Universidad de Valladolid

Escuela de Ingeniería Informática

Trabajo Fin de Grado

Grado en Ingeniería Informática

Desarrollo de Técnicas Basadas en Frameworks Deep Learning para la Caracterización de Galaxias Distantes

Autor

D. Jesús Fernández Iglesias

Tutores

D. Benjamín Sahelices Fernández

D. Fernando Buitrago Alonso

Resumen

La astronomía está experimentando un rápido crecimiento actualmente en cuanto a la masividad y la complejidad de los datos generados. Los telescopios sinópticos, telescopios que cartografían todo el cielo sistemáticamente y no solo regiones específicas, como Euclid, WFIRST, J-PAS o Vera C. Rubin son una prueba de ello. Éste último, que próximamente se instalará en Chile, generará 30 TB de información cada noche, y al final de sus primeros 10 años proporcionará un archivo con unos ∼500 PB de información (imágenes más otros productos derivados de ellas).

Ser capaces de aprovechar de manera útil toda esa información provocará avances en la astronomía de dimensiones quizás no vistas hasta la actualidad. Para analizar tales cantidades de información en búsqueda de patrones ocultos puede ser de gran utilidad el aprendizaje automático, y en concreto modelos de *deep learning* que se encuentran en el estado del arte.

En este trabajo se desarrollan técnicas y algoritmos *deep learning* punteros para resolver dos problemas de gran alcance, la clasificación morfológica de galaxias distantes y la detección de bordes de galaxias.

La morfología de las galaxias determina qué procesos físicos están aconteciendo en ellas, y el desarrollo de técnicas de estimación robusta de su estructura es todo un reto. Tanto la clasificación morfológica directa como la imitación del comportamiento de humanos expertos en dicha tarea son tratadas.

De igual manera, ser capaces de delimitar con precisión las fronteras de dichos cuerpos tiene multitud de aplicaciones en el estudio de la formación y evolución del universo distante.

Los conjuntos de datos utilizados provienen del proyecto CANDELS (*Cosmic Assembly Near-infrared Deep Extragalactic Legacy Survey*) [1], el proyecto más grande (984 órbitas) jamás llevado a cabo por el Telescopio Hubble y cuyo objetivo principal era documentar el primer tercio de la evolución galáctica. Se utilizan las 5 regiones de exposición disponibles en el estudio: GOODS-N, GOODS-S, UDS, EGS y COSMOS.

Abstract

Astronomy is currently experiencing rapid growth in terms of the massiveness and complexity of the data generated. Synoptic telescopes, telescopes that map the whole sky systematically and not just specific regions, such as Euclid, WFIRST, J-PAS or Vera C. Rubin are proof of this. The latter, which will soon be installed in Chile, will generate 30 TB of information every night, and at the end of its first 10 years will provide an archive with about 500 PB of information (images plus other products derived from them).

Being able to use all this information in a useful way will lead to advances in astronomy of dimensions perhaps unseen until now. To analyze such amounts of information in search of hidden patterns, machine learning, and in particular state-of-the-art models of deep learning, can be very useful.

In this work, modern techniques and algorithms are developed to solve two far-reaching problems, the morphological classification of distant galaxies and the detection of galaxy edges.

The morphology of galaxies tells us what physical processes are going on in them, and the development of robust structure estimation techniques is a challenge. Both direct morphological classification and mimicking the behavior of human experts in such a task are discussed.

Likewise, being able to precisely delimit the boundaries of such bodies has many applications in the study of the formation and evolution of the distant universe.

The data sets used are from the CANDELS (Cosmic Assembly Near-infrared Deep Extragalactic Legacy Survey) project [1], the largest project (984 orbits) ever undertaken by the Hubble Telescope and whose main objective was to document the first third of galactic evolution. The 5 exposure fields available in the survey are used: GOODS-N, GOODS-S, UDS, EGS, and COSMOS.

Agradecimientos

A los profesores del Grado en Ingeniería Informática que me han acompañado durante estos años por transmitirme la pasión y el conocimiento necesarios para formar mi futuro profesional.

A Benjamín y a Fernando, por la constante dedicación y ayuda que han puesto en este proyecto, la predisposición a resolver dudas y la confianza depositada en mi. El fantástico ambiente de trabajo creado ha sido clave para el desarrollo de un trabajo de calidad.

A mis compañeros de carrera por haber compartido esta travesía a mi lado, superando juntos todas las dificultades surgidas.

Y por último, a mi familia, en espacial a mis padres, por inculcarme los valores del sacrificio y el esfuerzo tan necesarios, y ser los pilares que me han sostenido durante toda esta etapa.

Índice general

Índice de figuras

Índice de cuadros

1. Introducción

La astronomía está experimentando un rápido crecimiento en cuanto a que los conjuntos de datos que se generan con cada vez más masivos y complejos. Este fenómeno ha provocado que el descubrimiento de conocimiento científico de este área ya no se enfoque bajo el paradigma tradicional, si no que se siga una aproximación *data driven*, donde los astrónomos desarrollan herramientas para analizar y explorar los conjuntos de datos y extraer conclusiones a partir de los patrones y relaciones encontrados.

En años recientes, la aplicación de algoritmos de aprendizaje automático a problemas astronómicos ha crecido de manera exponencial y son usados para una amplia variedad de tareas. Baron, Dalya. (2019) en *Machine Learning in Astronomy: a practical overview* [2] hace un recorrido a través de la multitud de uniones entre algoritmos de *machine learning* y problemas de carácter astronómico.

Este trabajo de investigación sigue la misma línea y trata del desarrollo del técnicas de aprendizaje automático, concretamente del subcampo del aprendizaje profundo, aplicadas a dos grandes problemas: la clasificación morfológica de galaxias distantes en primera instancia y la detección de bordes de las mismas posteriormente.

En el Capítulo 2 se hace un repaso a los pilares de las redes neuronales artificiales, y se introduce y detalla la arquitectura y funcionamiento de las redes neuronales convolucionales.

Por su parte, los objetivos, tareas desarrolladas y riesgos del trabajo detectados se muestran en el Capítulo 3.

El Capítulo 4 trata de astronomía. Primero, se introducen conceptos básicos de esta disciplina necesarios para ubicar el dominio de aplicación de las técnicas desarrolladas en este trabajo y las complejidades que han surgido desde el primer momento. Posteriormente, en el marco de la astroinformática, se enumera y explica el *software* astronómico que se ha utilizado durante el desarrollo del proyecto.

En el Capítulo 5 se realiza una presentación de los *frameworks* y herramientas que se han utilizado para el desarrollo de los algoritmos de aprendizaje profundo y la realización en general del proyecto.

La clasificación morfológica de galaxias distantes es tratada en el Capítulo 6. Primero, se ataca el problema mediante la clasificación directa, midiendo y comprobando el efecto que causan diversas técnicas sobre una arquitectura diseñada. Segundo, el problema se ataca mediante la regresión, donde 3 arquitecturas de diferente complejidad son utilizadas con el objetivo de ver su

adecuación a la tarea tratando de imitar el comportamiento de expertos en la materia. Además, se estudia si el efecto de diversas transformaciones genera los mismos cambios en cada arquitectura.

El problema más complejo del trabajo, la detección de bordes de galaxias, es abordado en el Capítulo 7. La construcción de redes neuronales convolucionales en forma de U (U-*Nets*) y la aplicación del aprendizaje por *ensembles* en el marco de la segmentación semántica son tratados en profundidad.

A modo de conclusión, y como broche final al trabajo, el último capítulo está dedicado a la extracción de conclusiones y sugerencia de líneas futuras de investigación.

2. Redes Neuronales

El aprendizaje automático es una aplicación de la Inteligencia Artificial que provee a los sistemas de la habilidad para automáticamente aprender y mejorar a través de la experiencia y sin ser explícitamente programados. El proceso de aprendizaje comienza con datos, experiencia directa y/o instrucciones, tratando de buscar y encontrar patrones en dicha información para mejorar la toma de decisiones futura.

Actualmente, el amplio abanico de metodologías y técnicas que se encuentran bajo el seno del *machine learning* puede clasificarse en 4 grandes categorías:

- **Aprendizaje supervisado**: se caracteriza por aprovechar la información que proporcionan la etiquetas de los datos. La naturaleza de la variable respuesta permite diferenciar entre problemas de clasificación, donde ésta es categórica, y problemas de regresión, donde ésta es numérica. Algoritmos y técnicas tan conocidas como los árboles de decisión, la regresión logística, SVM *(Support Vector Machines)*, k-vecinos, Naive-Bayes, *Bagging* o *Boosting* pertenecen a este paradigma.
- **Aprendizaje no supervisado**: trata de inferir patrones presentes en conjuntos de datos no etiquetados. Se consiguen aprender propiedades estructurales internas muy valiosas del *dataset*. En este grupo se encuentran métodos tan famosos como las k-medias o el agrupamiento espacial basado en densidad de aplicaciones con ruido (*DBSCAN*).
- **Aprendizaje semisupervisado**: utiliza tanto datos etiquetados como sin etiquetar para generar las hipótesis. Este paradigma se encuentra entre medias del aprendizaje supervisado y el no supervisado. Los algoritmos pertenecientes a esta categoría operan con una pequeña cantidad de datos que están etiquetados, para el resto se desconoce o se omite intencionadamente la etiqueta. Los modelos que se engloban en esta categoría suelen responder a alguno de los siguientes *generative*, *low-density separation*, *graph-based* o *heuristic*.
- **Aprendizaje por refuerzo**: no se dispone de datos etiquetados, un agente o modelo interactúa con el medio y aprende mediante la realización de acciones que pueden desencadenar recompensas. Se busca maximizar el número de premios. Un ejemplo sencillo para imaginarse el funcionamiento de este tipo de aprendizaje es un robot que está aprendiendo a andar. Cuanto más tiempo aguante andando sin caerse, la recompensa que obtendrá será mayor.

La presencia de tantas aproximaciones para la resolución de problemas de aprendizaje automático ofrece una amplia variedad y diversidad de soluciones, y su continuo e incesante desarrollo permite aplicar estos algoritmos a cada vez más problemas pertenecientes a un amplio abanico de contextos, obteniendo resultados excelentes en muchos casos.

La similitud existente entre un algoritmo ideal de aprendizaje automático y el cerebro humano es palmaria. En ambos casos, una estructura más o menos compleja es alimentada con datos. En el

primer caso, las estructuras tendrán un carácter tecnológico, mientras que en el segundo, tendrán un carácter biológico. Dichas estructuras, sean de la naturaleza que sean, son capaces de aprender en base a esa información, desarrollando experiencia para comportarse de diferentes maneras ante escenarios diversos. Lo realmente interesante es que estos sistemas son capaces de generalizar a situaciones no contempladas durante el aprendizaje y, en consecuencia, saber comportarse con un alto grado de éxito ante situaciones de incertidumbre.

Ante estas similitudes, históricamente se ha tratado de replicar un cerebro humano biológico artificialmente. Sin embargo, una estructura tan compleja y hasta cierto punto desconocida es muy difícil de replicar.

2.1. Inicios de las redes neruonales artificiales

La primera red neuronal artificial fue diseñada en el año 1943 por Warren McCulloch and Walter Pitts [3]. Ésta se componía de varias pequeñas unidades de procesamiento de la información, simulando una neurona humana. El cómputo que realizaba cada una de estas neuronas era la suma ponderada de las entradas, seguida de la aplicación de una función no lineal al resultado de dicho sumatorio (Figura 2.1).

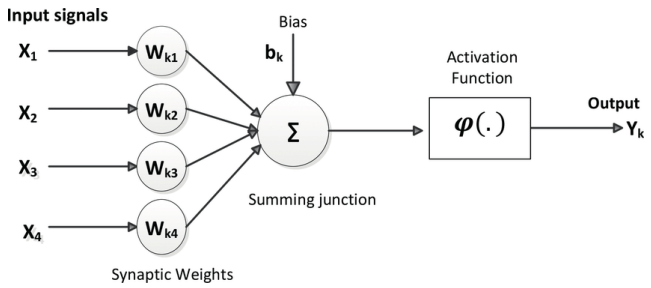

Figura 2.1: Réplica de una neurona biológica en el modelo de McCulloch and Pitts.

Por tanto, este modelo se compone de los siguientes elementos:

- Entradas: X_i es el valor de la entrada i-ésima.
- Pesos: W_{ki} es el valor del peso para la entrada i-ésima en la k-ésima neurona.
- Bias: b_k es una valor que se le añade al resultado del sumatorio del producto de las entradas por sus correspondientes pesos. También es conocido como valor umbral o *threshold*.
- Función de activación: $\varphi(.)$ es una función no lineal. Algunas de las funciones de activación más famosas se detallarán más adelante.
- Salida: Y_k es la salida obtenida después de que la k-ésima neurona haga el procesamiento de las entradas.

Por tanto, según este modelo de neruona, la salida se obtiene como:

$$Y_k = \varphi(b_k + \sum_{i=1}^{n} X_i W_{k,i})$$

Entre las propiedades interesantes que tiene esta representación de neurona artificial destacan la capacidad de simular el comportamiento de varias funciones lógicas, como lo son las funciones *AND*, *OR*, *NOR* y *NOT*. Además, sea cual sea la configuración de la neurona, se ha demostrado que existe un autómata finito determinista equivalente (AFD).

2.2. Generalidades de las redes neuronales

Para recorrer los conceptos principales que residen bajo las generalidades de las redes neuronales y que son expuestos en este capítulo se ha utilizado [4].

2.2.1. Neurona

Una neurona es la unidad básica de cómputo de una red neuronal, y consta de un conjunto de entradas y unos pesos asociados a cada entrada. La neurona, multiplicando las entradas por los pesos, transforma los múltiples valores que recoge inicialmente en uno sólo. Dicha operación puede verse como una suma ponderada, por ejemplo, si se tienen n entradas, entonces la neurona realiza el siguiente cálculo

$$f(x_1, \ldots, x_n) = w_1 \cdot x_1 + \cdots + w_n \cdot x_n = \sum_{i=1}^{n} w_i \cdot x_i \tag{1}$$

El vector n-dimensional de pesos $(w_1, \ldots, w_n)$ que tiene asociado cada neurona va variando durante el entrenamiento de la red neuronal, con el objetivo de que el resultado final que ofrece la red sea lo más óptimo posible. Es fácil darse cuenta que el comportamiento de una neurona es muy similar al de una regresión múltiple, donde se asignan coeficientes a una serie de variables explicativas. Este comportamiento explica que, por sí solas, las neuronas ofrecen un comportamiento lineal. En (1), tras la estimación de los pesos w_i, la neurona representa un hiperplano en $\mathbb{R}^n$.

En ocasiones, al sumatorio ponderado de las entradas se le puede añadir un término llamado *bias* o *offset*, que suele tener el valor 1. La manera de interpretar este término es la misma que la de el parámetro *Intercept* en una regresión lineal, permitiendo mover de izquierda a derecha el hiperplano calculado por la neurona. El bias también tiene asociado un peso, y este es el que marca dicha variación. Por tanto, el cálculo que una neurona con término *offset* realiza es el siguiente:

$$f(x_1, \ldots, x_n) = w_1 \cdot x_1 + \cdots + w_n \cdot x_n + w_{n+1} \cdot b = \sum_{i=1}^{n} (w_i \cdot x_i) + w_{n+1} \cdot b \tag{2}$$

2.2.2. Función de activación

Las funciones de activación tienen como cometido recibir la suma ponderada de las entradas de la neurona marcada por los pesos y, a partir de ella, calcular un nuevo valor. Cada neurona de una red neuronal suele tener asociada una función de activación específica. Ésta puede ser la misma para todas las neuronas de la red, la misma para todas las neuronas de una capa o incluso personalizada para cada neurona, aunque este último caso no suele ser común. Las funciones de activación pueden verse como puertas que comunican la salida de una neurona con las neuronas que se encuentran en la siguiente capa. Las funciones de activación se dividen en dos grandes familias, las lineales y las no lineales. Las primeras, mantienen el comportamiento lineal inducido por el hiperplano que es calculado por las neuronas, mientras que las segundas permiten crear modelos mucho más complejos introduciendo la no linealidad al comportamiento de cada neurona.

2.2.2.1 Lineal

Una función de activación lineal toma como entrada la suma ponderada de las entradas de la neurona, y ofrece una salida proporcional a dicha cantidad.

$$\varphi\left(\sum_{i=1}^{n} w_i \cdot x_i\right) = c * \left(\sum_{i=1}^{n} w_i \cdot x_i\right) \tag{3}$$

Su uso está justificado cuando las salidas que ofrece la red no tienen porqué encontrarse en un determinado intervalo, es decir, no se restringe la salida. Sin embargo, la familia de funciones de activación que cumplen (3) presentan 2 grandes problemas. El primero es que impedimenta el uso del algoritmo de descenso de gradiente para ajustar los pesos de la red, ya que las derivadas de las funciones que se realizan durante la etapa de retropropagación son constantes, y no dependen del valor de las entradas, por lo que la red no es capaz de aprender. El segundo incoveniente del uso de estas funciones de activación es que por muchas capas que se tengan en una red neuronal, la última será una combinación lineal de la primera, debido a que una combinación lineal de combinaciones lineales sigue siendo una combinación lineal. Esto quiere decir que una red neuronal con funciones de activación lineales en todas las neuronas se traduce en una red neuronal con una sola capa.

2.2.2.2 Sigmoide

La función sigmoide es una función de activación no lineal ampliamente usada en las redes neuronales, la transformación que realiza es la siguiente:

$$\varphi\left(\sum_{i=1}^{n} w_i \cdot x_i\right) = \frac{e^{\sum_{i=1}^{n} w_i \cdot x_i}}{1 + e^{\sum_{i=1}^{n} w_i \cdot x_i}} \tag{4}$$

Las principales ventajas de esta función para su aplicación como función de activación es que es diferenciable, monótona y normaliza la salida de las neuronas al intervalo (0, 1). Entre las

desventajas de esta función destacan su alto coste computacional y la poca variación que sufre el valor de salida ante grandes variaciones del valor de entrada. Esto se traduce en que redes donde se utiliza esta función de activación pueden sufrir el problema del desvanecimiento del gradiente, el cual se explicará más adelante.

2.2.2.3 Tangente hiperbólica

La tangente hiperbólica es una función de activación de activación no lineal con una estructura muy similar a la función sigmoide:

$$\varphi\left(\sum_{i=1}^{n} w_i \cdot x_i\right) = \frac{e^{\sum_{i=1}^{n} w_i \cdot x_i} - e^{-\left(\sum_{i=1}^{n} w_i \cdot x_i\right)}}{e^{\sum_{i=1}^{n} w_i \cdot x_i} + e^{-\left(\sum_{i=1}^{n} w_i \cdot x_i\right)}} \tag{5}$$

Las salidas de la función en (5) se encuentran en el intervalo (-1, 1), y la ventaja de su uso respecto a la función sigmoide es que su derivada es más inclinada, lo que facilita que el aprendizaje de la red pueda ser más rápido. Las desventajas, al igual que en la función de activación anterior, son su alto coste computacional y la tendencia a originar problemas de desvanecimiento del gradiente.

2.2.2.4 Rectificador lineal

ReLU es una función de activación no lineal muy usada en aprendizaje profundo, y se define de la siguiente manera:

$$\varphi\left(\sum_{i=1}^{n} w_i \cdot x_i\right) = \begin{cases} 0 & \text{si } \sum_{i=1}^{n} w_i \cdot x_i \leq 0 \\ \sum_{i=1}^{n} w_i \cdot x_i & \text{si } \sum_{i=1}^{n} w_i \cdot x_i > 0 \end{cases} \tag{6}$$

ReLU es computacionalmente muy eficiente, lo que ha provocado su uso mayoritario por encima de la función sigmoide y la tangente hiperbólica en redes de gran tamaño. Sus valores oscilan en el intervalo [0, ∞). Sin embargo, esta función presenta problemas cuando la suma ponderada de las entradas se aproxima a 0, causando que la retropropagación no sea efectiva y la red no pueda aprender. Para solucionar este problema, se introduce la función *Leaky* ReLU:

$$\varphi\left(\sum_{i=1}^{n} w_i \cdot x_i\right) = \begin{cases} 0{,}01 \cdot \sum_{i=1}^{n} w_i \cdot x_i & \text{si } \sum_{i=1}^{n} w_i \cdot x_i \leq 0 \\ \sum_{i=1}^{n} w_i \cdot x_i & \text{si } \sum_{i=1}^{n} w_i \cdot x_i \geq 0 \end{cases} \tag{7}$$

Con esta modificación se consigue que cuando la entrada es negativa, la pendiente de la función es ligeramente positiva, permitiendo la retropropagación incluso para entradas negativas.

2.2.3. Entrenamiento

El proceso de entrenamiento de una red neuronal mediante el método clásico de *backpropagation* consta de 3 etapas sucedidas de manera secuencial: la alimentación hacia delante, el cálculo de la

pérdida y la retropropagación. La primera de las etapas (*feedforward*) consiste fluir las entradas de la red neuronal a través de la misma. Los *inputs* originales se introducen en la primera capa, la salida de la primera capa es la entrada de la segunda. Esto es, las salida de la capa i se utiliza como entrada de la capa $i + 1$.

Tras la etapa de alimentación hacia delante, hay que medir cómo de diferentes son los resultados obtenidos por la red de los deseados, es decir, comparar las predicciones con las etiquetas. Para medir de forma objetiva las discrepancias se utilizan las funciones de pérdida o *loss functions*. Estas funciones toman las predicciones y las etiquetas, y ofrecen un valor indicativo de su similitud. Es crucial que las funciones de pérdida sean las adecuadas para cada tipo de problema, por ejemplo, no se debe utilizar la misma función para un problema de clasificación categórica que para un problema de regresión numérica, la naturaleza de la variable respuesta marcará qué tipos de funciones de pérdida deben utilizarse (Cuadros 2.1 y 2.2).

Mean Squared Error	$\frac{\sum_{i=1}^{n}(y_i-\hat{y}_i)^2}{n}$
Mean Absolute Error	$\frac{\sum_{i=1}^{n}\lvert y_i-\hat{y}_i\rvert}{n}$
Mean Bias Error	$\frac{\sum_{i=1}^{n}(y_i-\hat{y}_i)}{n}$

Cuadro 2.1: Funciones de pérdida comunes en problemas de regresión.

SVM Loss	$\sum_{j\neq y_i} \text{máx}\,(0, \text{score}_j - \text{score}_{y_i} + 1)$
Cross Entropy Loss	$-\left(y_i log\hat{y}_i + (1-y_i)\, log\,(1-\hat{y}_i)\right)$

Cuadro 2.2: Funciones de pérdida comunes en problemas de clasificación.

El objetivo de la red neuronal durante el entrenamiento es el de minimizar esa función de pérdida, lo cual se consigue asignando a los pesos los valores correctos. Esta modificación de los pesos es la que se realiza en la etapa de *backpropagation*. Para resolver el problema de minimización lo más común es utilizar un método numérico denominado descenso de gradiente. El descenso de gradiente es un método numérico de optimización que trata de encontrar la porción de una función donde se alcanzan sus valores más bajos. Podría pensarse que lo más útil para resolver el problema de minimización sería calcular dónde la derivada de la función es igual a 0, sin embargo, si el número de variables es muy grande las funciones suelen ser muy complejas y resolver el problema exacto de optimización es muy difícil.

Para minimizar una función objetivo, el descenso de gradiente calcula el gradiente evaluado en el punto que marque en cada momento la salida de la red. Ese gradiente marcará la dirección del ascenso más pronunciado, es decir, indica el camino más rápido para maximizar una función. Como lo que se desea es realizar una minimización, entonces se modificarán los pesos en la dirección opuesta. Los gradientes irán acumulándose desde las capas finales de la red neuronal hasta las capas iniciales mediante la regla de la cadena, para así modificar todos los pesos de la red, en lo que se conoce como *backpropagation*.

Para ilustrar de manera sencilla este proceso, suponer que se tiene un conjunto de datos $\mathcal{D} = \{x^i, y^i : i = 1, 2, \ldots, N\}$, donde $x^i \in \mathbb{R}^3, \quad y^i \in \mathbb{R} \quad \forall i$ y una red sencilla (Figura 2.2).

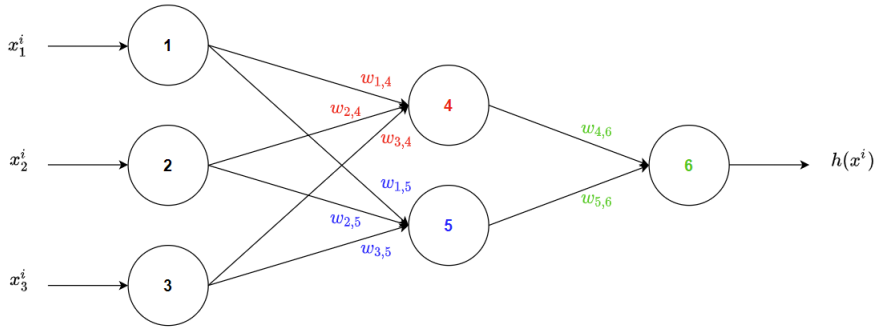

Figura 2.2: Red neuronal sencilla.

El resultado $h(x^i)$ se calcula como:

$$h(x^i) = \varphi(w_{4,6} \cdot \varphi(w_{1,4} \cdot x_1^i + w_{2,4} \cdot x_2^i + w_{3,4} \cdot x_3^i) + w_{5_6} \cdot \varphi(w_{1,5} \cdot x_1^i + w_{2,5} \cdot x_2^i + w_{3,5} \cdot x_3^i))$$

El conjunto de coeficientes a los que hay que estimar su valor es:

$$\vartheta = \{w_{1,4}, w_{2,4}, w_{3,4}, w_{1,5}, w_{2,5}, w_{3,5}, w_{4,6}, w_{5,6}\}$$

con el objetivo de minimizar una función de pérdida, por ejemplo:

$$J(\vartheta) = \frac{1}{2} \sum_{i=1}^{N} \left(y^i - h(x^i)\right)^2$$

El algoritmo de descenso de gradiente actualiza los parámetros en la dirección opuesta a la de máximo crecimiento de la función, esto es:

$$\vartheta := \vartheta - \eta \cdot \frac{\partial J}{\partial \vartheta}$$

Particularizando en un caso concreto para ver como funciona el *backpropagation*, suponer que se quiere calcular la variación que sufre el peso $w_{2,5}$. Sea:

$$a(x^i) = w_{1,4} \cdot x_1^i + w_{2,4} \cdot x_2^i + w_{3,4} \cdot x_3^i$$

$$b(x^i) = w_{1,5} \cdot x_1^i + w_{2,5} \cdot x_2^i + w_{3,5} \cdot x_3^i$$

$$c(x^i) = w_{4,6} \cdot \varphi(a(x^i)) + w_{5,6} \cdot \varphi(b(x^i))$$

Entonces:

$$\frac{\partial J}{\partial w_{2,5}} = \sum_{i=1}^{N} \left[\left(h(x^i) - y^i \right) \frac{\partial h(x^i)}{\partial w_{2,5}} \right]$$

$$\frac{\partial h(x^i)}{\partial w_{2,5}} = \frac{\partial h(x^i)}{\partial c(x^i)} \cdot \frac{\partial c(x^i)}{\partial b(x^i)} \cdot \frac{\partial b(x^i)}{\partial w_{2,5}}$$

Si se quiere calcular la variación que sufrirá el peso $w_{1,4}$, entonces el procedimiento de *backpropagation* involucrará la siguiente cadena de derivadas parciales:

$$\frac{\partial J}{\partial w_{1,4}} = \sum_{i=1}^{N} \left[\left(h(x^i) - y^i \right) \frac{\partial h(x^i)}{\partial w_{1,4}} \right]$$

$$\frac{\partial h(x^i)}{\partial w_{1,4}} = \frac{\partial h(x^i)}{\partial c(x^i)} \cdot \frac{\partial c(x^i)}{\partial a(x^i)} \cdot \frac{\partial a(x^i)}{\partial w_{1,4}}$$

Entre las ventajas por las que se destaca el algoritmo de descenso de gradiente se encuentran la sencillez de implementación y la eficiencia computacional. Sin embargo, este optimizador tiene facilidad para quedar atrapado en mínimos locales. Si la función a optimizar tiene varios mínimos, es posible que el mínimo absoluto nunca sea alcanzado. Además, la etapa de actualización de los pesos es llevada a cabo una vez se han introducido todos los ejemplos de entrenamiento a la red, por lo que la convergencia a la solución puede requerir mucho tiempo.

2.2.4. Optimizadores

Los optimizadores son los algoritmos y métodos encargados de variar los parámetros de la red con el fin de alcanzar el mínimo absoluto en la función de pérdida. Tradicionalmente esta variación únicamente afecta a los pesos, aunque recientemente hay algoritmos que también son capaces de optimizar otros parámetros como la tasa de aprendizaje.

2.2.4.1 Descenso de gradiente estocástico

Se trata de una variante del algoritmo de descenso del gradiente donde los pesos se modifican de manera más frecuente. La actualización de los parámetros se lleva a cabo con cada ejemplo del conjunto de entrenamiento tras computar la función de pérdida. Por ejemplo, si un conjunto de datos está compuesto de N muestras, el algoritmo de descenso de gradiente estocástico actualizará los pesos N veces por época (pasada por todas las muestras de entrenamiento), mientras que el algoritmo de descenso de gradiente tradicional los actualiza una sola vez por época. Debido a que los parámetros del modelo son actualizados constantemente, la evolución de la pérdida o *loss* puede sufrir fluctuaciones muy bruscas.

$$\vartheta := \vartheta - \eta \cdot \frac{\partial J\left(\vartheta, x^{(i)}, y^{(i)}\right)}{\partial \vartheta}$$

Entre las ventajas de este optimizador se encuentra la rápida a la convergencia a la solución a costa de que las estimaciones de los parámetros tengan una alta varianza.

2.2.4.2 Descenso de gradiente *mini batch*

Se trata de una variante que se encuentra en un camino intermedio entre el descenso de gradiente y el descenso de gradiente estocástico. La actualización de los pesos se lleva a cabo tras introducir un *batch* $B^{(i)}$ en la red, es decir un conjunto de muestras de tamaño fijado.

$$\vartheta := \vartheta - \eta \cdot \frac{\partial J\left(\vartheta, B^{(i)}\right)}{\partial \vartheta}$$

La convergencia a la solución óptima se lleva a cabo más rápido que en el descenso de gradiente, y la varianza de las estimaciones es menor que en la variante estocástica. Es la mejor variante dentro de la familia de optimizadores bajo el abanico del descenso de gradiente.

2.2.4.3 Momentum

El optimizador momentum fue creado para mitigar la varianza en las estimaciones de los parámetros que causaba el oprimizador SGD (*Stochastic Gradient Descent*). Acelera la convergencia y reduce las fluctuaciones en la dirección irrelevante. La variación de los pesos en una etapa depende de la variación de los mismos en la etapa anterior.

$$V(t) = \gamma V(t-1) + \eta \cdot \frac{\partial J(\vartheta)}{\partial \vartheta}$$

$$\vartheta = \vartheta - V(t)$$

El término momento γ es normalmente fijado en 0.9. Además de reducir la varianza y las oscilación en las estimaciones converge más rápido a la solución que los algoritmos de descenso de gradiente.

2.2.4.4 ADAM

ADAM (*Adaptative Moment Estimation*) utiliza los momentos de primer y segundo orden de los gradientes, esto es, su media y su varianza, siendo β_1^t el parámetro de deterioro exponencial para las estimaciones del momento de primer orden y β_2^t el parámetro de deterioro exponencial para las estimaciones del momento de segundo orden.

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

Este método trata de dar pasos más pequeños que el resto de optimizadores para llegar al óptimo, es decir, se reduce la velocidad ligeramente para realizar una búsqueda más cuidadosa. ADAM almacena un promedio del deterioro exponencial de los gradientes.

2.2.5. Causas de mal rendimiento

Las redes neuronales, al ser estructuras tan complejas, no están exentas de problemas. En la amplia mayoría de los casos, éstos guardan relación con el sobreajuste e infraajuste o con el desvanecimiento y explosión del gradiente.

2.2.5.1 Sobreajuste e Infraajuste

El sobreajuste u *overfitting* consiste en que un modelo ajuste demasido bien los datos de entrenamiento pero su rendimiento sobre el conjunto de test o de validación es pobre. Los modelos que sufren de este problema tienen una capacidad de generalización baja, llegando a aprenderse de memoria las muestras de entrenamiento y, en consecuencia, careciendo de poder discriminatorio sobre las muestras con las que no se le han ajustado sus parámetros.

En el caso de las redes neuronales, cuando presentan sobreajuste, existen varias técnicas que se pueden aplicar para intentar paliarlo, aunque su aplicación no tiene por qué reducir dicha problemática, las más comunes son:

- Simplificar el modelo: reducir el número de capas de la red o el número de neuronas de las mismas simplifica el modelo y en ocasiones mitiga el sobreajuste.

- *Early stopping*: consiste en parar el entrenamiento de la red en cuanto el rendimiento sobre el conjunto de test comience a empeorar. Vista la evolución de la tasa de error que comete la red sobre el conjunto de test a lo largo de las épocas, el *early stopping* consiste en parar cuando se alcance el mínimo absoluto.

- *Data augmentation*: se trata de aumentar de manera artificial el número de imágenes en el conjunto de entrenamiento. Algunas de las transformaciones más usadas son la rotación, el recorte o el difuminado (*blur*).

- Regularización: técnica para reducir la complejidad del modelo mediante la adición de una penalización a la función de pérdida. En el caso de la regularización L1, dicha penalización es $\lambda \cdot \sum_{i=1}^{n} \mid \vartheta_i \mid$, mientras que en el caso de la regularización L2 dicha penalización es $\lambda \cdot \sum_{i=1}^{n} \vartheta_i^2$. El parámetro λ dicta la cantidad de regularización que será aplicada, provocando que los parámetros del modelo ϑ_i tiendan más a cero cuando λ aumenta.

- *Dropouts*: consiste en eliminar aleatoriamente algunas neuronas de la red durante el entrenamiento de la misma. Cuando se eliminan distintos conjuntos de neuronas de una red que sufre *overfitting*, las redes resultantes también sufrirán el mismo problema. El objetivo es que la combinación de esas redes actúe como un balance y el resultado sea que se reduce el sobreajuste de la red original (Figura 2.3).

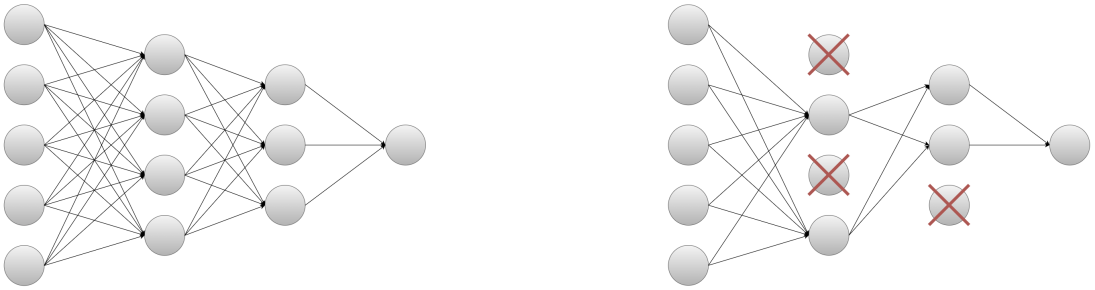

Figura 2.3: Efecto del *dropout* sobre una red.

El *infraajuste* consiste en que los modelos no son capaces de realizar un ajuste razonable sobre el conjunto de entrenamiento, en consecuencia el rendimiento suele ser igual de pobre o peor sobre los conjuntos de test o validación. En el caso de la redes neuronales, lo más común para solventar esta problemática suele ser o bien cambiar la arquitectura de la red o bien modificar la que se está usando añadiéndola más complejidad, ya sea aumentando el número de capas o aumentando el número de neuronas en cada capa.

2.2.5.2 Desvanecimiento y explosión del gradiente

Los problemas de desvanecimiento y explosión del gradiente son muy comunes en las redes neuronales muy profundas. Como se ha venido explicando a los largo del capítulo, cuando se entrena una red neuronal para ajustar sus pesos mediante retropropagación hay que calcular una serie de derivadas parciales. Estas derivadas se extienden desde la última capa del red hasta la primera, por lo que a mayor número de capas que tenga la red mayor será el número de derivadas

parciales que se irán acumulando. En una red con n capas ocultas, n derivadas serán multiplicadas para modificar los pesos de las neuronas de la primera capa oculta.

Si las derivadas tienen un valor alto, el producto de multiplicar varias aumentará de manera exponencial conforme vamos retrocediendo por las capas ocultas de la red hasta que en un momento dado el valor explota y se hace inmanejable. Este problema es el conocido como explosión del gradiente. La acumulación de valores de derivadas muy altos resulta en que la red se vuelve muy inestable, siendo incapaz de aprender de manera efectiva carcaterísticas de los datos de entrada. En el peor caso, el producto de las derivadas causará un desbordamiento (*overflow*), provocando que a los pesos se les asigne un *NaN* y no se puedan volver a modificar.

De manera análoga, si las derivadas tienen un valor pequeño, el producto acumulado que llegará a las primeras capas ocultas de la red será prácticamente 0, el gradiente se desvanece. Este problema es conocido como la evanescencia del gradiente. En el peor caso de que el gradiente llegue a valer 0, la red dejará de aprender y se detendrá el entrenamiento.

Para diagnosticar el problema de la explosión del gradiente, las siguientes técnicas pueden resultar de utilidad [5]:

- El modelo no es capaz de aprender del conjunto de entrenamiento.
- Los pesos y/o la pérdida se vuelven NaN.
- Los pesos de la red tienen valores enormes y por tanto el *loss* sufre grandes cambios en cada actualización.

En el caso del problema del desvanecimiento del gradiente, los puntos en los que fijarse para diagnosticarlo son:

- El modelo mejora de manera muy lenta durante la fase de entrenamiento, o incluso no mejora obteniendo un desempeño muy pobre.
- Los pesos del modelo valen 0, o valores muy cercanos al 0.

Para intentar mitigar ambos problemas una de las técnicas más utilizadas es reducir el número de capas de la red, perdiendo con la decisión capacidad de ajuste a soluciones complejas. Una inicialización cuidadosa de los pesos de la red también puede ser clave para evitar tanto la explosión como el desvanecimiento del gradiente. En el caso del problema de la explosión, fijar una cota superior para el valor absoluto de los pesos también puede resultar de gran utilidad.

2.3. Redes neuronales convolucionales

Las redes neuronales convolucionales son un tipo de redes neuronales cuyo objetivo es replicar los procesos que ocurren en la corteza visual primaria de un cerebro biológico [6]. Esta característica las hace idóneas para tareas como la clasificación de imágenes o la segmentación. Las redes convolucionales son una variación del perceptrón multicapa, prepradas para que las entradas sean matrices multidimensionales en vez de vectores unidimensionales.

Para comprender la diferencia que reside entre una red convolucional y un perceptrón multicapa clásico supóngase que se tienen una serie de imágenes de 100x100 píxeles en formato RGB, es decir, la imagen se encuentra en 3 canales. Las imágenes pueden contener coches o aviones, y se desea clasificar cada imagen en base a la presencia de un coche o un avión en la misma. Si se utiliza un perceptrón multicapa para dicha tarea, las imágenes tienen que ser aplanadas de manera previa a su introducción a la red. Cada imagen se convertirá en un vector de $100^2 \times 3$ posiciones, y será dicho vector el que será introducido a la red. Al aplanar la imagen completa se están perdiendo características espaciales imprescindibles necesarias para la discriminación del objeto que contiene, por lo que esta aproximación no suele ofrecer buenos resultados.

Si se utiliza una red convolucional, lo primero que se hará será intentar extraer las características más primarias de la imagen, los rasgos más identificativos de un coche y de un avión, características que a la postre si podrán utilizarse para que un perceptrón multicapa pueda clasificar el objeto contenido en la imagen inicial. De este modo, una red convolucional está compuesta por dos conjuntos de capas diferenciadas, las capas convolucionales y las capas FC (*fully connected*). Además, esta aproximación permite trabajar con imágenes de gran tamaño, ya que al reducir sus dimensiones antes de introducirla a un perceptrón multicapa la carga computacional derivada de su entrenamiento no es tan alta.

2.3.1. Capas convolucionales

Las capas convolucionales son la primera parte de una red neuronal convolucional, y en ocasiones se les denomina *feature extractor* o extractor de características. Esta denominación proviene del trabajo que realizan estas capas, que es obtener los rasgos más primarios de las imágenes introducidas, rasgos que por si solos pueden ser utilizados para determina la naturaleza de la imagen. Una capa convolucional completa suele constar de una operación de convolución, una función de activación no lineal y un operación de *pooling*.

2.3.1.1 Operación de convolución

Una convolución es la aplicación de un filtro o *kernel* a una entrada, por ejemplo una imagen, lo que genera una característica. La aplicación repetida del mismo filtro a distintas zonas de la entrada provoca un mapa de características (Figura 2.4). El mapa de características indica la

localización y la intensidad de las características detectadas. Una operación de convolución queda definida por 3 parámetros:

- *Kernel*: conjunto de pesos de la neurona convolucional que realizan la operación de convolución con el volumen de entrada de la neurona. Pueden ser multidimensionales.

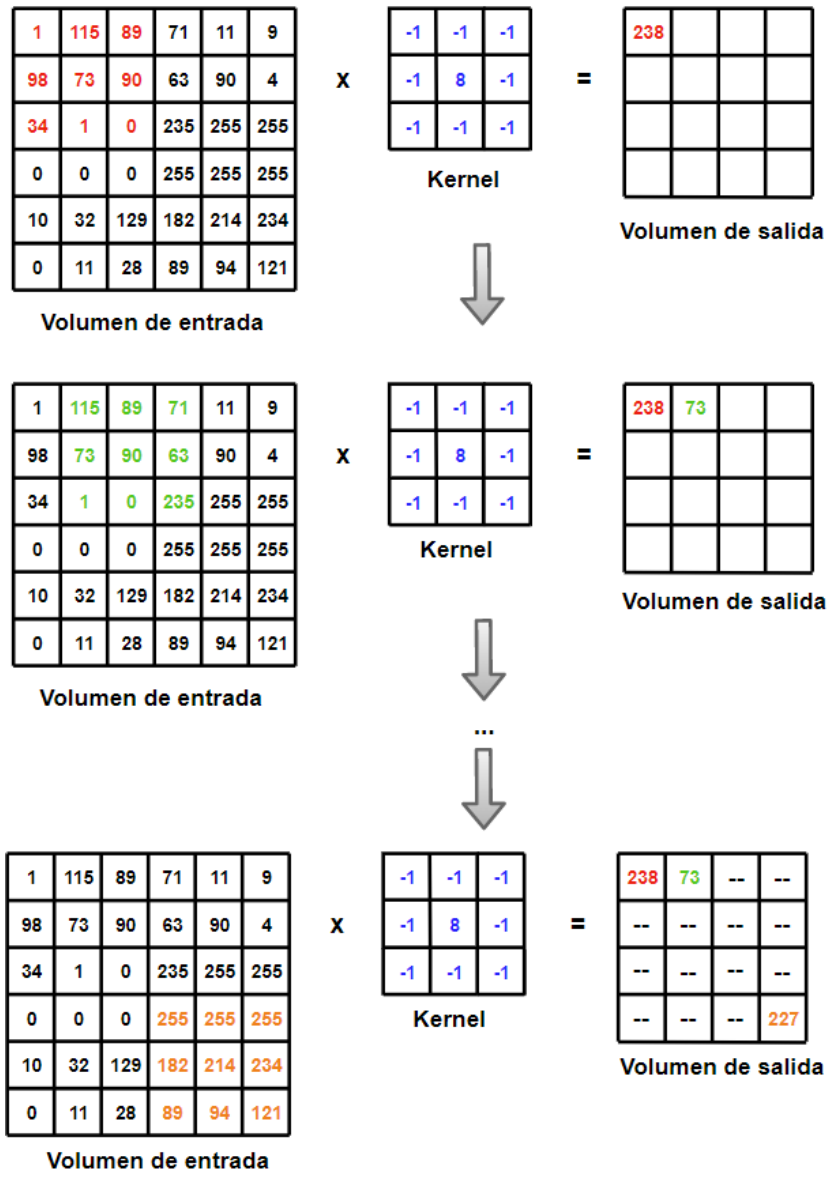

Figura 2.4: Aplicación de un *kernel* 3x3 sobre un volumen de entrada 6x6 ($stride = 0$, $padding = 0$).

- *Stride*: parámetro que rige el movimiento del *kernel* sobre el volumen de entrada. Por ejem-

plo, si el *stride* vale (1,1), el *kernel* se deslizará sobre la imagen de píxel en píxel, de izquierda a derecha y de arriba a abajo. Si el *stride* vale (1,2), el *kernel* se deslizará sobre la imagen de píxel en píxel de izquierda a derecha, pero de 2 píxeles en 2 píxeles de arriba a abajo.

- *Padding*: es la adición de píxeles artificiales en los bordes de las imágenes (Figura 2.5). Se utiliza para permitir que el *kernel* cubra una mayor parte de la imagen y la resuma más adecuadamente al operar más veces sobre los píxeles que conforman los bordes de la imagen.

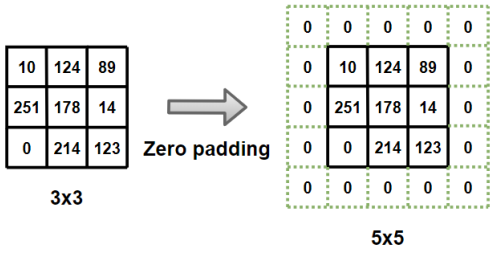

Figura 2.5: Aplicación de *zero padding*.

Cuando se aplica una operación de convolución sobre una entrada, la dimensionalidad de la salida puede calcularse a partir de la siguiente fórmula [7]:

$$O = \frac{I - K + 2P}{S} + 1$$

El significado de los parámetros de la ecuación es el siguiente:

- O: representa la dimensionalidad del volumen de salida de la operación de convolución.
- I: representa la dimensionalidad del volumen de entrada de la la operación de convolución.
- K: representa la dimensionalidad del *kernel* o el tamaño del campo receptivo de las neuronas.
- P: representa el *padding*.
- S: representa el *stride*.

Para ver de manera gráfica el efecto de una convolución, suponer que se tienen dos imágenes de 28x28 píxeles en blanco y negro, por lo que se encuentran en un único canal. Las imágenes representan una camiseta y un zapato y han sido extraídas de [8] (Figura 2.6).

Figura 2.6: Imágenes extraídas del *dataset fashion-mnist*.

