ai/. Accedido: 20/03/2021.

[22] fast.ai website. fast.ai. https://www.fast.ai/. Accedido: 23/02/2021.

[23] Python Software Foundation. Pypi. https://pypi.org/. Accedido: 01/06/2021.

[24] GeeksforGeeks. Underfitting and overfitting in machine learning. 2020. https://www.geeksforgeeks.org/underfitting-and-overfitting-in-machine-learning/. Accedido: 14/11/2020.

[25] Rick O. Gilmore. Psych 260 - neuroanatomy iii. https://psu-psychology.github.io/psych-260-2020-fall/lectures/260-2020-09-10-anatomy-III.html. Accedido: 28/11/2020.

[26] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org. Accedido: 23/11/2020.

[27] Google. Google trends. https://trends.google.com/trends/. Accedido: 25/05/2021.

[28] NASA / GSFC. The fits support office. https://fits.gsfc.nasa.gov/. Accedido: 10/03/2021.

[29] Jason Gunthorpe. *Guía de usuario de APT*. Debian, 1998. https://www.debian.org/doc/manuals/apt-guide/. Accedido: 01/06/2021.

[30] Red Hat. El concepto de linux. `https://www.redhat.com/es/topics/linux`. Accedido: 25/05/2021.

[31] Ryan Hausen and Brant Robertson. Morpheus — morpheus documentation. `https://morpheus-astro.readthedocs.io/en/latest/`. Accedido: 04/04/2021.

[32] Ryan Hausen and Brant E. Robertson. Morpheus: A deep learning framework for the pixel-level analysis of astronomical image data. *The Astrophysical Journal Supplement Series*, 248(1):20, May 2020.

[33] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[34] Jose Martinez Heras. Gradiente descendiente para aprendizaje automático, 2020. `https://www.iartificial.net/gradiente-descendiente-para-aprendizaje-automatico/#more-1191`. Accedido: 16/12/2020.

[35] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors, 2012.

[36] Dr. Benne W. Holwerda. *Source Extractor for Dummies*. Space Telescope Science Institute, 5 edition, 12 2005.

[37] Jeremy Howard and Sylvain Gugger. *Fastbook*. `https://github.com/fastai/fastbook`. Accedido: 25/02/2021.

[38] Jeremy Howard and Sylvain Gugger. Practical deep learning for coders. `https://course.fast.ai/`. Accedido: 23/02/2021.

[39] Jeremy Howard and Sylvain Gugger. *Deep Learning for Coders with Fastai and Pytorch: AI Applications Without a PhD*. O'Reilly Media, Incorporated, 1 edition, 2020.

[40] Jeremy Howard and Sylvain Gugger. Fastai: A layered api for deep learning. *Information*, 11(2):108, Feb 2020.

[41] M. Huertas-Company, R. Gravet, G. Cabrera-Vives, P. G. Pérez-González, J. S. Kartaltepe, G. Barro, M. Bernardi, S. Mei, F. Shankar, P. Dimauro, and et al. A catalog of visual-like morphologies in the 5 CANDELS fields using deep-learning. *The Astrophysical Journal Supplement Series*, 221(1):8, Oct 2015.

[42] Bob Hughes and Mike Cotterell. *Software Project Management*. McGraw-Hill Education, 5th edition edition, 2009.

[43] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[44] Jeremy Jordan. Setting the learning rate of your neural network, Mar 2018. `https://www.jeremyjordan.me/nn-learning-rate/`. Accedido: 21/12/2020.

[45] Kaggle. What's kaggle? `https://www.kaggle.com/`. Accedido: 25/04/2021.

[46] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences, 2014.

[47] Jeyhan S. Kartaltepe, Mark Mozena, Dale Kocevski, Daniel H. McIntosh, Jennifer Lotz, Eric F. Bell, Sandy Faber, Harry Ferguson, David Koo, Robert Bassett, and et al. CANDELS visual classifications: Scheme, data release, and first results. *The Astrophysical Journal Supplement Series*, 221(1):11, Oct 2015.