A continuación definimos 3 *kernels* de dimensiones 3x3:

$$Kernel\ blur = \begin{pmatrix} 0{,}0625 & 0{,}125 & 0{,}0625 \\ 0{,}125 & 0{,}25 & 0{,}125 \\ 0{,}0625 & 0{,}125 & 0{,}0625 \end{pmatrix}$$

$$Kernel\ bottom\ sobel = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

$$Kernel\ sharpen = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix}$$

Si se aplican los distintos *kernels* sobre las imágenes junto con la relación de parámetros *stride = 1* y *padding = 0* se obtienen los resultados de la Figura 2.7. Como se puede apreciar, cada *kernel* está especializado en una tarea diferente. Por ejemplo, la aplicación del *kernel blur* produce un efecto de emborronamiento de la imagen , mientras que la aplicación del *kernel sharpen* parece resaltar los bordes de los objetos que hay en la imagen. El tamaño de las imágenes resultantes es de 26x26 píxeles, como se puede inferir del siguiente cálculo:

$$(26, 26) = \frac{(28, 28) - (3, 3) + 2 \times 0}{1} + 1$$

El código creado para aplicar la convolución (*kernel blur*, *stride = 1*, *padding = 0*) es el siguiente:

```
kernel = np.array(([0.0625, 0.125, 0.0625],
            [0.125, 0.25, 0.125],
            [0.0625, 0.125, 0.0625]))
```

```
imagen_convolucion = np.zeros((26,26))
for i in range(1,27):
  for j in range(1,27):
    imagen_convolucion[i-1,j-1] = np.sum(imagen[i-1:i+2, j-1:j+2] * kernel)
```

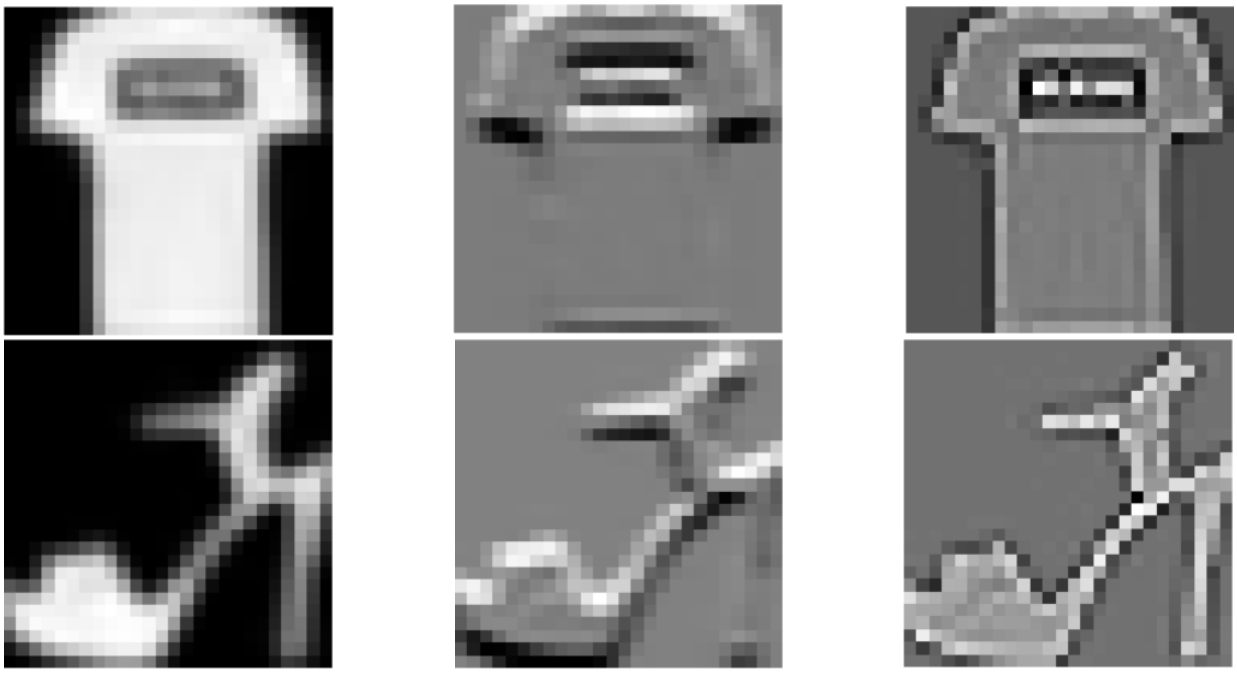

Figura 2.7: Aplicación de los *kernels blur* (izquierda), *bottom sobel* (centro) y *sharpen* (derecha).

La innovación que presentan las capas convolucionales de una red neuronal convolucional es la habilidad para aprender distintos filtros o *kernels* durante la etapa de entrenamiento, resaltando y resumiendo las características de los volúmenes de entrada que sean más importantes para su identificación. En cada operación de convolución se pueden aprender varios filtros diferentes, aumentando así el número de canales originales del volumen de entrada.

2.3.1.2 Operación de *pooling*

La operación de *pooling* permite reducir el tamaño de un mapa de características (*downsampling*) resumiendo las propiedades más importantes del mismo. Su aplicación es imprescindible por dos razones de peso.

Primeramente, al reducir el tamaño de las imágenes y ser una operación muy rápida permite reducir los requisitos computacionales para entrenar las redes donde se aplique.

Segundo, es imprescindible para reducir el sobreajuste que las redes convolucionales cometen de manera natural por la manera en la que están construidas. Las redes convolucionales recuerdan de manera exacta la posición de las características más importantes de las entradas, por lo que pequeñas variaciones de la posición de dichas características resultarán en cálculos diferentes (por ejemplo rotando una imagen o haciendo un *zoom*). Disminuyendo la resolución de la imagen mediante *pooling* se consiguen mantener las características más importantes de las entradas pero sin ser tan dependientes de su localización original.

Al igual que sucedía con la operación de convolución, el *pooling* también dispone de un filtro (conjunto de píxeles que se van a resumir), *stride* (rige el movimiento del filtro sobre el mapa de características) y *padding* (utilizado por si surge una incompatibilidad de dimensiones entre la combinación del filtro y el *stride* con las dimensiones del mapa de características). Las dimensiones del volumen de salida tras aplicar un operación de *pooling* a un volumen de entrada se pueden calcular con la misma fórmula que en el caso de la operación de convolución. Los tipos de *pooling* más frecuentes son el del máximo, el del mínimo y el de la media (Figura 2.8).

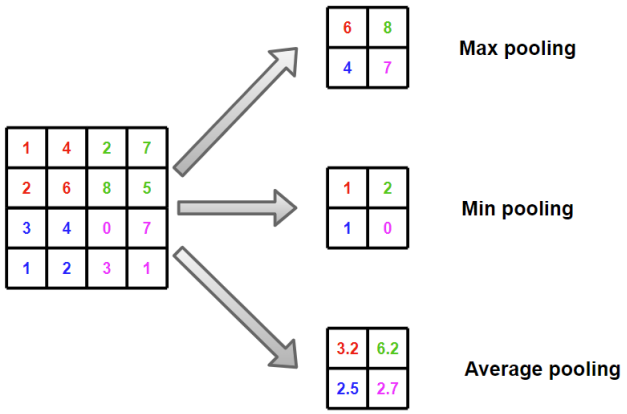

Figura 2.8: Aplicación de *max, min* y *average pooling* 2x2 (*stride = 2*).

Si se aplican los distintos tipos de *pooling* sobre las imágenes obtenidas tras aplicar la convolución con el *kernel blur* junto con la relación de parámetros *stride = 2* y *padding = 0* se obtienen los resultados de la Figura 2.9. El tamaño de las imágenes resultantes es de 13x13 píxeles, como se puede inferir del siguiente cálculo:

$$(13, 13) = \frac{(26, 26) - (2, 2) + 2 \times 0}{2} + 1$$

El código creado para aplicar el *pooling* (máximo, *stride = 2*, *padding = 0*) es el siguiente:

```
imagen_pooling = np.zeros((13,13))
i = 1; j = 1
while i<26:
  while j <26:
    imagen_pooling[int(i/2), int(j/2)] = np.max(imagen_convolucion[i-1:i+1,j-1:
   j+1])
    j += 2
  j = 1
  i += 2
```

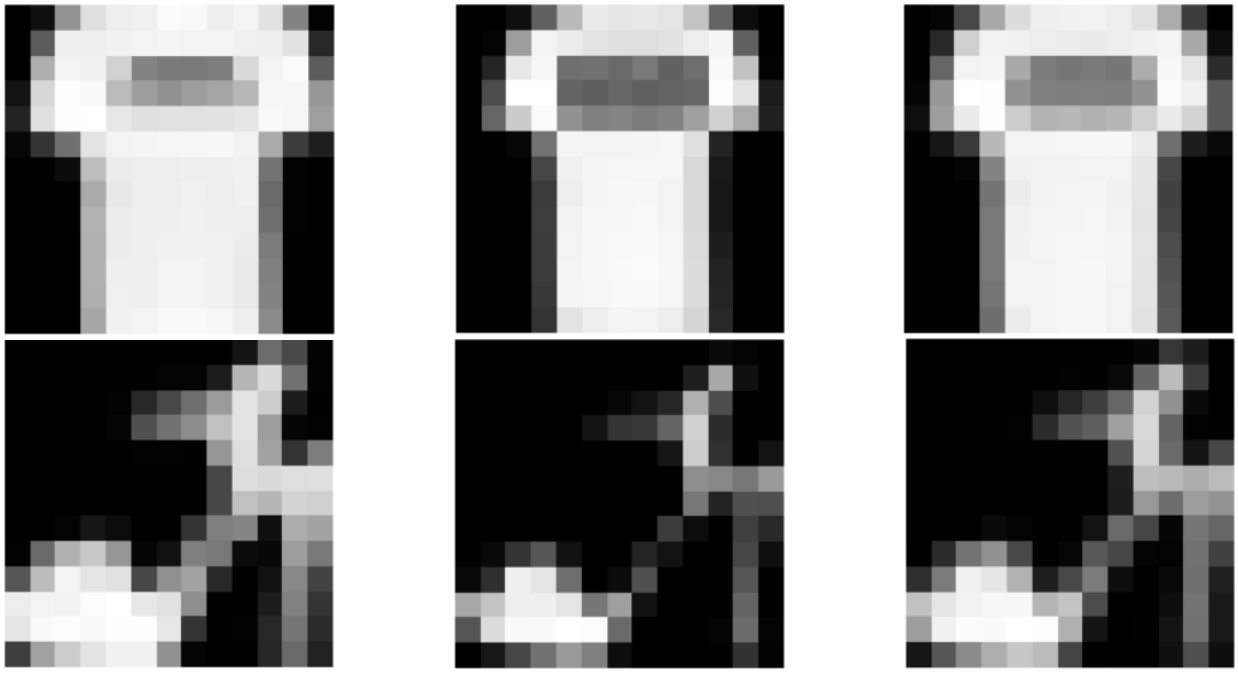

Figura 2.9: Aplicación del *pooling* del máximo (izquierda), mínimo (centro) y media (derecha).

2.3.2. Capas fully-connected

Una vez las capas convolucionales de la red han reducido el tamaño del mapa de características y extraído la información más relevante de los volúmenes de entrada es necesario aplanar el volumen de salida de la última capa convolucional antes de su introducción a la primera capa FC (*fully connected*). Si el volumen de salida está compuesto por α canales de dimensiones $\beta \times \gamma$, entonces de su aplanado se obtendrá como resultado un vector de $\alpha \times \beta \times \gamma$ posiciones. Tanto la Figura 2.10 como la representación de la arquitectura de las redes convolucionales o partes de ellas mostradas en este trabajo han sido realizadas con [9].

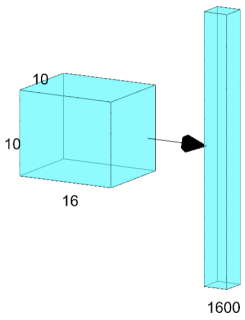

Figura 2.10: Aplanado entre las capas convolucionales y las capas FC.

Una vez se ha realizado el aplanado, las capas FC se corresponden con un perceptrón multicapa clásico. Todas las neuronas de dos capas consecutivas está conectadas entre sí, y en cada neurona se realiza el paso de la suma ponderada de sus entradas por una función de activación no lineal. La capa de entrada tendrá tantas neuronas como posiciones tenga el vector obtenido tras realizar el aplanado del volumen de salida de la última capa convolucional. La capa de salida tendrá tantas neuronas como se requieran para ajustarse al problema: en un problema de clasificación o regresión donde hay k categorías, la capa de salida tendrá k neuronas (Figura 2.11).

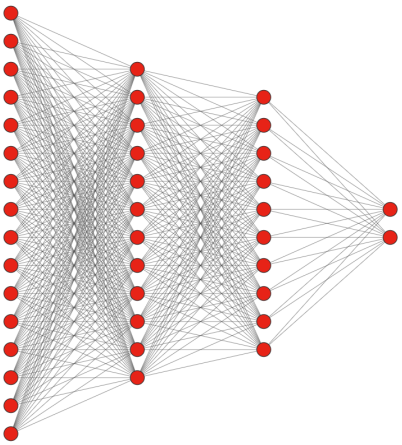

Figura 2.11: Capas FC para un problema con salida dicotómica.

3. Planificación

3.1. Objetivos

El objetivo principal de este proyecto es el desarrollo de técnicas de *deep learning* y su aplicación a imágenes astronómicas recogidas por el telescopio espacial Hubble para caracterizar galaxias distantes.

3.1.1. Actividades

La metodología de trabajo ha sido de tipo ágil, por lo que las actividades necesarias para lograr los objetivos y metas de cada momento han ido variando. Para proporcionar una visión global del proyecto, las actividades realizadas se dividen en 3 etapas.

La primera etapa está dedicada al estudio de conceptos necesarios para poder realizar con éxito el proyecto. En esta etapa se hizo una revisión y aprendizaje de conceptos de redes neuronales y *deep learning*. Se llevó a cabo un estudio del dominio de aplicación al que iban a aplicarse las técnicas a desarrollar: la astrofísica. Además, se aprendió desde cero como utilizar la biblioteca de aprendizaje automático de bajo nivel PyTorch, y el *framework* fastai derivado de PyTorch. La herramienta finalmente elegida para desarrollar las técnicas *deep learning* fue PyTorch debido a la flexibilidad de implementación que ofrece, a costa de la complejidad al tratarse de una biblioteca de bajo nivel.

Una parte importante de esta primera etapa también se ha dedicado al aprendizaje de programas de astronomía y astroinformática, los cuales son mostrados en el Capítulo 4.

La segunda etapa se centra en el desarrollo de algoritmos de *deep learning* con el fin de clasificar morfológicamente galaxias distantes. Este problema se ataca por dos vías diferentes. En primera instancia, mediante un problema de clasificación directo. En segunda instancia, se diseña un problema de regresión con el que, a posteriori, se obtiene la clasificación morfológica.

La tercera y última etapa se centra en la detección de bordes de las galaxias mediante segmentación semántica, el problema más complejo del proyecto. Al igual que en el caso anterior, también se opta por elegir dos enfoques para atacar el problema, un enfoque mediante U-*Nets* (redes neuronales convolucionales en U) y otro mediante aprendizaje por *ensembles*.

Todos los conceptos aprendidos así como las técnicas desarrolladas y el porqué de las elecciones tomadas han sido cuidadosamente documentadas de manera síncrona a su realización.

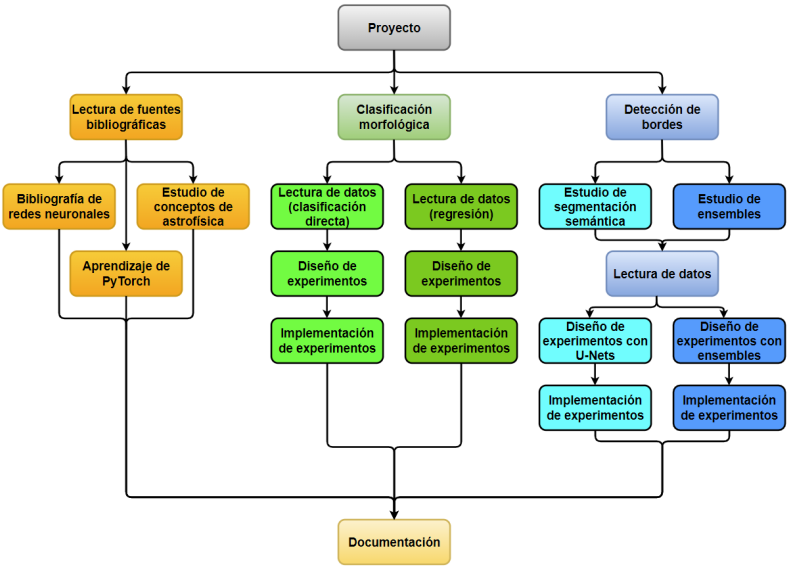

Figura 3.1: Diagrama de descomposición de actividades.

En la Figura 3.1 se encuentra un diagrama de descomposición de las actividades realizadas en el proyecto. El tiempo ocupado por cada actividad en el *timeline* de la realización del proyecto se aprecia en el diagrama de Gantt de la Figura 3.2.

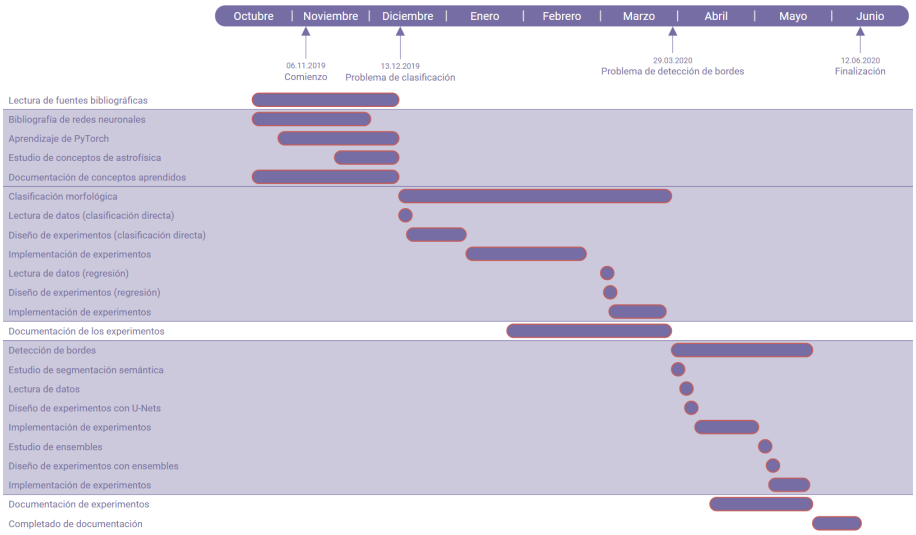

Figura 3.2: Diagrama de Gantt.

3.1.2. Riesgos

La identificación de riesgos del proyecto se encuentra en el Cuadro 3.1. Para la identificación y modo de valoración de los riesgos se utiliza la revisión de métodos para la gestión de riesgos de [10].

Descripción	Probabilidad	Impacto	Riesgo
Falta de bibliografía adecuada	1	10	10
Falta de imágenes suficientes	1	8	8
Imágenes de baja calidad	3	8	24
Imposibilidad de procesar las imágenes	1	8	8
Incapacidad creación algunos modelos	7	2	14
Tiempo excesivo creación modelos	8	3	24
Modelos creados no ajustan bien	5	3	15
Tiempo excesivo para documentación	8	3	24
Finalización tardía del proyecto	5	10	50

Cuadro 3.1: Resultados de los experimentos.

3.2. Metodología de trabajo

El desarrollo del proyecto se ha llevado a cabo mediante reuniones semanales con los tutores. Además, en los primeros estadios del proyecto, las reuniones se hicieron de manera conjunta con otros alumnos que realizaban su proyecto de una temática similar. Dichas reuniones servían para que los tutores verificasen el trabajo realizado, así como fijar nuevos objetivos en base a las metas que se iban alcanzando.

Tanto para el desarrollo de algoritmos *deep learning* para la clasificación morfológica de galaxias distantes como para la detección de bordes, la metodología seguida es del tipo ágil (Figura 3.3). Los requisitos y el carácter de los algoritmos a desarrollar han evolucionado en base a las metas que se iban alcanzando y los problemas que iban surgiendo.

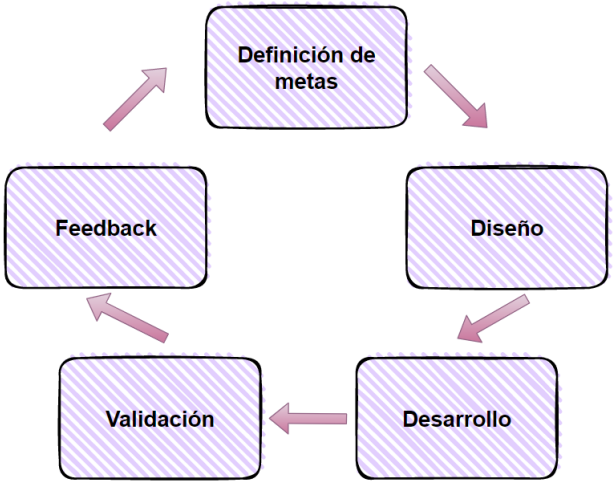

Figura 3.3: Metodología de tipo ágil.

4. Contextualización astronómica

4.1. Conceptos básicos

Los recientes avances en materia de *hardware* y *software* han revolucionado las técnicas clásicas aplicadas en el mundo de la física y de la ciencia en general y han permitido alcanzar horizontes inexplorados hasta el momento.

Estos avances han creado un nuevo paradigma a la hora de entender el papel que juega la información a la hora de generar nuevo conocimiento. Tradicionalmente, el proceso de realizar descubrimientos en el ámbito científico comenzaba formulando una o varias hipótesis. Posteriormente, se recolectaba información y datos con los que, tras construir una teoría, poder validar o refutar la hipótesis planteada (Figura 4.1).

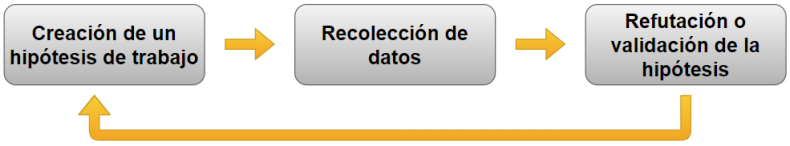

Figura 4.1: Paradigma clásico de generación de conocimiento científico.

Ahora, se disponen de cantidades ingentes de información. Tal cantidad de información no aporta por sí misma un valor añadido del que se pueda sacar provecho, es necesario procesarla y buscar aquellos patrones y comportamientos ocultos realmente interesantes. Para realizar esta tarea, la inteligencia artificial juega un papel dominante. Una vez se han obtenido aquellos datos relevantes, es hora de formular una hipótesis que, posteriormente, habrá que refutar o validar (Figura 4.2). Este cambio de paradigma permite alcanzar un mayor nivel de comprensión del problema al que se hace frente.

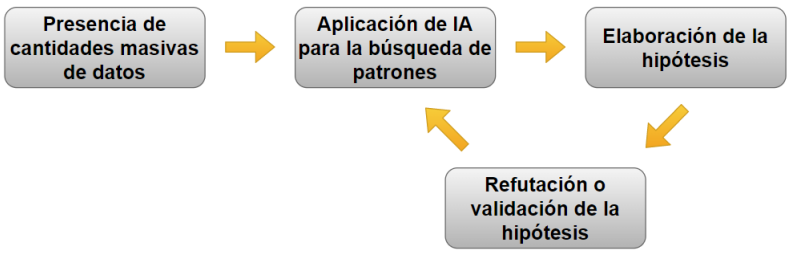

Figura 4.2: Aproximación *data driven* de generación de conocimiento científico.

Esta nueva tendencia ha creado una nueva rama de la física y de la informática que se denomina astroinformática. La astroinformática no se trata únicamente de desarrollar *hardware* con

la capacidad de almacenar y procesar cantidades masivas de datos mandadas desde telescopios, si no que la creación de nuevo *software* juega un papel clave. La creación de *software* para la gestión de bases de datos de gran tamaño, automatizar procesos de recepción y procesamiento de información proveniente de telescopios formando *pipelines*, la creación de observatorios virtuales y el análisis de datos son áreas de trabajo en constante crecimiento y muy importantes. Dentro del análisis de datos, se encuentra la aplicación de técnicas de *machine learning* a información astronómica, concepto sobre el que va a pivotar este proyecto.

Un sistema planetario está formado por una estrella y varios planetas orbitando entorno a ella. De la misma manera, las estrellas se agrupan entre sí, formando estructuras de mayor tamaño llamadas galaxias. Las galaxias pueden dividirse en dos grandes clases, las galaxias elípticas, que tienen una morfología de elipsoide en 3 dimensiones, y las galaxias espirales (Figura 4.3).

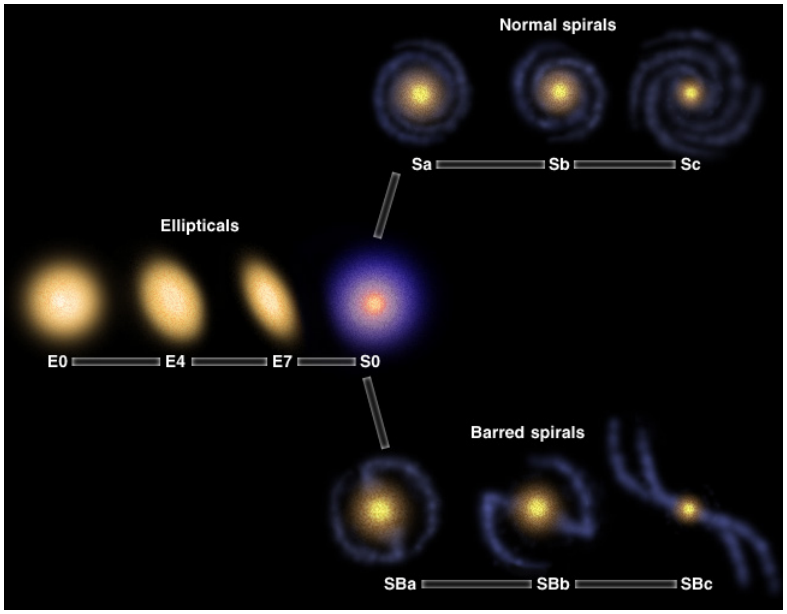

Figura 4.3: Esquema de clasificación de galaxias de Edwin Hubble (extraída de [11]).

La gran distancia que nos separa de las galaxias y el polvo interestelar presente en el universo son causantes de que veamos una masa con luz difusa y no los focos de luz puntual que son las estrellas individuales que forman la galaxia.

El tamaño de las galaxias no es un concepto claro ya que éstas no poseen límites marcados. Cuando tomamos una fotografía de una galaxia, su tamaño varía en función del tiempo de exposición de la fotografía. En la Figura 4.4 se aprecian 4 tomas de la misma galaxia con diferentes tiempos de exposición. La primera toma (parte superior izquierda) ha sido observada con el tiempo normal de exposición usado en el cartografiado de galaxias más utilizado en el universo local. Se

aprecia una galaxia de tipo espiral con unos pocos objetos (galaxias satélites) a su alrededor. Sin embargo, la misma galaxia si es observada durante un mayor tiempo de exposición y con el GTC (Gran Telescopio de Canarias), el telescopio óptico-infrarrojo más grande del mundo, se observan muchos más objetos vecinos a la galaxia (parte inferior derecha). Además, la galaxia pasa de ser un disco perfecto a tener cierta estructura en sus límites.

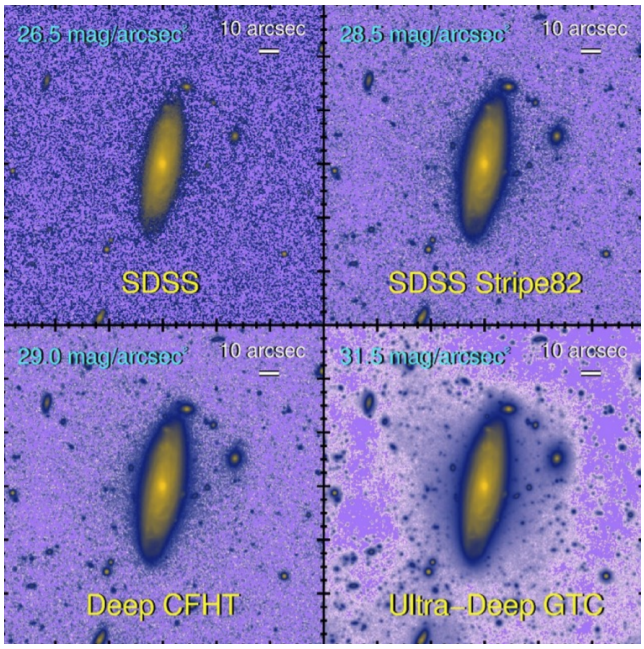

Figura 4.4: Tamaño de las galaxias en función del tiempo de exposición.

Por tanto, y sabiendo que las galaxias son conjuntos de estrellas, se deduce que obteniendo las imágenes con un mayor tiempo de exposición y telescopios más potentes somos capaces de captar las partes más débiles de la misma, reduciendo el ruido de las imágenes.

Otro concepto importante a la hora de tratar con imágenes astronómicas es que éstas no se ven en los colores habituales. Los dispositivos de captura de imágenes utilizados por el público en general (cámaras de fotos, teléfonos móviles...) disponen de 3 filtros para capturar la información, un filtro para el color rojo (R), un filtro para el color verde (G) y un filtro para el color azul (B). La imagen final se logra haciendo una composición de los 3 canales. Los colores utilizados para capturar información y crear imágenes astronómicas no son los colores RGB, si no que vienen de otras partes del espectro electromagnético, es decir, son colores con una longitud de onda distinta.

En la Figura 4.5 se aprecia una captura de una galaxia en diferentes filtros del espectro electromagnético. Los filtros van colocados del más azul (F365W, arriba a la izquierda) hasta el más rojo (F814W, abajo a la derecha).

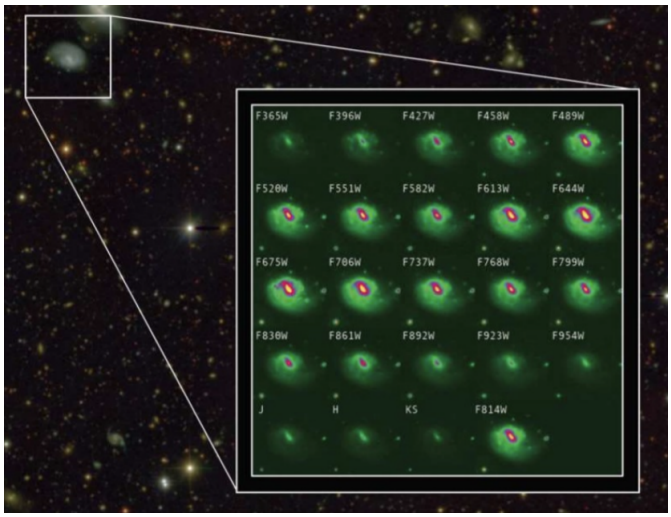

Figura 4.5: Captura de una galaxia en con diferentes filtros del espectro electromagnético (extraída de [12]).

En este trabajo, los filtros en los que se dispone las imágenes astronómicas extraídas del proyecto CANDELS son el filtro H (F160W), el filtro I (F814W), el filtro J (F125W) y el filtro V (F606W) (Figura 4.6).

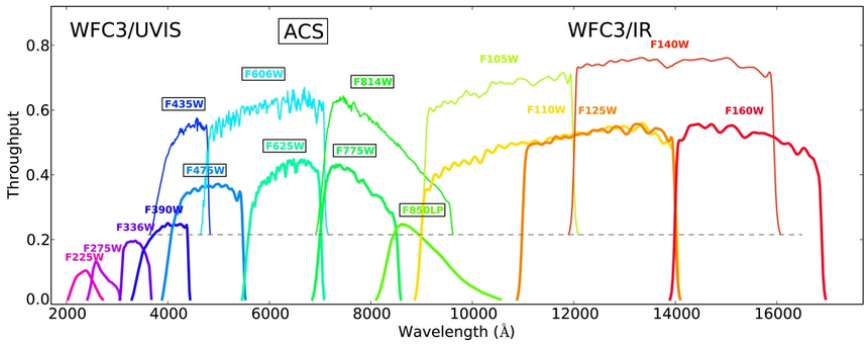

Figura 4.6: 16 filtros HST entre longitudes de onda ∼2000–17000 λ con WFC3/UVIS en el ultravioleta cercano, ACS en el óptico (extendiéndose hasta el infrarrojo cercano), y WFC3/IR en el infrarrojo cercano (extraída de [13]).

4.2. Astroinformática

Durante el desarrollo de este proyecto se ha utilizado *software* astronómico como medio para realizar diversas tareas. A continuación se expone la relación de programas utilizados y una descripción del uso que se les ha dado.

4.2.1. Formato FITS

En astronomía, el formato por excelencia en el que se encuentran las imágenes es el formato FITS (*Flexible Image Trnasport System*). Este formato también es utilizado para almacenar otro tipo de datos como por ejemplo espectros electromagnéticos o listas de fotones. La lectura y tratamiento de ficheros FITS está soportada por lenguajes como C++, Python, Java, Matlab y R entre otros [14].

Un archivo FITS está formado por un cabecera y los datos. La cabecera contiene metadatos acerca de la proveniencia del fichero y de los datos que almacena. Este formato de almacenamiento permite concatenar varios pares cabecera-datos, almacenando en un único lugar el equivalente a varios archivos tradicionales (Figura 4.7).

Figura 4.7: Estructura de un fichero FITS.

Para leer un fichero FITS en Python, lenguaje principal que se ha utilizado en este proyecto, hay que utilizar funciones del paquete *astropy*, biblioteca diseñada para el trabajo astronómico en Python. Concretamente, las siguientes órdenes:

```
from astropy.io import fits
imagen = fits.open("DISK_206_egs_J.fits")
```

En la cabecera de un archivo FITS se pueden apreciar multitud de metadatos. Por ejemplo, el tipo de telescopio utilizado para tomar la imagen, el tiempo que duró la exposición, parámetros del algoritmo de rechazo de rayos cósmicos e información acerca de la calidad de la imagen (Figura 4.8).

```
cabecera = imagen[0].header
```

El la segunda parte de un fichero FITS se encuentran los datos en crudo. En este proyecto, matrices con los valores de los píxeles que conforman la imagen astronómica.

```
  datos = imagen[0].data
```

```
                / COSMIC RAY REJECTION ALGORITHM PARAMETERS

MEANEXP =                  0.0 / reference exposure time for parameters
SCALENSE=                  0.0 / multiplicative scale factor applied to noise
INITGUES= '   '                / initial guess method (MIN or MED)
SKYSUB  = '    '               / sky value subtracted (MODE or NONE)
SKYSUM  =                  0.0 / sky level from the sum of all constituent image
CRSIGMAS= '               '    / statistical rejection criteria
CRRADIUS=                  0.0 / rejection propagation radius (pixels)
CRTHRESH=              0.000000 / rejection propagation threshold
BADINPDQ=                    0 / data quality flag bits to reject
REJ_RATE=                  0.0 / rate at which pixels are affected by cosmic ray
CRMASK  =                    F / flag CR-rejected pixels in input files (T/F)
```

Figura 4.8: Metadatos de un fichero FITS.

4.2.2. SAOImageDS9

SAOImageDS9 es una herramienta de visualización de imágenes y datos astronómicos. El formato de entrada que soporta DS9 son archivos *fits* o tablas binarias. Proporciona funcionalidad para la manipulación de regiones dentro de una imagen, multitud de algoritmos para realizar escalados y transformaciones de las imágenes y una amplia gama de mapas de colores [15].

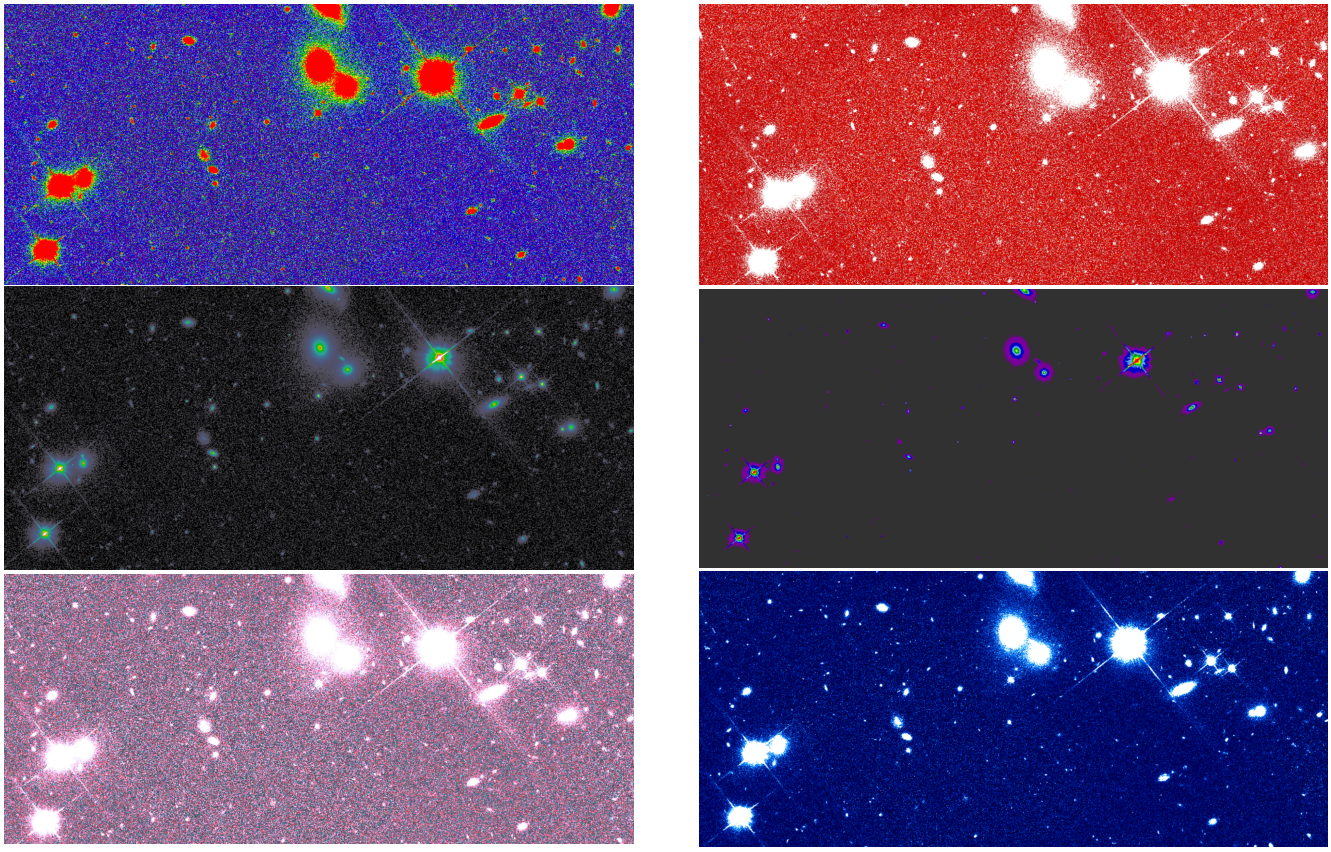

Figura 4.9: Imágenes creadas con SAOImageDS9.

Los diferentes filtros y transformaciones que se aplican a una imagen pueden resultar críticos para resaltar partes más débiles de la misma, u objetos que resulten de interés. Por ejemplo, en

la Figura 4.9 se aprecia una misma imagen a la que se han realizado 6 procesamientos diferentes. Para obtenerlas se han modificado el rango de valores que pueden tomar los píxeles, el mapa de colores y/o la escala en la que se encuentra cada una.

El uso que se le ha dado al programa durante el desarrollo del proyecto ha sido principalmente para el aprendizaje de conceptos de astronomía mediante la visualización de imágenes astronómicas. Por ejemplo, un concepto muy fácil de aprender es la discriminación entre estrellas y galaxias en imágenes como las mostradas en la Figura 4.9. Las estrellas presentan prolongaciones de luz en forma de cruz, mientras que las galaxia no.

4.2.3. Aladin Sky Atlas

Aladin Sky Atlas es atlas interactivo que permite visualizar imágenes astronómicas o regiones completas (*surveys*). Además, incluye numerosas bases de datos y catálogos astronómicos y accesos a conocidos repositorios de datos astronómicos como Simbad (base de datos astronómica de objetos que se encuentran más allá del sistema solar) o VizieR (servicio de catálogos astronómicos proporcionado por el *Centre de Données astronomiques de Strasbourg*) [16].

Entre las características que hacen interesantes el uso de esta plataforma se encuentran la facilidad para la interconexión con otras herramientas de visualización y análisis astronómico, la localización y selección de partes de interés de regiones de gran tamaño (Figura 4.10) y el acceso y exploración de *datasets* distribuidos.

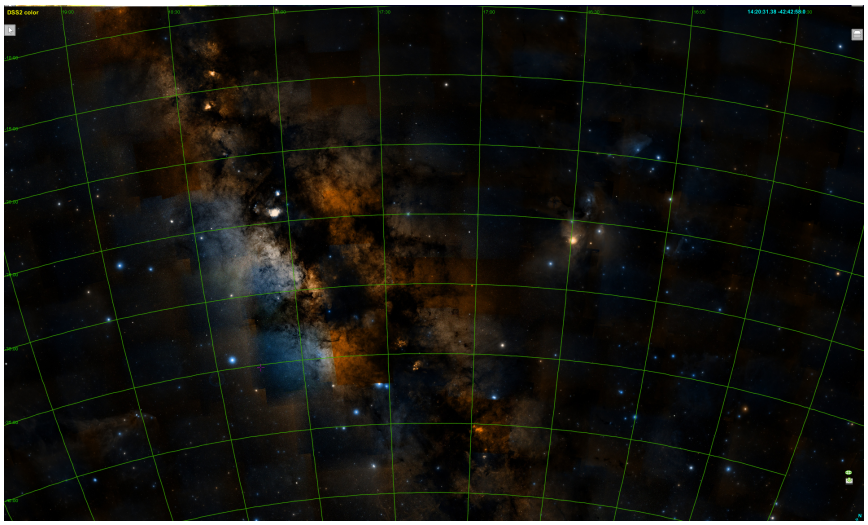

Figura 4.10: Malla de coordenadas creada sobre una nebulosa en Aladin.

Aladin permite la identificación de cuerpos astronómicos en múltiples *surveys* mediante los catálogos de Simbad y VizieR. En la Figura 4.11 se muestra la identificación del pulsar PSR

J1909-3744.

Figura 4.11: Identificación del púlsar PSR J1909-3744.

El uso principal que se le ha dado a la herramienta fue en el comienzo del proyecto para descubrir y asentar conocimientos astronómicos, como por ejemplo la localización, identificación y recorte de estrellas y galaxias.

4.2.4. SExtractor

SExtractor es un *software* astronómico que construye un catálogo de objetos a partir de una imagen astronómica [17]. Es utilizado para la detección automática y la fotometría, cantidad de radiación, de los cuerpos que se encuentren dentro de regiones que pueden ser de gran tamaño. Dicho programa está especialmente diseñado para ofrecer de manera rápida una fotometría de calidad razonable de todos los cuerpos detectados en imágenes de gran tamaño.

Para ilustrar el uso de SExtractor, se utilizará la imagen astronómica de la Figura 4.12.

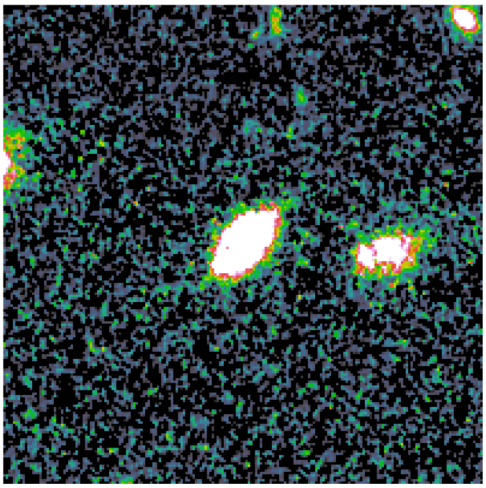

Figura 4.12: Imagen analizada por SExtractor.

Para utilizar SExtractor se necesita crear, o utilizar los disponibles por defecto, diferentes

ficheros. Lo primero de todo hay que definir que características se desean obtener de los objetos que el *software* detecte en la imagen. Dicho fichero ha de tener la extensión *.param*. En la Figura 4.13 se aprecia una porción de un posible fichero donde especificamos que parámetros deseamos obtener. En él, aparecen comentadas las características que no se desean obtener, y sin comentar las que se desea disponer para cada objeto que se detecte. La relación de parámetros que se pueden obtener se pueden encontrar en [18]. Este fichero tiene que ser especificado en el fichero de configuración que se explicará a continuación.

```
NUMBER

ALPHA_J2000
DELTA_J2000

#FLUX_ISO
#FLUXERR_ISO
#MAG_ISO
#MAGERR_ISO

#FLUX_ISOCOR
#FLUXERR_ISOCOR
#MAG_ISOCOR
#MAGERR_ISOCOR

#FLUX_APER(1)
#FLUXERR_APER(1)
#MAG_APER(1)
#MAGERR_APER(1)

#FLUX_AUTO
#FLUXERR_AUTO
#MAG_AUTO
#MAGERR_AUTO

#FLUX_BEST
#FLUXERR_BEST
MAG_APER
MAGERR_APER
```

Figura 4.13: Porción de fichero de parámetros de SExtractor.

Otro fichero que se necesita es el de configuración, que ha de tener la extensión *.sex*. En él se indican cuestiones relativas a la actuación del programa sobre la imagen durante su ejecución. Por ejemplo, se puede definir el número mínimo de píxeles agrupados con intensidad mayor a un umbral necesarios para considerarlos como un cuerpo individual. Si este fichero no se crea SExtractor utilizará el que incluye por defecto. Una porción de un posible fichero de configuración está en la Figura 4.14.

```
#------------------------------ Photometry -----------------------------------

PHOT_APERTURES  10.0            # MAG_APER aperture diameter(s) in pixels
PHOT_AUTOPARAMS        1.0, 1.0       # MAG_AUTO parameters: <Kron_fact>,<min_radius>
PHOT_AUTOAPERS  1.0,1.0               # MAG_AUTO minimum diameters for detection,estimation
PHOT_FLUXFRAC   0.5           # CALCULATE RADII CONTAINING 20,50,80% OF TOTAL FLUX

SATUR_LEVEL     30000.0               # level (in ADUs) at which arises saturation

MAG_ZEROPOINT   25.9463               # magnitude zero-point
MAG_GAMMA       4.0             # gamma of emulsion (for photographic scans)
GAIN            2.6             # detector gain in e-/ADU.
PIXEL_SCALE     0.1             # size of pixel in arcsec (0=use FITS WCS info).

#------------------------- Star/Galaxy Separation ----------------------------

SEEING_FWHM     0.2             # stellar FWHM in arcsec
STARNNW_NAME    default.nnw     # Neural-Network_Weight table filename
```

Figura 4.14: Porción de fichero de configuración de SExtractor.

Un último fichero necesario en un archivo con extensión *.conv* donde se indica el *kernel* de convolución que aplicará SExtractor como parte del cálculo del PSF (*Point Spread Function*)

(Figura 4.15). Dicho fichero tiene que ser especificado en el fichero de configuración, y al igual que en los casos anteriores SExtractor ofrece un núcleo de convolución por defecto en el caso de que no se especifique uno propio.

```
1 CONV NORM
2 # 5x5 convolution mask of a gaussian PSF with FWHM = 3.0 pixels.
3 0.092163 0.221178 0.296069 0.221178 0.092163
4 0.221178 0.530797 0.710525 0.530797 0.221178
5 0.296069 0.710525 0.951108 0.710525 0.296069
6 0.221178 0.530797 0.710525 0.530797 0.221178
7 0.092163 0.221178 0.296069 0.221178 0.092163
```

Figura 4.15: Fichero con el *kernel* de convolución.

Una vez se dispone de estos ficheros, se puede ejecutar el programa sobre una imagen. El formato de entrada más habitual es *.fits*. La manera de ejecutar el programa es desde consola, en sistemas Linux la orden es la mostrada en la Figura 4.16. Ha de especificarser la imagen a la que aplicar el procesamiento y el fichero de configuración.

```
source-extractor DISK_206_egs_J.fits  -PARAMETERS_NAME default.param
```

Figura 4.16: Orden para ejecutar SExtractor.

Una vez ejecutada la orden, el programa comenzará a analizar la imagen. Para imágenes de gran tamaño, varios *gigabytes*, el cómputo puede durar unos minutos. Una vez éste ha finalizado, la salida que se obtiene será similar a la mostrada en la Figura 4.17. En ella se indica el número de objetos que se han detectado, 34 en la ejecución mostrada, y el número de objetos a los que se han calculado las medidas especificadas en el fichero de parámetros según las restricciones que se hayan definido en el archivo de configuración, 21 en la ejecución mostrada.

```
----- Source Extractor 2.25.0 started on 2021-05-29 at 13:16:19 with 1 thread

----- Measuring from: DISK_206_egs_J.fits
      "EGS_ALL_WFC3_IR_F125" / no ext. header / 200x200 / 64 bits (floats)
(M+D) Background: 6.43594e-05 RMS: 0.00793959 / Threshold: 0.0111154
      Objects: detected 34       / sextracted 21

> All done (in 0.0 s: 4932.3 lines/s , 517.9 detections/s)
```

Figura 4.17: Salida obtenida al ejecutar SExtractor.

Tras realizar la ejecución, se genera un fichero con los resultados (Figura 4.18). El nombre y ruta del fichero son especificados en el fichero de configuración. En él, aparecen todos los objetos que cumplen las características especificadas en el fichero de configuración junto con la relación de medidas indicadas en el archivo de parámetros. Además, en las primeras líneas aparece una breve descripción de las medidas tomadas y las unidades en las que se encuentran.

```
 1 #   1 NUMBER                 Running object number
 2 #   2 ALPHA_J2000            Right ascension of barycenter (J2000)                    [deg]
 3 #   3 DELTA_J2000            Declination of barycenter (J2000)                        [deg]
 4 #   4 MAG_APER               Fixed aperture magnitude vector                          [mag]
 5 #   5 MAGERR_APER            RMS error vector for fixed aperture mag.                 [mag]
 6 #   6 X_IMAGE                Object position along x                                  [pixel]
 7 #   7 Y_IMAGE                Object position along y                                  [pixel]
 8 #   8 FLUX_RADIUS            Fraction-of-light radii                                  [pixel]
 9 #   9 ELLIPTICITY            1 - B_IMAGE/A_IMAGE
10 #  10 THETA_IMAGE            Position angle (CCW/x)                                   [deg]
11 #  11 KRON_RADIUS            Kron apertures in units of A or B
12 #  12 A_IMAGE                Profile RMS along major axis                             [pixel]
13         1 215.2626633 +53.0282447  22.4570   0.1351    101.2587    101.0384      5.293    0.440  48.81  1.42     8.289
14         2 215.2656348 +53.0277920  26.8911   1.0515     52.0088      1.9892     1.563    0.339   0.15  2.67     1.098
15         3 215.2607356 +53.0291273  26.5090   0.8821    106.3905    188.3161     1.755    0.281  56.61  2.81     1.220
16         4 215.2590485 +53.0280969  24.2246   0.3052    192.9584    194.2335     2.751    0.228 -46.24  1.43     3.519
17         5 215.2609629 +53.0286137  26.3801   0.8303    124.4304    162.0147     1.801    0.434 -81.53  2.11     1.742
18         6 215.2636715 +53.0296411  26.9520   1.0875     13.9863    128.0142     1.226    0.563  23.48  1.61     1.773
19         7 215.2634468 +53.0299104  25.6001   0.5769      7.0067    144.6940     3.249    0.160  63.21  2.97     2.191
20         8 215.2637507 +53.0298400  25.0171   0.4400      3.0728    133.6217     5.005    0.656 -88.24  1.77     7.225
21         9 215.2613506 +53.0285088  26.7347   0.9812    120.0906    147.3039     1.436    0.291  75.20  2.63     1.062
22        10 215.2628560 +53.0294615  99.0000  99.0000     41.3318    143.3219     0.892    0.412 -44.54  3.03     0.572
23        11 215.2604936 +53.0290783  25.7253   0.6115    114.3089    193.0244     3.184    0.568 -84.96  2.18     3.287
24        12 215.2623881 +53.0278555  27.0786   1.1551    125.4435     93.3617     0.946    0.324 -36.83  1.44     1.175
25        13 215.2618681 +53.0275352  24.4319   0.3359    152.2499     95.0798     4.151    0.384 -76.60  2.05     4.472
26        14 215.2619115 +53.0272771  26.4168   0.8447    162.9807     83.8019     2.627    0.473  42.29  3.29     2.175
27        15 215.2616319 +53.0274532  23.9415   0.2678    161.5329     98.3403     6.130    0.275   7.49  1.96     6.517
28        16 215.2617021 +53.0277598  26.2683   0.7879    145.9292    108.4013     2.790    0.392  -8.42  2.71     2.108
29        17 215.2647296 +53.0285801  26.8947   1.0583     37.4117     57.5752     1.638    0.589 -66.34  2.42     1.427
30        18 215.2623743 +53.0266290  27.1923   1.2193    181.6006     45.7998     1.422    0.424  71.44  3.08     0.775
31        19 215.2657804 +53.0286012  27.7143   1.5695     11.7519     29.6310     0.794    0.320  18.06  1.13     0.955
32        20 215.2649303 +53.0277676  27.8991   1.7187     69.6793     20.3236     0.928    0.404  45.14  3.20     0.568
33        21 215.2647642 +53.0275281  27.3892   1.3405     84.4904     15.5096     0.783    0.000  45.20  3.00     0.500
```

Figura 4.18: Fichero de salida obtenido al ejecutar SExtractor.

Un ejemplo de las medidas recogidas para cada objeto es *THETA_IMAGE*, el ángulo en sentido antihorario que forma el eje x con el semieje mayor de la galaxia (Figura 4.19).

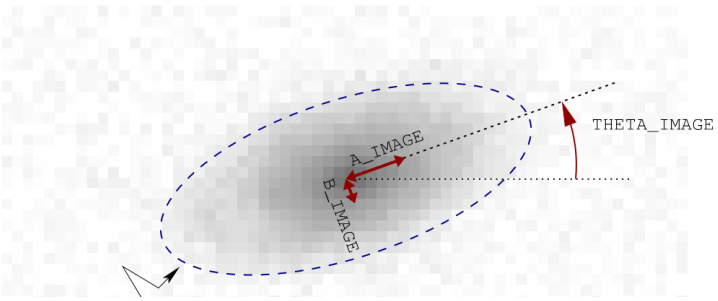

Figura 4.19: Ángulo entre el eje x y el semieje mayor de la galaxia (CCW) (imagen extraída de [18]).

Comprobando uno de los valores calculados, nos fijamos en el primer objeto detectado, cuyo centro se encuentra en el punto 101.2587, 101.0384 de las coordenadas x e y respectivamente. Localizando ese punto en la imagen con la ayuda de programas como SAOImageDS9 o Aladin vemos que ese objeto se corresponde con la galaxia central de la imagen. Si calculamos manualmente el ángulo entre el eje x y el semieje mayor de la galaxia nos damos cuenta que ese valor es el ofrecido por SExtractor en la columna 10, 48.81°(Figura 4.20).

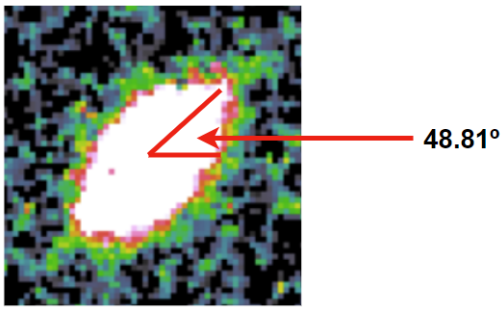

Figura 4.20: Ángulo entre el eje x y el semieje mayor de la galaxia (CCW) en la galaxia central de la imagen.

Tareas como la mostrada cuya complejidad no es muy alta se vuelven irrealizables de una manera manual cuando han de aplicarse a todos y cada uno de los objetos que se encuentren en imágenes como la de la de la Figura 4.21.

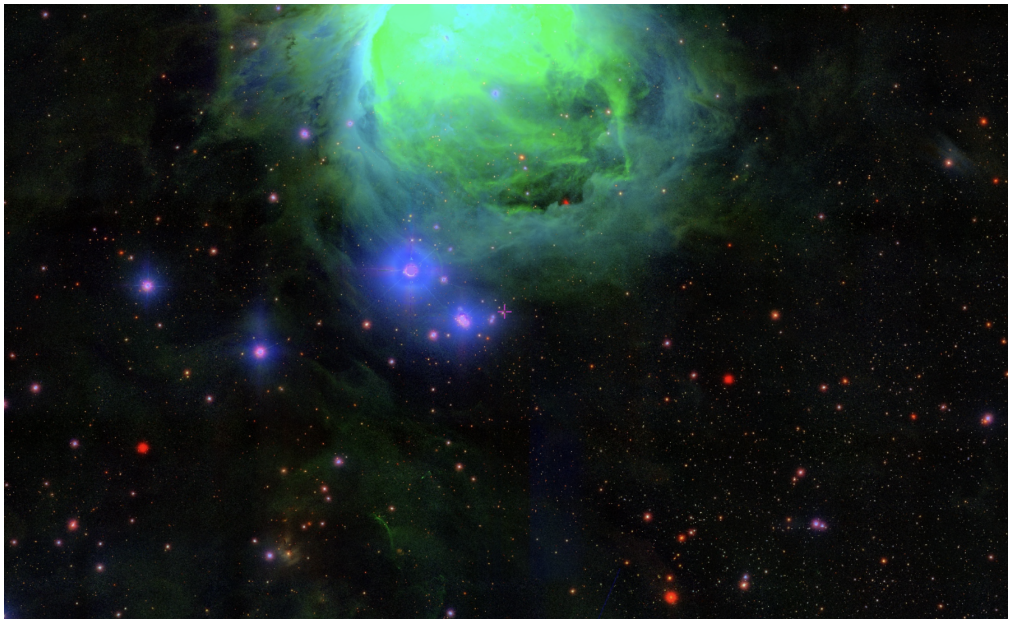

Figura 4.21: Imagen perteneciente a *Sloan Digital Sky Survey III* generada con Aladin.

5. Contexto Tecnológico

5.1. PyTorch

PyTorch es una biblioteca de *machine learning* de código abierto desarrollado por el Laboratorio de Investigación de Inteligencia Artificial de Facebook [19]. Su uso está extendido para aplicaciones relacionadas con la visión artificial y el procesamiento del lenguaje natural. Presenta dos interfaces, una en Python (la que se va a usar a lo largo de este trabajo) y otra en C++.

PyTorch se trata de una biblioteca de bajo nivel que se caracteriza por la necesidad de interactuar directamente con tensores (Figura 5.1). Un tensor es una entidad algebraica de varios componentes que generaliza los conceptos de escalar, vector y matriz de una manera que sea independiente de cualquier sistema de coordenadas elegido [20].

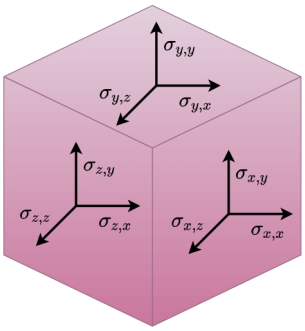

Figura 5.1: Tensor en $\mathbb{R}^3$.

```
import torch
tensor_escalar = torch.tensor(3)
tensor_tridimensional = torch.rand((10,10,10))
```

Las operaciones con los tensores se realizan de manera muy similar a las operaciones que se efectúan con la biblioteca *NumPy*.

```
maximo = torch.max(tensor_tridimensional)
minimo = torch.min(tensor_tridimensional)
indice_maximo = torch.argmax(tensor_tridimensional)
indice_minimo = torch.argmin(tensor_tridimensional)
```

Otra de las características que tiene PyTorch es que permite el uso de la GPU para realizar operaciones, característica fundamental ya que las tarjetas gráficas están optimizadas para en-

trenar inteligencia artificial y *deep learning* al permitir realizar de manera simultánea múltiples operaciones.

```
torch.cuda.get_device_name(0)
```

Es conveniente almacenar una referencia a la GPU, ya que todas las variables que se creen y quieran introducirse a la memoria RAM de la tarjeta gráfica para realizar algún cómputo habrán de ser transferidas de forma manual.

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
tensor = torch.tensor(10)
tensor.to(device)
```

Una manera sencilla de saber si un tensor se encuentra en la tarjeta gráfica es imprimiendo su valor (Figura 5.2).

```
tensor(10, device='cuda:0')
```

Figura 5.2: Tensor alojado en la GPU.

5.1.1. Clase *Dataset*

Una estructura que provee PyTorch de gran utilidad es la clase *Dataset*. Se trata de una clase abstracta de la que se puede heredar para crear un *dataset* propio personalizable. Una clase que represente un *dataset* y que herede de la clase *Dataset* tiene que implementar como mínimo 2 métodos: *__len__* (que retorna el número de instancias en el conjunto de datos) y *__getitem__* (que sirve para acceder a la instancia i-ésima del conjunto de datos). A continuación se crea una clase que representa un *dataset* propio. Al ser creada, recibe como argumentos un *array* de *NumPy* multidimensional representando la variable independiente x y un un *array* de *NumPy* unidimensional representando la variable dependiente y, se transforman a tensor y se almacenen como atributos de la clase.

```
class MyDataset(Dataset):
  def __init__(self, data, target):
    self.data = torch.from_numpy(data).float()
    self.target = torch.from_numpy(target).long()

  def __getitem__(self, index):
    x = self.data[index]
    y = self.target[index]
    return x, y
```

```
  def __len__(self):
    return len(self.data)

dataset = MyDataset(x, y)
```

También existe la opción de crear instancias de la clase *Dataset* a partir de conjuntos de datos que se encuentran en repositorios mantenidos por *TorchVision* u otros lugares. Por ejemplo, para crear una instancia de *Dataset* del conocido conjunto de datos Fashion-MNIST, basta con escribir las siguientes órdenes:

```
dataset = datasets.FashionMNIST(
  root="directorio_de_datos/",
  train=True,
  download=True,
  transform=ToTensor()
)
```

En *root* se especifica el directorio donde se almacenarán los datos si no están ya almacenados, *train* indica si el conjunto de datos es de entrenamiento o no, *download* = *True* descargará los datos del repositorio de *TorchVision* y los almacenará en el directorio indicado en *root*, y *transform* indica que transformaciones a aplicar a cada elemento del conjunto de datos.

Existe la posibilidad de concatenar varias transformaciones seguidas sobre cada elemento del conjunto de datos. Para ello, se utiliza la función *Compose* de *torchvision.transforms*. Por ejemplo, en el siguiente trozo de código se realiza un recorte aleatorio de 224 x 224 píxeles de una imagen, se transforma a tensor y se normalizan cada uno de sus 3 canales con distintos valores de media y desviación típica.

```
datos_transformados = transforms.Compose([
  transforms.RandomSizedCrop(224),
  transforms.ToTensor(),
  transforms.Normalize(mean=[0.4,0.5,0.4], std=[0.1,0.2,0.2])
])
```

5.1.2. Clase *DataLoader*

A partir de una instancia de una clase que herede de *Dataset* se puede crear una instancia de la clase *DataLoader*. Representa un iterable de Python sobre la clase que hereda de *Dataset*. Tiene como características [21]:

- Se puede personalizar el orden de las instancias.

- Generado de *batchs* automáticos.

- Posibilidad de realizar la carga de datos en paralelo (argumento *num_workers*).

- *Pinning* de memoria automático, es decir copias de los datos desde el *host* a la GPU de manera eficiente (argumento *pin_memory*).

Un ejemplo de creación de dos instancias de la clase *DataLoader* es la siguiente:

```
dataloader_1 = DataLoader(dataset, batch_size=28, shuffle=True)
dataloader_2 = DataLoader(dataset, batch_size=12, num_workers = 4, pin_memory =
    True)
```

5.1.3. Modelos

Una vez se tienen los datos preparados, hay que crear el modelo que se quiere utilizar y definir una serie de parámetros adicionales. En este caso, se va a mostrar como utilizar una red neuronal convolucional de *transfer learning*, es decir que la red ya ha sido entrenada en un contexto específico y los pesos ya están ajustados. También se mostrará como crear una red de diseño propio. En el primer caso, el modelo que se utilizará será AlexNet [22]. Nada más cargar el modelo, lo migramos a la GPU.

```
model = torch.hub.load('pytorch/vision:v0.9.0', 'alexnet', pretrained=True)
model.to(device)
```

Si queremos inspeccionar la arquitectura del modelo (Figura 5.3), basta con utilizar la función *summary*, a la cual la indicamos el modelo y las dimensiones de los tensores de entrada de la red. En este caso, la red AlexNet espera imágenes en 3 canales de, al menos, 224x224 píxeles, por lo que un tensor válido de entrada a la red puede tener las dimensiones (3,224,224).

```
summary(model, (3, 224, 224))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 64, 55, 55]          23,296
              ReLU-2          [-1, 64, 55, 55]               0
         MaxPool2d-3          [-1, 64, 27, 27]               0
            Conv2d-4         [-1, 192, 27, 27]         307,392
              ReLU-5         [-1, 192, 27, 27]               0
         MaxPool2d-6         [-1, 192, 13, 13]               0
            Conv2d-7         [-1, 384, 13, 13]         663,936
              ReLU-8         [-1, 384, 13, 13]               0
            Conv2d-9         [-1, 256, 13, 13]         884,992
             ReLU-10         [-1, 256, 13, 13]               0
           Conv2d-11         [-1, 256, 13, 13]         590,080
             ReLU-12         [-1, 256, 13, 13]               0
        MaxPool2d-13           [-1, 256, 6, 6]               0
AdaptiveAvgPool2d-14           [-1, 256, 6, 6]               0
          Dropout-15                [-1, 9216]               0
           Linear-16                [-1, 4096]      37,752,832
             ReLU-17                [-1, 4096]               0
          Dropout-18                [-1, 4096]               0
           Linear-19                [-1, 4096]      16,781,312
             ReLU-20                [-1, 4096]               0
           Linear-21                [-1, 1000]       4,097,000
================================================================
Total params: 61,100,840
Trainable params: 61,100,840
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 8.38
Params size (MB): 233.08
Estimated Total Size (MB): 242.03
----------------------------------------------------------------
```

Figura 5.3: Resumen de la red AlexNet.

La manera de crear una red a nuestro gusto es creando una clase que herede de *torch.nn.Module*. Dicha clase ofrece una amplia gama de operaciones y funcionalidad en el ámbito de la creación de redes neuronales: funciones de activación, operaciones, capas... En el interior de la clase habrá que crear al menos 2 métodos: el constructor (*__init__*) donde se definirán las capas y operaciones que conformen la red y *forward*, que indica el flujo de los datos a través de la red diseñada. En el código que se muestra a continuación, los datos fluyen a través de una primera capa convolucional, seguida de una capa FC y por último son pasados a través de la función sigmoide.

```
class ConvNet(nn.Module):
  def __init__(self):
    super(ConvNet, self).__init__()
    self.layer1 = nn.Sequential(
      nn.Conv2d(1, 3, kernel_size=10, stride=1, padding=0),
      nn.ReLU(),
      nn.MaxPool2d(10, stride=1))
    self.fc1 = nn.Linear(3*82*82, 2)
    self.act = nn.Sigmoid()
  def forward(self, x):
    out = self.layer1(x)
    out = out.reshape(out.size(0), -1)
    out = self.act(self.fc1(out))
  return out
```

Para inspeccionar la arquitectura del modelo, igualmente se puede utilizar la función *summary* (Figura 5.4). La red creada espera imágenes en 1 canal de 100x100 píxeles, por lo que un tensor válido de entrada a la red tendrá las dimensiones (1,100,100).

```
summary(model, (1, 100, 100))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 3, 91, 91]             303
              ReLU-2          [-1, 3, 91, 91]               0
         MaxPool2d-3          [-1, 3, 82, 82]               0
            Linear-4                  [-1, 2]          40,346
           Sigmoid-5                  [-1, 2]               0
================================================================
Total params: 40,649
Trainable params: 40,649
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.04
Forward/backward pass size (MB): 0.53
Params size (MB): 0.16
Estimated Total Size (MB): 0.73
----------------------------------------------------------------
```

Figura 5.4: Resumen de la red creada heredando de la clase *torch.nn.Module*.

Otro punto importante es la elección de la función de pérdida y el optimizador que se utilizarán para ajustar el modelo, PyTorch ofrece múltiples opciones. En el caso del *loss*, serán funciones del paquete *torch.nn*, mientras que el optimizador será una función del paquete *torch.optim*.

```
opt_1 = torch.optim.RMSprop(model.parameters(), lr=0.001)
opt_2 = torch.optim.SGD(model.parameters(), lr=0.01, momentum=0.9)
opt_3 = torch.optim.Adam([valor_1, valor_2], lr=0.0001)
```

```
loss_1 = torch.nn.CrossEntropyLoss()
loss_2 = torch.nn.MSELoss()
loss_3 = torch.nn.SmoothL1Loss()
```

5.1.4. Bucle de entrenamiento

En PyTorch, el bucle de entrenamiento de un algoritmo de *deep learning* hay que hacerle de manera manual. Para ello, se obtiene un *batch* proporcionado por la instancia de la clase *DataLoader*. Si en la creación de la instancia de *DataLoader* los datos no han sido migrados a la memoria RAM de la GPU, hay que hacerlo manualmente. Si no, el modelo no podrá procesar los datos al encontrarse en la memoria de la GPU. Tanto los datos como el modelo han de encontrarse en el mismo dispositivo.

```
for i, data in enumerate(dataloader, 0):
  inputs, labels = data[0].to(device), data[1].to(device)
```

El siguiente paso consiste en asignar el valor 0 a todos los gradientes antes de comenzar la etapa de *backpropagation*. Esto es necesario porque PyTorch acumula los gradientes en diversas pasadas de la retropropagación, por lo que antes de la introducción de cada *batch* esta acción es necesaria.

```
optimizer.zero_grad()
```

El flujo del *batch* a través de la red se realiza introduciendo en el modelo los datos proporcionados por la instancia de *DataLoader*. El resultado del paso de la información a través de la red generará una salidas, las cuales se comparan con las etiquetas también proprocionadas por la instancia de *DataLoader* mediante la función de pérdida que se haya definido.

```
outputs = model(inputs)
loss = criterion(outputs, labels)
```

Una vez realizados estos pasos, solo queda optimizar el *loss* en la etapa de retropropagación y actualizar el valor de los pesos del modelo en base a los gradientes calculados.

```
loss.backward()
optimizer.step()
```

Recopilando toda la información acerca del entrenamiento, un bucle muy sencillo tiene el siguiente aspecto:

```
for i, data in enumerate(dataloader, 0):
  inputs, labels = data[0].to(device), data[1].to(device)
  optimizer.zero_grad()
  outputs = model(inputs)
  loss = criterion(outputs, labels)
  loss.backward()
  optimizer.step()
```

Debido a la flexibilidad de implementación y la multitud de oportunidades que ofrece PyTorch, la totalidad del trabajo se ha realizado con él. Con esta decisión, se paga el precio de la complejidad que requiere la programación a bajo nivel de los modelos de *deep learning*.

5.2. Fastai

Fastai es una biblioteca de *deep learning* basada en PyTorch que facilita el uso de técnicas *state-of-the-art* para la obtención de resultados precisos. Dispone de una API de alto nivel que proporciona un gran nivel de abstracción para la construcción de modelos de aprendizaje profundo, aunque también es posible acceder a una API de medio y bajo nivel (Figura 5.5). Esta última es muy similar a PyTorch, permitiendo configurar a medida los modelos y experimentos que se deseen.

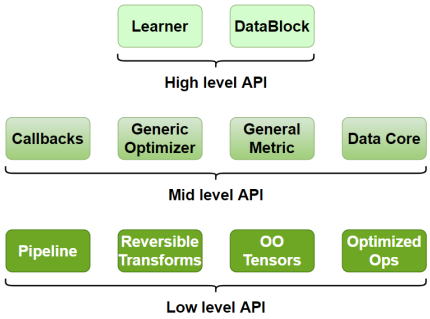

Figura 5.5: Estructura de APIs de fastai.

Para ubicar a fastai dentro del espectro de bibliotecas de *deep learning*, podría decirse que fastai es a PyTorch como Keras es a Tensorflow, siendo fastai y Keras dos APIs de alto nivel de PyTorch y Tensorflow respectivamente.

Fastai permite el uso de tarjetas gráficas para la computación de los modelos de aprendizaje automático. La manera de comprobar que el entorno de fastai está detectando la GPU es idéntica a la mostrada para PyTorch. Si la GPU es detectada, fastai automáticamente se encargará de trasladar los modelos y los datos a la memoria de RAM de la tarjeta, evitando la migración manual que requería realizar PyTorch.

```
import torch
torch.cuda.is_available()
```

Fastai permite el ajuste de múltiples modelos pertenecientes al aprendizaje profundo como las redes neuronales convolucionales y las redes neuronales recurrentes, así como técnicas de NLP (*Natural Language Processing*). A continuación, y en el contexto de este proyecto, se detallarán distintas clases que ofrece fastai que pueden ser de gran utilidad para la creación y ajuste de redes neuronales convolucionales. Toda la información se ha extraído de la documentación de fastai [23].

5.2.1. Clase *Dataset*

Fastai permite la creación de objetos de tipo *Dataset* de PyTorch, para su posterior uso en otras clases que provee fastai. Esta característica facilita la personalización completa del modo en el que se accederán tanto a los datos como a sus correspondientes etiquetas. Además de los métodos *__init__* y *__getitem__* necesarios para crear un *Dataset* de PyTorch, también es necesaria la creación del atributo *c*, el cual representa el número final de salidas del modelo que va a utilizar el *Dataset* (el número de clases del problema).

Por ejemplo, a continuación se muestra una clase *Dataset* personalizada que se inicializa a partir de una ruta donde se encuentran imágenes que se usarán para alimentar una posterior red neuronal convolucional. En esa ruta han de encontrarse dos directorios, un con el nombre OK (en el que se encontrarán imágenes pertenecientes a la clase OK) y otro con cualquier nombre (en el que se encontrarán imágenes que no pertenecen a la clase OK). Esta dualidad de nombres permite asignar la etiqueta a cada imagen examinando si la palabra OK se encuentra en la ruta de la misma.

```
from torch.utils.data import Dataset
from fastai.vision.all import *
import PIL

class DatasetPersonalizado(Dataset):
  def __init__(self, ruta):
    self.archivos = get_image_files(ruta)
    self.c = 2

  def __getitem__(self, i):
    archivo = self.archivos[i]
    imagen = PIL.Image.open(archivo)
    clase = int('OK' in str(archivo))
    return imagen, clase

  def __len__(self):
    return len(self.archivos)

dataset = DatasetPersonalizado('ruta/a/las/imagenes')
```

A la hora de almacenar las rutas como atributos se hace uso de la función *get_image_files* de fastai, que permite recorrer el directorio recursivamente almacenando las rutas de todos los archivos que se encuentren.

5.2.2. Clase *DataLoader*

La clase *DataLoader* de fastai es muy similar a la de PyTorch, incluso permitiendo reutilizar la que ofrece el segundo. El funcionamiento y argumentos de la clase son idénticos a los ya explicados en el caso de PyTorch. Como argumento principal recibe una instancia de la clase *Dataset*, y opciones como el tamaño de *batch*, si se desordenan las muestras o si se desean realizar los posteriores cálculos en la CPU en vez de en la GPU son configurables.

```
dataloader = DataLoader(dataset, batch_size=64, shuffle=True)
```

Fastai también permite la creación objetos personalizados de la clase *Dataloader* implementando como mínimo el método *create_item*. Por ejemplo, a continuación se construye un *Dataloader* que retorna con una probabilidad de 0.9 un valor $\in [0, 0{,}9)$, y con probabilidad de 0.10 termina su retorno de valores.

```
class DataLoaderAleatorio(DataLoader):
  def create_item(self, s):
    r = random.random()
    return r if r<0.9 else stop()
```

5.2.3. Clase *DataBunch*

La clase *DataBunch* permite la creación de objetos que encapsulan los datos de entrenamiento y test, o entrenamiento, validación y test, en una única estructura. Puede ser creado a partir de objetos de la clase *Dataset* o de la clase *DataLoader*, ofreciendo una gran flexibilidad y multitud de opciones de implementación. Esta estructura se encarga de comprobar que todos los objetos *Dataset* o *DataLoader* se encuentran en el mismo dispositivo, CPU o GPU, y permite la aplicación de transformaciones a las muestras.

En el caso de crear el *DataBunch* a partir de estructuras de la clase *Dataset* hay que utilizar el método factoría *create*. Éste realiza funciones muy similares a las de un objeto de la clase *DataLoader*, permitiendo fijar el tamaño de *batch* o el dispositivo a utilizar.

```
databunch = DataBunch.create(dataset_train, dataset_test, bs=64)
```

En el caso de crear el *DataBunch* a partir de estructuras de la clase *DataLoader* no es necesario utilizar un método factoría, si no que se puede instanciar la clase de la manera tradicional.

```
databunch = DataBunch(dataloader_tr, dataloader_vl, dataloader_ts)
```

Pueden aplicarse transformaciones a cada *batch* mediante el argumento *dl_tfms*. Por tanto, si las transformaciones son aleatorias, cada muestra probablemente será diferente en dos épocas diferentes. Por ejemplo, en el siguiente código se realiza un espejismo vertical y un *zoom* máximo de un 5 % a cada imagen haciendo uso de las transformaciones implementadas en la función *aug_transforms*.

```
databunch = DataBunch(dataloader_tr, dataloader_vl, dataloader_ts, dl_tfms =
    aug_transforms(flip_vert=True, max_zoom=1.05))
```

5.2.4. Modelos

Al igual que sucedía con PyTorch, los modelos pueden ser creados al gusto del usuario. La sintaxis es idéntica, apilando distintas operaciones dentro de un bloque *Sequential*.

```
import torch.nn as nn
capa = nn.Sequential(
          nn.AdaptiveAvgPool2d(5),
          nn.Conv2d(2,3,1),
          nn.MaxPool2d(4),
          nn.Linear(2,3)
          )
```

También se permite importar modelos preentrenados de *transfer learning*, por ejemplo las redes residuales.

```
resnet18 = fastai.vision.models.resnet18
resnet34 = fastai.vision.models.resnet34
resnet50 = fastai.vision.models.resnet50
resnet101 = fastai.vision.models.resnet101
resnet152 = fastai.vision.models.resnet152
```

5.2.5. Entrenamiento

La principal ventaja de fastai la encontramos en esta etapa, ya que resume todo el bucle de entrenamiento de una red en una única línea, utilizando un objeto de la clase *Learner*. Una instancia de esta clase, en su creación, recibe una instancia de la clase *DataBunch*, un modelo, un optimizador y una función de pérdida. Además, también podemos indicarle si queremos obtener alguna métrica a mayores de las que fastai ofrece por defecto en cada época.

```
learner_1 = Learner(databunch, modelo_proprio, metrics=accuracy, loss_func = nn
    .CrossEntropyLoss, opt_func = 'Adam')
```

En el caso de que la red la precarguemos entonces tenemos que utilizar un *Learner* específico. Esto es debido a que fastai almacena una arquitectura genérica de los modelos de *transfer learning*, a partir de la cual se pueden construir redes para problemas diferentes. Por ejemplo, en el caso de que queramos utilizar la red residual de 34 capas (*resnet34*) como red convolucional, entonces utilizaremos la opción *cnn_learner*. En el caso de que queramos utilizar la arquitectura para construir una red en U para abordar un problema de segmentación utilizaremos la opción *unet_learner*.

```
learner_2 = cnn_learner(databunch, models.resnet34, metrics=accuracy)
learner_3 = unet_learner(databunch, models.resnet34, metrics=accuracy)
```

Una vez se ha realizado este paso, únicamente falta decirle al objeto de la clase *Learner* que comience el entrenamiento. Fastai ofrece principalmente 3 opciones, la función *fit*, la función *fit_one_cycle* y la función *fine_tune*. A las 3 funciones se les indica como argumento el número de épocas que se desea entrenar la red.

La función *fit* realiza un entrenamiento clásico de la red, realizando las etapas de *feedforward*, cálculo del *loss*, retropropagación, variación de pesos y puesta a 0 de los gradientes.

```
learner.fit(10)
```

Las funciones *fit_one_cycle* y *fine_tune* encuentran la tasa de aprendizaje óptima, y además realizan el *freeze* y *unfreeze* de las primeras capas de los modelos automáticamente. Congelar una capa significa que sus pesos no son alterados tras la etapa de retropropagación. Esta técnica suele ser útil en el contexto del *transfer learning*, ya que las primeras capas de los modelos suelen estar especializadas en la detección de bordes, tarea que es útil para la mayoría de contextos en los que se aplican las redes convolucionales. Esto significa que, por término general, congelar las primeras capas de las redes preentrenadas en las primeras épocas de entrenamiento puede ser una buena práctica, ademas de inducir un ahorro considerable en el tiempo de computación al tener que modificarse menos pesos.

```
learner.fit_one_cycle(10)
learner.fine_tune(10)
```

Los propios creadores de fastai sugieren que las diferencias entre ambas funciones son mínimas, y en la mayoría de ocasiones los resultados a los que llegan son prácticamente idénticos.

Fastai se ha estudiado en el marco del proyecto realizado, pero como ya se ha comentado, todo el código desarrollado en el proyecto se ha creado en PyTorch.

5.3. Google Colaboratory

Google Colaboratory, también llamado Colab, es un servicio de computación en la nube que permite ejecutar código Python en un navegador [24]. Es una plataforma ampliamente utilizada por la comunidad de científicos de datos e investigadores en inteligencia artificial debido a la posibilidad de utilizar GPUs de potencia razonable para realizar cómputos.

La versión básica del servicio es gratuita, únicamente habiendo el requisito de disponer de una cuenta de Google. Si bien este servicio es más que suficiente para la mayoría de las tareas, la potencia de las GPUs asignadas, su memoria RAM y el tiempo de cómputo ininterrumpido están limitados en esta versión. Para solventar el problema existe una versión de pago no accesible desde España. Todas las computaciones para el ajuste de los modelos de aprendizaje profundo realizadas en este proyecto han sido realizadas en esta plataforma.

La manera en la que se han gestionado los archivos locales (imágenes astronómicas) para poder usarse desde Colab ha sido mediante la plataforma de almacenamiento en la nube Google Drive, subiendo los archivos necesarios a dicho servicio. Desde un *notebook* de Colab, se pueden acceder a archivos de Drive ejecutando la siguiente orden:

```
from google.colab import drive
drive.mount('/content/drive')
```

Una vez se ha ejecutado la porción de código, se pueden acceder a los archivos como se haría en un sistema local, a través de la ruta en la que se encuentren en el sistema de ficheros de Drive. Otro comando que es útil cuando se trabaja con este servicio de computación en la nube es:

```
!nvidia-smi
```

```
Mon May 31 18:37:14 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 465.19.01    Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   68C    P8    11W /  70W |      0MiB / 15109MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

Figura 5.6: Características de la GPU asignada por Colab.

Esta orden se utiliza para ver la tarjeta gráfica que Colab ha asignado a la sesión iniciada y algunas de sus características (Figura 5.6). Para que Colab asigne una GPU es necesario indicar que se requiere un acelarador por *hardware* en la configuración del cuaderno.

6. Clasificación morfológica de galaxias distantes

La morfología de los objetos galácticos determina qué procesos físicos están ocurriendo en su interior, por lo que no es necesario recalcar el papel tan importante que juega el desarrollo de técnicas que estimen la morfología de una galaxia en la comprensión y entendimiento del universo distante. En este capítulo se parte de 2 aproximaciones para el desarrollo de dichas técnicas, un enfoque de clasificación directa y uno de regresión para mimetizar el comportamiento humano.

6.1. Conjunto de datos

Disponer de un buen conjunto de datos es clave en el éxito del desarrollo de algoritmos de *deep learning*. El conjunto de datos original es filtrado en primera instancia para seleccionar aquellas imágenes de interés, y aumentado (de manera dinámica) en segunda instancia mediante *data augmentation*.

6.1.1. Selección de imágenes

El conjunto de datos inicial del que se parte proviene del proyecto CANDELS (*Cosmic Assembly Nearinfrared Deep Extragalactic Legacy Survey*), el proyecto de mayor magnitud nunca llevado a cabo por el Telescopio Hubble. Tanto las imágenes como las máscaras que a continuación se van a detallar han sido proporcionadas por el profesor Fernando Buitrago Alonso.

El conjunto de datos está formado por 13648 imágenes y 12936 máscaras, ambas de 200x200 píxeles. Cada una de las imágenes puede contener una o varias galaxias y/o cuerpos espaciales, en este último caso la galaxia principal y de interés es aquella situada en la parte central. Las imágenes se pueden encontrar en los canales H, I, J y V. Posteriormente, y para formar imágenes multicanal, se juntarán aquellas imágenes en canales diferentes que representen la misma porción del espacio exterior. Debido a esta posibilidad de aparición de varios cuerpos en una misma imagen es necesario enmascararlas, eliminando toda aquella información que no sea de interés, en este caso todos los objetos que no sean la galaxia principal de la imagen.

La tarea a acometer en este capítulo es la de clasificar las galaxias según su morfología mediante técnicas de *deep learning*. Para ello, es necesario que la galaxia principal de la imagen se encuentre aislada. En caso contrario, se corre el riesgo de las redes neuronales convolucionales recojan información de galaxias y cuerpos vecinos que no sean útiles para la determinación de la morfología de la galaxia principal.

El objetivo de las máscaras, por tanto, es el de aislar la galaxia central de la imagen del resto de objetos que aparecen en la misma, como se aprecia en la Figura 6.1. Aplicando esta técnica se logra que la red neuronal que posteriormente será alimentada con esas imágenes no se pierda en recoger características irrelevantes para la clasificación.

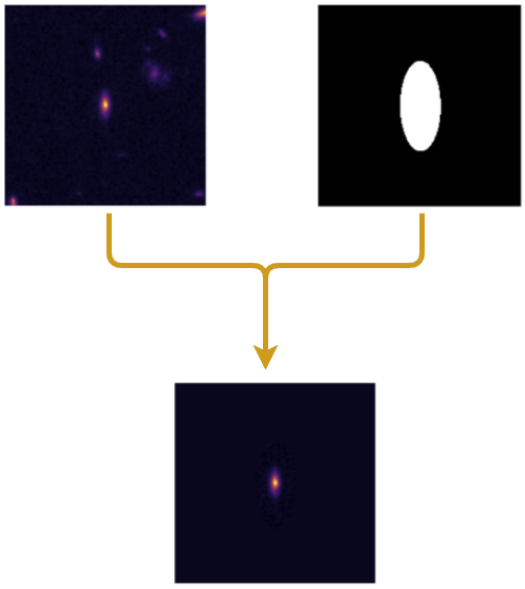

Figura 6.1: Unión de una imagen con su máscara.

La morfología de las galaxias principales de las imágenes puede ser categorizada en 6 clases: *DISK*, *SPH*, *DISKSPH*, *DISKIRR*, *IRR* y *NONE* (Figura 6.2). Además, y como ya se ha comentado, cada imagen se encuentra en uno de los 4 canales siguientes: *H*, *I*, *J* y *V*. Por su parte, las máscaras son personalizadas para cada imagen, por lo que debería haber equivalencia uno a uno entre ambas. Sin embargo, como ya se ha mencionado al comienzo del capítulo, se tienen 712 máscaras menos que imágenes.

Morfología	**Imágenes**				**Máscaras**			
	Canal H	Canal I	Canal J	Canal V	Canal H	Canal I	Canal J	Canal V
DISK	890	890	890	890	890	807	890	810
SPH	956	956	956	956	956	860	956	861
DISKSPH	969	969	969	969	969	851	969	853
DISKIRR	340	340	340	340	340	306	340	302
IRR	100	100	100	100	100	96	100	92
NONE	157	157	157	157	157	137	157	137

Cuadro 6.1: Número de imágenes y máscaras según morfología de la galaxia principal y canal.

En el Cuadro 6.1 se puede apreciar la relación de imágenes y máscaras disponibles en el conjunto de datos según morfología de la galaxia principal y canal de exposición. Es evidente que el siguiente paso será el de seleccionar aquellas imágenes que tengan una máscara disponible, emparejarlas y desechar las que no tengan asociada ninguna máscara. La razón de eliminar estas

últimas es que podrían confundir a los posteriores algoritmos de aprendizaje profundo y desviarlos de su objetivo: determinar la morfología de la galaxia principal de la imagen.

Figura 6.2: Morfología de las galaxias.

Morfología	Imágenes + Máscaras			
	Canal H	Canal I	Canal J	Canal V
DISK	890	807	890	810
SPH	956	860	956	861
DISKSPH	969	851	969	853
DISKIRR	340	306	340	302
IRR	100	96	100	92
NONE	157	137	157	137

Cuadro 6.2: Número de imágenes con máscara disponible según morfología y canal.

Las imágenes y las máscaras tienen un prefijo común, por lo que comparando dicho prefijo se puede realizar la equivalencia. Tanto las imágenes como las máscaras son proporcionadas en formato *.fits*, para acceder a su contenido es necesario utilizar la función *fits.open* del paquete *astropy.io*.

```
from astropy.io import fits
```

```
imagen = fits.open('SPH_8341_cosmos_H.fits')[0].data
mascara = fits.open('SPH_8341_cosmos_H_central_mask.fits')[0].data
```

Tras realizar la selección y unión de imágenes con sus respectivas máscaras, Cuadro 6.2, se cuenta con 12936 imágenes, cada una de ellas con una máscara disponible, y se avanza a una nueva etapa en el procesado del conjunto de datos. Lo siguiente que se va a realizar es eliminar aquellas imágenes que, sin enmascarar, tengan más de un 50 % de píxeles con valor igual a 0. Estas imágenes son aquellas que están prácticamente en su totalidad vacías, y debido a eso no son de interés para la tarea que se va a acometer en este capítulo. Para eliminar estas imágenes, contamos si de entre los 4×10^5 píxeles (200x200), 2×10^5 tienen un valor de 0.

Morfología	**Imágenes + Máscaras**			
	Canal H	Canal I	Canal J	Canal V
DISK	890	802	890	803
SPH	956	851	956	854
DISKSPH	969	848	969	847
DISKIRR	340	303	340	300
IRR	100	96	100	89
NONE	157	137	157	137

Cuadro 6.3: Número de imágenes con máscara disponible según morfología y canal con menos de un 50 % de píxeles con valor 0.

El número de imágenes eliminadas con este procedimiento son muy pocas, los resultados pueden verse en el Cuadro 6.3. Tras realizar este filtrado y desestimado de las muestras con muy poca información, pasamos de tener 12936 imágenes a disponer de 12891, 45 imágenes menos. El último paso de la selección de instancias consiste en escoger aquellas que se encuentren disponibles en todos los canales, es decir *H*, *I*, *J* y *V*. Algunas de las imágenes no se encuentran en la totalidad de los canales, por lo que este proceso también desechará algunas muestras. La relación de resultados se aprecia en el Cuadro 6.4. La unión de las imágenes que representan la misma porción del espacio exterior pero se encuentran en distintos canales, al igual que la unión entre las imágenes y las máscaras, se hace mediante un prefijo común que comparten (Figura 6.3).

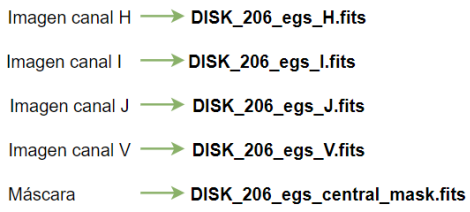

Figura 6.3: Nomenclatura de las imágenes y de las máscaras.

Morfología	Imágenes con máscara
DISK	793
SPH	847
DISKSPH	841
DISKIRR	299
IRR	89
NONE	135

Cuadro 6.4: Número final de imágenes con máscara disponibles en los canales *H*, *I*, *J*, y *V* según morfología tras realizar el procesado de muestras.

Una vez realizado este proceso, el resultado final son 12016 imáganes con 12016 máscaras asociadas, o bien 3004 imágenes en 4 canales cada una. Para reducir los requisitos de la máquina que realizará los cómputos de los parámetros de los algoritmos de *deep learning* y acelerar el tiempo de entrenamiento se opta por reducir el tamaño de las imágenes. Se seleccionan los 100x100 píxeles centrales de la imagen de 200x200 píxeles original, de tal manera que el objeto central de la imagen, el que es de interés en esta sección, se mantiene inalterado, únicamente eliminando la parte externa de las imágenes que, mayoritariamente, no aportará información valiosa.

Además, los píxeles de todas las imagen son escalados al intervalo [0,1]. Este procedimiento es de gran utilidad cuando se va a trabajar con algoritmos de aprendizaje automático que no se basan en reglas, como es el caso de las redes neuronales convolucionales, y ayuda a que la convergencia al valor óptimo de los pesos de la red sea más rápida. El método de escalado elegido es *min-max*.

Para almacenar las muestras seleccionadas de manera persistente se ha optado por serializar toda la información en un objeto de tipo *pickle*, representando una estructura *4*-dimensional con dimensiones *[3004, 4, 100, 100]*, donde 3004 son el número de imágenes, 4 el número de canales en los que se encuentra cada imagen, 100 el número de píxels que conforman la altura de la imagen y 100 el número de píxels que conforman la anchura de la imagen. El objeto de tipo *pickle* podrá ser desserializado posteriormente, evitando así realizar el procesamiento del *dataset* original de nuevo.

Respecto a las etiquetas asignadas a cada una de las imágenes, éstas serán diferentes para los problemas que se abordarán en las secciones 6.2 y 6.3, ya que al tratarse aproximaciones basadas en clasificación morfológica directa y regresión, la naturaleza de la variable respuesta diferirá, siendo categórica y numérica respectivamente. Los detalles de la misma serán explicados en las secciones mencionadas.

6.1.2. *Data augmentation*

La técnica conocida como *data augmentation* es usada en la mayoría de modelos de aprendizaje automático que se encuentran en el *state of the art* [25]. Consiste en, de manera artificial, generar nuevas instancias etiquetadas a partir de modificaciones de instancias ya existentes. La necesidad de realizar esta generación de nuevas muestras responde a que los modelos de *machine learning*, y sobre todo de *deep learning*, son cada vez más grandes, llegando a tener billones de parámetros. Para estimar de manera precisa todos ellos es necesario contar con conjuntos de datos de entrenamiento masivos.

Aplicando *data augmentation* al dominio de aplicación que concierne en este capítulo, se generarán nuevas imágenes artificiales de galaxias a partir de las ya existentes, introduciendo modificaciones para que las imágenes nuevas difieran de las primarias con las que han sido generadas, pero sin afectar a la morfología del objeto que contienen: la galaxia. Es por ello que no todas las técnicas de *data augmentation* son válidas en todos los contextos.

Un punto importante a tener en cuenta al realizar esta técnica es que únicamente han de generarse imágenes artificiales nuevas a partir de las disponibles en el conjunto de entrenamiento como se aprecia en la Figura 6.4. Nunca han de usarse imágenes del conjunto de prueba o del de validación, principalmente por 2 motivos:

1. Si se realiza *data augmentation* sobre el conjunto de datos total y se generan nuevas imágenes a partir de modificaciones de todas las disponibles, cuando se haga la división en conjuntos de entrenamiento y prueba habrá imágenes en este último que no sean independientes de imágenes con las que se ha entrenado el algoritmo de aprendizaje automático. Por ejemplo, es probable que suceda que una imagen del conjunto de entrenamiento se encuentre rotada en el conjunto de test, si se incluye la rotación como técnica de *data augmentation*. Estas imágenes que, a priori, pueden parecer independientes, guardan cierta relación por lo que se estaría falseando y estimando de manera optimista la tasa de error sobre el conjunto de test, lo cual es inadmisible. Incluso aunque a cada modificación de una imagen original se le añada una cantidad de ruido gaussiano, la independencia no se logra.

2. La precisión o desempeño de un algoritmo que clasifique imágenes no debe medirse con imágenes generadas artificialmente. Esto es debido a que no hay certeza absoluta de que, por muy seguro que se esté de que las transformaciones realizadas sobre las imágenes originales sean muy realistas, las imágenes artificiales puedan realmente darse en el entorno de producción de las imágenes originales: el universo en este trabajo. Puede darse el caso de estar midiendo la eficacia de un clasificador con imágenes que nunca va a encontrar en la realidad.

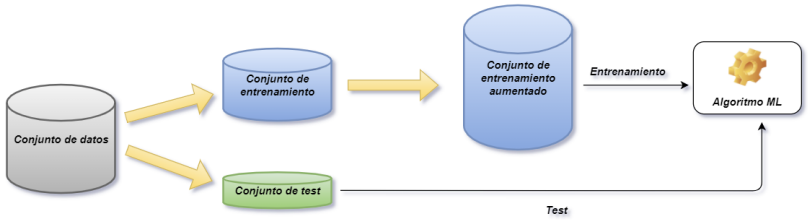

Figura 6.4: *Pipeline* de la técnica *data augmentation*.

Una vez se tienen claros estos conceptos y por consiguiente la manera de aplicar esta técnica, hay que definir qué transformaciones de las imágenes del conjunto de entrenamiento se van a realizar y por qué.

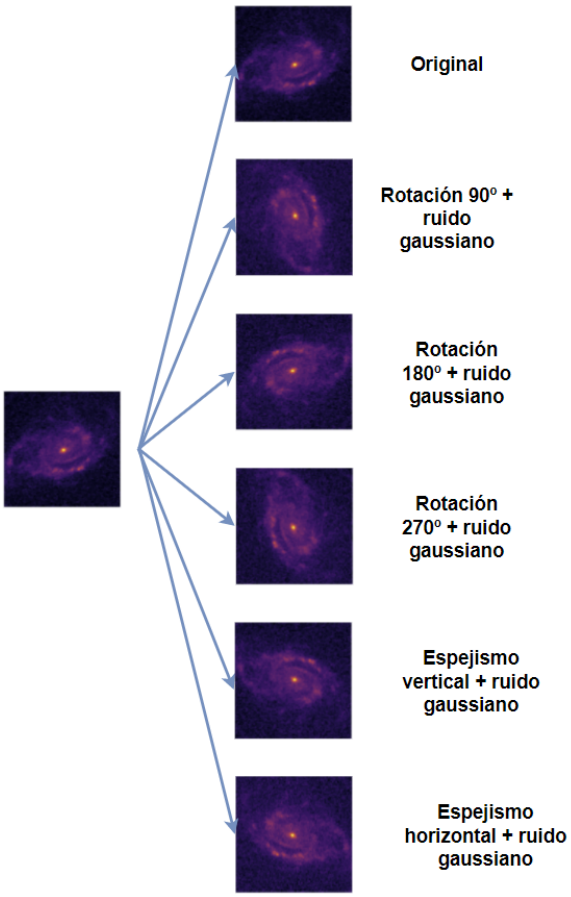

Figura 6.5: *Data augmentation* realizado sobre las imágenes.