[48] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[49] Anton M. Koekemoer, S. M. Faber, Henry C. Ferguson, Norman A. Grogin, Dale D. Kocevski, David C. Koo, Kamson Lai, Jennifer M. Lotz, Ray A. Lucas, Elizabeth J. McGrath, Sara Ogaz, Abhijith Rajan, Adam G. Riess, Steve A. Rodney, Louis Strolger, Stefano Casertano, Marco Castellano, Tomas Dahlen, Mark Dickinson, Timothy Dolch, Adriano Fontana, Mauro Giavalisco, Andrea Grazian, Yicheng Guo, Nimish P. Hathi, Kuang-Han Huang, Arjen van der Wel, Hao-Jing Yan, Viviana Acquaviva, David M. Alexander, Omar Almaini, Matthew L. N. Ashby, Marco Barden, Eric F. Bell, Frédéric Bournaud, Thomas M. Brown, Karina I. Caputi, Paolo Cassata, Peter J. Challis, Ranga-Ram Chary, Edmond Cheung, Michele Cirasuolo, Christopher J. Conselice, Asantha Roshan Cooray, Darren J. Croton, Emanuele Daddi, Romeel Davé, Duilia F. de Mello, Loic de Ravel, Avishai Dekel, Jennifer L. Donley, James S. Dunlop, Aaron A. Dutton, David Elbaz, Giovanni G. Fazio, Alexei V. Filippenko, Steven L. Finkelstein, Chris Frazer, Jonathan P. Gardner, Peter M. Garnavich, Eric Gawiser, Ruth Gruetzbauch, Will G. Hartley, Boris Häussler, Jessica Herrington, Philip F. Hopkins, Jia-Sheng Huang, Saurabh W. Jha, Andrew Johnson, Jeyhan S. Kartaltepe, Ali A. Khostovan, Robert P. Kirshner, Caterina Lani, Kyoung-Soo Lee, Weidong Li, Piero Madau, Patrick J. McCarthy, Daniel H. McIntosh, Ross J. McLure, Conor McPartland, Bahram Mobasher, Heidi Moreira, Alice Mortlock, Leonidas A. Moustakas, Mark Mozena, Kirpal Nandra, Jeffrey A. Newman, Jennifer L. Nielsen, Sami Niemi, Kai G. Noeske, Casey J. Papovich, Laura Pentericci, Alexandra Pope, Joel R. Primack, Swara Ravindranath, Naveen A. Reddy, Alvio Renzini, Hans-Walter Rix, Aday R. Robaina, David J. Rosario, Piero Rosati, Sara Salimbeni, Claudia Scarlata, Brian Siana, Luc Simard, Joseph Smidt, Diana Snyder, Rachel S. Somerville, Hyron Spinrad, Amber N. Straughn, Olivia Telford, Harry I. Teplitz, Jonathan R. Trump, Carlos Vargas, Carolin Villforth, Cory R. Wagner, Pat Wandro, Risa H. Wechsler, Benjamin J. Weiner, Tommy Wiklind, Vivienne Wild, Grant Wilson, Stijn Wuyts, and Min S. Yun. CANDELS: THE COSMIC ASSEMBLY NEAR-INFRARED DEEP EXTRAGALACTIC LEGACY SURVEY—THE HUBBLE SPACE TELESCOPE OBSERVATIONS, IMAGING DATA PRODUCTS, AND MOSAICS. *The Astrophysical Journal Supplement Series*, 197(2):36, dec 2011.

[50] Stanford Vision Lab. *ImageNet*. Stanford University. `https://www.image-net.org/`. Accedido: 13/05/2021.

[51] Sam Lau, Joey Gonzalez, and Deb Nolan. *Principles and Techniques of Data Science*. 2020. `https://www.textbook.ds100.org/`. Accedido: 28/11/2020.

[52] Carlos Marijuán López. *Fundamentos de Matemáticas, Grado en Ingeniería Informática, Estadísta e INdat*. 2017.

[53] Warren S. Mcculloch and Walter Pitts. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115—-133, 1943.

[54] Peter Melchior. The cluster lensing and supernova survey with hubble: An overview. https://www.researchgate.net/figure/Each-CLASH-cluster-is-observed-in-16-HST-filters-spanning-2000-17000A-with-WFC3-UVIS-in_fig1_231113522. Accedido: 19/05/2021.

[55] Marvin Minsky and Seymour Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, USA, 1969.

[56] Anshu Mishra. Convolutional neural networks-artificial neural network for computer vision. *Medium*, 2019. https://medium.datadriveninvestor.com/convolutional-neural-networks-artificial-neural-network-for-computer-vision-7ee553df5df9. Accedido: 12/04/2021.

[57] Abdelmalik Moujahid, Iñaki Inza, and Pedro Larrañaga. Redes neuronales. In *Métodos Matemáticos en Ciencias de la Computación*, chapter 14. http://www.sc.ehu.es/ccwbayes/docencia/mmcc/docs/t14-neuronales. Accecido: 13/12/2020.

[58] Mozilla. *Regular expressions - JavaScript — MDN*. https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Regular_Expressions. Accedido: 02/04/2021.

[59] NASA. About the hubble space telescope. https://www.nasa.gov/mission_pages/hubble/about. Accedido: 16/05/2021.

[60] NASA. Hubble Space Telescope, observatory - instruments. https://www.nasa.gov/content/goddard/hubble-space-telescope-science-instruments. Accedido: 15/03/2021.

[61] NASA. Wfc3 - technology - filters. https://wfc3.gsfc.nasa.gov/tech/filters.html. Accedido: 16/05/2021.

[62] NASA. A flight through the candels ultra deep survey field, 2019. https://svs.gsfc.nasa.gov/31035. Accedido: 18/05/2021.

[63] Andrew Yan-Tak Ng, Kian Katanforoosh, and Younes Bensouda Mourri. Improving deep neural networks: Hyperparameter tuning, regularization and optimization. *Coursera*.

[64] Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015. http://neuralnetworksanddeeplearning.com/. Accedido: 05/11/2020.

[65] Koldo Pina Ortiz. Como entrenar a tu perceptrón. Mar 2018. https://koldopina.com/como-entrenar-a-tu-perceptron/. Accedido: 05/10/2020.