Al tratarse de un problema profundamente relacionado con la morfología de las galaxias, las

transformaciones que se vayan a realizar no pueden alterar la forma del objeto galáctico. Es decir, si optamos por realizar un recorte de una parte de la imagen y, posteriormente, redimensionar el recorte para que tenga el tamaño de la imagen original, puede que la morfología de la galaxia se haya alterado lo suficiente como para que la galaxia ya no sea, por ejemplo, de tipo *DISK* y ahora sea de tipo *DISKSPH*. Es decir, el *cropping* o recorte no es una técnica válida para el par que forman el problema a resolver y el dominio de aplicación en el que se centra este capítulo.

Una vez explicada la razón por la que no todas las transformaciones posibles son válidas en este contexto, se procede a detallar las transformaciones realizadas:

1. **Rotaciones de 90°, 180° y 270° con ruido gaussiano**. Las primeras transformaciones consistirán en aplicar a las imágenes del conjunto de entrenamiento una serie de rotaciones, concretamente 90°, 180° y 270° en sentido horario. Estas transformaciones no alteran la morfología de la galaxia, si no simplemente su orientación, por lo que son válidas en este contexto. Además, para tener certeza de que las redes neuronales convolucionales que serán alimentadas por estas imágenes vean la imagen rotada diferente a la original [1], se añade ruido gaussiano a cada píxel aleatorio en cada época. Por tanto, la probabilidad de que dos imágenes con la misma rotación sean idénticas en dos épocas diferentes es prácticamente nula, carcaterística fundamental para intentar prevenir el sobreajuste y mejorar el rendimiento de la red. La desviación típica elegida respecto al valor original del píxel es 0.02. Es decir, si el píxel p_i tenía el valor μ, ahora tendrá el valor $\mu + x$, donde $x \sim N(0, \sigma^2)$, con $\sigma^2 = 0{,}0004$.

2. ***Mirroring* respecto a los ejes vertical y horizontal con ruido gaussiano**. Las segunda serie de transformaciones consistirá en aplicar a las imágenes del conjunto de entrenamiento un *mirroring* o espejismo frente al eje de ordenadas y un espejismo frente al eje de abscisas. Estas transformaciones provocan un resultado diferente a las rotaciones anteriores, e igualmente no alteran la morfología de la galaxia, simplemente su orientación. De igual manera, se añade ruido aleatorio en cada época proveniente de una distribución $N(0, \sigma^2)$, con $\sigma^2 = 0{,}0004$.

Por tanto, como se aprecia en la Figura 6.5, por cada imagen del conjunto de entrenamiento se generan 6, la original, 3 rotaciones y 2 espejismos. Las rotaciones y los espejismos además vienen acompañados de ruido gaussiano aleatorio en cada época, por lo que una misma imagen transformada en cada época será diferente.

6.2. Clasificación morfológica

En esta primera aproximación se va a tratar de clasificar las galaxias directamente en base a su morfología. Es decir, si una imagen contiene una galaxia de la categoría DISK la red neuronal convolucional que se diseñe tendrá que predecir directamente la categoría, sin que haya ningún tipo de predicción intermedia como si será en el caso de la aproximación basada en regresión.

6.2.1. Arquitectura

La red diseñada en esta sección es una adaptación de la red usada en M. Huertas-Company *et al.* (2015): *A catalog of visual-like morphologies in the 5 CANDELS fields using deep-learning* [26], que a su vez es extraída de Dieleman *et al.* (2015): *Rotation-invariant convolutional neural networks for galaxy morphology prediction* [27].

Operación	**Dimensionalidad volumen salida**	**# Parámetros**
Convolucional	8x81x81	12808
ReLU	8x81x81	0
Max Pool	8x62x62	0
Convolucional	16x53x53	12816
ReLU	16x53x53	0
Max Pool	16x45x45	0
Convolucional	32x40x40	18464
ReLU	32x40x40	0
Max Pool	32x20x20	0
Convolucional	64x16x16	51264
ReLU	64x16x16	0
Max Pool	64x9x9	0
Convolucional	128x7x7	73856
ReLU	128x7x7	0
Convolucional	128x5x5	147584
ReLU	128x5x5	0
Max Pool	128x4x4	0
FC	2048	4196352
ReLU	2048	0
FC	2048	4196352
ReLU	2048	0
FC	6	12294
Total	–	**8721790**

Cuadro 6.5: Variación de la dimensionalidad del volumen de entrada y # parámetros.

Las dimensiones de las imágenes de entrada en los artículos mencionados son de 45x45 píxeles en 3 canales, mientras que las imágenes que se van a utilizar son de 100x100 píxeles en 4 canales. Por tanto, la variación en la dimensionalidad del volumen de entrada provoca que sea necesario el rediseño de algunas capas de la red. Además, respecto a las 5 clases que predice la red en [26], también es necesario adaptar la salida de la red para contemplar las 6 posibles morfologías que pueden tener las galaxias.

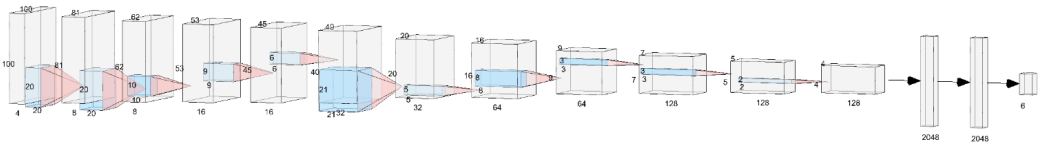

Figura 6.6: Arquitectura diseñada para el problema de clasificación directa.

La red consta de 9 capas, 6 capas convolucionales y 3 capas *fully connected*. Las 4 primeras capas convolucionales y la última capa convolucional están formadas por una operación de convolución seguidas de una función de activación no lineal (ReLU) y de una operación de *pooling* del máximo. La quinta capa convolucional sigue la misma estructura, pero sin la presencia de una operación de *pooling*.

Tras las capas convolucionales, se realiza un aplanado del volumen de salida y el resultado de dicha operación se introduce en 3 capas *fully connected* (Figura 6.6). La última de estas 3 capas consta de 6 neuronas, modelizando las 6 posibles morfologías que puede tener una galaxia en este problema. La reducción de las dimensiones que sufre el volumen inicial de dimensiones 4x100x100 (imágenes de 100x100 píxeles en 4 canales) según las imágenes fluyen por las distintas operaciones que se realizan en la red se especifica en el Cuadro 6.5. La relación de parámetros de cada una de las operaciones de las capas convolucionales se encuentra el Cuadro 6.6.

Operación	***Kernel***	***Padding***	***Stride***
Convolucional	20	0	1
Max Pool	20	0	1
Convolucional	10	0	1
Max Pool	9	0	1
Convolucional	6	0	1
Max Pool	21	0	1
Convolucional	5	0	1
Max Pool	8	0	1
Convolucional	3	0	1
Convolucional	3	0	1
Max Pool	2	0	1

Cuadro 6.6: Configuración de las capas convolucionales y de *pooling*.

La creación en PyTorch de la red convolucional especificada es la siguiente:

```
class ConvNet(nn.Module):
  def __init__(self):
    super(ConvNet, self).__init__()
    self.layer1 = nn.Sequential(
      nn.Conv2d(4, 8, kernel_size=20, stride=1, padding=0),
      nn.ReLU(),
      nn.MaxPool2d(20, stride=1))
    self.layer2 = nn.Sequential(
      nn.Conv2d(8, 16, kernel_size=10, stride=1, padding=0),
      nn.ReLU(),
      nn.MaxPool2d(9, stride=1))
    self.layer3 = nn.Sequential(
      nn.Conv2d(16, 32, kernel_size=6, stride=1, padding=0),
      nn.ReLU(),
      nn.MaxPool2d(21, stride=1))
    self.layer4 = nn.Sequential(
      nn.Conv2d(32, 64, kernel_size=5, stride=1, padding=0),
      nn.ReLU(),
      nn.MaxPool2d(8, stride = 1))
    self.layer5 = nn.Sequential(
      nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=0),
      nn.ReLU())
    self.layer6 = nn.Sequential(
      nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=0),
      nn.ReLU(),
      nn.MaxPool2d(2, stride = 1))
    self.fc1 = nn.Linear(4*4*128, 4*4*128)
    self.fc2 = nn.Linear(4*4*128, 4*4*128)
    self.fc3 = nn.Linear(4*4*128, 6)
    self.act = nn.ReLU()
  def forward(self, x):
    out = self.layer1(x)
    out = self.layer2(out)
    out = self.layer3(out)
    out = self.layer4(out)
    out = self.layer5(out)
    out = self.layer6(out)
    out = out.reshape(out.size(0), -1)
    out = self.act(self.fc1(out))
    out = self.act(self.fc2(out))
    out = self.fc3(out)
    return out
```

6.2.2. Experimentación y métricas

Con el objetivo de valorar como afectan distintas configuraciones de la red neuronal y del experimento, se define un experimento base. Los resultados de este marco inicial serán utilizados

como base para realizar comparaciones con otros experimentos.

Se realizarán 3 modificaciones del experimento base, variando únicamente la tasa de aprendizaje. El objetivo de estos 3 experimentos es el de comprobar si la variación del *learning rate* induce a resultados diferentes de los vistos en el experimento base, o por el contrario el retoque de este parámetro no produce cambios significativos.

La siguiente modificación del experimento base que se va a realizar consiste en modificar las imágenes que sirven como entrada de la red. Recordar que, en el experimento base, se proporcionaban imágenes de galaxias en 4 canales: H, I, J y V. Ahora, dichas imágenes se van a modificar para construir los canales R, G y B. El objetivo del cambio es remarcar las diferencias entre los píxeles que forman parte de la galaxia y los píxeles que no. Para realizar la transformación, se hace uso de la función *make_lupton_rgb* del paquete *astropy.visualization* y de código proporcionado por el profesor Fernando Buitrago Alonso.

```
def createRGB(b_im, g_im, r_im, stretch=1, Q=10, scale=None):
  if scale is None:
    pass
  else:
    b_im = b_im*scale[0]
    g_im = g_im*scale[1]
    r_im = r_im*scale[2]
  rgb = make_lupton_rgb(image_r=r_im,  image_g=g_im,  image_b=b_im,  stretch=
   stretch, Q=Q)
  return rgb
```

A los canales RGB se le añade un cuarto canal (α) que ejerce como canal de suavizado (Figura 6.7). Este canal proviene de la aplicación de un filtro gaussiano a los canales RGB.

```
scale = [1.,0.7,0.7]
stretch = 0.05
QQ = 10
kernel_sigma = 1
level_of_gray = 10
im_h = fits.open('DISK_206_egs_H.fits')[0].data
im_i = fits.open('DISK_206_egs_I.fits')[0].data
im_j = fits.open('DISK_206_egs_J.fits')[0].data
rgb_original = createRGB(im_i, im_j, im_h, scale=scale, stretch=stretch, Q=QQ)
rgb_smoothed = gaussian_filter(rgb_original, sigma = kernel_sigma)
rgb_alpha = np.stack([rgb_original[:,:,0], rgb_original[:,:,1], rgb_original
   [:,:,2], np.sum(rgb_smoothed, axis=2)*level_of_gray], axis=2)
```

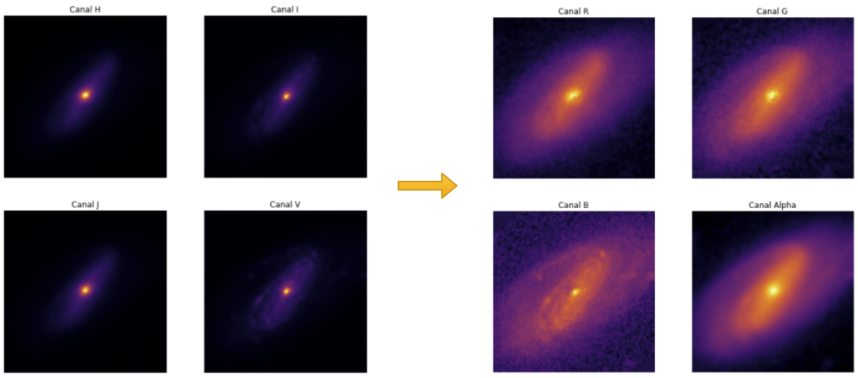

Figura 6.7: Imagen en los canales HIJV frente a RGBα.

Además, para cada experimento, se calcularán las siguientes métricas:

- Evolución del *loss* medio por época sobre el conjunto de entrenamiento.
- Evolución del *loss* medio por época sobre el conjunto de test.
- Evolución de la tasa de acierto media por época sobre el conjunto de entrenamiento.
- Evolución de la tasa de acierto media por época sobre el conjunto de test.
- Curvas ROC (*Receiver Operating Characteristic*) y el AUC (*Area Under Curve*) de la adaptación del problema al caso de la clasificación binaria, es decir, se mide la capacidad de la red neuronal convolucional para discriminar entre una clase frente a todas las demás. Al tener 6 clases correspondientes a los 6 tipos de morfología posibles de las galaxias, surgen 6 problemas de clasificación binaria: *DISK* frente a no *DISK*, *SPH* frente a no *SPH*, *DISKSPH* frente a no *DISKSPH*, *DISKIRR* frente a no *DISKIRR*, *IRR* frente a no *IRR* y *NONE* frente a no *NONE*.

6.2.3. Resultados

A continuación se exponen los diferentes resultados obtenidos para las distintas configuraciones que se han realizado, así como una comparación global de los cambios que producen la aplicación de las distintas técnicas en las métricas finales.

6.2.3.1 Experimento base

La configuración de la red neuronal convolucional usada como experimento base se muestra en el Cuadro 6.7. Se puede apreciar como el número de instancias de entrenamiento es de 12432, esto es $\approx \frac{2}{3}$ (*hold out* sin repetición como método de estimación de la tasa de acierto) de las 3004 imágenes del *dataset* original multiplicado por 6 debido a las 5 transformaciones añadida mediante

las técnicas de *data augmentation* ya explicadas. De esta manera, las 932 imágenes del conjunto de datos ($\approx \frac{1}{3}$ de las 3004 imágenes del *dataset* original) a las que no se les ha aplicado ninguna transformación se reservan como conjunto de test para medir la eficacia de la red sobre ejemplos que no ha visto, y medir así su capacidad de generalización.

# imágenes de entrenamiento	12432
# imágenes de test	932
Función de pérdida	Entropía cruzada
Optimizador	RMSP
Tamaño de *batch*	28
Tasa de aprendizaje	0.001
Épocas	200

Cuadro 6.7: Configuración del experimento base.

En la Figura 6.8 se aprecia como el *loss* o pérdida sigue un patrón muy similar tanto en el conjunto de entrenamiento como en el conjunto de test, estabilizándose entorno a la unidad a partir de las 50 épocas. La pérdida siempre es algo inferior sobre el conjunto de entrenamiento que sobre el conjunto de test, pero no lo suficiente como para considerar que hay un problema de sobreajuste, esto es, un problema de falta de generalización sobre imágenes no vistas.

Igualmente, se aprecia como la precisión o *accuracy* de la red convolucional es similar en sendos conjuntos de entrenamiento y test, siguiendo un patrón equivalente, y estabilizándose entorno a las 125 épocas. Tampoco puede hablarse de sobreajuste claro, ya que es normal que la precisión sobre el conjunto de entrenamiento sea mayor que sobre el conjunto de test, aunque si se aprecia un comportamiento oscilatorio más marcado conforme avanzan las épocas sobre este último. La varianza de las mediciones de la tasa de acierto es mayor sobre el conjunto de test que sobre el conjunto de entrenamiento.

Respecto a las curvas ROC y las áreas encerradas bajo las mismas, se aprecia como el problema binario en el que la red se desenvuelve con mayor destreza es en el de discriminar galaxias de tipo SPH frente a a galaxias que no son del tipo SPH. A éste le siguen los problemas donde la categoría en la que clasificar o no es DISKIRR, DISK, DISKSPH, IRR y NONE respectivamente (Cuadro 6.8).

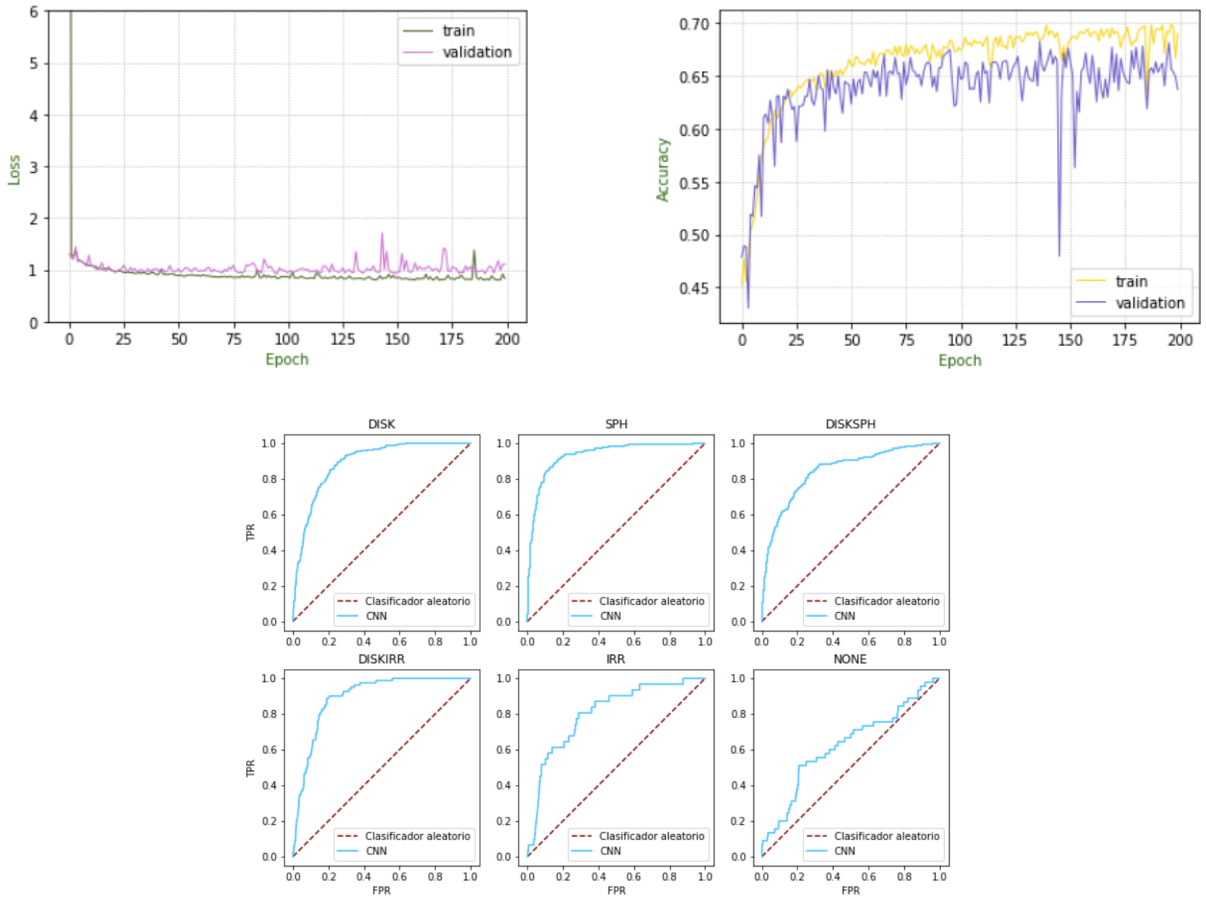

Figura 6.8: *Loss* (izquierda) y *accuracy* (derecha) medio por época para los datos de entrenamiento y test y curvas ROC (experimento base).

Problema binario	**AUC**
DISK frente los demás	0.891
SPH frente los demás	0.934
DISKSPH frente los demás	0.844
DISKIRR frente los demás	0.894
IRR frente los demás	0.806
NONE frente los demás	0.625

Cuadro 6.8: Áreas bajo la curva de los problemas de clasificación dicotómica para el experimento base.

6.2.3.2 Variaciones en el *learning rate*

Cuando se aumenta el *learning rate* 10 veces respecto al experimento base, tomando el valor de 0.01, se aprecia como la red neuronal convolucional no ofrece un buen rendimiento. La pérdida, ya sea sobre el conjunto de entrenamiento o sobre el conjunto de test, no disminuye, estancándose en un valor fijo antes de las 10 primeras épocas. Así mismo, la precisión es nefasta, sin lograr superar el 30 % de acierto en ningún caso, en ambos conjuntos (Figura 6.9). Esto se debe a que no se consigue optimizar la función de pérdida, por lo que la red no aprende las características relevantes y útiles de las imágenes para discriminar las galaxias.

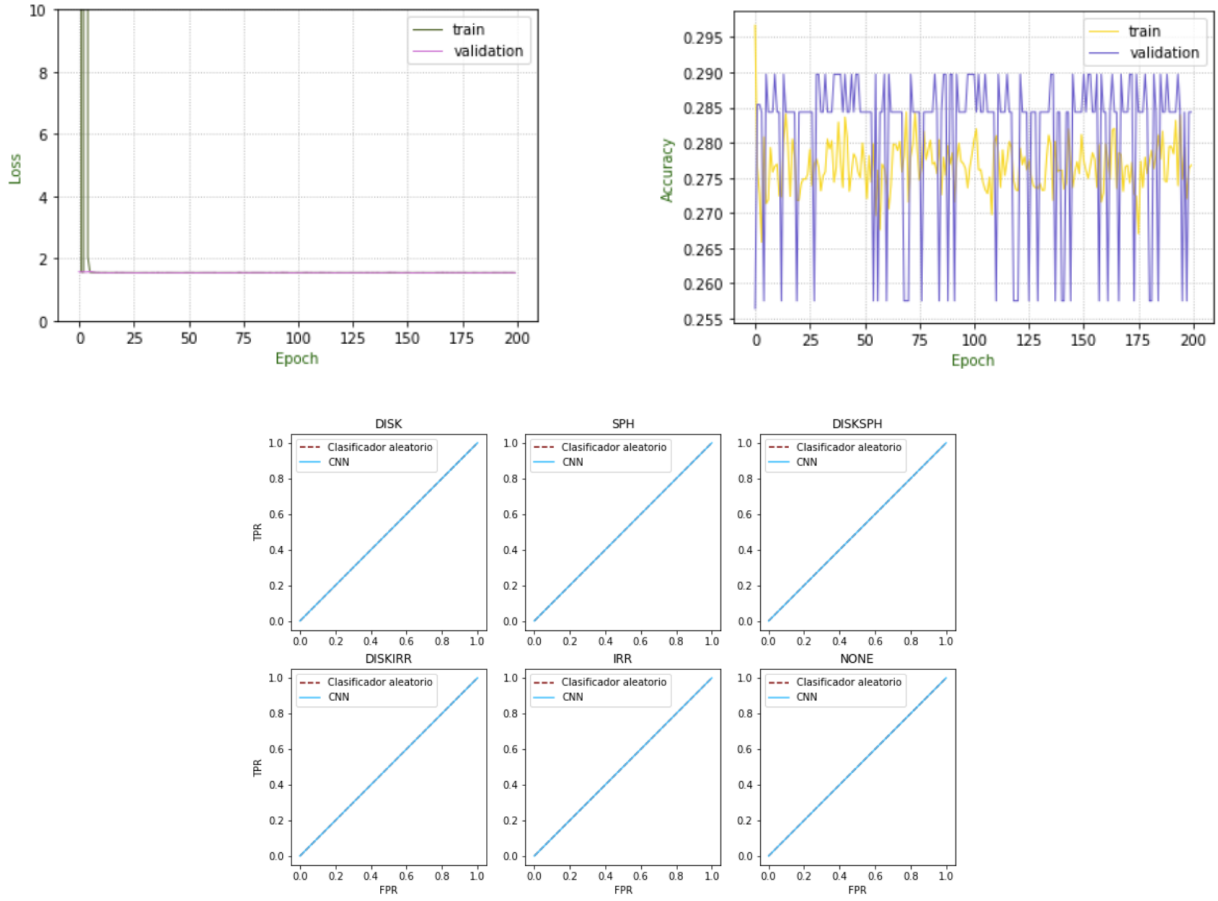

Figura 6.9: *Loss* (izquierda) y *accuracy* (derecha) medio por época para los datos de entrenamiento y test y curvas ROC ($lr = 0.01$.).

En este caso, las curvas ROC son de gran utilidad par diagnosticar lo que está sucediendo. La red está exhibiendo un comportamiento completamente aleatorio, siendo todas las áreas bajo la curva iguales a 0.5, el AUC que tiene un clasificador aleatorio (Cuadro 6.9).

Problema binario	AUC
DISK frente los demás	0.5
SPH frente los demás	0.5
DISKSPH frente los demás	0.5
DISKIRR frente los demás	0.5
IRR frente los demás	0.5
NONE frente los demás	0.5

Cuadro 6.9: Áreas bajo la curva de los problemas de clasificación dicotómica para el experimento con lr = 0.01.

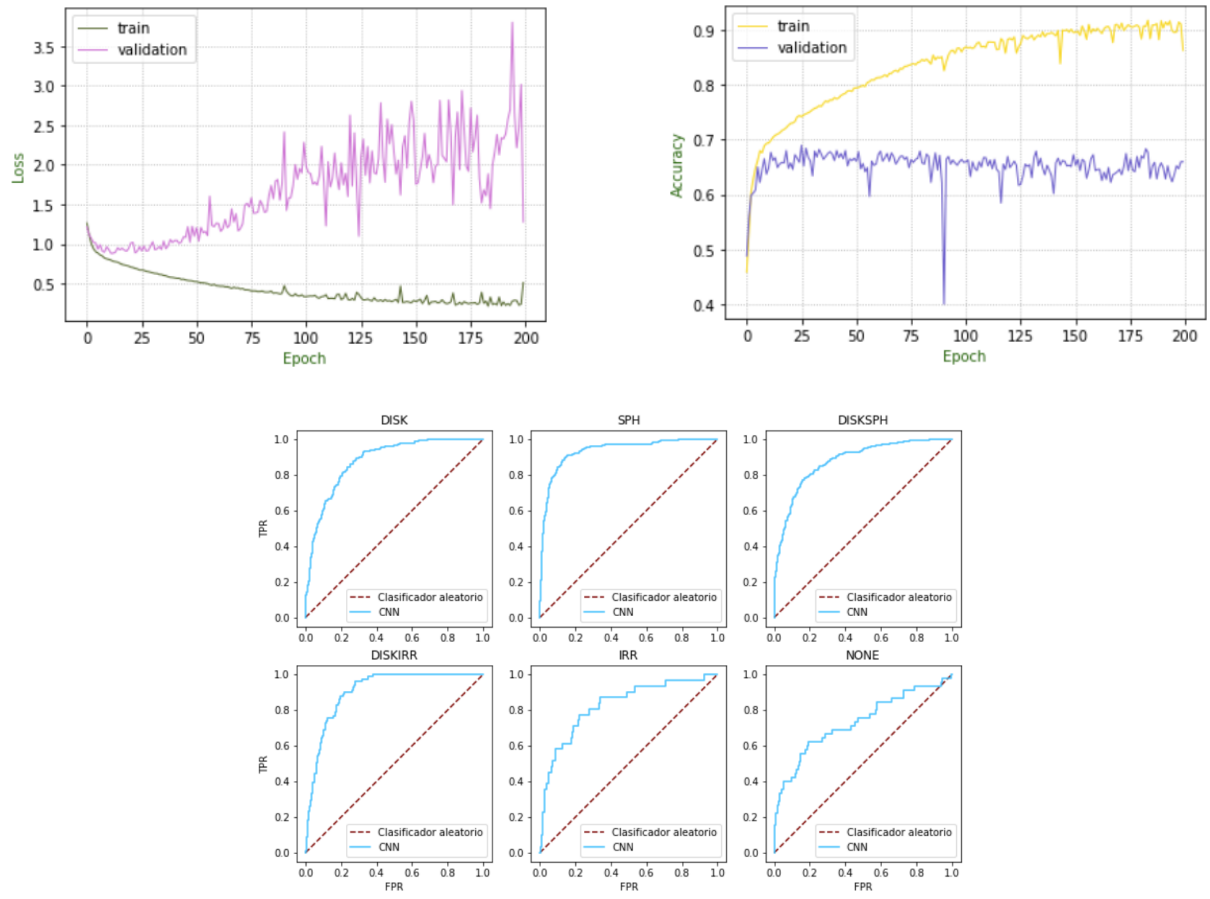

Figura 6.10: *Loss* (izquierda) y *accuracy* (derecha) medio por época para los datos de entrenamiento y test y curvas ROC ($lr = 0.0001$.).

Problema binario	**AUC**
DISK frente los demás	0.880
SPH frente los demás	0.937
DISKSPH frente los demás	0.877
DISKIRR frente los demás	0.907
IRR frente los demás	0.828
NONE frente los demás	0.732

Cuadro 6.10: Áreas bajo la curva de los problemas de clasificación dicotómica para el experimento con lr = 0.0001.

Cuando la tasa de aprendizaje es 10 veces más pequeña que en el experimento base, valiendo 0.0001, nos encontramos con un claro caso de sobreajuste u *overfitting*. En la Figura 6.10 se aprecia como tanto el *loss* como la *accuracy* sobre el conjunto de entrenamiento mejoran sin parar, probablemente obteniendo valores casi perfectos a la larga, es decir una pérdida de 0 y una precisión de 1. Sin embargo, estas métricas no ofrecen un comportamiento tan ideal sobre el conjunto de test, ya que la pérdida aumenta poco a poco pero de manera constante, oscilando según avanzan las épocas de una manera más marcada. Por su parte, la precisión sobre el conjunto de entrenamiento se estabiliza cerca del 70 % a partir de la época 25.

Respecto a las curvas ROC y las áreas encerradas bajo las mismas, se aprecia como el probelma binario en el que la red se desenvuelve con mayor destreza es en el de discriminar galaxias de tipo SPH frente a a galaxias que no son del tipo SPH. A éste le siguen los problemas donde la categoría en la que clasificar o no es DISKIRR, DISK, DISKSPH, IRR y NONE respectivamente (Cuadro 6.10), al igual que en el experimento base.

Cuando la tasa de aprendizaje es 100 veces más pequeña que en el experimento base, valiendo 0.00001, nos encontramos igualmente con sobreajuste, también muy evidente. En la Figura 6.11 se aprecia como tanto el *loss* como la *accuracy* sobre el conjunto de entrenamiento mejoran sin parar, pero estas métricas no ofrecen un comportamiento igual de idóneo sobre el conjunto de test, ya que a partir de la época 50 los valores se mantienen y paran su mejora, incluso empeoran en el caso del *loss*.

Respecto a las curvas ROC y las áreas encerradas bajo las mismas, se aprecia como el probelma binario en el que la red se desenvuelve con mayor destreza es en el de discriminar galaxias de tipo SPH frente a a galaxias que no son del tipo SPH. A éste le siguen los problemas donde la categoría en la que clasificar o no es DISK DISKIRR, DISKSPH, IRR y NONE respectivamente (Cuadro 6.11).

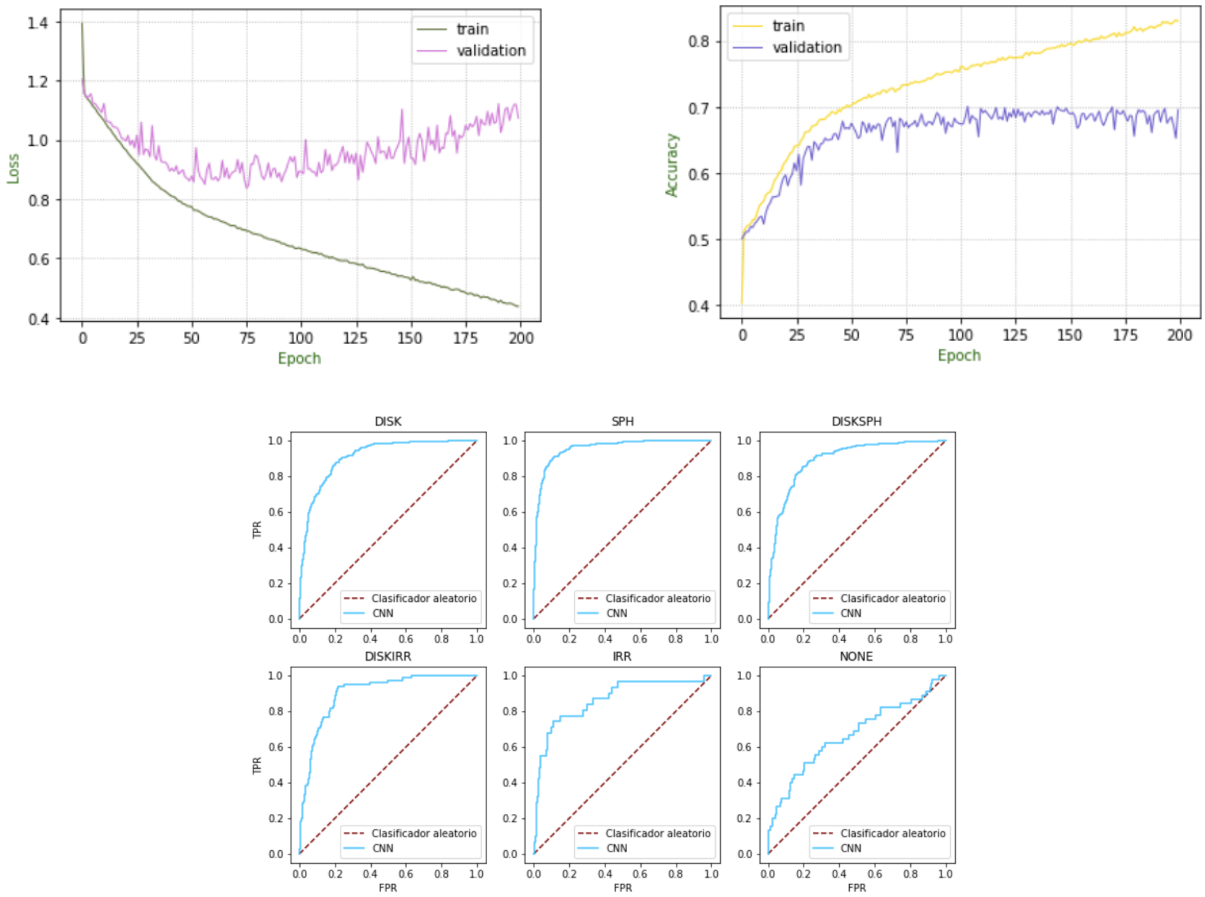

Figura 6.11: *Loss* (izquierda) y *accuracy* (derecha) medio por época para los datos de entrenamiento y test y curvas ROC ($lr = 0.00001$.).

Problema binario	**AUC**
DISK frente los demás	0.91
SPH frente los demás	0.956
DISKSPH frente los demás	0.895
DISKIRR frente los demás	0.902
IRR frente los demás	0.865
NONE frente los demás	0.668

Cuadro 6.11: Áreas bajo la curva de los problemas de clasificación dicotómica para el experimento con lr = 0.00001.

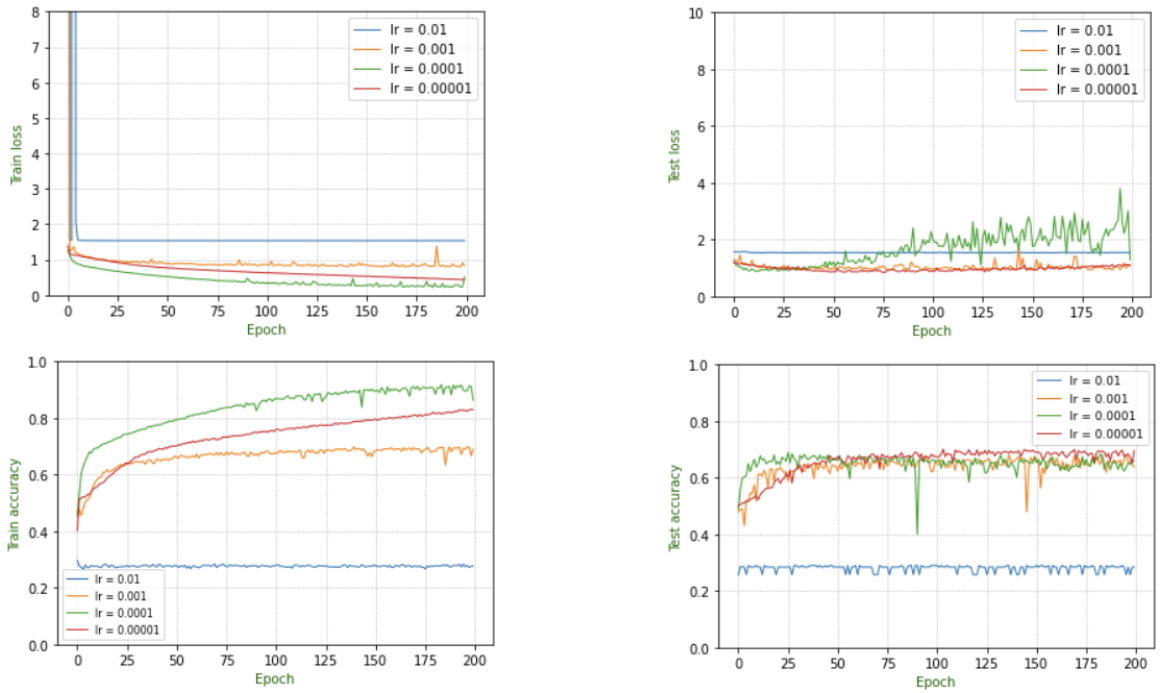

Figura 6.12: *Loss* (arriba) y *accuracy* (debajo) medio por época para los datos de entrenamiento (izquierda) y test (derecha) (distintos *lr*)

Para tener una visión unificada de cómo afecta la variación de la tasa de aprendizaje en la evolución del *loss* y la tasa de acierto se representan conjuntamente en la Figura 6.12. La pérdida en el conjunto de entrenamiento sigue un patrón similar a lo largo de las épocas cuando el *learning rate* vale 0.001, 0.0001 y 0.00001, siendo la que mejor resultados ofrece la segunda de las mencionadas. Sin embargo, sobre el conjunto de test, son las otras 2 (lr = 0.001 y lr = 0.00001) las que ofrecen un resultado más óptimo. El *overfitting* provoca que el *loss* sobre el conjunto de test cuando el parámetro lr vale 0.0001 empeore mucho más que en el resto de experimentos. Como ya se ha comentado anteriormente, cuando la tasa de aprendizaje vale 0.01, los pesos no convergen a ningún valor óptimo y el comportamiento de la red es el de un clasificador aleatorio.

Respecto a la precisión sobre el conjunto de entrenamiento, los resultados son los mismos que los extraídos a partir del *loss*. La mejor configuración es aquella con tasa de aprendizaje igual a 0.0001. Sobre el conjunto de test, todas las configuraciones menos la que no consigue aprender obtienen unos resultados muy similares, con tasas de acierto entorno al 70 %. En esta comparación se aprecia muy fácilmente como las configuraciones con tasa de aprendizaje 0.0001 y 0.00001 sufren sobreajuste y no generalizan de acuerdo a su rendimiento en el conjunto de entrenamiento sobre el conjunto de test.

6.2.3.3 Canales RGBα

Realizado este cambio respecto al experimento base, se mantiene el resto de la configuración mostrada en el Cuadro 6.8. Curiosamente, en la Figura 6.13 se aprecia como vuelve a suceder que la red no aprende, estancándose en el proceso de optimización de la función de pérdida. El *loss* en la etapa de test es más variable que en la de entrenamiento según se suceden las épocas, y el *accuracy* es ligeramente superior, no superando el 30 % en ningún caso, por lo que el desempeño es muy pobre.

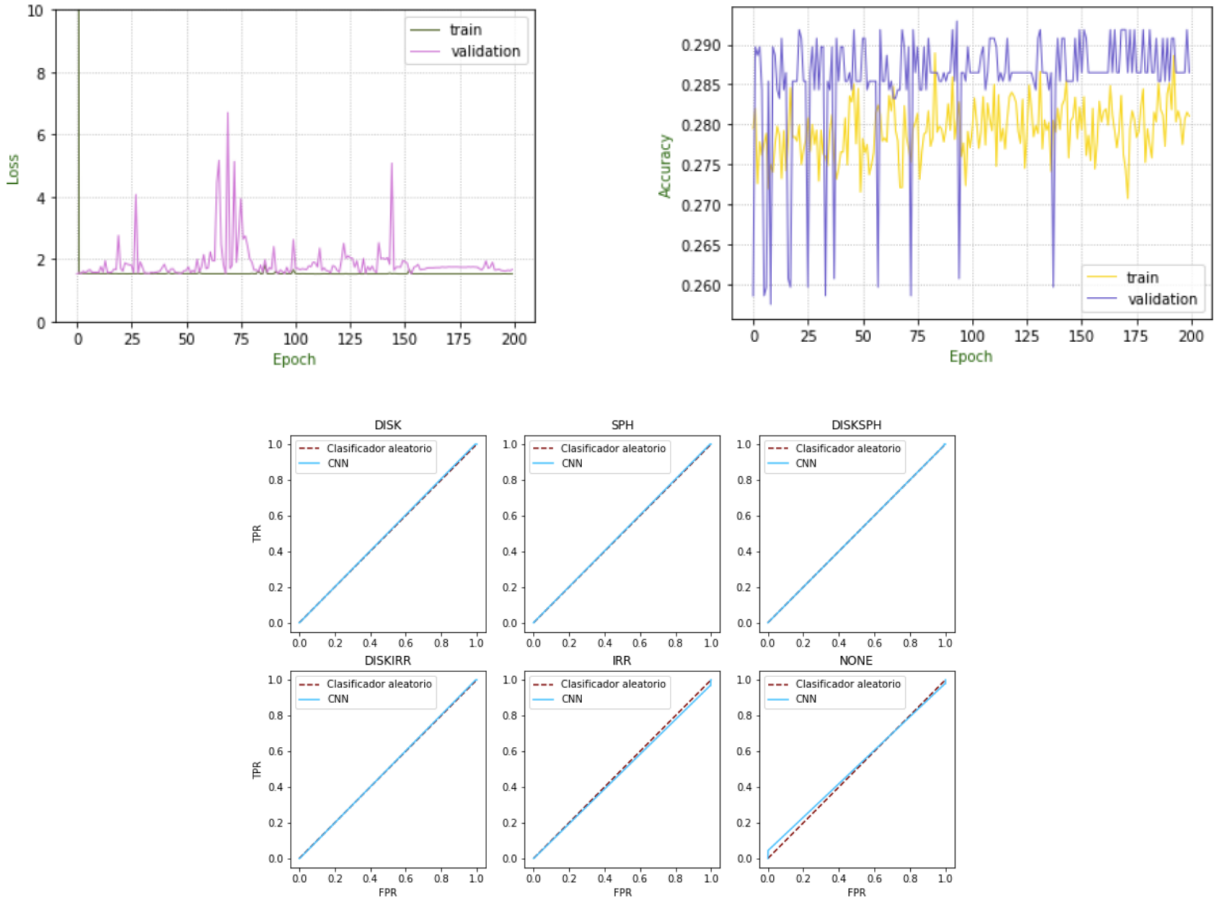

Figura 6.13: *Loss* (izquierda) y *accuracy* (derecha) medio por época para los datos de entrenamiento y test y curvas ROC (Canales RGBα).

Las curvas ROC y los AUC disponibles en la Figura 6.13 y el Cuadro 6.12 contribuyen a reforzar las conclusiones extraídas visualizando la evolución del *loss* y del *accuracy*. Como ya ha ocurrido con otro experimento anterior, las curvas ROC de las adaptaciones binarias del problema son prácticamente las de un clasificador aleatorio, denotando una falta de aprendizaje evidente. La decisión que toma la red a la hora de clasificar una galaxia es prácticamente aleatoria.

Problema binario	AUC
DISK frente los demás	0.504
SPH frente los demás	0.503
DISKSPH frente los demás	0.500
DISKIRR frente los demás	0.502
IRR frente los demás	0.486
NONE frente los demás	0.512

Cuadro 6.12: Áreas bajo la curva de los problemas de clasificación dicotómica para el experimento con canales RGBα.

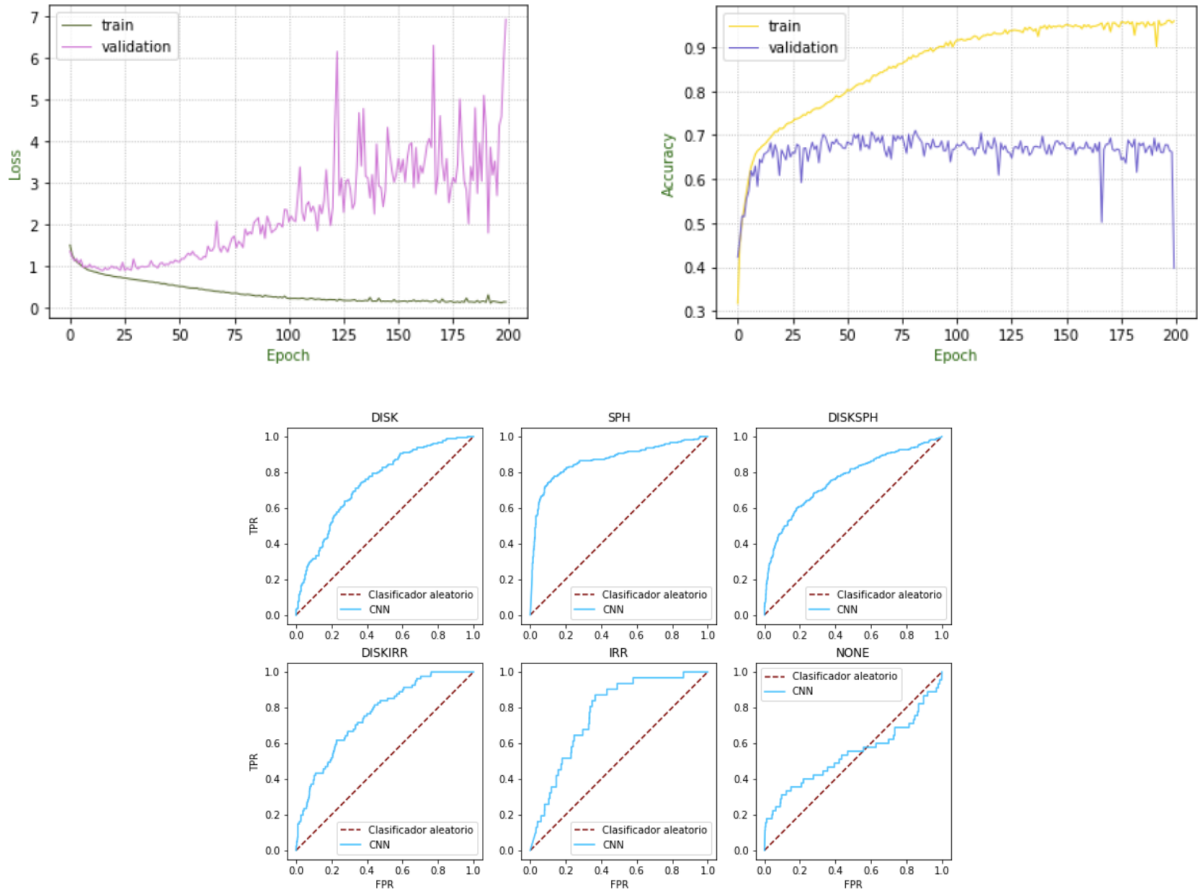

Figura 6.14: *Loss* (izquierda) y *accuracy* (derecha) medio por época para los datos de entrenamiento y test y curvas ROC (Canales RGBα y $lr = 0.0001$).

Para intentar aliviar este problema, y en base a la importancia vista que tiene la tasa de

aprendizaje, se va a probar una nueva configuración, realizar el mismo experimento pero variando el *learning rate*, ahora, en vez de ser 0.001, pasará a ser 0.0001, 10 veces menor.

En la Figura 6.14 se observa como con esta modificación se logra solucionar el problema anterior, y la red consigue aprender. Se tiene un problema de sobreajuste, como viene siendo habitual en todas las configuraciones de la red con un *learning rate* muy bajo. El *loss* sobre el conjunto de test, lejos de disminuir paulatinamente, incrementa a medida que se suceden las épocas, y la precisión se estanca entorno a un 70 % a partir de la época 50.

Analizando las curvas ROC y sus áreas, se determina que el problema binario en el que la red se desenvuelve con mayor destreza es en el de discriminar galaxias de tipo SPH frente a a galaxias que no son del tipo SPH. A éste le siguen los problemas donde la categoría en la que clasificar o no es IRR, DISKSPH, DISKIRR, DISK y NONE respectivamente (Cuadro 6.13).

Problema binario	**AUC**
DISK frente los demás	0.741
SPH frente los demás	0.868
DISKSPH frente los demás	0.760
DISKIRR frente los demás	0.759
IRR frente los demás	0.770
NONE frente los demás	0.540

Cuadro 6.13: Áreas bajo la curva de los problemas de clasificación dicotómica para el experimento con los canales RGBα y $lr = 0.0001$.

Al igual que en la serie de configuraciones anteriores, se representan conjuntamente en la Figura 6.15 la evolución del *loss* y el *accuracy* sobre los conjuntos de entrenamiento y test de las dos configuraciones que incluyen la que utiliza los canales RGBα con el experimento base.

Sobre el conjunto de entrenamiento, la configuración más óptima es la que utiliza los canales RGBα y la tasa de aprendizaje vale 0.0001, seguido del experimento base, y por último la configuración que no aprende: canales RGBα y tasa de aprendizaje 0.001.

Valorando conjuntamente el *loss* y la precisión sobre el conjunto de test, se observa como las dos mejores configuraciones sobre el conjunto de entrenamiento también son las mejores sobre el conjunto de test, padeciendo de sobreajuste la que incluye un canal suavizado y tasa de aprendizaje 0.0001.

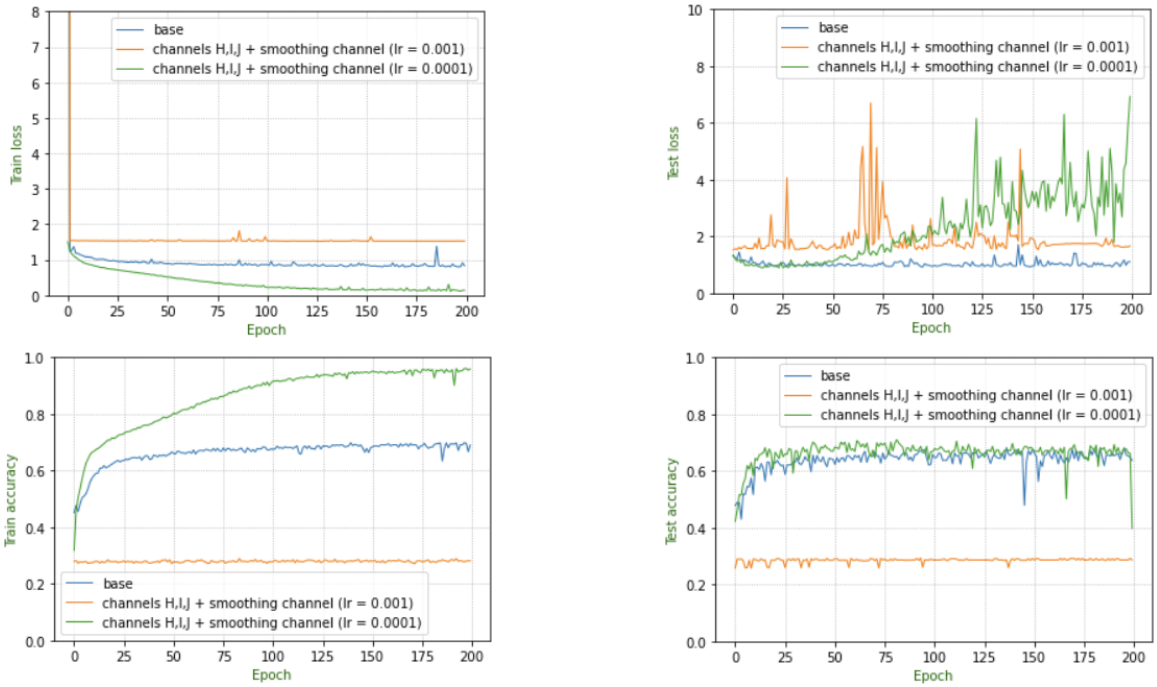

Figura 6.15: *Loss* (arriba) y *accuracy* (debajo) medio por época para los datos de entrenamiento (izquierda) y test (derecha) (base vs canales RGBα con $lr = 0.001$ y 0.0001).

6.2.3.4 Comparación de resultados

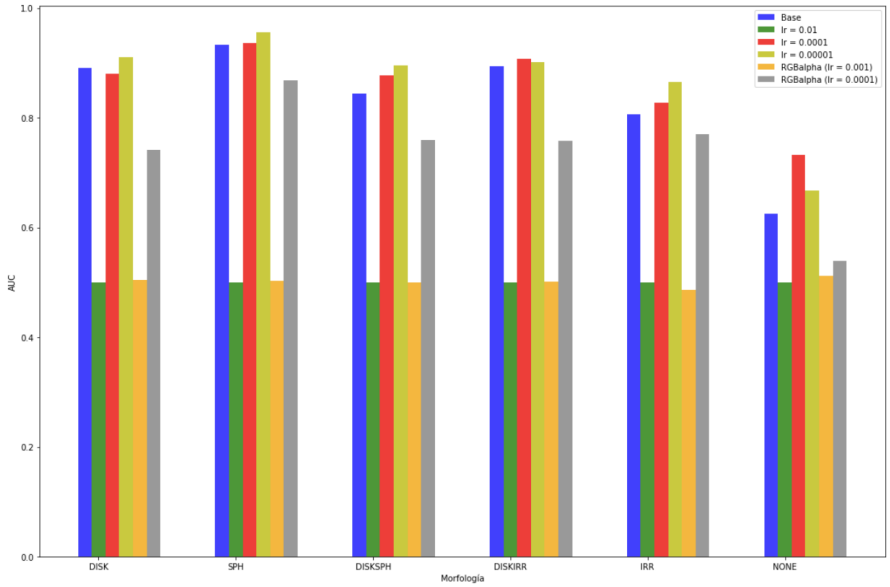

Figura 6.16: AUCs de las distintas configuraciones y morfologías.

En líneas generales, el experimento que mejores resultados ha obtenido en base al AUC es el que tiene una tasa de aprendizaje más baja (0.00001), seguido de la configuración con $lr = 0.0001$ y del experimento base. Los experimentos con tasa de aprendizaje más alta (0.01) y con canales RGBα y $lr = 0.001$ obtienen resultados muy similares a los que obtendría un clasificador aleatorio. La configuración con canales RGBα y $lr = 0.0001$ se encuentra entre medias de los mejores experimentos y los peores (Figura 6.16).

6.3. Regresión

Una alternativa al problema de clasificación morfológica directa de las galaxias es atacar primero un problema de regresión y, una vez obtenida la solución, transformar de alguna manera la o las salidas en una etiqueta que caracterice la forma de la galaxia [26].

Para plantear un problema de regresión se requiere de un número real o un vector p-dimensional de valores reales para cada muestra del conjunto de datos, imágenes de galaxias en este caso. Con este fin, se va a sacar ventaja del trabajo realizado por un conjunto de expertos en la tarea de discriminar galaxias visualmente en base a su morfología. Para ello, se cuenta con 5 indicadores. Cada uno de ellos representa la proporción de expertos que seleccionaron la característica morfológica de la galaxia asociada al indicador.

Para cada galaxia se disponen de 5 indicadores o *flags*: $f_{spheroid}$, f_{disk}, f_{irr}, f_{PS} y f_{Unc}, haciendo referencia a la frecuencia que los expertos asignaron a cada galaxia de tener una forma de tipo esferoidal, de tipo disco, de presentar algunas irregularidades, de ser irresoluble o *point source* y de ser inclasificable respectivamente.

Por tanto, el objetivo de las técnicas desarrolladas en esta sección será imitar el comportamiento humano. La metodología propuesta tratará de predecir la frecuencia de votación de cada una de las 5 características morfológicas de una galaxia dada. Si los indicadores obtenidos por la votación de los expertos para un imagen son erróneos, también lo será la predicción del método que consiga replicar razonablemente bien dichos *flags*. Por tanto, es probable que haya un ligero componente de error y sesgo introducido por el humano en este problema. En el caso de que lo hubiese, los métodos propuestos intentarán replicarlo.

Un aspecto importante a remarcar en la variable respuesta *5*-dimensional es que, salvo en el caso del indicador f_{Unc}, un indicador no excluye el resto. Esto es, un experto puede decir que una galaxia tiene forma de esfera y forma de disco, lo que significa que la suma de las 5 frecuencias no tiene por qué ser 1.

En este enfoque de un problema de regresión, se probarán 3 configuraciones de redes neuronales convolucionales diferentes. Primero, una red de diseño propio, posteriormente, la red de diseño propio adaptada de [26] utilizada en el problema de clasificación directo (adaptada al nuevo problema), y por último una red neuronal residual, concretamente la versión que consta de 18 capas ocultas popularmente conocida como Resnet18. La complejidad de las redes en cuanto al número de parámetros entrenables, es decir estimables, va creciendo respectivamente, contando aproximadamente la primera con 70 mil, la segunda con 8 millones y la tercera con 11 millones.

Otro aspecto a comentar es la manera en que se obtiene la métrica precisión, típica de los problemas de clasificación, a partir de las salidas de un problema de regresión. La manera de obtenerla es muy sencilla, basta con seleccionar la posición que tiene un valor mayor del vector 5-dimensional calculado como predicción por la red, y ver si esa posición es la misma que la que contiene el valor mayor del vector que conforma la etiqueta de la muestra. Si la posición es la misma, entones se considera un acierto de la red en la clasificación de la red, en caso contrario se considera un fallo. Como se explica en [26], y para no tener en cuenta aquellas galaxias en la que los expertos no se ponen de acuerdo en cuanto a su morfología, solo contribuirán a calcular la precisión aquellas imágenes en cuya etiqueta haya al menos una diferencia de 0.5 respecto a los dos *flags* más altos.

A continuación se ve la función creada en PyTorch para calcular la métrica recibiendo como argumentos un tensor con las predicciones de la red para varias imágenes y un tensor con las etiquetas de las mismas imágenes.

```
def accuracy(outputs, labels):
  total = 0
  hits = 0
  for i in range(labels.shape[0]):
  labels_sort = labels[i,:].sort(0, descending = True)[0]
    if labels_sort[0].item() > (labels_sort[1].item() + 0.5):
      total += 1
      if torch.argmax(labels[i,:]).item() == torch.argmax(outputs[i,:]).item():
        hits += 1
  return(hits/total)
```

Por ejemplo, la combinación de etiqueta $y_i = (0.8, 0.2, 0.11, 0, 0)$ y predicción $\hat{y}_i = (0.7, 0.1, 0.4, 0, 0.12, 0)$ es considerada como un acierto, mientras que la combinación de etiqueta $y_i = (0.8, 0.2, 0.11, 0, 0)$ y predicción $\hat{y}_i = (0.8, 0.1, 0.85, 0, 0.12, 0)$ es considerada como un fallo.

6.3.1. Experimentación y métricas

Al igual que ya se realizó durante la etapa de clasificación morfológica directa y con el objetivo de valorar como afectan distintas configuraciones de la red neuronal y del experimento, se define un experimento base. Los resultados de esta primera aproximación serán utilizados como base para realizar comparaciones con otros experimentos.

La primera serie de experimentos que se llevará a cabo consistirá en variar la tasa de aprendizaje, y la segunda en modificar las imágenes de entrada para adaptarlas al formato RGBα, tal y como se detalló en el problema de clasificación directa.

La última experimentación consistirá en aplicar la técnica conocida como *label smoothing*. Dicha práctica consiste en aplicar ruido a las etiquetas, con el objetivo de reducir la certeza del etiquetado original, como se aprecia en la Figura 6.17. Por ejemplo, suponer que se tiene un problema de clasificación multiclase en el que las etiquetas toman 3 valores. Codificando la clase con la codificación *one hot*, la etiqueta de una observación perteneciente a la primera clase sería (1,0,0), y la de una observación perteneciente a la tercera clase (0,0,1). Esta asignación *hard label* denota una confianza plena en la clase a la que pertenece la instancia, y puede que induzca a que un algoritmo de *deep learning* sea extremadamente seguro en sus predicciones causando sobreajuste. Para evitar esto, la técnica de *label smoothing* suaviza el etiquetado añadiendo un factor de corrección. La diferencia entre el factor de corrección y uno es multiplicado por el valor de cada etiqueta, y a dicho valor se le añade el cociente entre el factor de corrección y el número de clases. Con esta técnica, y factor de corrección 0.1, la asignación anterior (0,0,1) pasaría a ser (0.03, 0.03, 0.93).

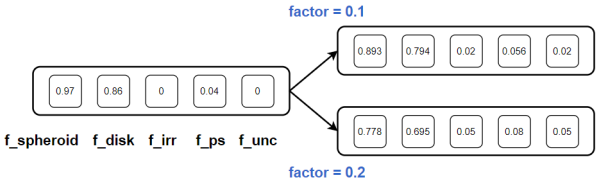

Figura 6.17: *Label smoothing*.

El código creado para aplicar *label smoothing*, con factor de corrección del 10 % por defecto, a un tensor que contiene los 5 parámetros de la regresión es el siguiente:

```
def label_smoothing(labels, factor=0.1):
  labels *= (1 - factor)
  labels += (factor / labels.shape[1])
  return labels
```

Para cada experimento, se calcularán las siguientes métricas:

- Evolución del *loss* medio por época sobre el conjunto de entrenamiento.
- Evolución del *loss* medio por época sobre el conjunto de test.
- Evolución de la tasa de acierto media por época sobre el conjunto de entrenamiento.
- Evolución de la tasa de acierto media por época sobre el conjunto de test.
- Coeficiente R^2 del ACC (Análisis de Correlaciones Canónicas). El ACC es un método de análisis multivariante que trata de buscar y cuantificar las relaciones que haya entre dos grupos de variables. Mientras que un análisis de correlación múltiple se predice una única variable respuesta en base a varias variables explicativas, en el Análisis de Correlaciones Canónicas se predicen varias variables respuesta en base a varias variables explicativas. Como en este problema tanto la salida de la red como la variable respuesta está formada por un conjunto de 5 variables, este método se convierte en idóneo para cuantificar el grado de correlación entre las predicciones de la red y las etiquetas. El coeficiente de correlación R^2 del ACC cuantifica el grado de relación, pudiendo oscilar entre valores negativos y 1 (el nivel máximo de correlación). Si el coeficiente de determinación R^2 es negativo significa que la proporción de variabilidad explicada por la matriz de predicciones (predicciones para las imágenes apiladas en filas) de la matriz de etiquetas (etiquetas de las imágenes apiladas en filas) es peor que la que explicaría un vector con valores constantes, es decir un hiperplano en $\mathbb{R}^5$.

6.3.2. Red convolucional de diseño propio

El diseño de una arquitectura propia responde a la necesidad de probar una red convolucional con un número pequeño de parámetros (~ 70000), lo cuál se logra apilando 4 capas convolucionales a una capa *fully connected*. Tanto la arquitectura de la red de *deep learning* como los resultados que obtiene son expuestos en esta sección.

6.3.2.1 Arquitectura

La primera arquitectura de red neuronal convolucional que se va a utilizar en esta sección es de diseño propio. Se trata de una red sencilla, con un número de parámetros mucho menor que las otras dos redes convolucionales que se van a utilizar para atacar el problema de regresión. Para lograr la reducción de parámetros las imágenes se reducen de tamaño. De los 100x100 píxeles que se tenían se pasa a 28x28 píxeles. Aunque esta redimensión de las imágenes muy probablemente conlleve a una pérdida de información resulta interesante estudiar si dicha pérdida es notable en los resultados finales que obtenga la red.

El porqué de la elección de una red sencilla responde a la utilidad de comparar los resultados obtenidos para este problema por arquitecturas de distinta complejidad. Al final de esta sección se observará si arquitecturas más complejas inducen a métricas más deseables o, por el contrario, son preferibles los resultados obtenidos, así como la evolución de los mismos, por este modelo de red.

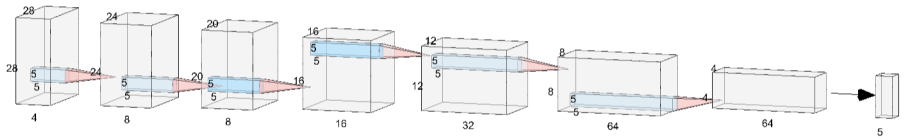

Figura 6.18: Arquitectura 1 diseñada para el problema de regresión.

La red consta de 5 capas, 4 capas convolucionales y 1 capa *fully connected*. Las 2 primeras capas convolucionales y la última capa convolucional están formadas por una operación de convolución seguidas de una función de activación no lineal (ReLU) y de una operación de *pooling* del máximo. La tercera capa convolucional únicamente contiene una operación de convolución seguida de una función de activación. La capa lineal consta de 5 neuronas, necesarias para modelizar las 5 salidas presentes en el problema (los 5 indicadores obtenidos a partir de la votación de los expertos). Una representación de la arquitectura está en la Figura 4.18.

La reducción de las dimensiones que sufre el volumen inicial de dimensiones 4x28x28 según las imágenes fluyen por las distintas operaciones que se realizan en la red se especifica en el Cuadro

6.14. La configuración de cada una de las operaciones se encuentra el Cuadro 6.15.

Operación	**Dimensionalidad volumen salida**	**# Parámetros**
Convolucional	8x24x24	808
ReLU	8x24x24	0
Max Pool	8x20x20	0
Convolucional	16x16x16	3216
ReLU	16x16x16	0
Convolucional	32x12x12	12832
ReLU	32x12x12	0
Convolucional	64x8x8	51264
ReLU	64x8x8	0
Max Pool	64x4x4	0
FC	5	5125
Total	–	**73245**

Cuadro 6.14: Variación de la dimensionalidad del volumen de entrada y # parámetros.

Operación	***Kernel***	***Padding***	***Stride***
Convolucional	5	0	1
Max Pool	5	0	1
Convolucional	5	0	1
Convolucional	5	0	1
Convolucional	5	0	1
Max Pool	5	0	1

Cuadro 6.15: Configuración de las capas convolucionales y de *pooling*.

La red convolucional diseñada se crea en PyTorch de la siguiente manera:

```
class ConvNet(nn.Module):
  def __init__(self):
    super(ConvNet, self).__init__()
    self.layer1 = nn.Sequential(
      nn.Conv2d(4, 8, kernel_size=5, stride=1, padding=0),
      nn.ReLU(),
      nn.MaxPool2d(5, stride=1))
    self.layer2 = nn.Sequential(
      nn.Conv2d(8, 16, kernel_size=5, stride=1, padding=0),
      nn.ReLU())
```

```
    self.layer3 = nn.Sequential(
      nn.Conv2d(16, 32, kernel_size=5, stride=1, padding=0),
      nn.ReLU())
    self.layer4 = nn.Sequential(
      nn.Conv2d(32, 64, kernel_size=5, stride=1, padding=0),
      nn.ReLU(),
      nn.MaxPool2d(kernel_size=5, stride=1, padding=0))
    self.fc1 = nn.Linear(4*4*64, 5)
  def forward(self, x):
    out = self.layer1(x)
    out = self.layer2(out)
    out = self.layer3(out)
    out = self.layer4(out)
    out = out.reshape(out.size(0), -1)
    out = self.fc1(out)
    return out
```

6.3.2.2 Resultados

Los resultados obtenidos por la arquitectura tratada en esta sección se exponen a continuación.

6.3.2.2.1. Experimento base

La configuración de la red neuronal convolucional usada como experimento base se muestra en el Cuadro 6.16. Al igual que en el problema de clasificación directo, el número de instancias de entrenamiento es de 12432 y el número de muestras reservadas para test son 932, aproximadamente un 30 % del conjunto de datos previo a la realización de *data augmentation* (*hold out sin repetición* como método para estimar las métricas sobre el conjunto de test). Recordar que estas 932 instancias son las que realmente van a proporcionar un medida realista del rendimiento de la red.

# imágenes de entrenamiento	12432
# imágenes de test	932
Función de pérdida	RMSE
Optimizador	SGD
Tamaño de *batch*	28
Tasa de aprendizaje	0.001
Épocas	800

Cuadro 6.16: Configuración del experimento base.

En la Figura 6.19 se aprecia como el *loss* en los conjuntos de entrenamiento y test sigue un patrón muy similar, disminuyendo de manera pareja conforme se van sucediendo las épocas.

Esto se traduce en que la precisión sobre los conjuntos de entrenamiento y test también son muy similares. El resultado es excelente, y la mejora sobre los resultados obtenidos en el problema de clasificación anterior palmaria. La tasa de acierto es superior al 95 % sobre el conjunto de test.

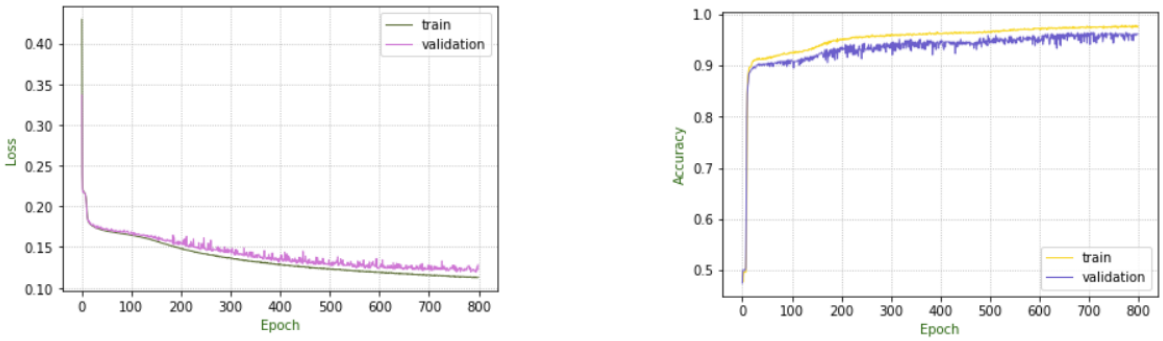

Figura 6.19: *Loss* (izquierda) y *accuracy* (derecha) medio por época para los datos de entrenamiento y test (base).

6.3.2.2.2. Variaciones en el *learning rate*

Para ver como afecta el parámetro *learning rate* en este problema, se lanzarán 3 modificaciones del experimento base, variando únicamente la tasa de aprendizaje respecto a la configuración especificada en el Cuadro 6.16. Las 3 tasas de aprendizaje que se probará son 0.01, 0.0001 y 0.00001, siendo 0.001 la del experimento base.

Cuando la tasa de aprendizaje es 0.01 (Figura 6.20), se aprecia que hay un ligero problema de *overfitting*, tanto el loss como la tasa de acierto sobre el conjunto de entrenamiento no paran de disminuir y aumentar respectivamente, mientras que sobre el conjunto de test se estabilizan entorno a las época 200, exhibiendo una tendencia con pendiente nula desde ese momento.

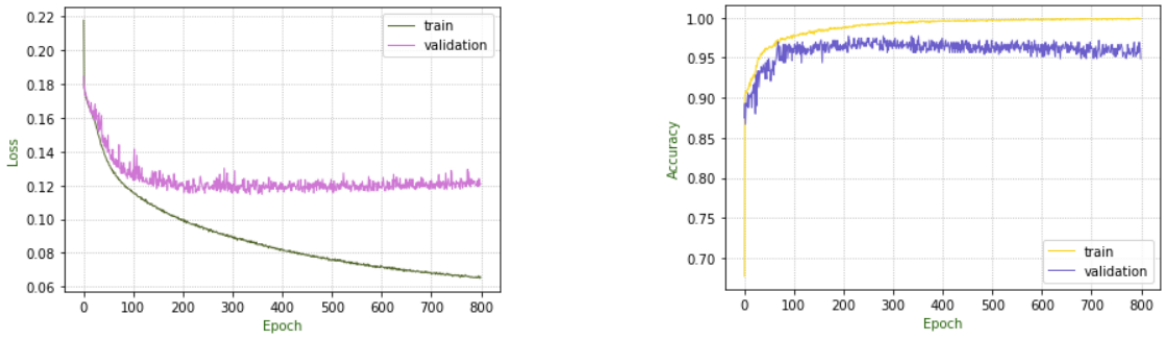

Figura 6.20: *Loss* (izquierda) y *accuracy* (derecha) medio por época para los datos de entrenamiento y test ($lr = 0.01$).

Cuando la tasa de aprendizaje disminuye 10 y 100 veces respecto a la del experimento base, situándose en 0.0001 y 0.00001 (Figuras 6.21 y 6.22), se elimina el sobreajuste presente en el experimento inmediatamente anterior. La capacidad de discriminación que tiene la red sobre los conjuntos de entrenamiento y test es muy similar. En el último caso, cuando el *learning rate* toma el valor 0.00001, la tasa de acierto sufre una caída fuerte poco después de la época 100, se recupera rápidamente y luego exhibe una evolución con pendiente 0 hasta aproximadamente la época 600, donde comienza a aumentar de manera notable su precisión en ambos conjuntos.

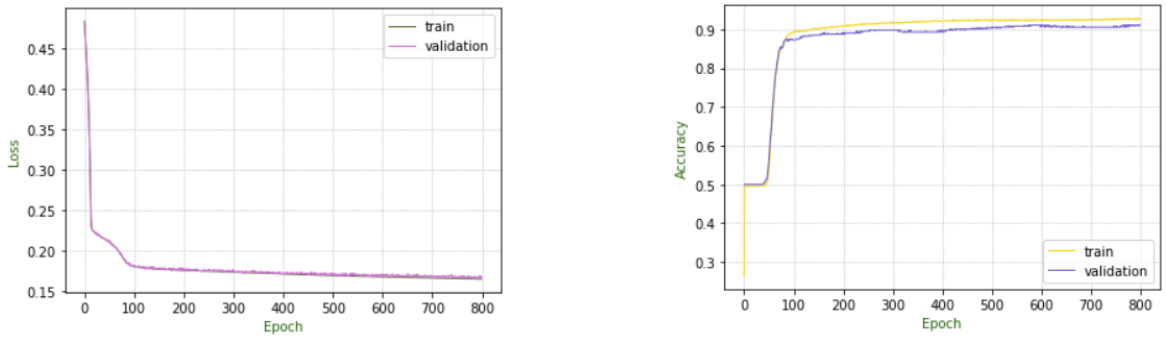

Figura 6.21: *Loss* (izquierda) y *accuracy* (derecha) medio por época para los datos de entrenamiento y test ($lr = 0.0001$).

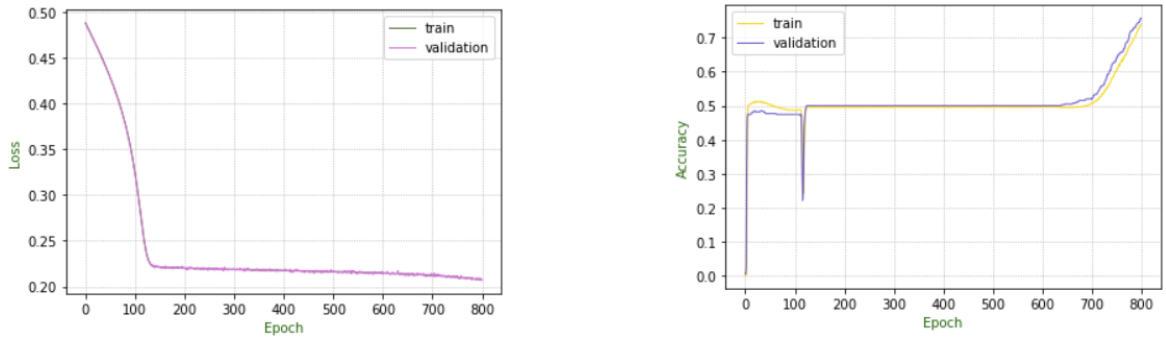

Figura 6.22: *Loss* (izquierda) y *accuracy* (derecha) medio por época para los datos de entrenamiento y test ($lr = 0.00001$).

Comparando de manera conjunta las 3 configuraciones donde se varía la tasa de aprendizaje y el experimento base (Figura 6.23), se determina que los dos experimentos que inducen a menor *loss* y mayor tasa de acierto sobre el conjunto de test son el base y el que tiene la tasa de aprendizaje más alta de todas. Posteriormente, se sitúan los que tienen un *learning rate* de 0.0001 y 0.00001 respectivamente. No hay un problema de sobreajuste evidente en ninguna de las configuraciones. También se aprecia como las configuraciones con una tasa de aprendizaje más baja tardan más

aprender las características relevantes de las muestras, debido a que la actualización de los pesos durante la etapa de *backpropagation* se realiza con cantidades más pequeñas.

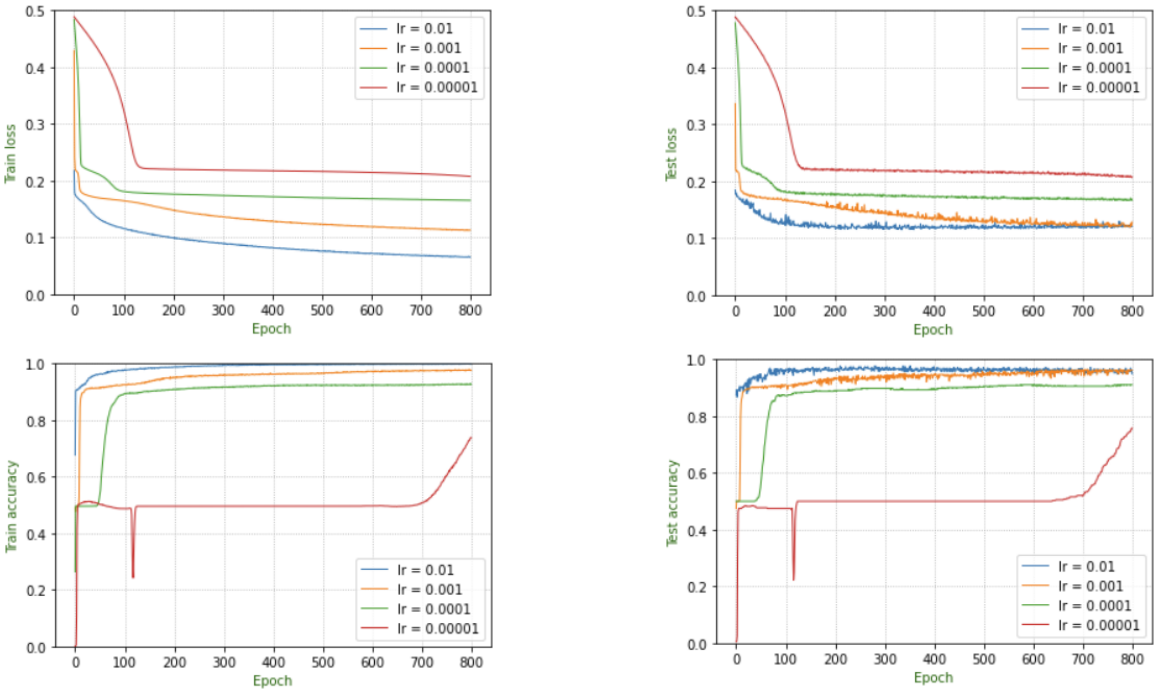

Figura 6.23: *Loss* (arriba) y *accuracy* (debajo) medio por época para los datos de entrenamiento (izquierda) y test (derecha) (base vs $lr = 0.001$, 0.0001 y 0.00001).

6.3.2.2.3. *Label smoothing*

Para ver como afecta la técnica de *label smoothing*, se diseña un experimento en el que se mantienen fijos los parámetros del experimento base. El único cambio respecto a este es la aplicación de esta técnica. Recordar que este método introduce ruido en las etiquetas de las muestras, los 5 parámetros de la regresión en este caso. Por ello, sólo se deben suavizar las etiquetas del conjunto de entrenamiento y nos las del de test, ya que en ese caso se estaría calculando el loss y la tasa de acierto sobre el conjunto de test en base a valores que no son reales, lo cual es inadmisible.

Aplicando la técnica de *label smoothing* se observa un comportamiento a priori similar al del experimento base. No parece haber un problema de *overfitting*, y la evolución de las métricas sobre ambos conjuntos de datos es muy pareja como se ve en la Figura 6.24.

Comparando el experimento base con el experimento donde se aplica *label smoothing* salta a la vista la similitud de resultados. Las evoluciones que se muestran en las 4 gráficas de la Figura 6.25 son muy similares, prácticamente idénticas. La única en la que se puede apreciar una ligera

diferencia es en la del *loss* durante el entrenamiento de la red, siendo algo menor siempre el de la configuración con *label smoothing* que el del experimento base. Ambas configuraciones carecen de sobreajuste y ofrecen un rendimiento excelente, atendiendo a las tasas de acierto sobre el conjunto de test superiores al 95 %.

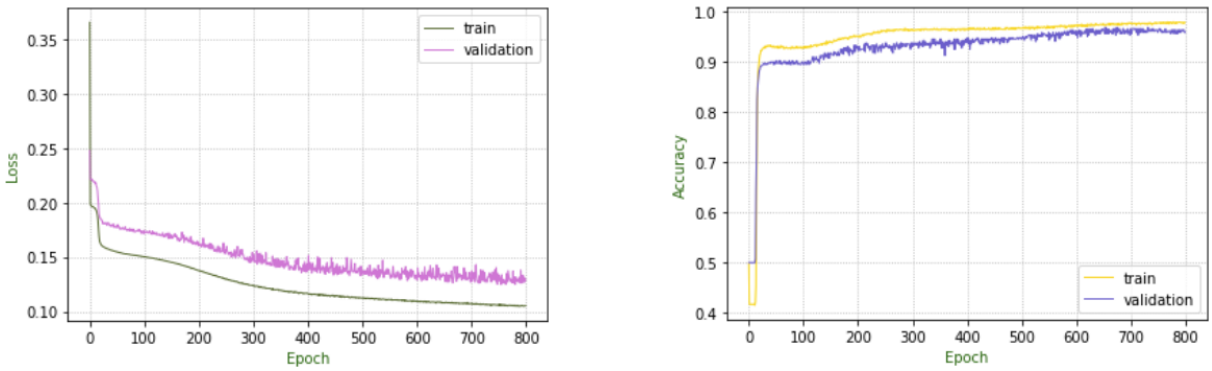

Figura 6.24: *Loss* (izquierda) y *accuracy* (derecha) medio por época para los datos de entrenamiento y test (*label smoothing*).

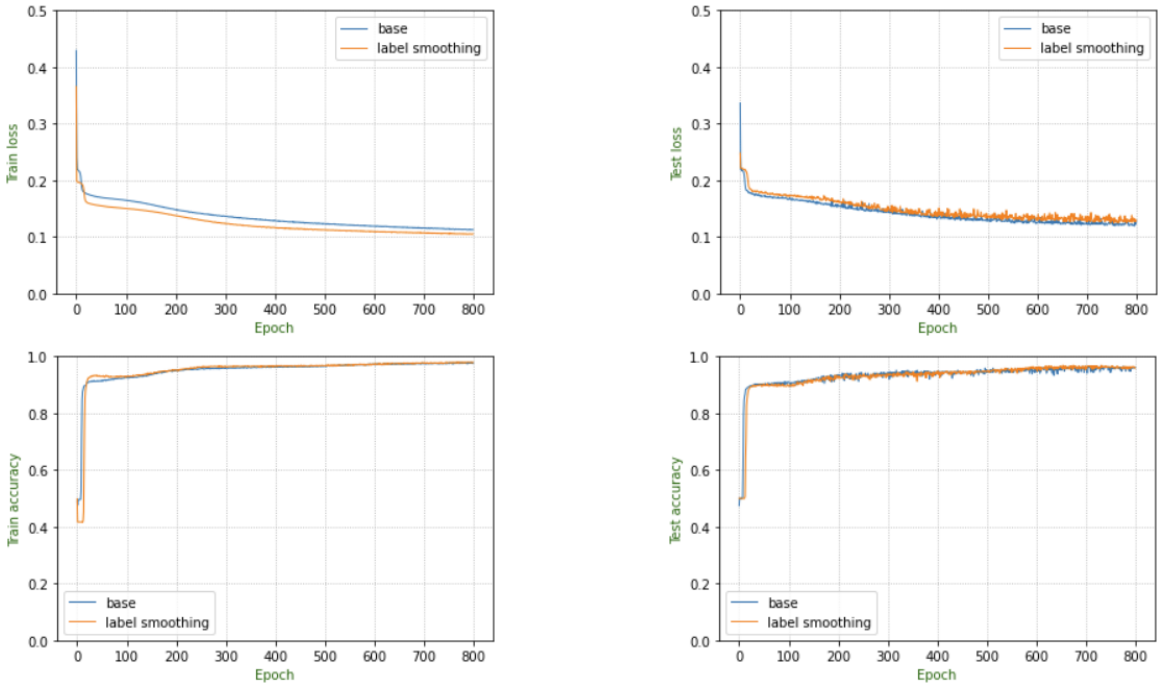

Figura 6.25: *Loss* (arriba) y *accuracy* (debajo) medio por época para los datos de entrenamiento (izquierda) y test (derecha) (base vs *label smoothing*).

6.3.2.2.4. Canales RGBα

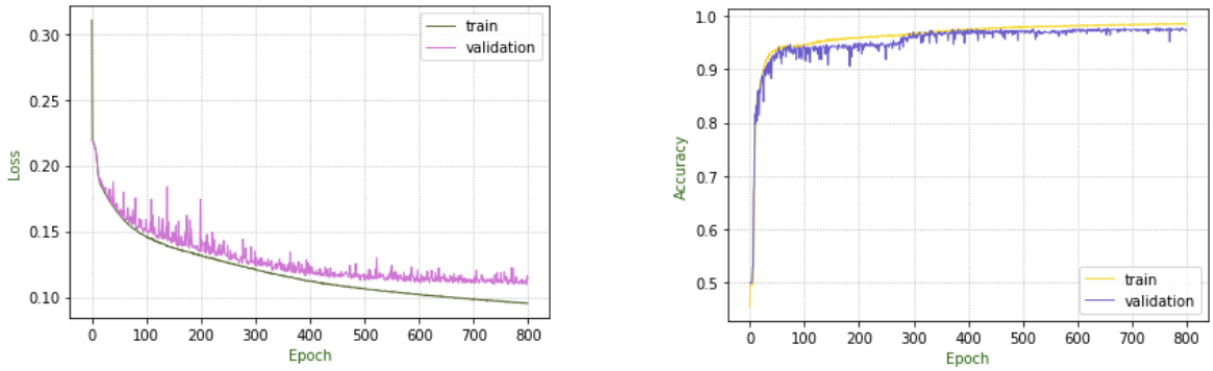

Figura 6.26: *Loss* (izquierda) y *accuracy* (derecha) medio por época para los datos de entrenamiento y test (canales RGBα).

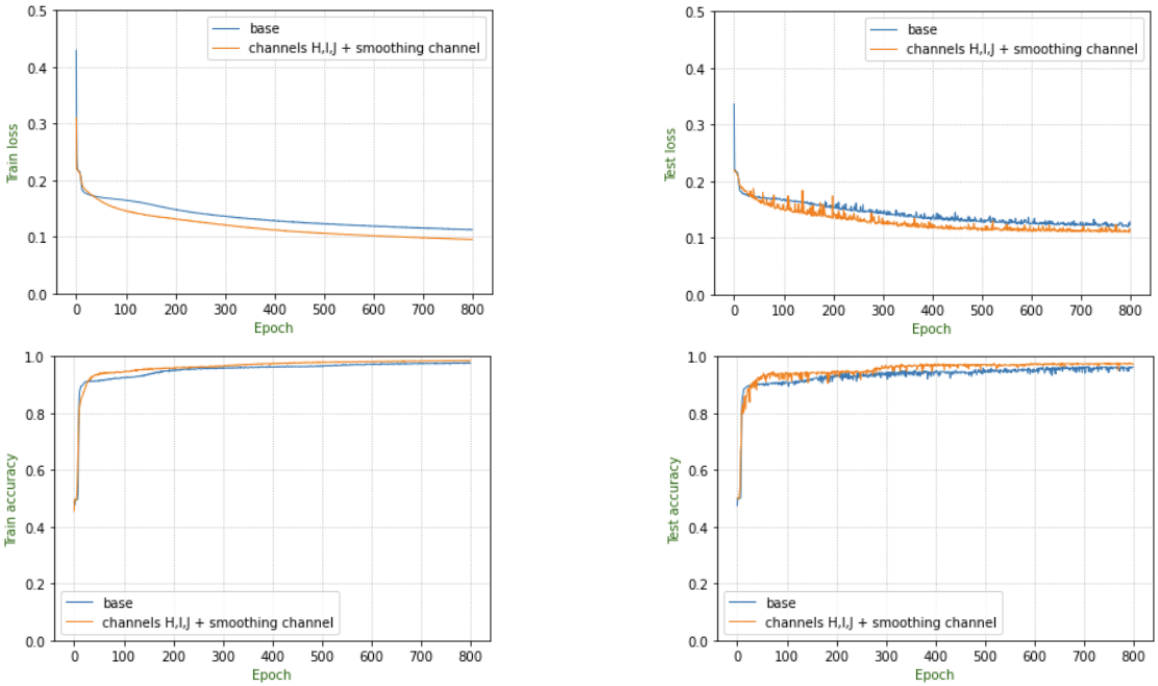

Figura 6.27: *Loss* (arriba) y *accuracy* (debajo) medio por época para los datos de entrenamiento (izquierda) y test (derecha) (base vs canales RGBα).

Cuando cambiamos los canales HIJV por los canales RGBα se observa que el *loss* sobre el conjunto de test tiene más varianza que sobre el conjunto de entrenamiento, es decir, un comportamiento más oscilatorio. Estas oscilaciones se mitigan a partir de la época 400, momento en el

dicha métrica sobre el conjunto de test se estabiliza entorno a 0.12. La precisión es muy buena en ambos conjuntos, sin apreciarse sobreajuste (Figura 6.26).

La comparación de la configuración con los canales RGBα y el experimento base arroja resultados muy similares. El *loss* y la tasa de acierto sobre el conjunto de test muestran que cuando introducimos a la red los nuevos canales, el desempeño es ligeramente mejor, pero las diferencias son mínimas, la evolución de las métricas es prácticamente calcada (Figura 6.27).

A modo resumen, y para establecer una jerarquía basada en el rendimiento de las distintas configuraciones, se observa en el Cuadro 6.17 un resumen con 3 métricas recogidas en cada experimento: el *loss* en la última época, la tasa de acierto en la última época y el R^2 proporcionado por el Análisis de Correlaciones Canónicas, todas ellas medidas sobre el conjunto de test. La configuración que induce a mejores resultados es aquella que utiliza los canales RGBα, consiguiendo los mejores valores en las 3 métricas. En el caso contrario se encuentra la configuración con tasa de aprendizaje 0.00001, siendo el experimento que peores resultados ofrece en las 3 métricas. El coeficiente de determinación R^2 es negativo, lo que refleja que la proporción de variabilidad explicada por la matriz de predicciones de la matriz de etiquetas es peor que la que explicaría un vector con valores constantes, es decir un hiperplano en $\mathbb{R}^5$. Por su parte las configuraciones base, con tasa de aprendizaje 0.01 y aplicando *label smoothing* consiguen métricas muy parejas. El experimento con *learning rate* 0.0001 obtiene peores resultados que todos menos la configuración con menor *learning rate*.

Experimento	***Loss***	***Accuracy***	**CCA** R^2
Base	0.1282	0.9617	0.6789
$lr = 0.01$	0.1211	0.9490	0.6433
$lr = 0.0001$	0.1670	0.9107	0.2322
$lr = 0.00001$	0.2074	0.7577	-2.1438
Label smoothing	0.1294	0.9592	0.6957
Canales RGBα	0.1135	0.9719	0.7092

Cuadro 6.17: Métricas de los experimentos sobre el conjunto de test.

6.3.3. Red convolucional de Huertas-Company

Una red de tamaño medio, con más parámetros que la diseñada en el apartado anterior (arquitectura propia) y con menos que la última que se utilizará (Resnet 18) es tratada en esta sección.

6.3.3.1 Arquitectura

La red diseñada en esta sección es prácticamente idéntica a la utilizada en la etapa de clasificación morfológica directa, y únicamente hay que adaptarla a la particularidad que presenta el problema de regresión. La última capa *fully connected* ahora ha de tener 5 neuronas (representando los 5 parámetros de la regresión) en vez de 6 (los 6 tipos de morfología posibles). Al igual que en el problema de clasificación, las imágenes que se van a utilizar para alimentar a la red son de 100x100 píxeles en 4 canales.

Operación	**Dimensionalidad volumen salida**	**# Parámetros**
Convolucional	8x81x81	12808
ReLU	8x81x81	0
Max Pool	8x62x62	0
Convolucional	16x53x53	12816
ReLU	16x53x53	0
Max Pool	16x45x45	0
Convolucional	32x40x40	18464
ReLU	32x40x40	0
Max Pool	32x20x20	0
Convolucional	64x16x16	51264
ReLU	64x16x16	0
Max Pool	64x9x9	0
Convolucional	128x7x7	73856
ReLU	128x7x7	0
Convolucional	128x5x5	147584
ReLU	128x5x5	0
Max Pool	128x4x4	0
FC	2048	4196352
ReLU	2048	0
FC	2048	4196352
ReLU	2048	0
FC	5	10245
Total	–	**8719741**

Cuadro 6.18: Variación de la dimensionalidad del volumen de entrada y # parámetros.

La configuración y explicación de las capas convolucionales es idéntica a la ya dada anteriormente, únicamente varía el número de parámetros total de la red debido a la reducción en una unidad del tamaño de la última capa lineal (Cuadro 6.18). Una representación de su arquitectura está en la Figura 6.28.

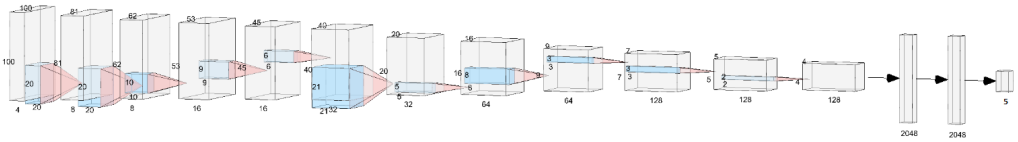

Figura 6.28: Segunda arquitectura diseñada para el problema de regresión.

La creación en PyTorch de la red convolucional especificada es la siguiente:

```
class ConvNet(nn.Module):
  def __init__(self):
    super(ConvNet, self).__init__()
    self.layer1 = nn.Sequential(
      nn.Conv2d(4, 8, kernel_size=20, stride=1, padding=0),
      nn.ReLU(),
      nn.MaxPool2d(20, stride=1))
    self.layer2 = nn.Sequential(
      nn.Conv2d(8, 16, kernel_size=10, stride=1, padding=0),
      nn.ReLU(),
      nn.MaxPool2d(9, stride=1))
    self.layer3 = nn.Sequential(
      nn.Conv2d(16, 32, kernel_size=6, stride=1, padding=0),
      nn.ReLU(),
      nn.MaxPool2d(21, stride=1))
    self.layer4 = nn.Sequential(
      nn.Conv2d(32, 64, kernel_size=5, stride=1, padding=0),
      nn.ReLU(),
      nn.MaxPool2d(8, stride = 1))
    self.layer5 = nn.Sequential(
      nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=0),
      nn.ReLU())
    self.layer6 = nn.Sequential(
      nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=0),
      nn.ReLU(),
      nn.MaxPool2d(2, stride = 1))
    self.fc1 = nn.Linear(4*4*128, 4*4*128)
    self.fc2 = nn.Linear(4*4*128, 4*4*128)
    self.fc3 = nn.Linear(4*4*128, 5)
    self.act = nn.ReLU()
  def forward(self, x):
    out = self.layer1(x)
    out = self.layer2(out)
    out = self.layer3(out)
    out = self.layer4(out)
    out = self.layer5(out)
```

```
    out = self.layer6(out)
    out = out.reshape(out.size(0), -1)
    out = self.act(self.fc1(out))
    out = self.act(self.fc2(out))
    out = self.fc3(out)
    return out
```

6.3.3.2 Resultados

Los resultados obtenidos por la red tratada en esta sección son expuestos a continuación.