[66] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks, 2015.

[67] Overleaf. *Documentation*. https://es.overleaf.com/learn. Accedido: 15/10/2020.

[68] Pandas. *pandas documentation*. https://pandas.pydata.org/docs/. Accedido: 02/05/2021.

[69] Alfonso Jesús Población. Cálculo diferencial en varias variables. In *Ampliación de Matemáticas, Grado de Ingeniería Informática*, chapter 5. 2018.

[70] Jonathon Price, Alfred Wong, Tiancheng Yuan, Joshua Mathews, and Taiwo Olorunniwo. Stochastic gradient descent. *Cornell University*, 2020. `https://optimization.cbe.cornell.edu/index.php?title=Stochastic_gradient_descent`. Accedido: 11/04/2021.

[71] Jupyter Project. Jupyter. `https://jupyter.org/`. Accedido: 20/03/2021.

[72] Python Software Foundation. *Python 3 documentation*. `https://docs.python.org/3/`.

[73] PyTorch. *PyTorch documentation*. `https://pytorch.org/docs/`. Accedido: 20/01/2020.

[74] Quora. What is the difference between deep learning and usual machine learning? `https://www.quora.com/What-is-the-difference-between-deep-learning-and-usual-machine-learning`. Accedido: 10/06/2021.

[75] Google Research. Colaboratory, Frequently asked questions. `https://research.google.com/colaboratory/faq.html`. Accedido: 25/05/2021.

[76] ResearchGate. Application of transfer learning using convolutional neural network method for early detection of terry's nail. `https://www.researchgate.net/figure/Illustration-of-Max-Pooling-and-Average-Pooling-Figure-2-above-shows-an-example-of-max_fig2_333593451`. Accedido: 16/12/2020.

[77] ResearchGate. Maritime semantic labeling of optical remote sensing images with multi-scale fully convolutional network. `https://www.researchgate.net/figure/The-receptive-field-of-each-convolution-layer-with-a-3-3-kernel-The-green-area-marks_fig4_316950618`. Accedido: 15/05/2021.

[78] Mark Robins. The difference between artificial intelligence, machine learning and deep learning. *Intel*, May 2020. `https://www.intel.es/content/www/es/es/artificial-intelligence/posts/difference-between-ai-machine-learning-deep-learning.html`. Accedido: 10/11/2020.

[79] D. Rumelhart, Geoffrey E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[80] Amazon Web Services. Regression - amazon machine learning. `https://docs.aws.amazon.com/es_es/machine-learning/latest/dg/regression.html`. Accedido: 20/03/2021.

[81] SExtractor. *SExtractor User Manual — SExtractor 2.24.2 documentation*, 2017. `https://sextractor.readthedocs.io/en/latest/index.html`. Accedido: 31/05/2021.

[82] Kousai Smeda. Understand the architecture of cnn, Oct 2019. `https://towardsdatascience.com/understand-the-architecture-of-cnn-90a25e244c7`. Accedido: 15/12/2020.

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

[84] Leslie N. Smith and Nicholay Topin. Super-convergence: Very fast training of neural networks using large learning rates, 2018.

[85] BarD Software s.r.o. Ganttproject. `https://www.ganttproject.biz/`. Accedido: 20/11/2020.

[86] Eli Stevens, Luca Antiga, and Thomas Viehmann. *Deep Learning with PyTorch*. Manning, 2020. `https://pytorch.org/assets/deep-learning/Deep-Learning-with-PyTorch.pdf`. Accedido: 20/01/2021.

[87] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1139–1147, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

[88] Mark Taylor. Topcat. `http://www.star.bristol.ac.uk/~mbt/topcat/`. Accedido: 16/03/2021.

[89] Jupyter Team. *Jupyter Notebook Documentation*, 2015. `https://jupyter-notebook.readthedocs.io/`. Accedido: 20/03/2021.

[90] Andrew Trask. A neural network in 13 lines of python (part 2 - gradient descent), 7 2015. `https://iamtrask.github.io/2015/07/27/python-network-part2/`. Accedido: 17/05/2021.

[91] CANDELS UDF. Nasa, 2019. `https://svs.gsfc.nasa.gov/31020`. Accedido: 18/05/2021.

[92] Guido van Rossum, Barry Warsaw, and Nick Coghlan. *Style Guide for Python Code*. Python Software Foundation, Jul 2001. `https://www.python.org/dev/peps/pep-0008/`. Accedido: 16/01/2021.

[93] Carlos Santana Vega. Dot CSV. `https://www.youtube.com/DotCSV`. Accedido: 19/03/2021.

[94] Wikipedia. Gradient descent. `https://en.wikipedia.org/wiki/Gradient_descent`. Accedido: 11/03/2021.

[95] Wikipedia. Hubble sequence. `https://en.wikipedia.org/wiki/Hubble_sequence`. Accedido: 19/05/2021.

[96] Wikipedia. Perceptrón multicapa. `https://es.wikipedia.org/wiki/Perceptr%C3%B3n_multicapa`. Accedido: 04/10/2020.