6.3.3.2.1. Experimento base

Lo primero que se hace es definir un experimento base. La relación de hiperparámetros que definen este experimento son los mismos que los de la red anterior, únicamente se reducen las épocas de 800 a 400 debido al incremento de la complejidad de la red y el mayor tiempo que requiere su entrenamiento (Cuadro 6.19).

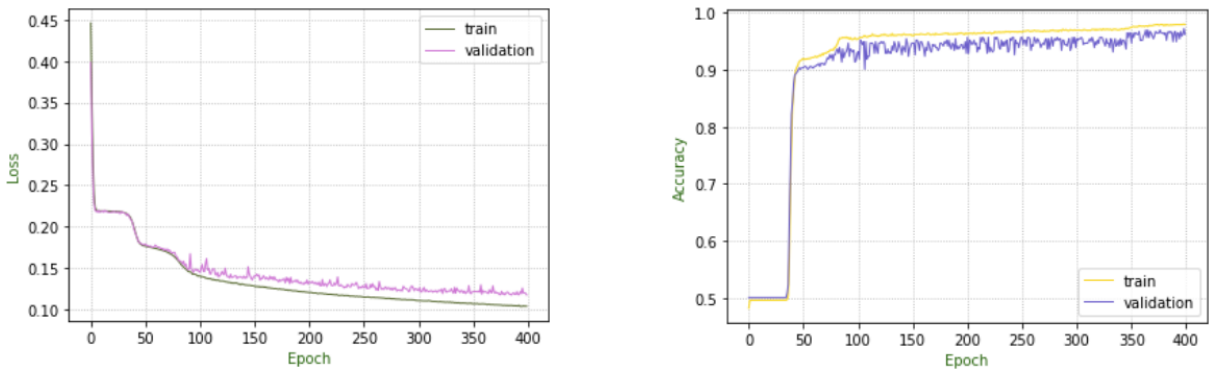

Figura 6.29: *Loss* (izquierda) y *accuracy* (derecha) medio por época para los datos de entrenamiento y test (experimento base).

En la Figura 6.29 se distingue como tanto el *loss* como la tasa de acierto sobre los conjuntos de entrenamiento y test sigue un patrón prácticamente idéntico hasta las primeras 100 épocas. Posteriormente, ambas las medidas sobre ambos conjuntos divergen levemente, aunque no lo suficiente como para hablar de sobreajuste. El resultado es muy bueno, logrando tasas de acierto sobre el conjunto de test superiores al 95 %.

# imágenes de entrenamiento	12432
# imágenes de test	932
Función de pérdida	RMSE
Optimizador	SGD
Tamaño de *batch*	28
Tasa de aprendizaje	0.001
Épocas	400

Cuadro 6.19: Configuración del experimento base.

6.3.3.2.2. Variaciones en el *learning rate*

Asignando a la tasa de aprendizaje un valor 10 veces más alto que el del experimento base, se dislumbra una ligera problemática de sobreajuste en la gráfica de evolución del *loss*, no siendo así en la gráfica de evolución de la tasa de acierto. (Figura 6.30) Esto es así debido a que, a partir de la época 50, las métricas sobre ambos conjuntos avanzan de manera pareja, y a partir de ese momento el margen de mejora es mayor para el *loss* que para la tasa de acierto, por lo que el sobreajuste se hace más evidente en el primer caso.

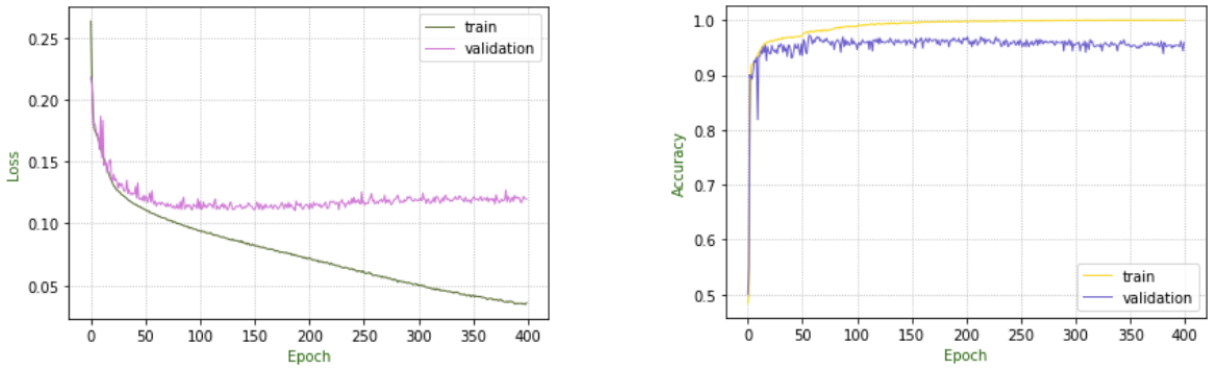

Figura 6.30: *Loss* (izquierda) y *accuracy* (derecha) medio por época para los datos de entrenamiento y test ($lr = 0.01$).

Cuando el learning rate pasa a valer 10 y 100 veces menos que en el experimento base, es decir 0.0001 y 0.00001, la red neuronal convolucional no converge a una solución óptima. Esto se aprecia fácilmente en los gráficos de la evolución de la tasa de acierto de las Figuras 6.31 y 6.32, siendo la tasa de acierto sobre el conjunto de test del 50 %, y 4 milésimas menos sobre el conjunto de entrenamiento. Este desempeño nefasto se debe al fenómeno del *gradient vanishing* o evanescencia del gradiente, por el que se impide que los pesos de la red neuronal cambien su valor debido a que el gradiente toma valores prácticamente nulos, y como consecuencia la red no es capaz de aprender apenas ninguna característica relevante de la imagen.

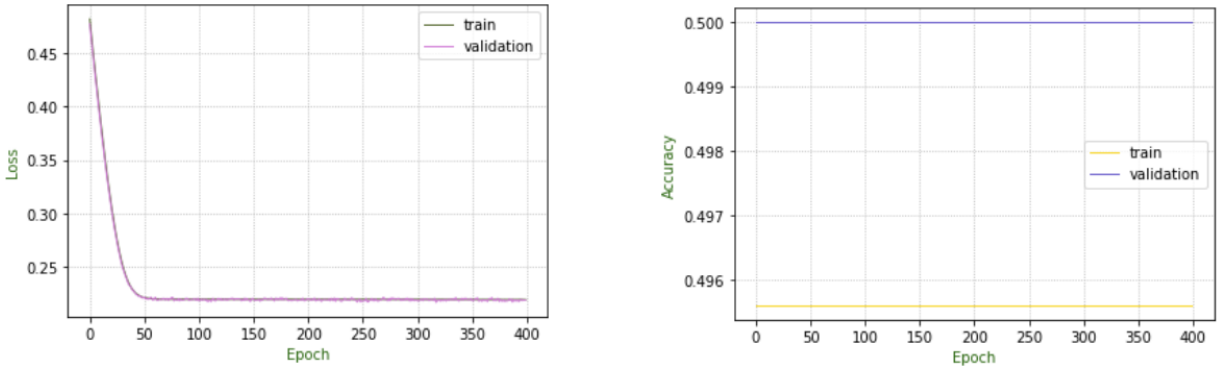

Figura 6.31: *Loss* (izquierda) y *accuracy* (derecha) medio por época para los datos de entrenamiento y test ($lr = 0.0001$).

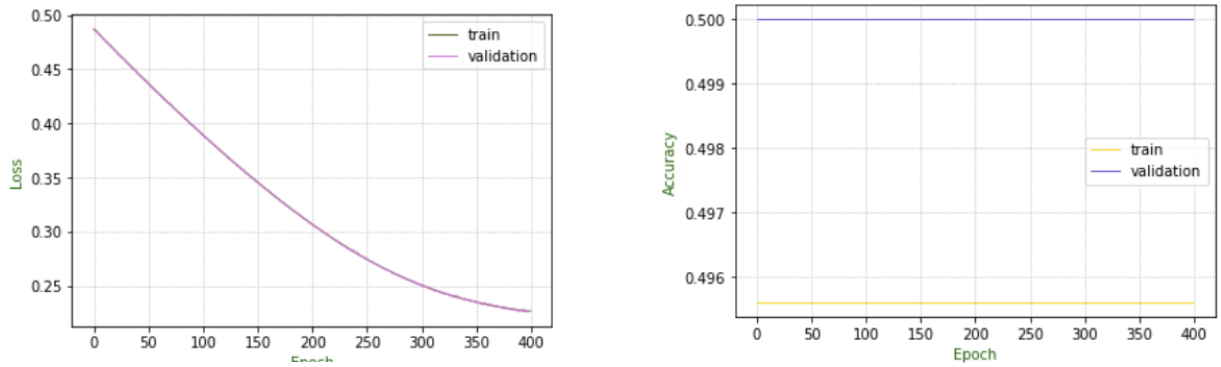

Figura 6.32: *Loss* (izquierda) y *accuracy* (derecha) medio por época para los datos de entrenamiento y test ($lr = 0.00001$).

La comparación simultánea en una misma gráfica de las 4 configuraciones con distintos *learning rates* revela que dicho parámetro tiene una importancia evidente. Elegir un valor adecuado del mismo puede inducir a que los resultados sean excelentes o por el contrario que sean desastrosos.

En la Figura 6.33 se determina que los dos experimentos que guían a alcanzar menores *loss* y mayores tasas de acierto sobre los conjuntos de entrenamiento y test son el base y el que tiene la tasa de aprendizaje más baja de todas. La mejor configuración atendiendo a estos dos criterios es la del experimento base, debido a que no presenta sobreajuste, mientras que aquella con *learning rate* 0.01 sí. Los experimentos con *learning rate* 0.0001 y 0.00001 no convergen a una solución óptima debido al problema de evanescencia del gradiente, como se aprecia por la evolución plana de la tasa de acierto.

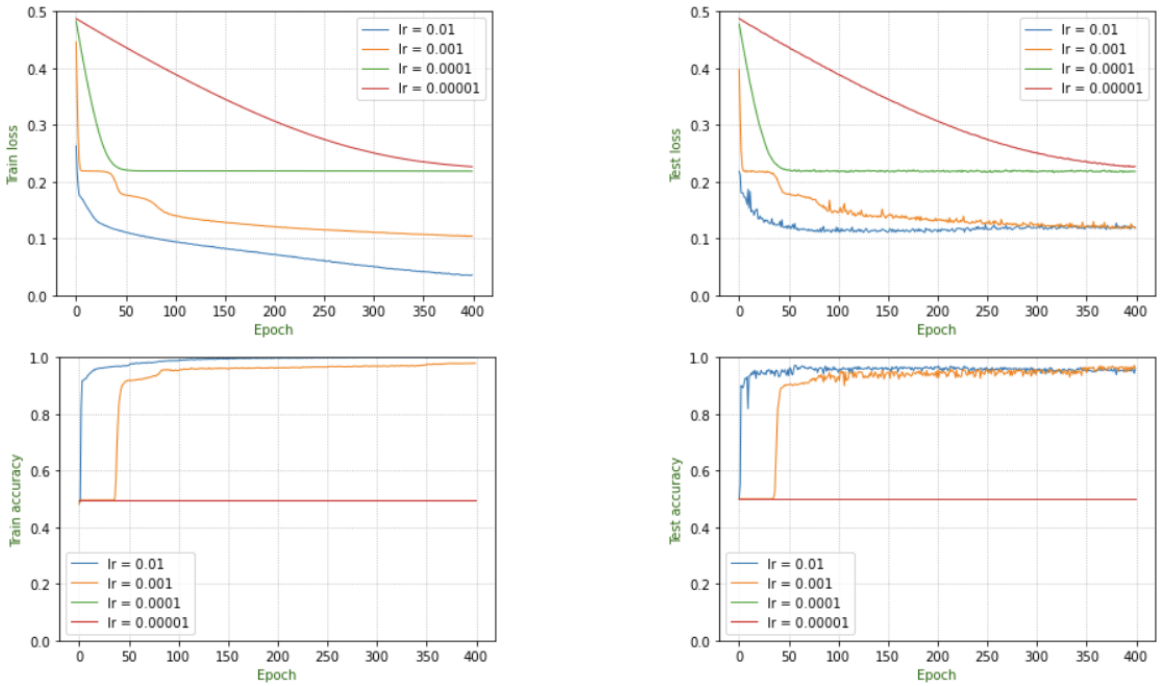

Figura 6.33: *Loss* (arriba) y *accuracy* (debajo) medio por época para los datos de entrenamiento (izquierda) y test (derecha) (base vs $lr = 0.001$, 0.0001 y 0.00001).

6.3.3.2.3. *Label smoothing*

Aplicando el suavizado de etiquetas se observa un comportamiento correcto (Figura 6.34). No hay un problema de *overfitting*, y la evolución de las métricas sobre conjuntos de entrenamiento y test es equivalente, siempre obteniendo resultados algo mejores sobre el utilizado para ajustar los parámetros el modelo como es de esperar.

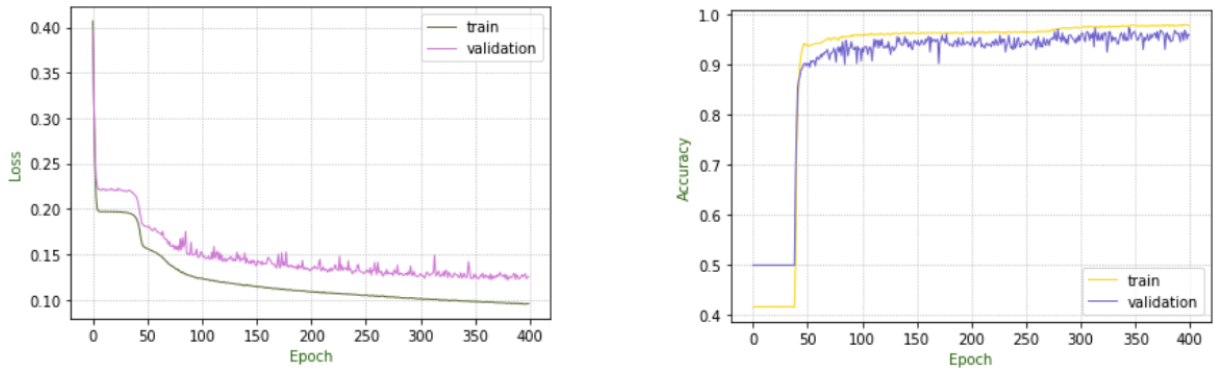

Figura 6.34: *Loss* (izquierda) y *accuracy* (derecha) medio por época para los datos de entrenamiento y test (*label smoothing*).

Comparando los resultados obtenidos mediante la aplicación de esta técnica respecto a los logrados con la configuración base en la Figura 6.35 se aprecia como el desempeño es prácticamente idéntico. Sobre el conjunto de test el comportamiento de la métricas al realizar el suavizado de etiquetas es más volátil, teniendo unos cambios de pendiente mucho más remarcados respecto al experimento base.

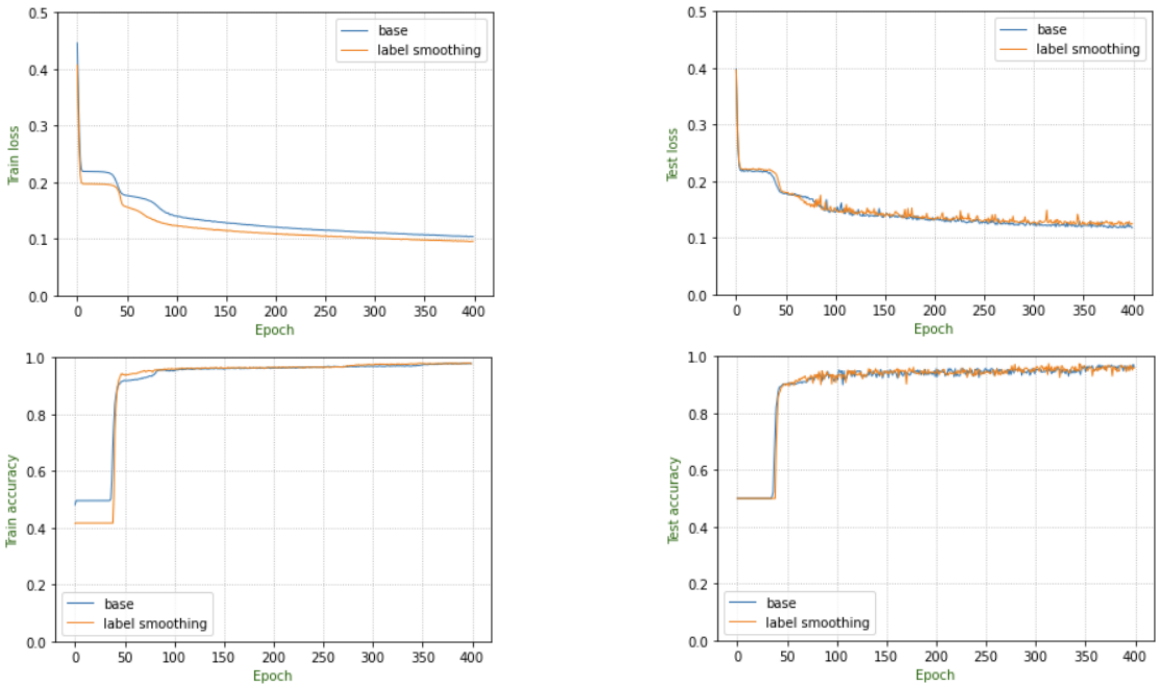

Figura 6.35: *Loss* (arriba) y *accuracy* (debajo) medio por época para los datos de entrenamiento (izquierda) y test (derecha) (base vs *label smoothing*).

6.3.3.2.4. Canales RGBα

Utilizando los canales RGBα en detrimento de los canales HIJV se observa que el *loss* y el *accuracy* sobre el conjunto de test muestra más dispersión en su evolución que sobre el conjunto que se ha utilizado para ajustar la red. Las oscilaciones producidas por esa mayor varianza se suavizan a partir de la época 200. La precisión es muy buena sobre ambos conjuntos, sin apreciarse sobreajuste (Figura 6.36).

La comparación anterior entre el experimento con suavizado de etiquetas y el experimento base y esta compración entre la configuración con un canal suavizado y el experimento base son prácticamente similares, ya que cualquiera de las 3 configuraciones evoluciona de manera muy similar en las 4 métricas evaluadas. Si que se observa que tanto sobre el conjunto de entrenamiento como sobre el conjunto de test el experimento base converge ligeramente más despacio a los valores

que alcanza finalmente que la configuración abordada en esta comparación (Figura 6.37).

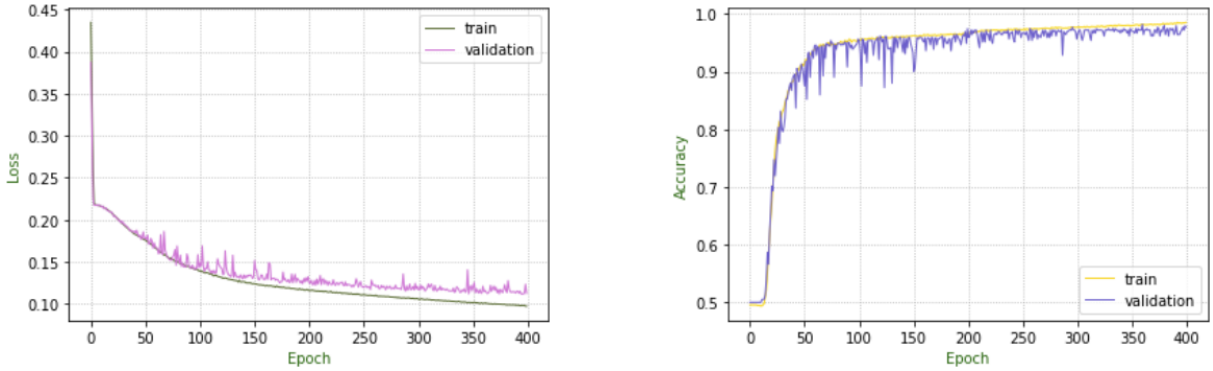

Figura 6.36: *Loss* (izquierda) y *accuracy* (derecha) medio por época para los datos de entrenamiento y test (canales RGBα).

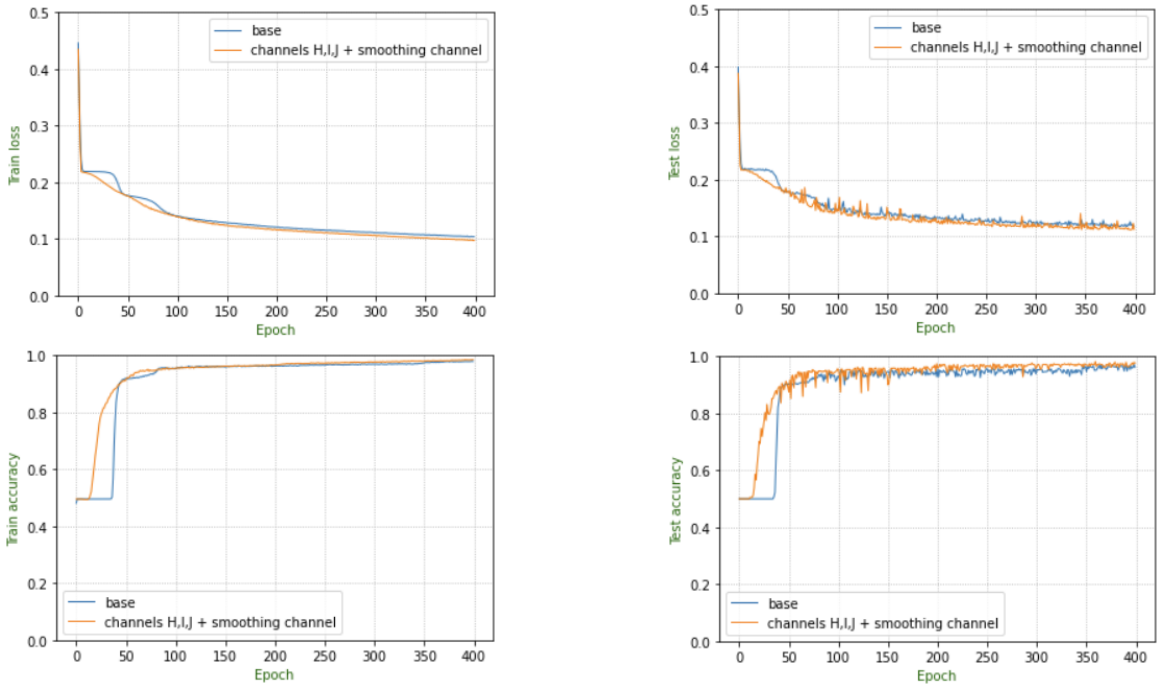

Figura 6.37: *Loss* (arriba) y *accuracy* (debajo) medio por época para los datos de entrenamiento (izquierda) y test (derecha) (base vs canales RGBα).

Comparando sobre el conjunto de test el *loss* en la última época, la tasa de acierto en la última época y el R^2 proporcionado por el Análisis de Correlaciones Canónicas, todas ellas medidas sobre el conjunto de test (Cuadro 6.20), se observa que la configuración que induce a mejores resultados

en dos de las 3 métricas es aquella que utiliza los canales RGBα, mientras que el experimento base es el que mejor desempeño tiene atendiendo al criterio del R^2 del Análisis de Correlaciones Canónicas. Opuestamente, los peores resultados se obtienen cuando se fija el *leaning rate* en el valor más pequeño, 0.00001, debido al problema de la evanescencia del gradiente. El rendimiento de la configuración con una tasa de aprendizaje de 0.0001 también deja mucho que desear debido a este mismo problema. Por su parte, el experimento con *learning rate* 0.01 y el experimento con suavizado de etiquetas obtienen resultados muy parecidos en las 3 métricas, siendo idéntica la tasa de acierto.

Experimento	***Loss***	***Accuracy***	**CCA** R^2
Base	0.1177	0.9617	0.7386
$lr = 0.01$	0.1196	0.9592	0.6757
$lr = 0.0001$	0.2182	0.5	-1.9209
$lr = 0.00001$	0.2273	0.5	-4.5349
Label smoothing	0.1253	0.9592	0.7103
Canales RGBα	0.1126	0.9770	0.7290

Cuadro 6.20: Métricas de los experimentos sobre el conjunto de test.

6.3.4. Resnet18

Con el objetivo de medir el desempeño de una red compleja para una posterior comparación con las arquitecturas ya mostradas se va a utilizar una adpatación al problema de la red residual de 18 capas.

6.3.4.1 Arquitectura

La tercera y última arquitectura que se va a usar en este capítulo es la red convolucional *ResNet-18*. Esta arquitectura pertenece al tipo de redes agrupadas bajo la familia de redes residuales, existiendo diversas configuraciones de las mismas en función del número de capas que presente la red. En este caso, el número de capas es de 18.

Las redes residuales fueron propuestas por primera vez en *Deep Residual Learning for Image Recognition* [28], siendo las 5 configuraciones más famosas de este tipo de redes las configuraciones con 18, 34, 50, 101 y 152 capas. El motivo del surgimiento de estas redes es que las redes tradicionales muy profundas eran muy difíciles de entrenar. En las redes residuales, se reformulan las capas para que sean capaces de aprender de residuos generados a partir de las entradas a cada capa, mediante las llamadas funciones residuales. En la publicación citada se muestra que las redes residuales, aún siendo más profundas que otras redes clásicas, son más fáciles de optimizar y por tanto conseguir resultados excelentes.

Es sabido que las redes neuronales son capaces de aproximar de manera muy certera funciones matemáticas. Por tanto, identificar y reproducir el comportamiento de la función identidad es una tarea sencilla para una red neuronal.

$$\varphi(x) = x$$

Siguiendo el mismo razonamiento, si se añade la entrada de la primera capa de la red a la salida de la última, la red tendría que ser capaz de aprender la función que estuviese aprendiendo antes de la adición a la salida de la entrada de la primera capa.

$$\varphi(x) + x = \Psi(x)$$

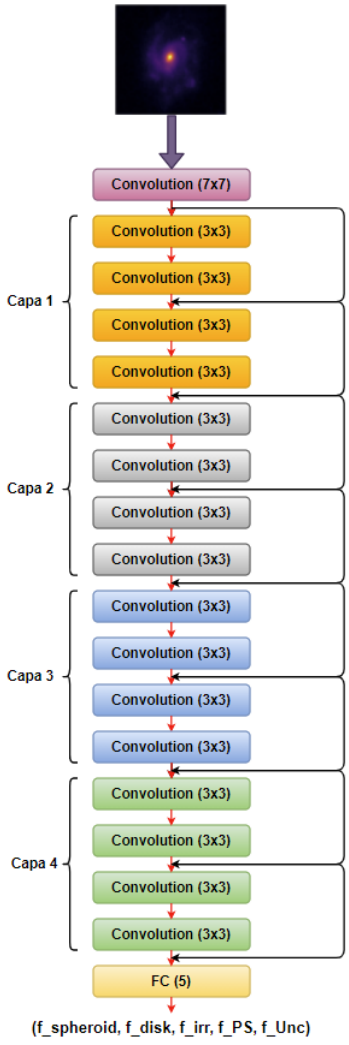

Figura 6.38: Arquitectura de la *ResNet* 18 adapatada.

Este razonamiento es el que se utiliza en las redes residuales con el fin de trasladar valores entre capas, dicho mecanismo se denomina *skip connections*. Con esta solución las redes residuales consiguen solucionar en gran medida el problema del desvanecimiento del gradiente, ya que los gradientes pueden fluir de manera directa desde las últimas capas de la red a las primeras en la etapa de retropropagación.

ResNet-18 está preparada para recibir imágenes en 3 canales, por lo que es necesario adaptar la red para que reciba imágenes en 4 canales. Además, la última capa de la red (una capa lineal) tiene 1000 neuronas, siendo necesaria una modificación para que tenga únicamente 5. La arquitectura adaptada puede verse en la Figura 6.38.

También se aprecian las *skip connections* cada dos capas de convolución. En total, la red consta de 8 conexiones directas que facilitan la transmisión de información entre capas sin que esta se vea alterada.

6.3.4.2 Resultados

Los resultados obtenidos por la red residual de 18 capas en las diferentes configuraciones diseñadas son detallados a continuación.

6.3.4.2.1. Experimento base

Como se ha realizado con los dos arquitecturas de redes neuronales convolucionales previas, ha de definirse un experimento base para ver cómo afecta la variación de distintos parámetros y la adopción de nuevas configuraciones en los resultados finales. La relación de hiperparámetros que definen este experimento se encuentra en el Cuadro 6.21.

# imágenes de entrenamiento	12432
# imágenes de test	932
Función de pérdida	RMSE
Optimizador	SGD
Tamaño de *batch*	28
Tasa de aprendizaje	0.001
Épocas	80

Cuadro 6.21: Configuración del experimento base.

La diferencia existente entre el *loss* y la tasa de acierto sobre los conjuntos de entrenamiento y test es pequeña, por lo que la red Resnet18, con la configuración del experimento base no sufre del problema del sobreajuste (Figura 6.39). El resultado, como ya sucedía con los experimentos base

de las otras dos configuraciones previas probadas en el problema de regresión, es razonablemente bueno, logrando tasas de acierto sobre el conjunto de test superiores al 95 %. En la evolución de ambas métricas sobre el conjunto de test si que se aprecia más varianza que sobre el conjunto de entrenamiento, teniendo cambios de pendiente más marcados.

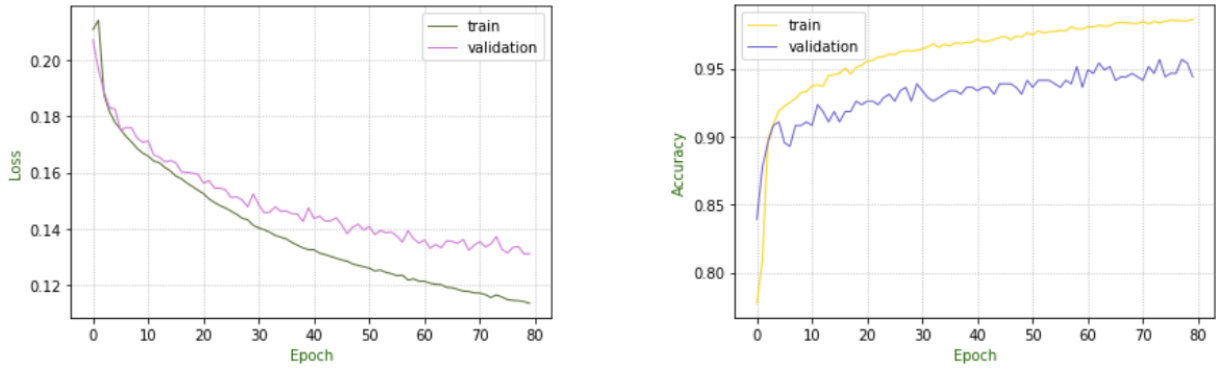

Figura 6.39: *Loss* (izquierda) y *accuracy* (derecha) medio por época para los datos de entrenamiento y test (base).

6.3.4.2.2. Variaciones en el *learning rate*

Cuando la tasa de aprendizaje toma un valor de 0.01 se aprecia un comportamiento mucho más brusco en lo que a la evolución de las métricas se refiere sobre el conjunto de test que sobre el conjunto de entrenamiento (Figura 6.40). Aún habiendo presencia de ese comportamiento aleatorio, los resultados sobre el conjunto de test son muy buenos. Con esta configuración es reseñable decir que, sobre el conjunto de entrenamiento, la red consigue alcanzar una precisión del 100 %, discriminando a la perfección entre las muestras con las que el modelo es alimentado. En esta configuración, ni en ninguna de las posteriores variaciones del *learning rate* se aprecian problemas de infraajuste o sobreajuste.

Si el *learning rate* pasa a valer 10 y 100 veces menos que en el experimento base (Figuras 6.41 y 6.42), es decir 0.0001 y 0.00001, la red neuronal convolucional ofrece una evolución de las gráficas similar. Tanto la pérdida como la tasa de acierto son prácticamente idénticas en cualquier época y para cualquiera de las dos configuraciones. Las tasas de acierto conseguidas sobre ambos conjunto de datos son algo menores que en la configuración anteriormente probada con una tasa de aprendizaje 10 veces mayor debido a que la reducción de este parámetro provoca que la convergencia de los pesos de la red a un valor óptimo sea más lenta.

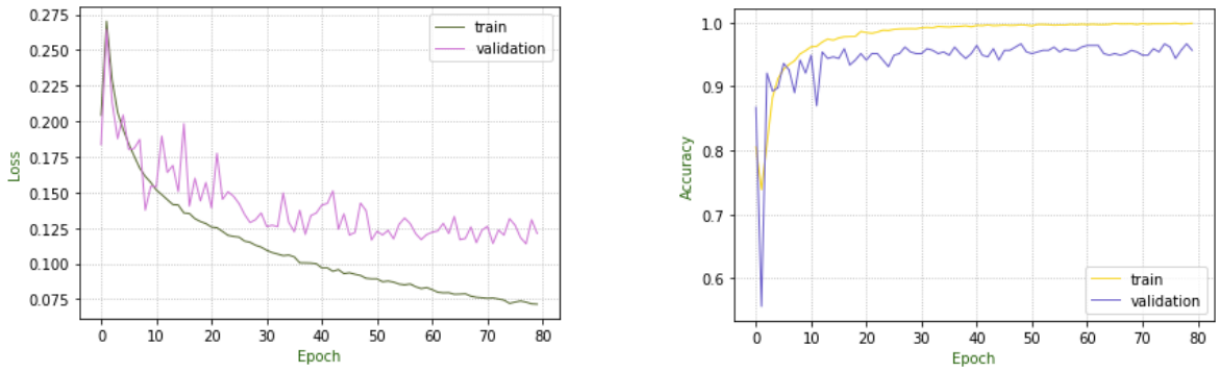

Figura 6.40: *Loss* (izquierda) y *accuracy* (derecha) medio por época para los datos de entrenamiento y test ($lr = 0.01$).

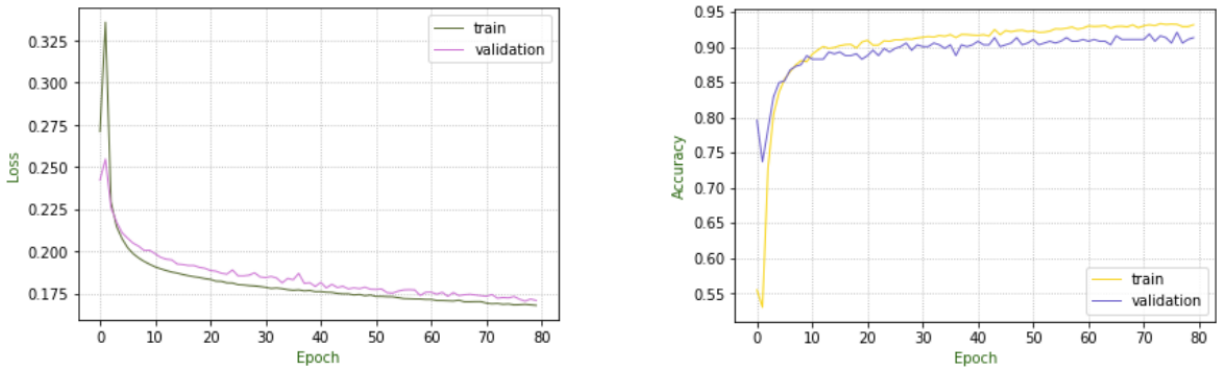

Figura 6.41: *Loss* (izquierda) y *accuracy* (derecha) medio por época para los datos de entrenamiento y test ($lr = 0.0001$).

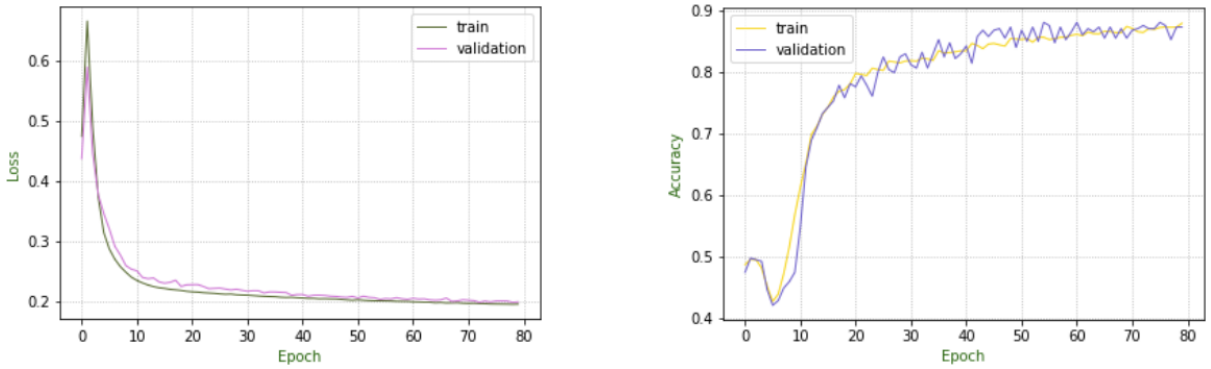

Figura 6.42: *Loss* (izquierda) y *accuracy* (derecha) medio por época para los datos de entrenamiento y test ($lr = 0.00001$).

Comparar 4 configuraciones con distintos *learning rates* de manera paralela en una misma gráfica revela unos resultados muy similares a los ya mostrados para los dos redes neuronales convolucionales anteriores, y es que dicho parámetro tiene un peso elevado en el rendimiento que ofrece la red. En la Figura 6.43 se muestra de una manera clara como las dos configuraciones que mejor funcionan son aquellas en las que la tasa de aprendizaje vale 0.01 y 0.001, mientras que cuando este parámetro se reduce un 10 % y un 100 % respecto al usado en las configuración base la convergencia de las métricas es más lenta. Como ya se ha comnetado anteriormente, ninguna de las configuraciones sufre de infrajuste ni de sobreajuste.

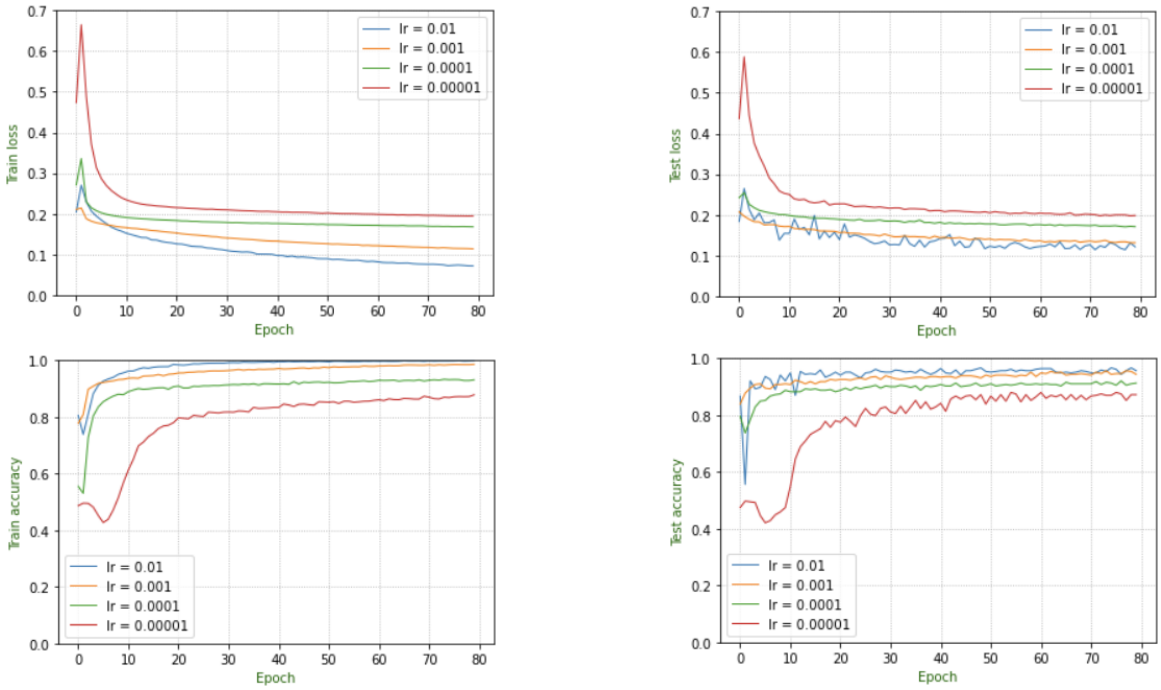

Figura 6.43: *Loss* (arriba) y *accuracy* (debajo) medio por época para los datos de entrenamiento (izquierda) y test (derecha) (base vs $lr = 0.001$, 0.0001 y 0.00001).

6.3.4.2.3. *Label smoothing*

En la Figura 6.44 se observa como aplicando el suavizado de etiquetas se logra un comportamiento bueno, sin presencia de sobreajuste y una evolución pareja de las métricas sobre los conjuntos de entrenamiento y test.

La comparación de los resultados con los obtenidos en el experimento base arroja una similitud evidente (Figura 6.45). La evolución de las métricas es calcada en ambas configuraciones, ambas ofrecen un tasa de acierto sobre el conjunto de test cercana al 100 %, y no hay problemas de sobreajuste o infraajuste. Sobre el conjunto de test el comportamiento de la métricas al realizar el

suavizado de etiquetas es ligeramente más volátil que en el experimento base, aunque la diferencia es tan sutil que es prácticamente inapreciable.

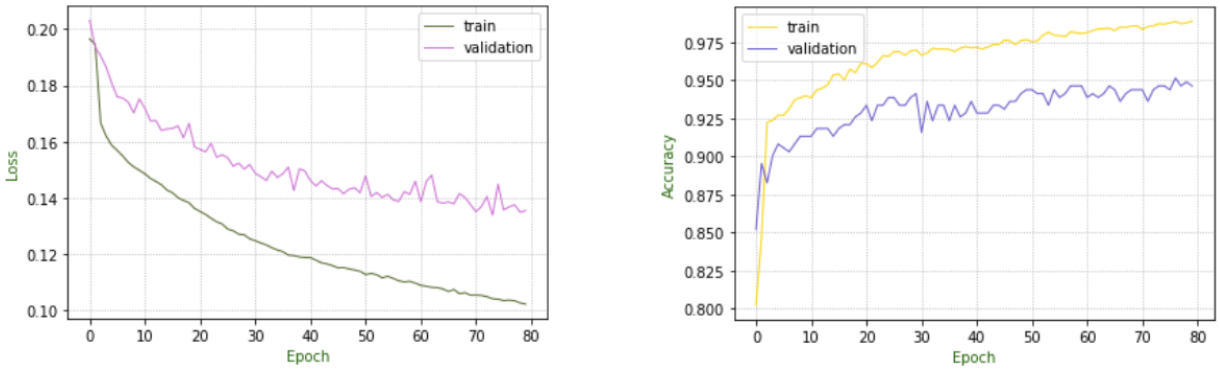

Figura 6.44: *Loss* (izquierda) y *accuracy* (derecha) medio por época para los datos de entrenamiento y test (*label smoothing*).

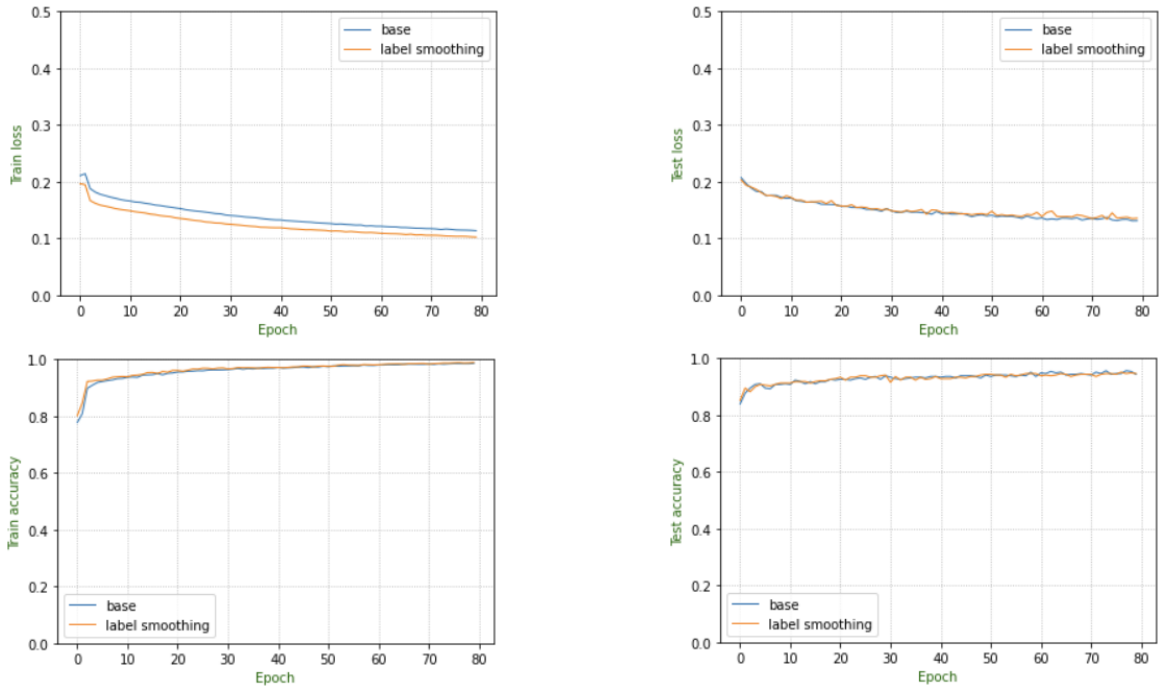

Figura 6.45: *Loss* (arriba) y *accuracy* (debajo) medio por época para los datos de entrenamiento (izquierda) y test (derecha) (base vs *label smoothing*).

6.3.4.2.4. Canales RGBα

Sustituyendo los canales HIJV por los canales RGBα se ve como tanto el *loss* como el *accuracy* muestran una mayor varianza en su evolución sobre el conjunto de test que sobre el conjunto de entrenamiento. Esta configuración no presenta síntomas ni de sobreajuste ni de infraajuste (Figura 6.46).

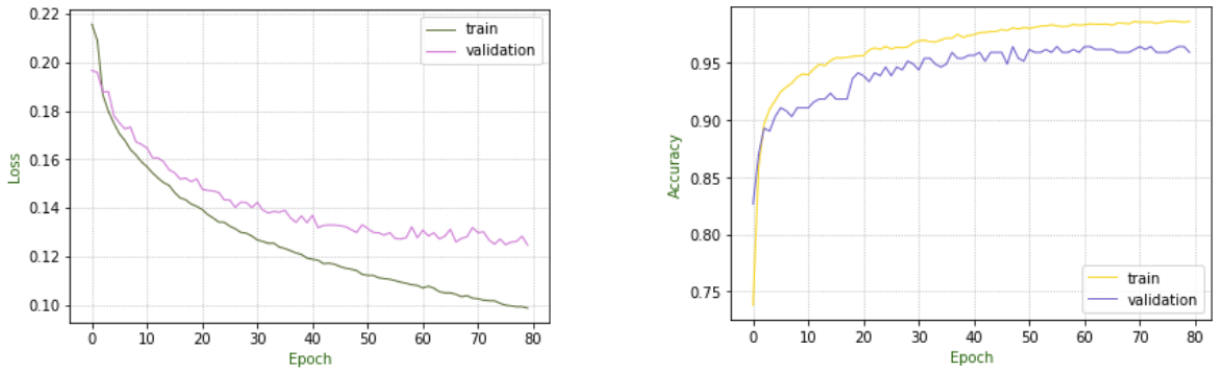

Figura 6.46: *Loss* (izquierda) y *accuracy* (derecha) medio por época para los datos de entrenamiento y test (canales RGBα).

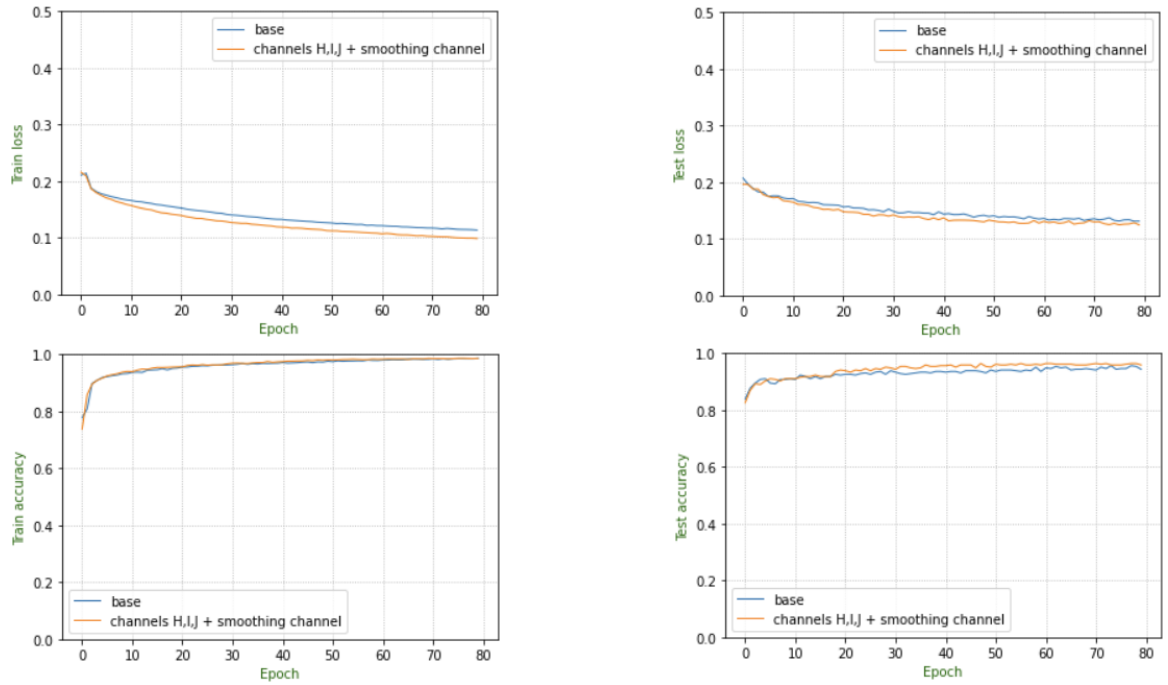

Figura 6.47: *Loss* (arriba) y *accuracy* (debajo) medio por época para los datos de entrenamiento (izquierda) y test (derecha) (base vs canales RGBα).

De nuevo, como ya ocurría en la anterior comparación entre el experimento con *label smoothing*

y el base, se aprecia una igualdad evidente entre el experimento un canal suavizado y la configuración base (Figura 6.47). Ambos casos evolucionan de manera muy similar en las 4 métricas evaluadas. Atendiendo al *loss*, tanto sobre el conjunto de entrenamiento como sobre el conjunto de test el experimento base converge ligeramente más despacio a los valores que alcanza finalmente que la configuración abordada en esta comparación, evento que no ocurre en la evolución de la tasa de acierto.

El análisis conjunto de las métricas *loss* en la última época, tasa de acierto en la última época y R^2 proporcionado por el Análisis de Correlaciones Canónicas, todas ellas medidas sobre el conjunto de test (Cuadro 6.22), se observa que los peores resultados son obtenidos de nuevo por el experimento con *learning rate* 0.00001, al igual que ha pasado con las dos redes neuronales anteriores. La configuración que induce mejores resultados en dos de las 3 métricas es aquella con tasa de aprendizaje 0.01, mientras que la que usa canales RGBα es la que mejor desempeño tiene atendiendo a la tasa de acierto. El experimento base y en el que se realiza el suavizado de etiquetas obtienen también muy buenas resultados, cercanos a los mejores, mientras que la configuración con una tasa de aprendizaje 0.0001 tiene el segundo peor resultado en las 3 métricas.

Experimento	***Loss***	***Accuracy***	**CCA** R^2
Base	0.1312	0.9439	0.6503
$lr = 0.01$	0.1212	0.9566	0.7222
$lr = 0.0001$	0.1708	0.9133	0.4335
$lr = 0.00001$	0.1982	0.8724	0.1944
Label smoothing	0.1355	0.9464	0.6260
Canales RGBα	0.1247	0.9592	0.6847

Cuadro 6.22: Métricas de los experimentos sobre el conjunto de test.

6.3.5. Comparación de resultados

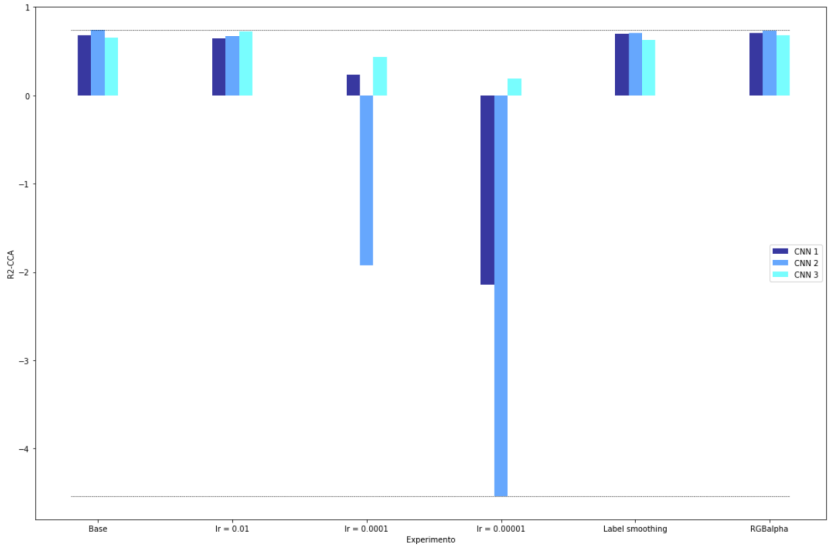

Figura 6.48: R^2 del ACC para las distintas redes y configuraciones sobre el conjunto de test.

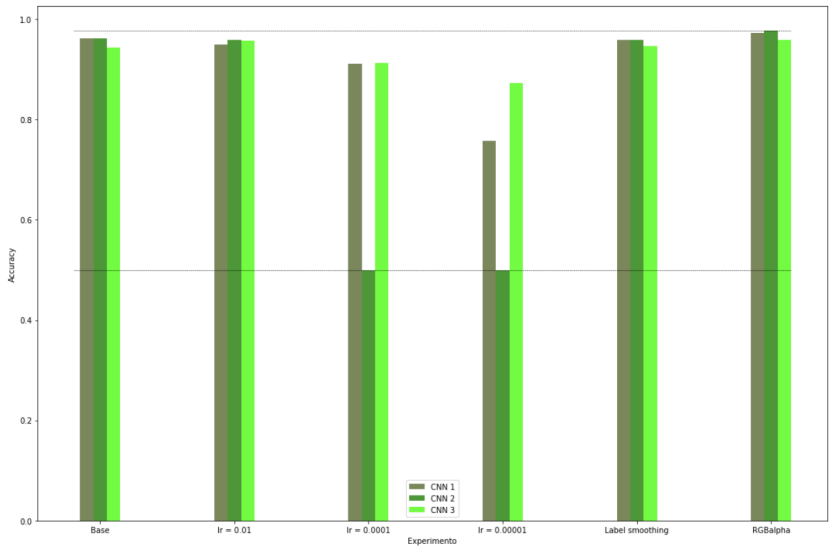

Figura 6.49: *Accuracy* para las distintas redes y configuraciones sobre el conjunto de test.

Atendiendo al criterio del R^2 (Figura 6.48), la red que mejores resultados ofrece es la segunda de las construidas, en su versión del experimento base, seguida de la segunda en su versión entrenada

con imágenes en los canales RGBα. La que peor se comporta también es la segunda, en su versión con tasa de aprendizaje 0.00001.

La tasa de acierto muestra que la red que mejor se comporta también es la segunda, en su versión entrenada con imágenes en los canales RGBα, mientras que la peor se comporta es la segunda, en sus versiones con *learning rates* 0.0001 y 0.00001 (Figura 6.49).

En general, las configuraciones con tasas de aprendizaje muy bajas perjudican el rendimiento de las 3 CNNs. Los experimentos con *lr = 0.01*, *label smoothing* e imágenes de entrada en formato RGBα así como la configuración base obtienen métricas bastante similares entre sí, sin parecer beneficiar o perjudicar de manera clara a alguna red.

6.4. Conclusiones

A modo de resumen del capítulo de clasificación morfológica de galaxias distantes, se han explorado dos aproximaciones distintas, una primera basada en clasificación morfológica directa y una segunda basada en regresión, tratando de mimetizar el comportamiento de humanos expertos en morfología galáctica.

En ambos enfoques de utilizaron las mismas transformaciones de *data augmentation* dinámicas, es decir, aplicadas en cada época, con el objetivo de incluir un carácter aleatorio en las imágenes de entrenamiento necesario para que la red no las memorizase. Concretamente, las transformaciones realizadas fueron rotaciones de 90°, 180°y 270°y espejismos vertical y horizontal, todas ellas con una cantidad de ruido gaussiano aleatorio en cada época.

Para el primer enfoque se probó una red convolucional derivada y adaptada de [26] y los resultados no fueron demasiado buenos. Todas las configuraciones probadas (variando la tasa de aprendizaje, realizando suavizando de etiquetas y cambiando los canales en los que se encontraban las imágenes de entrada a la red) sufrían de un serio sobreajuste, siendo la red incapaz de generalizar sobre las imágenes de test el conocimiento aprendido sobre las imágenes de entrenamiento. Algunas configuraciones con *learning rates* muy bajas incluso provocaban severos problemas de desvanecimiento del gradiente.

De manera completamente opuesta, el enfoque basado en regresión produjo resultados excelentes. Las 3 arquitecturas probadas (uno de diseño propio con pocos parámetros, una adaptada de [26] de tamaño medio, y la red residual de 18 capas Resnet18) superaron el 95 % de tasa de acierto sobre el conjunto de test en múltiples configuraciones (variando la tasa de aprendizaje, con suavizado de etiquetas y cambiando los canales en los que se encontraban las imágenes de entrada a la red). Comparando todas las redes y experimentos, la configuración más óptima fue la segunda red en su versión del experimento base y en su versión con canales de entrada RGBα. En general, las peores configuraciones para las 3 redes fueron aquellas con tasas de aprendizaje más bajas.

7. Detección de bordes

Tener capacidad de delimitar con precisión los bordes y fronteras de los objetos galácticos presentes en el espacio lejano nos permite comprender desde las etapas más tempranas del universo, su formación, hasta las actuales y la futuras, su evolución. En este capítulo se desarrollan técnicas *deep learning* con el fin explicado.

7.1. Segmentación semántica

Uno de los más problemas más comunes y conocidos en el campo del *deep learning* es la clasificación de imágenes, donde se busca que un modelo determine qué objeto o qué escena es la que se encuentra presente. Se podría decir que este problema es básico y de alto nivel, ya que únicamente es necesario determinar un patrón común que formen los píxeles de la imagen para determinar la clasificación.

Un problema mucho más interesante y de más bajo nivel consiste en clasificar cada uno de los píxeles que haya en las imágenes, tarea conocida como segmentación de imágenes o segmentación semántica. El objetivo es que todos los píxeles que se encuentren dentro de la porción de una imagen original en la que se encuentre un objeto tomen un mismo valor en la imagen segmentada, y si en otra zona de la imagen original hay un objeto del mismo tipo, los píxeles de la imagen segmentada que le representen deberán tener el mismo valor que los asociados al otro objeto del mismo tipo.

La diferencia más notable respecto al problema de clasificación tradicional de imágenes es que ahora el objetivo no es etiquetar una imagen, si no asociar cada píxel a cada tipo de objeto presente la imagen, creando una especie de fronteras o bordes en la imagen original, donde todos los píxeles dentro de un borde están etiquetados con el mismo valor porque representan al mismo objeto (Figura 7.1).

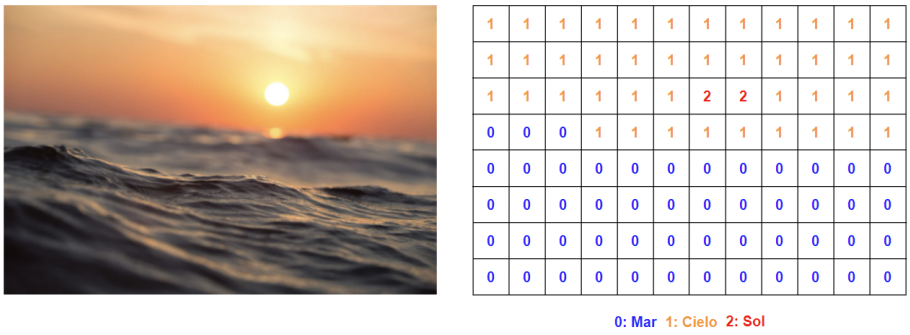

Figura 7.1: Ejemplo de imagen y su correspondiente segmentación semántica.

La segmentación semántica es una técnica puntera y en los últimos años ha sido desarrollada principalmente mediante técnicas de *deep learning*. Entre sus aplicaciones más importantes destacan la conducción autónoma, el diagnóstico de imágenes médicas y la inspección industrial. Siguiendo el desarrollo natural de este trabajo, la segmentación se utilizará en el contexto de la astronomía, más concretamente se tratará de clasificar cada píxel de las imágenes en una de las siguientes 3 categorías: galaxia principal, galaxia secundaria y ruido.

7.2. Conjunto de datos

Para atacar el problema de la segmentación semántica aplicado a imágenes astronómicas se dispone inicialmente de un conjunto de datos con 1053 imágenes disponibles en el canal H, 1045 en el canal I, 1053 en el canal J y 1053 máscaras (Cuadro 7.1). Tanto las imágenes como las máscaras son de 200x200 píxeles. Al igual que en el problema de clasificación morfológica, las imágenes provienen del proyecto CANDELS, por lo que han sido tomadas por el telescopio Hubble. Las zonas de exposición siguen siendo GOODS-N, GOODS-S, UDS, EGS y COSMOS. Tanto las imágenes como las máscaras de segmentación han sido facilitadas por el profesor Fernando Buitrago Alonso.

Tipo	#
Imágenes canal H	1053
Imágenes canal I	1045
Imágenes canal J	1053
Máscaras	1053

Cuadro 7.1: Número de imágenes según canal y máscaras disponibles.

Un detalle importante que no se puede pasar por alto es el no confundir las máscaras que se van a utilizar a lo largo de todo el capítulo con las máscaras de las que se habla en el capítulo anterior. Anteriormente, las máscaras se utilizaban para aislar el objeto de interés en una imagen que contenía muchos objetos. Ahora, las máscaras se corresponden con las etiquetas de las imágenes y los elementos que los algoritmos de *deep learning* tiene que ser capaces de generar. Como ya se ha explicado anteriormente, los valores de los píxeles de una máscara harán referencia al tipo de objeto que se encuentra en la porción de la imagen donde se encuentran los píxeles.

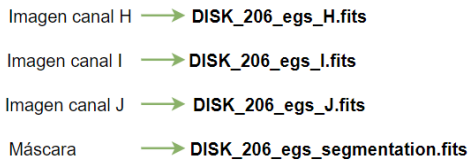

Figura 7.2: Nomenclatura de las imágenes y las máscaras.

El primer paso que hay que hacer para generar el *dataset* base es seleccionar aquellas imágenes que se encuentren disponibles en los canales H , I y J. Tras realizar el emparejamiento, el resultado son 1045 imágenes disponibles en los 3 canales. El modo de relacionar las imágenes que se encuentren disponibles en 3 canales es mediante un prefijo común que comparten, compuesto por la morfología de la galaxia principal de la imagen (información no necesaria en el problema de la segmentación semántica), un número y la zona de exposición como se aprecia en la Figura 7.2.

El segundo paso, y como ya se realizó en el capítulo anterior, consiste en filtrar las imágenes para que ninguna tenga más de un 50 % de píxeles con valor 0.0. El motivo para la realización del filtrado es que dichas imágenes no tendrían suficiente información de calidad como para alimentar los posteriores algoritmos de aprendizaje profundo con ellas. Se encuentra que ninguna de las 1045 imágenes en ninguno de los 3 canales cumple esa condición, por lo que tras este filtrado se siguen disponiendo de 1045 imágenes.

El tercer y último paso consiste en asignar a cada imagen, en sus 3 canales, su máscara correspondiente. Esto se consigue gracias a que las imágenes y las máscaras tienen un prefijo común, por lo que comparando dicho prefijo se puede realizar la equivalencia (Figura 7.2). Tras realizar este procesamiento, se dispone de un conjunto de datos compuesto por 1045 imágenes, en 3 canales cada una, y 1045 máscaras. Un ejemplo de imagen con su correspondiente máscara de segmentación está en la Figura 7.3.

Tanto las imágenes como las máscaras son proporcionadas en formato *.fits*, para acceder a su contenido es necesario utilizar la función *fits.open* del paquete *astropy.io*.

```
imagen = fits.open('DISK_206_egs_H.fits')[0].data
mascara = fits.open('DISK_206_egs_segmentation.fits')[0].data
```

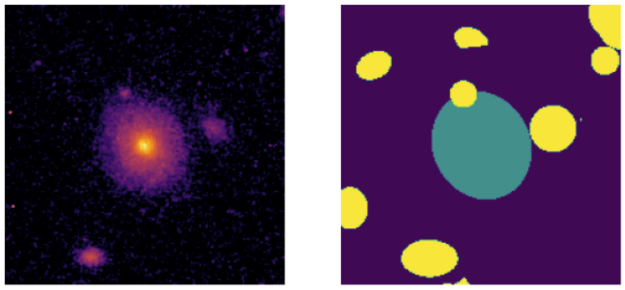

Figura 7.3: Ejemplo de imagen en un canal con su respectiva máscara.

Las imágenes que se van a utilizar como entrada de los algoritmos *deep learning* no son las originales disponibles en los canales H, I y J, si no que se van a modificar mediante un procedi-

miento proporcionado por el profesor Fernando Buitrago. El objetivo del cambio es remarcar las diferencias entre los píxeles que forman parte de la galaxia y los píxeles que no. Lo primero, es transformar los canales H, I y J en los canales R, G y B mediante la función *make_lupton_rgb* del paquete *astropy.visualization*.

Una vez se tiene la imagen en los canales RGB, se añade un cuarto canal suavizado (α) proveniente de la aplicación de un filtro gaussiano a los canales RGB (Figura 7.4). Los efectos de las trasformaciones provocan que se pasen de 1045 imágenes en los canales H, I y J a 1045 imágenes en los canales R, G, B y un filtro gaussiano usado como cuarto canal. El procedimiento es el mismo que ya se mostró en el capítulo de clasificación, al transformar las imágenes del formato HIJV al formato RGBα, por lo que no se volverá a mostrar el código.

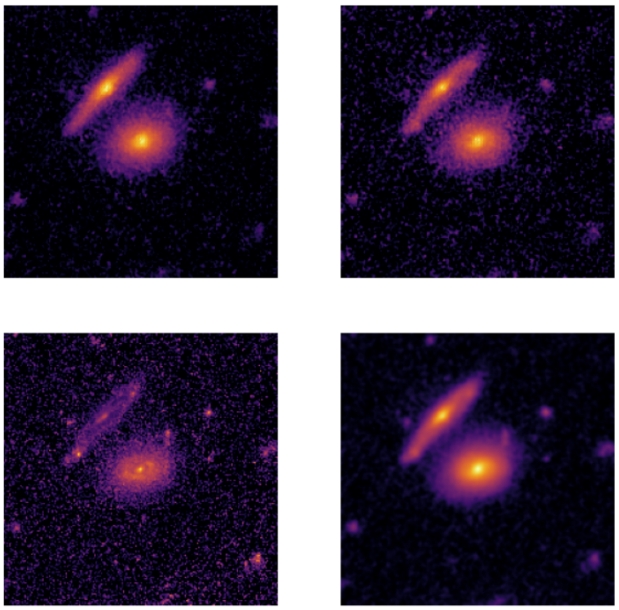

Figura 7.4: Ejemplo de imagen en los 4 canales.

7.3. Segmentación mediante U-*Net*

En esta sección se utilizan las redes neuronales convolucionales en U para la tarea de delimitar los bordes de las galaxias. Tanto la arquitectura diseñada como los resultados obtenidos probando diferentes configuraciones del experimento base son detallados.

7.3.1. Arquitectura

Las redes en U o U-*Nets* fueron desarrolladas por Olaf Ronneberger et al. para Bio Medical Image Segmentation [29]. Constan de una etapa de contracción que captura el contexto de cada

elemento presente en la imagen y de una etapa de expansión simétrica a la de contracción que permite localizar de manera precisa cada objeto en la imagen. Además, las capas de contracción y expansión que se encuentran al mismo nivel tiene conexión directa mediante el mecanismo conocido como *skip conncetion*. Como se explica en el *paper* original, este tipo de redes pueden ser entrenadas con una cantidad relativamente escasa de muestras, y obtener muy buenos resultados. También se explica que el uso de *data augmentation* suele inducir U-*Nets* con mejor desempeño. El código base del que se ha partido está en [30].

7.3.1.1 Etapa de contracción

La primera parte de una U-*Net* consta de una fase contracción, en la cuál mediante una serie de operaciones de convolución y *pooling* se reduce el tamaño de las imágenes de entrada. El objetivo de esta primera parte de la red es reducir la resolución de las entradas. La etapa de contracción se compone de varios bloques de contracción apilados secuencialmente, un bloque de contracción en PyTorch puede definirse de la siguiente manera:

```
def bloque_contraccion(self, canales_entrada, canales_salida, kernel, padding):
contraccion = nn.Sequential(
torch.nn.Conv2d(canales_entrada, canales_salida, kernel_size=kernel, stride=1,
    padding=padding),
torch.nn.BatchNorm2d(canales_salida),
torch.nn.ReLU(),
torch.nn.Conv2d(canales_salida, canales_salida, kernel_size=kernel, stride=1,
    padding=padding),
torch.nn.BatchNorm2d(canales_salida),
torch.nn.ReLU(),
torch.nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
)
return contraccion
```

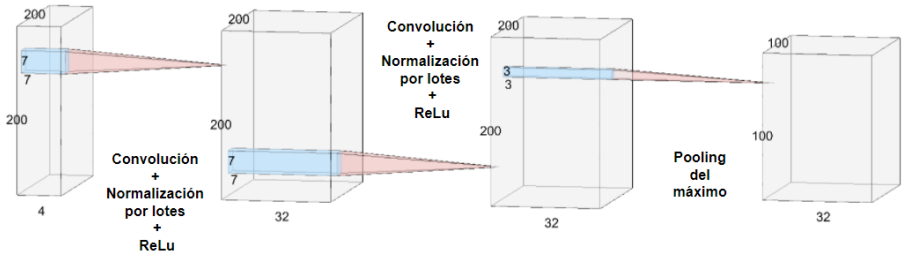

Figura 7.5: Bloque de contracción con kernel 7x7 y padding 3 de las operaciones de convolución. Canales de entrada 4, canales de salida 32.

Está compuesto por dos operaciones de convolución seguidas de un operación de *pooling* del máximo (Figura 7.5). Después de cada operación de convolución se aplica normalización por lotes (*batch normalization*) y la función de activación rectificador lineal (ReLu). La primera convolución aumenta el número de canales de los especificados como canales de entrada a los especificados como canales de salida, y la segunda convolución mantiene el número de canales de salida. El tamaño del *kernel* y el *padding* de las operaciones de convolución son pasados como parámetros de la función, mientras que en la operación de *pooling* del máximo están fijados en 3 y 1. El *stride* está fijado en 1 y 2 en las operaciones de convolución y la operación de *max pooling* respectivamente.

7.3.1.2 Etapa de expansión

La etapa de expansión tiene como objetivo aumentar la resolución de las imágenes, para lograrlo hace uso de las operaciones de convolución transpuestas. Como ya se ha explicado, para lograr que las dimensiones de la salida de la fase de expansión coincidan con las dimensiones de la entrada de la fase de contracción ambas etapas tienen que ser simétricas, por lo que el código de la etapa de expansión hace la operación inversa al definido en la etapa de contracción.

```
def bloque_expansion(self, canales_entrada, canales_salida, kernel, padding):
expansion = nn.Sequential(
torch.nn.Conv2d(canales_entrada, canales_salida, kernel, stride=1, padding=
    padding),
torch.nn.BatchNorm2d(canales_salida),
torch.nn.ReLU(),
torch.nn.Conv2d(canales_salida, canales_salida, kernel, stride=1, padding=
    padding),
torch.nn.BatchNorm2d(out_channels),
torch.nn.ReLU(),
torch.nn.ConvTranspose2d(canales_salida, canales_salida, kernel_size=3, stride
    =2, padding=1, output_padding=1)
)
return expansion
```

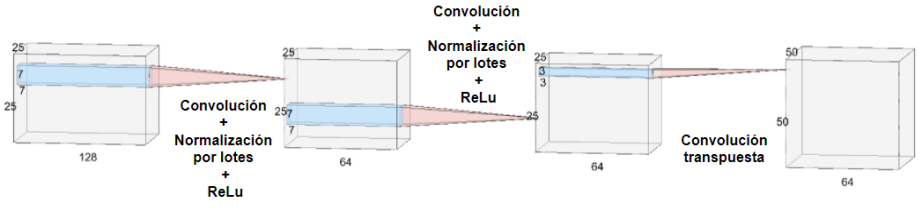

Figura 7.6: Bloque de expansión con kernel 7x7 y padding 3 de las operaciones de convolución. Canales de entrada 128, canales de salida 64.

La estructura es idéntica a la del bloque de expansión. Dos etapas de convolución con norma-

lización por lotes y la función de activación rectificador lineal se suceden, y posteriormente una operación de convolución inversa aumenta la resolución del mapa de características, haciendo el procedimiento inverso al realizado en la fase de reducción mediante la operación de *pooling* del máximo (Figura 7.6).

7.3.1.3 *Skip connections*

Una parte fundamental de las U-*Nets* son las *skip conncetions*, que comunican de manera directa los bloques de la fase de contracción con los bloques de la fase expansión. Concretamente, cada salida de un bloque de contracción es añadida a la entrada de su correspondiente bloque de expansión, salvo para el primer bloque de expansión. Esto es, la entrada de un bloque de expansión es la salida del bloque de expansión anterior junto con la salida del correspondiente bloque de contracción.

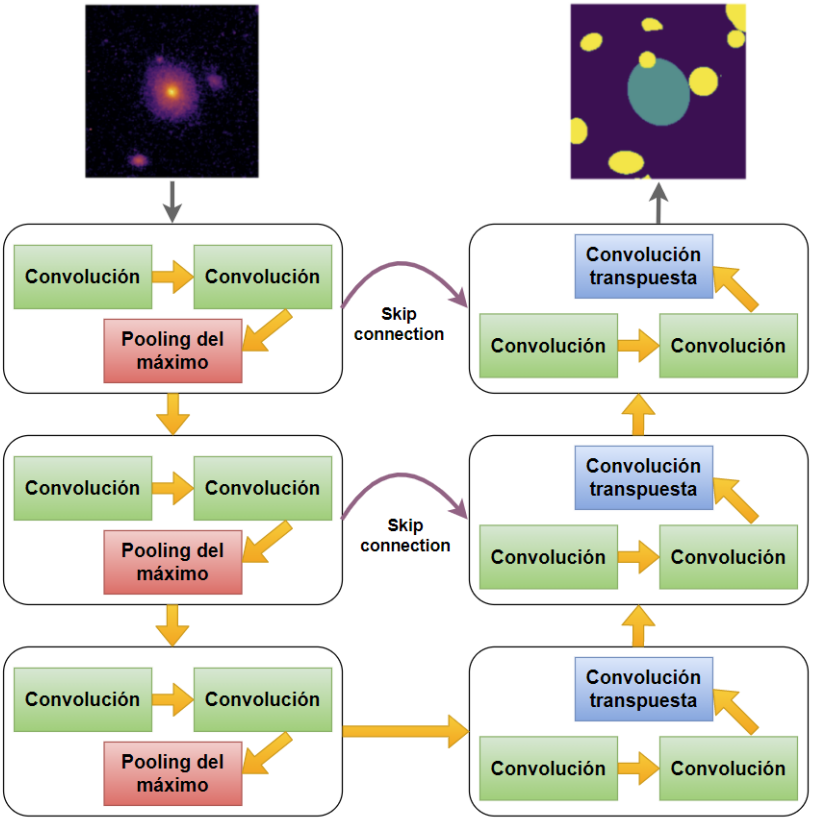

Figura 7.7: U-*Net* construida. Consta de 3 bloques de contracción y 3 bloques de expansión.

El significado de esta operación es que un bloque de la etapa de expansión realizará las etapas de convolución sobre la concatenación de los canales de la salida del bloque de expansión anterior con los canales de la salida del bloque de contracción equivalente.

La red diseñada está formada por 3 bloques de contracción y 3 bloques de expansión, por lo que habrá dos *skip conncetions*, desde el segundo bloque de contracción al segundo bloque de expansión y desde el primer bloque de contracción al primer bloque de expansión.

El primer bloque de contracción reduce las imágenes de 200x200 píxeles a 100x100 píxeles, pasando de 4 a 32 canales. El segundo bloque de contracción reduce las imágenes de 100x100 píxeles a 50x50 píxeles, pasando de 32 a 64 canales. Por su parte, el tercer bloque de contracción reduce las imágenes de 50x50 píxeles a 25x25 píxeles, pasando de 64 a 128 canales.

Complementariamente, el primer bloque de expansión aumenta las imágenes de 25x25 píxeles a 50x50 píxeles, pasando de 128 a 64 canales. El segundo bloque de expansión aumenta las imágenes de 50x50 píxeles a 100x100 píxeles, pasando de 64 a 32 canales. El tercer y último bloque de expansión aumenta las imágenes de 100x100 píxeles a 200x200 píxeles, el tamaño original de las imágenes, pasando de 32 a 3 canales. La arquitectura íntegra de la U-*Net* diseñada está en la Figura 7.7.

Una vez se disponen de los 3 canales que ofrece la red como salida, para construir la máscara se selecciona para cada píxel el canal que maximiza su valor. De esta manera, si un píxel tiene el valor 0.32 en el primer canal, 0.12 en el segundo y 0.78 en el tercero, el valor de la máscara en ese píxel será un 2 (suponiendo que al primer canal se le asigna el valor 0).

7.3.2. Resultados

Para todos los experimentos que se van a realizar con la U-*Net* descrita se va a medir la evolución de la pérdida y de la tasa de acierto sobre los conjuntos de entrenamiento y test durante las 100 épocas que se entrenarán las redes, es decir, todas las muestras de entrenamiento se introducirán 100 veces en las redes.

Primero, se definirá un experimento base entrenando la U-*Net* creada sobre el conjunto de datos descrito en 7.2 aplicando *data augmentation*. Posteriormente, utilizando el mismo conjunto de datos con el mismo *data augmentation* se lanzará una segunda tanda de experimentos donde se aplicará la técnica de regularización *drop out* eliminando neuronas con distintas probabilidades. Por último, se aplicará la U-*Net* sobre un conjunto de datos con fuertes transformaciones aleatorias en cada época, en concreto *zooms* aleatorios.

Todos los experimentos que se van a realizar comparten la relación de características mostradas en el Cuadro 7.2.

La manera de medir la tasa de acierto es comparando el valor de cada píxel predicho con su correspondiente píxel real. Por ejemplo, si de la imagen de 200x200 píxeles original la máscara predice correctamente el valor de 30000, entonces la tasa de acierto será del 75 %.

Función de pérdida	Entropía cruzada
Optimizador	ADAM
Tamaño de *batch*	25
Tasa de aprendizaje	0.01
Épocas	100

Cuadro 7.2: Configuración del experimentos.

7.3.2.1 Experimento base

Como ya se ha introducido anteriormente, para crear el experimento base se utilizará la U-*Net* creada que consta de 3 bloques de contracción y 3 bloques de expansión. Respecto al conjunto de datos, de las 1045 imágenes disponibles el 66 %, 700, se destinan para entrenar la red mientras que el 33 %, 345, se reservan para conjunto de test. Por tanto, el método de estimación de la tasa de las métricas sobre el conjunto de test es *hold out* sin repetición $\frac{2}{3} - \frac{1}{3}$.

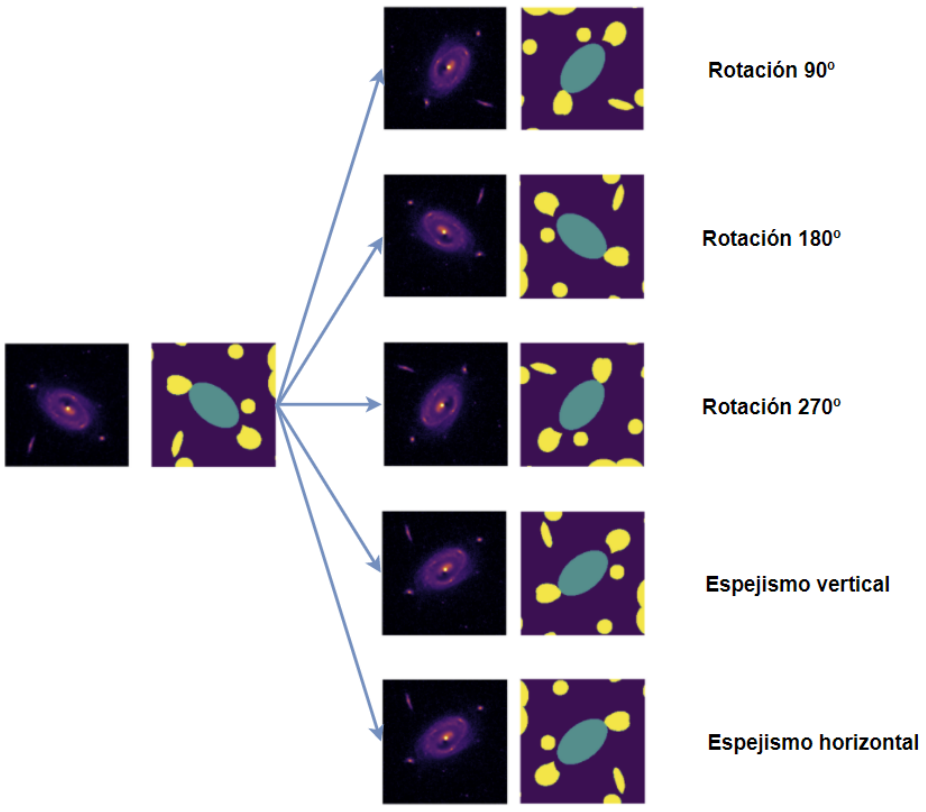

Figura 7.8: Transformaciones aplicadas a las imágenes en el experimento base.

Al tratarse de pocas imágenes para entrenar una red compleja como lo es la U-*Net* diseñada es necesario generar más de manera artificial mediante la técnica conocida como *data augmentation*. Para este experimento, las transformaciones elegidas son la rotación (90°, 180° y 270°) y el espejismo (frente al eje horizontal y al eje vertical) como se aprecia en la Figura 7.8.. Por cada imagen, se obtienen 5 nuevas, por lo que de las 700 imágenes originalmente destinadas para entrenar la red se pasa a 4200.

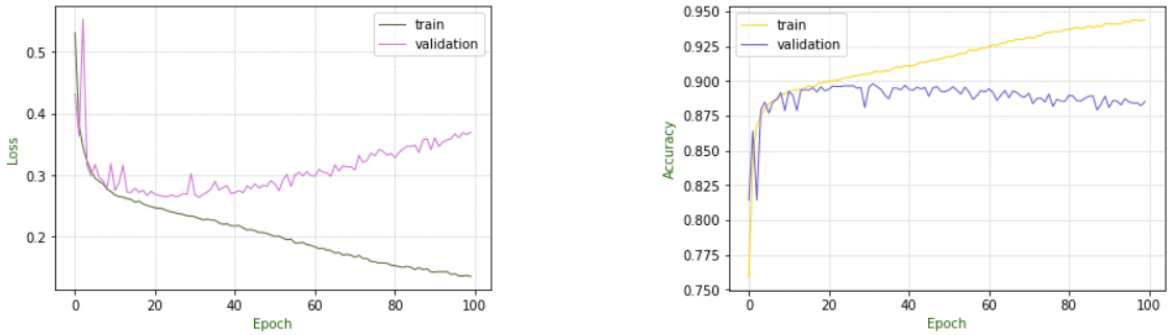

Figura 7.9: Evolución del *loss* y *accuracy* sobre conjuntos de entrenamiento y test en el experimento base.

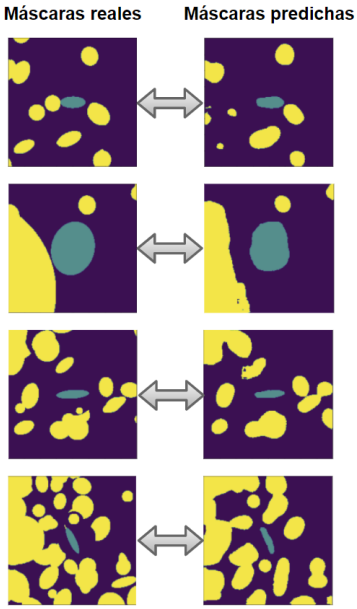

Figura 7.10: Máscaras del conjunto de test reales y predichas por la U-*Net*.

En la Figura 7.9 se aprecia la evolución de la pérdida y de la tasa de acierto sobre los conjuntos de entrenamiento y test. Tanto en el caso del *loss* como en el caso de la *accuracy* el sobreajuste

es evidente. Las métricas sobre el conjunto de entrenamiento no paran de mejorar conforme se suceden las épocas, mientras que sobre el conjunto de test la mejora se estanca entorno a la época 20. A partir de ese punto, lejos de mejorar o incluso mantenerse, las métricas comienzan a empeorar, siendo más evidente este fenómeno en el caso del *loss* que en el caso de la *accuracy*.

La tasa de acierto sobre el conjunto de test es del 88 %, llegando a alcanzar prácticamente el 90 % entre las épocas 20 y 40. Por tanto, se puede concluir que el rendimiento de la red es muy bueno a pesar del sobreajuste, visualmente esto se puede comprobar en la Figura 7.10 comparando máscaras reales con las predichas por la red.

7.3.2.2 Drop Out

Para intentar reducir el sobreajuste presente en el experimento base se va a aplicar la técnica de regularización conocida como *drop out*. Como ya se ha explicado en el Capítulo 2, consiste en ignorar neuronas con una cierta probabilidad p y mantener la neuronas intactas con una probabilidad de *1-p*, evitando que las unidades escogidas participen tanto en el proceso de *backpropagation* como en el proceso de *feed forward*. Es una manera de reducir la complejidad de la red.

En Hinton *et al.* (2012) [31], el *paper* original donde se introdujeron las capas con esta técnica de regularización, el *drop out* en las redes neuronales convolucionales se utilizaba en cada una de las capas *fully connected* o *dense* con $p = 0.5$ antes de la salida, pero no se utilizaba en las capas convolucionales.

Sin embargo, más recientemente, en Sungheon Park and Nojun Kwak (2017) [32] se muestra como aplicar *drop out* en las capas convolucionales también puede resultar útil si se reduce la probabilidad de eliminado de neuronas de manera drástica, hasta $p = 0.1$ o $p = 0.2$. El lugar donde situar el *drop out* dentro de cada capa convolucional es justo después de la función de activación y no entre la operación de convolución y la función de activación.

Siguiendo las indicaciones de la publicación más reciente, y ante la ausencia de capas *FC* en las U-*Net*, se han lanzado 4 experimentos aplicando *drop out* en las capas convolucionales, cada uno de ellos con una probabilidad diferente. Concretamente, las probabilidades son $p = 0.10$, $p = 0.15$, $p = 0.20$ y $p = 0.25$.

En la Figura 7.11 se aprecia la evolución de la pérdida y la tasa de acierto del experimento base y las 4 configuraciones con *drop out* sobre los conjuntos de entrenamiento y test. Sobre el conjunto de entrenamiento el efecto que causa aplicar esta técnica de regularización está claro. A medida que la probabilidad del *drop out* aumenta, el *loss* y el *accuracy* frenan su mejora de manera drástica. A medida que p aumenta, la red aprende menos de los ejemplos de entrenamiento.

Sin embargo, cuando ponemos el foco de atención en el conjunto de test, el realmente im-

portante, se aprecia el poder que tiene el *drop out* para mitigar el problema de sobreajuste que sufre el experimento base. Las configuraciones con $p = 0.10$ y $p = 0.20$ consiguen eliminar por completo el *overfitting*, manteniendo de una manera constante el *loss* y la tasa de acierto a partir de la época 20, momento en el que las métricas del experimento base comienzan a empeorar. La configuración con $p = 0.15$ sufre algo de sobreajuste, aunque es mucho menos evidente que en el experimento base. El experimento con $p = 0.25$ sufre el problema opuesto, padeciendo infrajuste o *underfitting*. La probabilidad con la que se ignoran neuronas es muy alta y la red no es capaz de aprender de las imágenes de entrenamiento lo suficiente.

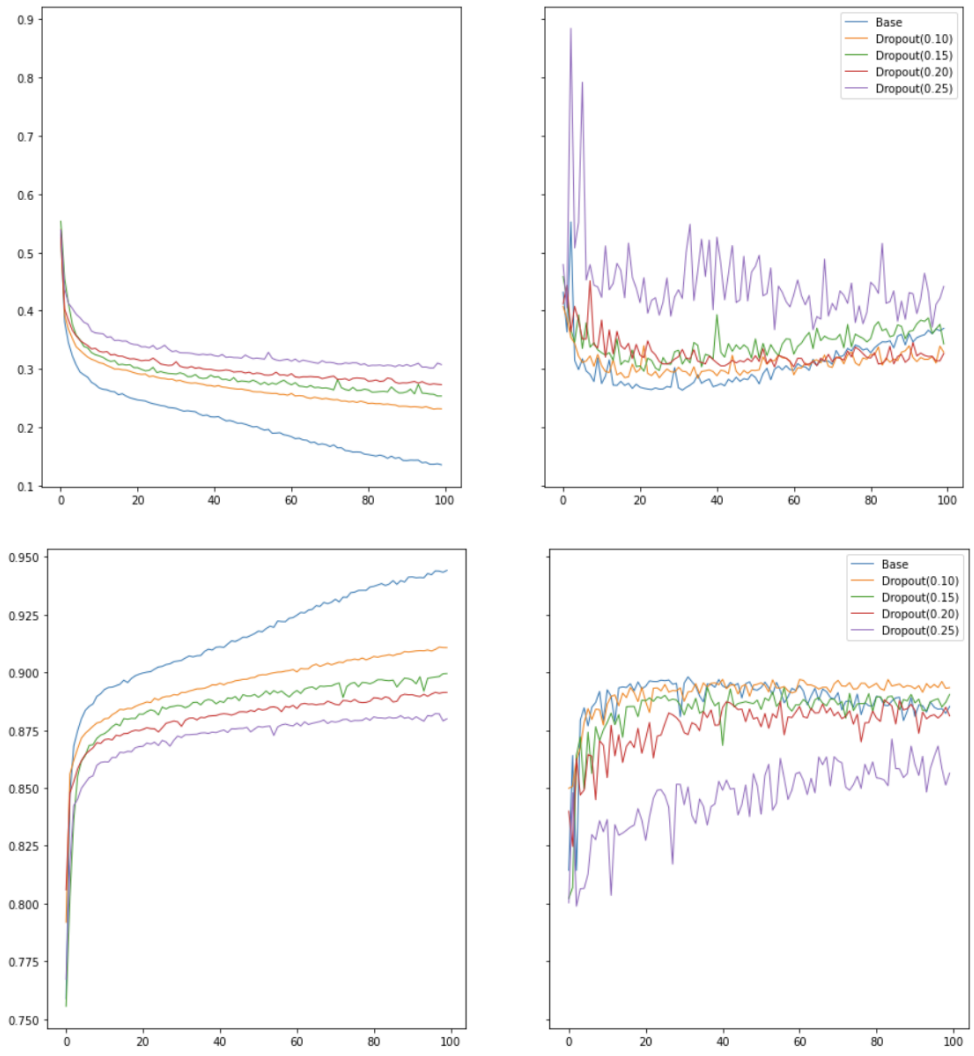

Figura 7.11: Evolución del *loss* (arriba) y *accuracy* (debajo) sobre el conjunto de entrenamiento (izquierda) y test (derecha). Experimento base frente a distintas probabilidades de *drop out*.

Se puede concluir que para reducir el sobreajuste en redes de tipo U el *drop out* es una técnica útil, aunque un p demasiado alto puede truncar la capacidad de aprendizaje de la red. En este caso, de las configuraciones probadas la que mejor rendimiento ofrece es $p = 0.10$.

7.3.2.3 *Zooms*

Otra de las técnicas más efectivas para combatir el sobreajuste en las redes neuronales convolucionales es realizar *zooms* o aumentos sobre las imágenes. Este tipo de transformación no es válida para todos los problemas a los que se puede aplicar las redes convolucionales, ya que dependiendo el factor de aumento del zoom y la zona sobre la que se realice la imagen puede quedar totalmente distinta a la original. Por ejemplo, en un problema de clasificación morfológica un *zoom* excesivo puede resultar perjudicial ya que se puede perder la esencia del objeto a clasificar.

Para la detección de bordes y la segmentación semántica que se está realizando este problema no existe, ya que o bien los límites entre los cuerpos o bien el tipo de cuerpo al que pertenece cada píxel va a continuar siendo el mismo en cualquier región de la imagen.

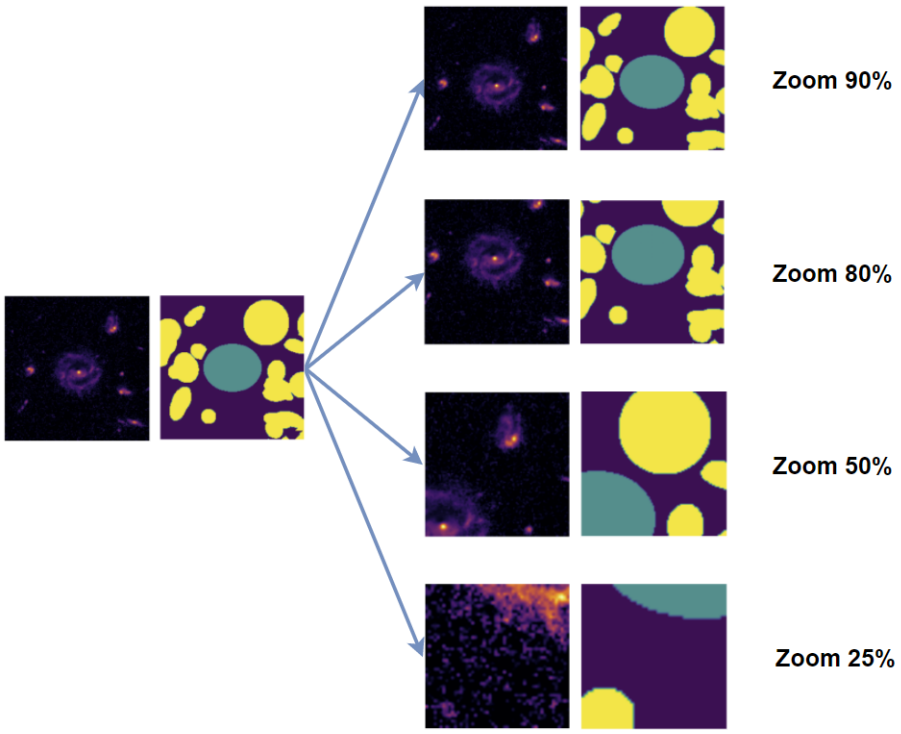

Figura 7.12: Aumentos realizados sobre las imágenes originales.

En este experimento a cada imagen del conjunto de entrenamiento se le realizará cuatro au-

mentos que recojan el 90 %, 80 %, 50 % y 25 % de la misma como se aprecia en la Figura 7.12, seleccionando porciones de 180x180, 160x160, 100x100 y 50x50 píxeles respectivamente. De esta manera, por cada una de las 700 imágenes de entrenamiento se generan 4 nuevas. Las 700 imágenes originales no se utilizan para entrenar la red, por lo que el conjunto de entrenamiento final consta de 2800 imágenes.

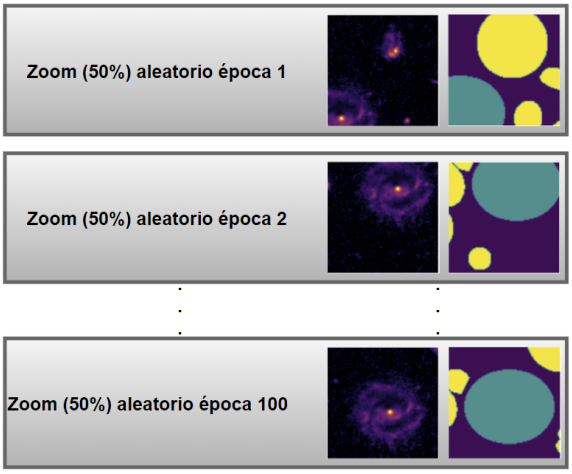

Figura 7.13: Ejemplo de aumentos aleatorios en cada época.

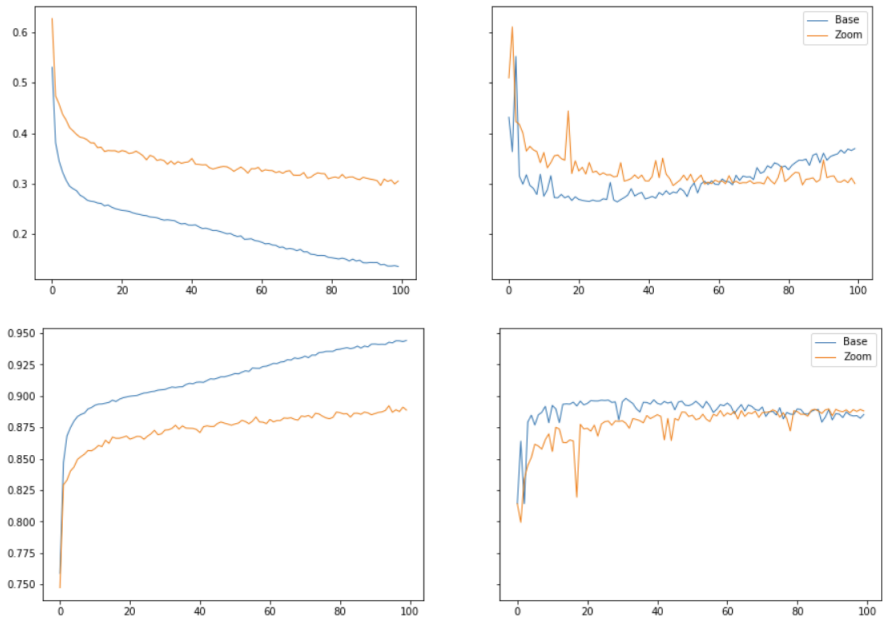

Figura 7.14: Evolución del *loss* (arriba) y el *accuracy* (debajo) sobre el conjunto de entrenamiento (izquierda) y test (derecha). Experimento base frente a experimento con zooms.

Además, para que la prevención del sobreajuste sea más efectiva, los *zooms* serán aleatorios en cada época (Figura 7.13), por lo que es poco probable que una misma imagen de entrenamiento sea exactamente igual en alguna de las 100 épocas. Este fenómeno es más evidente conforme el *zoom* es más drástico. Hay muchas más subregiones de 50x50 píxeles que de 180x180 píxeles dentro de una imagen de 200x200 píxeles.

En la Figura 7.14 se aprecia a la perfección como el sobreajuste consigue reducirse completamente introduciendo aumentos aleatorios en cada época. A diferencia del experimento base, a medida que las épocas se suceden la pérdida y la tasa de acierto sobre el conjunto de test no empeoran, si no que parece que incluso van mejorando levemente, síntoma de que la red va ganando capacidad de generalización poco a poco.

7.3.2.4 Comparación conjunta

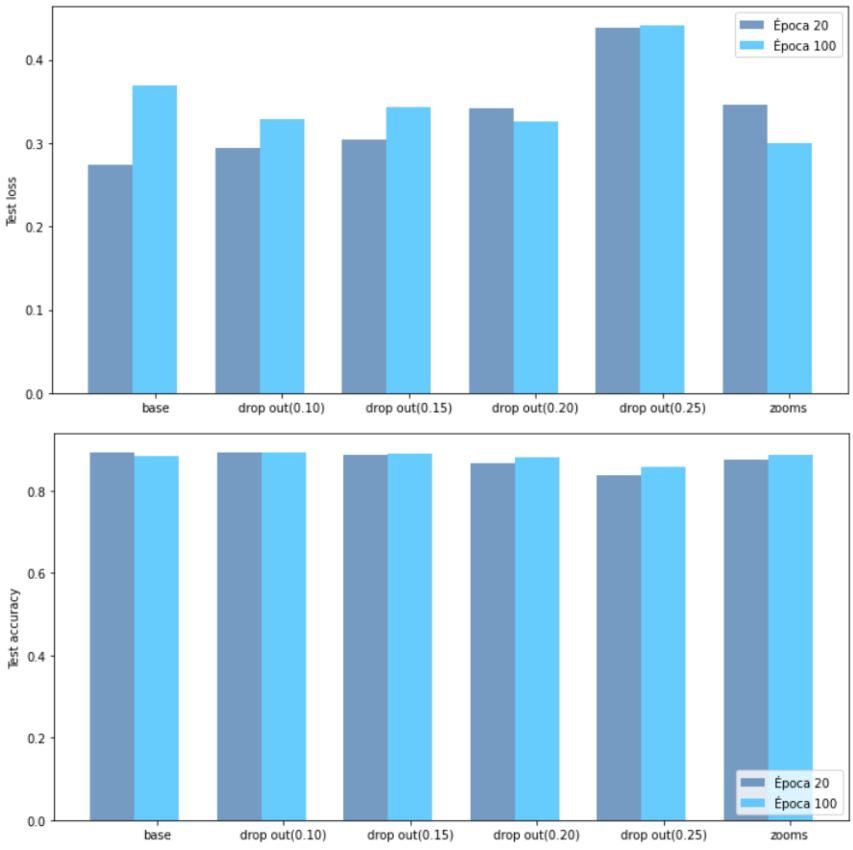

Figura 7.15: Gráficos de barras del *loss* (arriba) y el *accuracy* (debajo) sobre el conjunto de test en las épocas 20 y 100 según experimento.

Para realizar una comparación conjunta del rendimiento que ofrece cada experimento y cuantificar la presencia o ausencia de sobreajuste se dispone de la Figura 7.15 y el Cuadro 7.3.

Tanto en la figura como en las dos últimas columnas del cuadro se aprecia que el experimento que más empeora sus métricas conforme se van sucediendo las épocas y por tanto que más sobreajuste sufre es el base, ya que su *loss* en la época 100 es 1.35 veces su *loss* en la época 20, y su tasa de acierto es un 1 % menor. Las configuraciones con *drop out* con $p = 0.10$ y $p = 0.15$ también tienen peor pérdida en la época 100 que en la época 20, mientras que la tasa de acierto se mantiene invariante.

Por su parte, las configuraciones con *drop out* con $p = 0.20$ y con *zoom* no empeoran sus métricas, si no que las mejoran. Observando tanto el *loss* en la última época como el *accuracy* en la última época (2 primeras columnas del Cuadro 7.3) se determina que el experimento que mayor rendimiento ofrece es en el que se han realizado aumentos aleatorios, consigue la menor pérdida y la tercera mayor tasa de acierto. La configuración que mejor tasa de acierto obtiene es la que tiene *drop out* con $p = 0.10$, llegando a 0.8934.

Experimento	***Loss* final**	***Accuracy* final**	**% diferencia *loss***	**% diferencia *accuracy***
Base	0.3696	0.8853	35 %	-1 %
DO (0.10)	0.3288	0.8934	12 %	0 %
DO (0.15)	0.3429	0.8905	13 %	0 %
DO (0.20)	0.3265	0.8812	-5 %	2 %
DO (0.25)	0.4417	0.8564	1 %	2 %
Zooms	0.3002	0.8880	-15 %	2 %

Cuadro 7.3: Resultados de los experimentos.

7.4. Segmentación mediante *ensembles*

El aprendizaje por *ensembles* fue introducido por primera vez para tareas de clasificación en aprendizaje supervisado por Nilsson en 1965 [33]. El concepto que reside bajo este paradigma de aprendizaje consiste en entrenar diversos clasificadores de manera independiente y, una vez finalizado el entrenamiento, combinar sus predicciones para ofrecer una única salida. Los clasificadores que forman el *ensemble* se denominan clasificadores base, y la idea que se persigue es que el rendimiento que ofrezca el *ensemble* de clasificadores sea mayor que el rendimiento de cualquiera de los clasificadores base que forman el *ensemble*.

A pesar de que el uso de *ensembles* puede conllevar en una mejora de rendimiento respecto a los clasificadores base, no están exentos de problemas, destacando tres hándicaps. Primero, son modelos complejos, tanto de construir como de mantener. Segundo, debido a que es necesario en-

trenar varios clasificadores por separado los requerimientos de cómputo y almacenamiento pueden volverse muy altos, dependiendo la complejidad y tamaño de los clasificadores base que se utilicen. Por último, estos modelos pueden volverse los llamados modelos de caja negra o *blackbox*, debido a la dificultad para explicar el porqúe de sus decisiones.

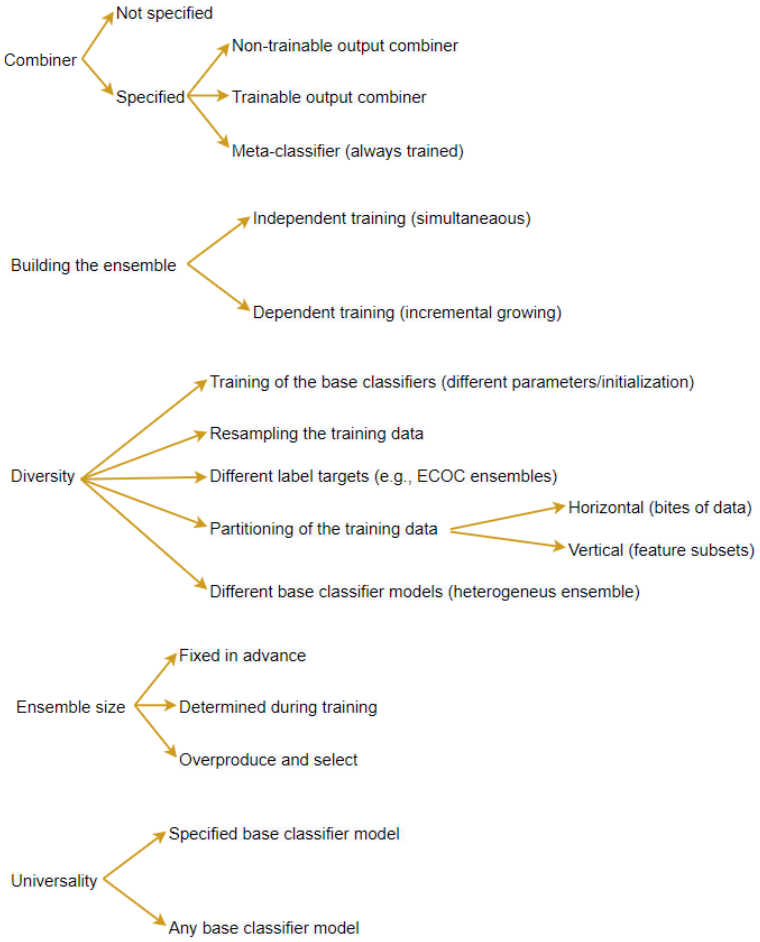

Figura 7.16: Taxonomía de Rokach.

Un concepto clave del aprendizaje por *ensembles* es la diversidad. Si disponemos de un algoritmo que no comete errores de clasificación entonces no necesitamos de un *ensemble* para intentar mejorar el rendimiento. Sin embargo, si el algoritmo si que comete errores entonces puede ser útil complementarle con otros algoritmos que cometan errores en instancias diferentes. Es por ello que la diversidad de salidas de los clasificadores base es necesaria para garantizar el éxito del *ensemble*. Aunque parezca contraproducente, reducir de manera intencionada la precisión de los clasificadores base con el objetivo de ganar diversidad es una técnica que mejora el rendimiento del *ensemble*.

La Figura 7.16 muestra la taxonomía de Rokach para clasificar métodos de *ensemble* learning en base a 5 criterios [34], a continuación se detalla cada uno de ellos:

1. ***Combiner***: el modo en el que se combinan las salidas de los clasificadores base. Algunos *ensembles* no especifican el *combiner*. De entre los que sí los especifican, éste puede ser:
   a) No entrenable: el voto mayoritario por ejemplo.
   b) Entrenable: votación mediante Naïve-Bayes utilizando probabilidades condicionadas para maximizar la verosimilitud de la decisión final.
   c) Meta clasificador: las salidas de los clasificadores son tratadas como entradas de un nuevo clasificador, el cual constituye por si mismo el *combiner*.
2. **Construcción del *ensemble***: los clasificadores pueden ser entrenados de manera independiente o el entrenamiento ha de seguir alguna secuencia.
3. **Diversidad**: Como se introduce la diversidad al *ensemble*.
   a) Usando diferentes hiperparámetros o semillas en cada clasificador base.
   b) Manipulando el conjunto de datos para cada clasificador base.
   c) Cada clasificador base se espcializa en una etiqueta.
   d) Haciendo una partición del conjunto de entrenamiento. Cada clasificador es entrenado con un subconjunto del conjunto de entrenamiento. La partición puede realizarse en el espacio de muestras o en el espacio de características.
   e) Cada clasificador base es de diferente tipo.
4. **Tamaño del *ensenmble***: el número de clasificadores del *ensemble* se fija de antemano o bien es dinámico y se modifica durante la vida del *ensemble*.
5. **Universalidad**: algunas aproximaciones de *ensemble* pueden ser usadas con cualquier clasificador base, mientras que otras están restringidas a un tipo específico.

7.4.1. *Ensemble* de 10 U-*Nets* combinando mediante votación

En esta sección se detalla la creación y el desempeño de un *ensemble* de redes neuronales en U combinándolas mediante un mecanismo de voto democrático: la moda.

7.4.1.1 Arquitectura

El *ensemble* creado está compuesto de 10 clasificadores base. Todos los clasificadores base siguen una misma arquitectura base, la de la U-*Net* creada en 7.3.1. Por tanto, los bloques de contracción y de expansión utilizados en las U-*Net* son los mismos que los de la sección anterior.

Recordar que la red diseñada está formada por 3 bloques de contracción y 3 bloques de expansión, por lo que habrá una *skip conncetion*, desde el primer bloque de contracción al primer bloque de expansión y desde el segundo bloque de contracción al segundo bloque de expansión.

El primer bloque de contracción reduce las imágenes de 200x200 píxeles a 100x100 píxeles, pasando de 4 canales a c_1 canales, el segundo bloque de contracción reduce las imágenes de 100x100 píxeles a 50x50 píxeles, pasando de c_1 canales a c_2 canales y el tercer bloque de contracción reduce las imágenes de 50x50 píxeles a 25x25 píxeles, pasando de c_2 canales a c_3 canales De manera complementaria, el primer bloque de expansión aumenta las imágenes de 25x25 píxeles a 50x50 píxeles, pasando de c_3 canales a c_2 canales, el segundo bloque de expansión aumenta las imágenes de 50x50 píxeles a 100x100 píxeles, pasando de c_2 canales a c_1 canales, y el tercer y último bloque de expansión aumenta las imágenes de 100x100 píxeles a 200x200 píxeles, pasando de c_1 canales a 3 canales (debido a los 3 posibles valores que pueden tomar los píxeles de la máscara).

El número de canales c_1, c_2 y c_3 se establece aleatoriamente para cada una de las 10 U-*Nets* que conforman el *ensemble*, dentro de un conjunto predeterminado de valores. Concretamente:

$$c_1 \in (4, 6, 8, 12)$$
$$c_2 \in (16, 32, 64)$$
$$c_2 \in (164, 128, 256)$$

Esta decisión, junto con otra que se va a explicar más adelante se ha tomado para garantizar la diversidad de los clasificadores base. En cualquier caso, el número de capas no será demasiado elevado bajo ninguna posible combinación, con la intención de que el comportamiento de las U-*Net* sea variable al ser su arquitectura más sencilla.

La segunda medida que se ha tomado para garantizar la diversidad de los clasificadores base es fragmentar el conjunto de entrenamiento, de tal manera que sea altamente probable que la intersección de los conjuntos de entrenamiento de dos U-*Nets* cualesquiera del *ensemble* sea $\emptyset$. El modo de fragmentar el conjunto de entrenamiento es mediante *bootstrap*, de tal manera que si el conjunto de entrenamiento tiene n imágenes, se seleccionen aleatoriamente n con reemplazamiento. De esta manera, cada muestreo *bootstrap* realizado contendrá aproximadamente $\frac{2}{3}$ de las imágenes del conjunto de entrenamiento original (Figura 7.17).

Otro punto a tratar vital a la hora de describir el *ensemble* creado es la manera en la que se combinan los resultados. El método elegido es un *combiner* no entrenable, y se trata de la moda. Para cada píxel de cada imagen las posibles salidas de los clasificadores base son un 0 (ruido), un 1 (galaxia central) o un 2 (galaxia secundaria), y la salida del *ensemble* será el valor con mayor frecuencia absoluta.

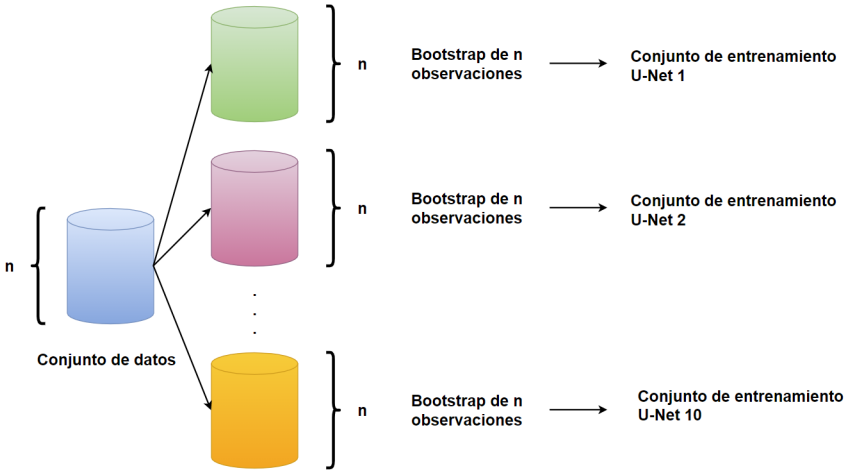

Figura 7.17: *Bootstrap sobre el conjunto de entrenamiento*

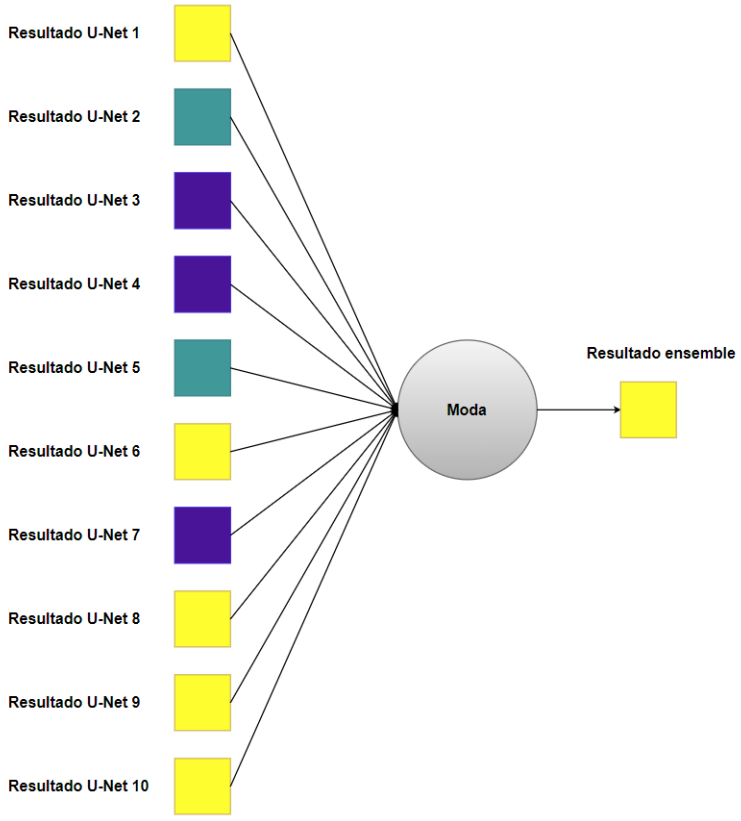

Figura 7.18: Combinación de las salidas de los clasificadores base.

Por ejemplo, si 5 redes predicen que un píxel pertenece a una galaxia secundaria, 3 predicen que pertenece al ruido, y 2 que pertenece a una galaxia principal, entonces la predicción final será que el píxel pertenece a una galaxia secundaria (Figura 7.18). En el caso de que haya un empate entre dos posibles valores, el píxel será considerado que pertenece al ruido.

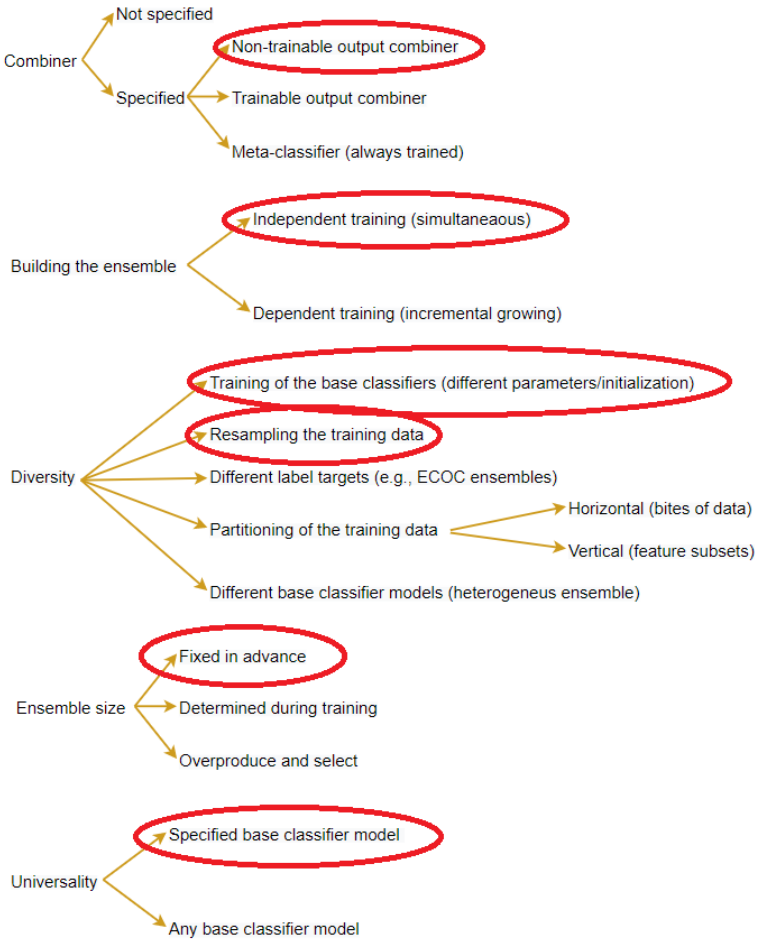

Figura 7.19: Clasificación del *ensemble* en la taxonomía de Rokach.

La función creada que realiza la combinación explicada de las salidas de las U-*Nets* para ofrecer el resultado del *ensemble*, en PyTorch, es la siguiente:

```
def accuracy_ensemble(p1, p2, p3, p4, p5, p6, p7, p8, p9, p10, y):
  metrica = 0
  for i in range(len(y)):
  mask1 = p1[i,:,:,:].argmax(dim=0)
  mask2 = p2[i,:,:,:].argmax(dim=0)
  mask3 = p3[i,:,:,:].argmax(dim=0)
```

```
mask4 = p4[i,:,:,:].argmax(dim=0)
mask5 = p5[i,:,:,:].argmax(dim=0)
mask6 = p6[i,:,:,:].argmax(dim=0)
mask7 = p7[i,:,:,:].argmax(dim=0)
mask8 = p8[i,:,:,:].argmax(dim=0)
mask9 = p9[i,:,:,:].argmax(dim=0)
mask10 = p10[i,:,:,:].argmax(dim=0)
masknew = torch.stack((mask1, mask2, mask3, mask4, mask5, mask6, mask7, mask8
  , mask9, mask10))
moda = torch.mode(masknew,0).values
metrica += (moda == y[i,:,:]).float().mean().item()
return(metrica/len(y))
```

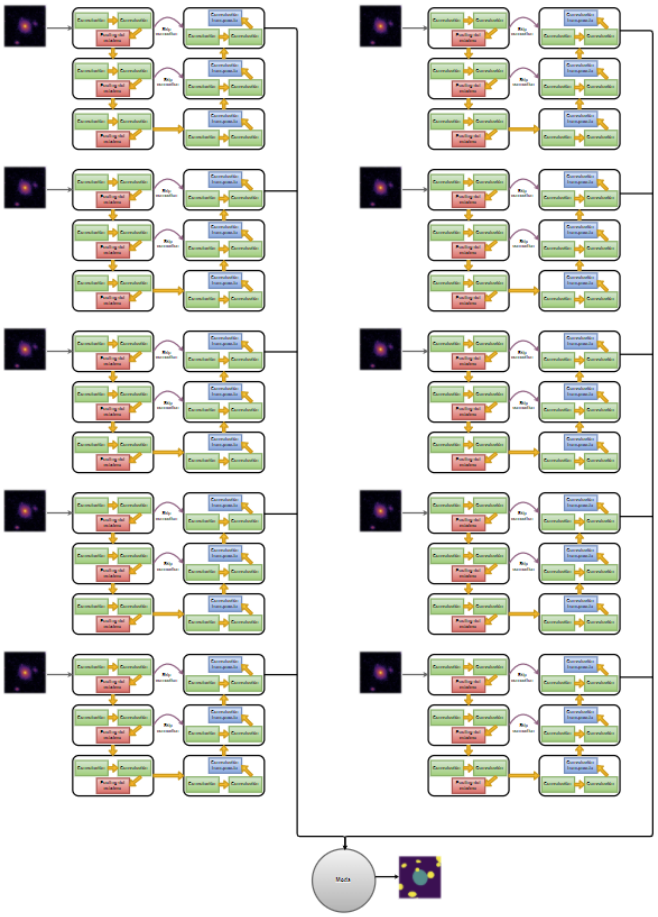

Figura 7.20: *Ensemble* construido.

Situando el *ensemble* creado en la taxonomía de Rokach, en la categoría de *combiner* se encuadra en la categoría especificado no entrenable. En la categoría de construcción del *ensemble* en

entrenamiento de los clasificadores base de manera independiente. En la categoría de diversidad, esta viene inducida por la diferencia en los parámetros (número de canales) entre los clasificadores base y el muestreo *bootstrap sobre el conjunto de datos*. El tamaño del *ensemble* es fijo, y está preparado para su uso con un solo tipo de clasificador base: redes neuronales convolucionales en forma de U (Figura 7.19). La arquitectura final del *ensemble* construido se aprecia en la Figura 7.20.

7.4.1.2 Resultados de los clasificadores base

En la Figura 7.21 se aprecia la evolución de la pérdida de y de la tasa de acierto de las U-*Nets* que actúan como clasificadores base del *ensemble*. Las diferencias de comportamiento entre las redes son notables, producto de la diversidad causada al realizar *bootstrap* sobre el conjunto de entrenamiento y crear las redes con una configuración diferente (número de canales a los que se pasa tras realizar las dos operaciones de convolución del bloque de contracción con su correspondiente simetría en la etapa de expansión.)

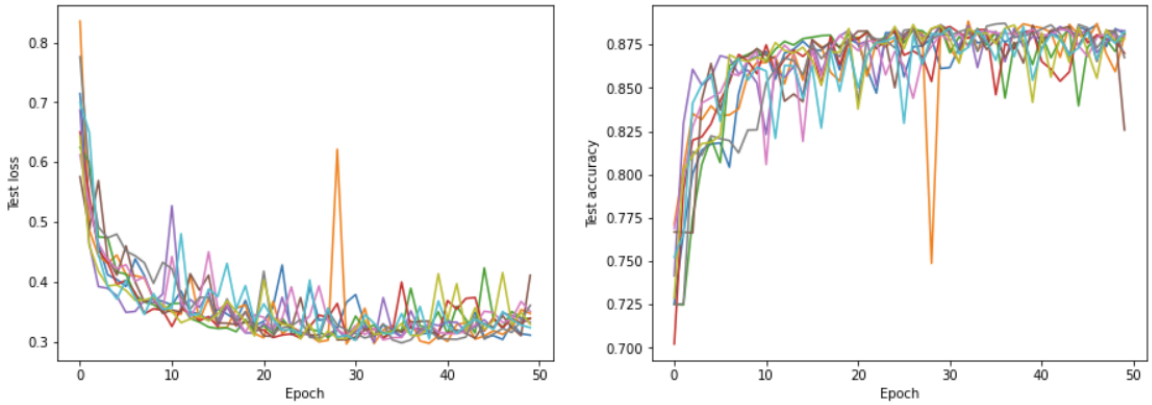

Figura 7.21: Evolución del *loss* (izquierda) y el *accuracy* (derecha) de los clasificadores base sobre el conjunto de test.

La tasa de acierto más alta sobre el conjunto de entrenamiento es la obtenida por la red 5, con 0.8830, seguida de la red 10 con 0.8825. Las dos tasas de acierto más bajas son las obtenidas por las redes 6 y 8, con 0.8257 y 0.8675 respectivamente (Cuadro 7.4).

Clasificador base	***Accuracy***
1	0.8812
2	0.8804
3	0.8813
4	0.8699
5	0.8830
6	0.8257
7	0.8800
8	0.8675
9	0.8783
10	0.8825

Cuadro 7.4: *Accuracy* de los clasificadores base sobre el conjunto de test.

7.4.1.3 Resultados del *ensemble*

La mejoría producida por el *ensemble* sobre los clasificadores base es evidente (Figura 7.22). A partir de la época 15 la evolución de la tasa de acierto sobre el conjunto de test del *ensemble* comienza a diferenciarse de manera positiva sobre las evoluciones de las U-*Nets*, haciéndose esa divergencia más notable conforme se suceden las épocas. Además, el comportamiento del *ensemble* es mucho menos variable que el de los clasificadores base, apreciándose mucha menos varianza en la evolución del *accuracy* respecto a las redes en U.

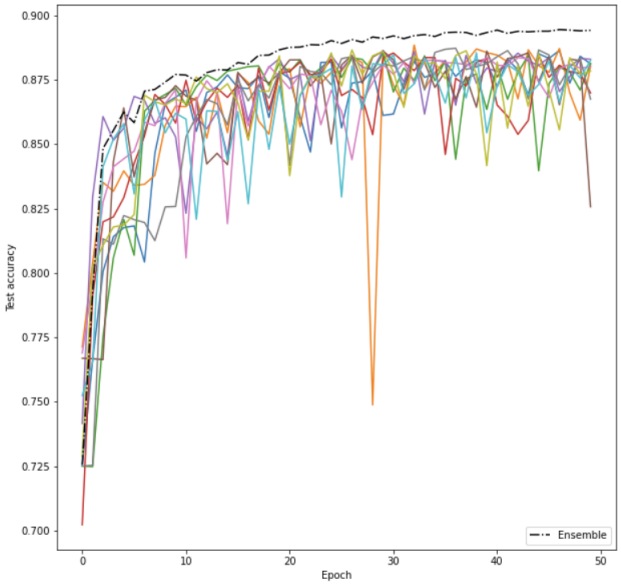

Figura 7.22: Evolución del *accuracy* del *ensemble* sobre el conjunto de test.

La tasa de acierto obtenida sobre el conjunto de test es de 0.8942, más de un 1 % mayor que la mayor de las tasas de acierto de los clasificadores base (0.8830), por lo que la construcción del *ensemble* ha sido un éxito (Cuadro 7.5).

Clasificador	*Accuracy*
Ensemble	0.8942

Cuadro 7.5: *Accuracy* del *ensemble* sobre el conjunto de test.

Se ha demostrado como el paradigma de aprendizaje por *ensembles* utilizando U-*Nets* como clasificadores base, la moda como método de combinación de resultados y *bootstrap* y diferentes configuraciones de las redes como modo de inducir diversidad aplicado a la segmentación semántica de imágenes astronómicas obtiene resultados excelentes.

7.4.2. *Ensemble* de 10 U-*Nets* combinando mediante algoritmos de ML

Como ya se explicó en la taxonomía de Rokach, el método democrático de obtener las predicciones finales del *ensemble* mediante la moda de las predicciones de los clasificadores base no es el único.

Otros métodos como la media, la mediana o incluso introducir las salidas de los clasificadores base en un algoritmo de *machine learning* son opciones válidas. Esta última opción es la que se va a probar, y en concreto se han elegido 2 algoritmos: Naive-Bayes y regresión logística (Figura 7.23).

Para ello, se pasan las 345 imágenes de test a través de los clasificadores base y se obtienen sus predicciones. De esas 345 imágenes, el 66 %, 231, se van a utilizar para ajustar el algoritmo de ML junto con las etiquetas correspondientes, y las otras 114 imágenes se usan como conjunto de test para estimar la tasa de acierto del *ensemble*.

Las imágenes han de ser aplanadas, convirtiéndose en un vector de 40000 posiciones. Para cada posición del vector, es decir cada píxel de la imagen, se introducen las 10 predicciones de las U-*Nets* en el algoritmo de Naive-Bayes o la regresión logística, y estos, en base a las entradas, ofrecen su predicción. Dicha predicción es la predicción final que ofrece el *ensemble*.

En el Cuadro 7.6 se aprecia como la tasa de acierto que obtiene el *ensemble* sobre el conjunto de test con cualquiera de los dos algoritmos como método de combinación es menor que la que se obtuvo utilizando el criterio democrático de la moda. No obstante, el segundo clasificador (Naive Bayes) obtiene mayor tasa de acierto que cualquiera de las U-*Nets* que actuaban como clasificadores base, por lo que es otro caso de éxito de construcción del *ensemble*.

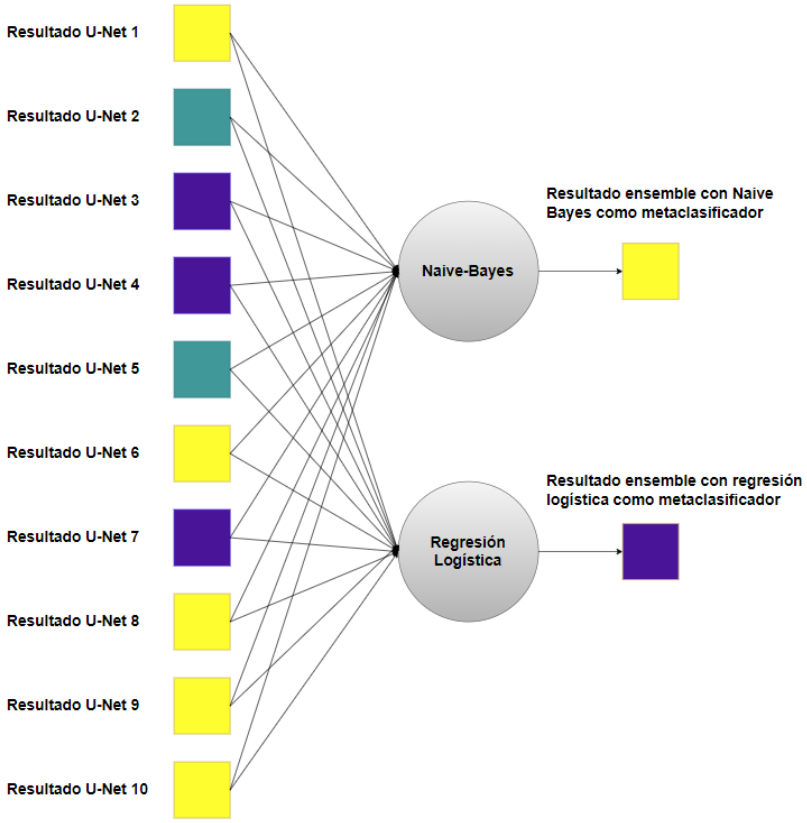

Figura 7.23: Combinación de las salidas de los clasificadores base con ML.

Algoritmo ML	***Accuracy***
Regresión Logística	0.8605
Naive-Bayes	0.8879

Cuadro 7.6: *Accuracy* del *ensemble* sobre el conjunto de test según algoritmo ML.

7.5. Conclusiones

A modo de conclusiones y cerrando el capítulo de detección de bordes, se ha constatado como las redes convolucionales en forma de U ofrecen un desempeño bastante bueno en la tarea de delimitar las fronteras de las galaxias.

La U-*Net* del experimento base sufría ciertos problemas de sobreajuste, los cuáles fueron solventados mediante la introducción de *drop outs* (probando distintas probabilidades de eliminación

de neruonas en distintos experimentos) y mediante aumentos aleatorios en cada época de las imágenes de entrenamiento. Esta última técnica fue la que resultó eliminar el *overfitting* de una manera más efectiva. Respecto a los *drop outs*, la configuración con p = 0.25 anuló por completo la capacidad de aprendizaje de la red al eliminar demasiadas neuronas. El resto de configuraciones funcionaron de una manera bastante similar, obteniendo buenos resultados.

Respecto al aprendizaje por *ensembles*, se construyeron 3 configuraciones distintas. Todas ellas compartían los mismos elementos (10 redes neuronales en U como clasificadores base) excepto la fase final de combinación de resultados, donde se usaron un método democrático (la moda) y dos algoritmos de *machine learning*: la regresión logística y Naive-Bayes. Tanto el criterio democrático como Naive Bayes consiguieron mejorar las métricas obtenidas por cualquiera de los clasificadores base constituyendo la creación de un *ensemble* exitoso, mientras que la regresión logística por el contrario obtuvo peores resultados.

8. Conclusiones y líneas futuras de investigación

8.1. Conclusiones

En este trabajo se han diseñado y aplicado algoritmos basados en *deep learning*, redes neuronales convolucionales concretamente, a dos problemas complejos en el contexto de la astronomía, la clasificación morfológica y la detección de bordes de galaxias.

Todas las imágenes que se han utilizado para diseñar, construir y evaluar los algoritmos desarrollados han sido tomadas por el telescopio Hubble en el marco del proyecto CANDELS.

El primer problema, el de clasificación morfológica, se atacó por dos vías diferentes, obteniendo resultados muy distintos. En primera instancia, se trató de clasificar directamente las galaxias en base a su morfología, obteniendo un desempeño no demasiado bueno.

Algunas de las técnicas desarrolladas sufrían un claro problema de sobreajuste, obteniendo unas métricas prácticamente perfectas sobre el conjunto de entrenamiento pero siendo bastante peores sobre imágenes que no habían visto las redes neuronales convolucionales. Otras configuraciones, a pesar de evitar el *overfitting*, no conseguían aprender los suficiente de las imágenes de entrenamiento como para que el desempeño final fuera razonablemente bueno.

La segunda vía que se siguió para afrontar el problema de clasificación morfológica fue tratar de predecir unos parámetros que representaban la proporción de votaciones dadas por expertos a la morfología de las galaxias. Mediante un problema de regresión se predijeron esos parámetros, y posteriormente dicha salida fue reconstruida para obtener una clasificación final de la forma de la galaxia.

Los resultados obtenidos fueron excelentes, rozando la perfección sobre el conjunto de test para varias configuraciones distintas. Se comprobó como, por norma general, tasas de aprendizaje muy bajas no producen buenos resultados, mientras que tasas de aprendizaje más altas así como la aplicación del suavizado de etiquetas convergen a resultados más que razonables. También se constató que introducir las imágenes tanto en los canales HIJV como RGBα produce resultados muy similares y de calidad, a pesar del evidente cambio visual que se produce en las imágenes.

En ambos casos, y debido a la importancia que se ha visto durante el desarrollo del proyecto que tienen las transformaciones de las imágenes mediante *data augmentation*, las imágenes fueron transformadas sin variar su morfología, así como modificadas con ruido gaussiano aleatorio en cada época para garantizar que las CNNs no veían dos imágenes iguales durante todo el entrenamiento.

Tras el problema de clasificación morfológica, se afrontó la detección de bordes de galaxias mediante segmentación semántica. Primero, se diseñó una U-*Net* y se comprobó como los resul-

tados obtenidos eran bastante buenos, clasificando acertadamente entorno al 88 % de los píxeles del conjunto de test.

Sin embargo, y a pesar de aplicar técnicas de *data augmentation* como rotaciones y espejismos, esta configuración adolecía de cierto sobreajuste, el cual se intentó paliar mediante *drop outs* con distintas probabilidades y *zooms* aleatorios en cada época. Esta última técnica resultó ser la más efectiva para eliminar el sobreajuste.

Continuando con la segmentación semántica e intentado mejorar los resultados obtenidos con la U-*Net*, se diseñó y construyó un *ensemble* de 10 U-*Nets* con combinación de voto democrático mediante la moda con el que se consiguió mejorar la tasa de acierto obtenida por cualquiera de los clasificadores base, llegando hasta un 89 % de píxeles bien clasificados en imágenes del conjunto de test.

También se probó a juntar las salidas ofrecidas por las redes convolucionales en U mediante dos algoritmos de *machine learning* la regresión logística y Naive Bayes. El segundo consiguió mejorar a cualquiera de los clasificadores base, sin llegar al excelente rendimiento ofrecido por la moda, mientras que la regresión logística no funcionó tan bien y empeoró los resultados de la mayoría de las U-*Nets*.

Estos resultados, en los que combinando la predicción de varias redes convolucionales en U se consiguen mejorar los resultados de cualquiera de las redes por individual, ofrecen líneas futuras de investigación muy potentes e innovadoras.

8.2. Líneas futuras de investigación

El potencial uso de técnicas *deep learning* aplicadas a problemas complejos en el marco de la astronomía y la astroinformática queda fuera de toda duda.

Queda como trabajo pendiente descubrir nuevas técnicas de *data augmentation* que permitan reducir el sobreajuste obtenido en el problema de clasificación morfológica directa. También, el desarrollo de técnicas aplicadas y modelos de redes neuronales convolucionales que se adecúen a dicha tarea.

En el campo de la detección de bordes de objetos galácticos y en concreto la segmentación semántica, conseguir combinar redes convolucionales en U de la manera más óptima posible para obtener unas métricas finales aún mejores de las ya obtenidas tiene un gran interés y multitud de aplicaciones.

Por último, no hace falta reseñar la posibilidad de explorar nuevos problemas que hasta ahora ni se habían planteado y que pueden ser resueltos mediante el aprendizaje profundo.

Referencias

[1] Cosmic Assembly Near-infrared Deep Extragalactic Legacy Survey. (2010). Recuperado el 10 de Junio de 2021 de http://arcoiris.ucolick.org/candels/

[2] Baron, Dalya. (2019). Machine Learning in Astronomy: a practical overview.

[3] Neurona de McCulloch-Pitts. (s.f). En Wikipedia. Recuperado el 20 de Noviembre de 2020 de https://es.wikipedia.org/wiki/Neurona_de_McCulloch-Pitts

[4] Nielsen, Michael. (2018). Neural Networks and Deep Learning.

[5] The Vanishing/Exploding Gradient Problem in Deep Neural Networks. (2020). En Towards Data Science. Recuperado el 15 de Diciembre de 2020 de https://towardsdatascience.com/the-vanishing-exploding-gradient-problem-in-deep-neural-networks-191358470c11

[6] Red neuronal convolucional. (s.f). En Wikipedia. Recuperado el 22 de Diciembre de 2020 de https://es.wikipedia.org/wiki/Red_neuronal_convolucional

[7] Dumoulin, V., & Visin, F. (2016). A guide to convolution arithmetic for deep learning. ArXiv, abs/1603.07285.

[8] Xiao, Han & Rasul, Kashif & Vollgraf, Roland. (2017). Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms.

[9] Publication-ready NN-architecture schematics. Recuperado el 20 de Mayo de 2021 de http://alexlenail.me/NN-SVG/AlexNet.html

[10] Stern R., Arias J.C. (2011). Review of Risk Management Methods.

[11] Hubble Tuning Fork Classification for Galaxies. (2009). En SlideShare. Recuperado el 10 de Abril de 2020 de https://www.slideshare.net/ninabean47/interacting-galaxies/12-Hubble_Tuning_Fork_Classification_for

[12] Molino, A.; Benítez, N.; Moles, M.; Fernández-Soto, A.; Cristóbal-Hornillos, D.; Ascaso, B.; Jiménez-Teja, Y.; Schoenell, W.; Arnalte-Mur, P.; Pović, M. et al. (2014). The ALHAMBRA Survey: Bayesian photometric redshifts with 23 bands for 3 deg. Monthly Notices of the Royal Astronomical Society, Volume 441, Issue 4, p.2891-2922.

[13] Postman, Marc & Coe, Dan & Benitez, Narciso & Bradley, Larry & Broadhurst, Thomas & Donahue, Megan & Ford, Holland & Graur, Or & Graves, Genevieve & Jouvel, Stephanie & Koekemoer, Anton & Lemze, Doron & Medezinski, Elinor & Molino, Alberto & Moustakas, Leonidas & Ogaz, Sara & Riess, Adam & Rodney, Steve & Rosati, Piero & van der Wel, Arjen. (2012). The Cluster Lensing and Supernova Survey with Hubble: An Overview. The Astrophysical Journal Supplement Series. 199. 25. 10.1088/0067-0049/199/2/25.

[14] FITS. (s.f.). En Wikipedia. Recuperado el 10 de Abril de 2020 de https://en.wikipedia.org/wiki/FITS

[15] SAOImageDS9. Recuperado el 10 de Abril de 2020 de https://sites.google.com/cfa.harvard.edu/saoimageds9/about

[16] Aladin Sky Atlas. Recuperado el 10 de Abril de 2020 de https://aladin.u-strasbg.fr/

[17] SExtractor. Recuperado el 10 de Abril de 2020 de https://sextractor.readthedocs.io/en/latest/Introduction.html

[18] B. W. Holwerda. (2005). Source Extractor for Dummies v5.

[19] PyTorch. (s.f.). En Wikipedia. Recuperado el 10 de Mayo de 2020 de https://en.wikipedia.org/wiki/PyTorch

[20] Cálculo tensorial. (s.f.). En Wikipedia. Recuperado el 10 de Mayo de 2020 de https://es.wikipedia.org/wiki/C%C3%A1lculo_tensorial#:~:text=En%20matem%C3%A1ticas%20y%20en%20f%C3%ADsica,convenio%20de%20suma%20de%20Einstein.

[21] Documentación de PyTorch. (s.f.). Recuperado el 10 de Mayo de 2020 de https://pytorch.org/docs/stable/data.html

[22] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2017. ImageNet classification with deep convolutional neural networks. Commun. ACM 60, 6 (June 2017), 84–90.

[23] Documentación de fastai. (2021). Recuperado el 10 de Mayo de 2020 de https://docs.fast.ai/

[24] Google Colaboratory. (s.f.). Recuperado el 10 de Mayo de 2020 de https://colab.research.google.com/notebooks/intro.ipynb?utm_source=scs-index

[25] Sharon Y. Li. (2020). Automating Data Augmentation: Practice, Theory and New Direction. En The Stanford AI Lab Blog. Recuperado el 5 de Enero de 2020 de http://ai.stanford.edu/blog/data-augmentation/

[26] Huertas-Company, M. et al. (2015). A catalog of visual-like morphologies in the 5 CANDELS fields using deep-learning. arXiv: Astrophysics of Galaxies.

[27] Dieleman, Sander & Willett, Kyle & Dambre, J.. (2015). Rotation-invariant convolutional neural networks for galaxy morphology prediction. Monthly Notices of the Royal Astronomical Society. 450. 10.1093/mnras/stv632.

[28] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 770-778.

[29] Ronneberger, O., Fischer, P., & Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. MICCAI.

[30] Maurício Cordeiro. (2020). Semantic Segmentation of Satellite Images. En Medium. Recuperado el 15 de Abril de 2020 de `https://medium.com/analytics-vidhya/creating-a-very-simple-u-net-model-with-pytorch-for-semantic-segmentation-of-\satellite-images-223aa216e705`

[31] Hinton, G.E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. ArXiv, abs/1207.0580.

[32] Park, S., & Kwak, N. (2016). Analysis on the Dropout Effect in Convolutional Neural Networks. ACCV.

[33] Nilsson, N. (1965). Learning Machines: Foundations of Trainable Pattern-Classifying Systems, New York: McGraw-Hill.

[34] Kuncheva, L. (2014). Combining Pattern Classifiers: Methods and Algorithms, 2nd Edition. Wiley.