TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA INFORMÁTICA

Clasificación de galaxias mediante Deep Learning

Autor
Jorge Picado Cariño

Director
Julián Luengo Martín

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, julio de 2023

Clasificación de galaxias mediante Deep Learning

Autor
Jorge Picado Cariño

Director
Julián Luengo Martín

Clasificación de galaxias mediante Deep Learning

Jorge Picado Cariño

Palabras clave: galaxias, segmentación de imágenes, clasificación de imágenes, Deep Learning, Aprendizaje Supervisado, Aprendizaje No Supervisado, Visión por Computador

Resumen

En las últimas décadas, el campo de la astronomía ha experimentado un crecimiento exponencial gracias a los avances en la tecnología y el acceso a grandes volúmenes de datos. Uno de los aspectos que engloba la astronomía es el estudio de las galaxias [1], que son agrupaciones de estrellas, cuerpos celestes y materia cósmica que está concentrada en una determinada región del espacio por efecto de la atracción gravitatoria y constituye una unidad en el universo.

La clasificación de galaxias es una tarea fundamental en la astronomía, ya que permite comprender mejor la formación y evolución del universo. Tradicionalmente, esta clasificación se ha llevado a cabo de manera manual por astrónomos expertos, quienes examinan imágenes de galaxias y las categorizan en diferentes tipos según su forma, estructura y otras características observables.

Sin embargo, debido al crecimiento exponencial de los datos astronómicos, se requiere de métodos más eficientes y automatizados para clasificar las galaxias de manera precisa y rápida. En cada imagen del espacio de un telescopio pueden aparecer decenas de galaxias. La tarea de los astrónomos especializados en su clasificación (espiral, elíptica, etc.) puede suponer un tiempo considerable que tiene un coste económico perceptible y una merma de otras tareas que el experto podría estar realizando. En este contexto, el aprendizaje profundo, o "Deep Learning", ha demostrado ser una herramienta poderosa y prometedora.

Se espera que este enfoque de clasificación automatizada de galaxias mediante Deep Learning proporcione resultados eficientes en comparación con los métodos tradicionales. Además, el sistema desarrollado podría utilizarse como una herramienta complementaria para los astrónomos, ayudándoles a procesar grandes volúmenes de datos y acelerar el proceso de clasificación.

[1] Definición extraída de Oxford Languages

El objetivo de este Trabajo de Fin de Grado (TFG) es el uso de técnicas de Deep Learning para segmentar las galaxias de las imágenes astronómicas, así como clasificar su forma si la resolución es suficiente. Se planteará el uso de técnicas supervisadas frente a no supervisadas para ahorrar el máximo tiempo al posible astrónomo experto. Para lograr esto, se utilizarán conjuntos de datos astronómicos que contienen imágenes de galaxias previamente clasificadas, este dataset está disponible en el proyecto Galaxy Zoo. Estas imágenes serán utilizadas para entrenar un modelo que aprenderá a reconocer los patrones y características distintivas de cada tipo de galaxia.

En resumen, este TFG tiene como objetivo principal la clasificación automatizada de galaxias mediante el uso de técnicas de Deep Learning. Se espera que los resultados obtenidos contribuyan al avance del campo de la astronomía y abran nuevas puertas para la exploración y comprensión del universo.

Classification of Galaxies Using Deep Learning

Jorge Picado Cariño

Keywords: galaxies, image segmentation, image classification, Deep Learning, Supervised Learning, Unsupervised Learning, Computer Vision

Abstract

In recent decades, the field of astronomy has experienced exponential growth thanks to advances in technology and access to large volumes of data. One aspect encompassed by astronomy is the study of galaxies, which are clusters of stars, celestial bodies, and cosmic matter that are concentrated in a certain region of space due to gravitational attraction and form a unit in the universe.

The classification of galaxies is a fundamental task in astronomy as it provides a better understanding of the formation and evolution of the universe. Traditionally, this classification has been carried out manually by expert astronomers who examine images of galaxies and categorize them into different types based on their shape, structure, and other observable characteristics.

However, due to the exponential growth of astronomical data, more efficient and automated methods are required to classify galaxies accurately and quickly. Dozens of galaxies can appear in each telescope image of space. The task of astronomers specializing in their classification (spiral, elliptical, etc.) can be time-consuming, resulting in a noticeable economic cost and detracting from other tasks the expert could be performing. In this context, deep learning, or "Deep Learning,"has proven to be a powerful and promising tool.

The automated galaxy classification approach using Deep Learning is expected to provide efficient results compared to traditional methods. Additionally, the developed system could be used as a complementary tool for astronomers, helping them process large volumes of data and accelerate the classification process.

This Final Degree Project (TFG) aims to use Deep Learning techniques to segment galaxies in astronomical images and classify their shape if the

resolution is sufficient. The use of supervised versus unsupervised techniques will be considered to save as much time as possible for the potential expert astronomer. To achieve this, astronomical datasets containing images of previously classified galaxies will be used. This dataset is available from the Galaxy Zoo project. These images will be used to train a model that will learn to recognize the distinctive patterns and characteristics of each galaxy type.

In summary, the main objective of this TFG is the automated classification of galaxies using Deep Learning techniques. It is hoped that the results obtained will contribute to the advancement of the field of astronomy and open new doors for the exploration and understanding of the universe.

Yo, **Jorge Picado Cariño**, alumno de la titulación Grado en Ingeniería Informática de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 75930960W, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Jorge Picado Cariño

Granada a X de julio de 2023.

D. **Julián Luengo Martín**, Profesor del Área de XXXX del Departamento Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada.

Informa:

Que el presente trabajo, titulado ***Clasificación de galaxias mediante Deep Learning***, ha sido realizado bajo su supervisión por **Jorge Picado Cariño**, y autorizo la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a X de julio de 2023.

El director:

Agradecimientos

Quiero agradecer a todas las personas que me han apoyado en la realización de mi Trabajo de Fin de Grado, gracias a ellos este logro no hubiera sido posible.

En primer lugar a mi tutor académico Julián, por su orientación, apoyo y valiosos consejos a lo largo de todo el proceso de elaboración de este trabajo.

A mis compañeros y compañeras de clase, por su colaboración, intercambio de ideas y apoyo mutuo. Fue enriquecedor poder contar con sus opiniones y consejos no sólo durante este trabajo, sino a lo largo de estos 4 años.

A mi familia, por su amor, paciencia y comprensión a lo largo de toda mi etapa académica. Su apoyo incondicional me impulsó a alcanzar mis metas.

Sin todas estas personas, este Trabajo de Fin de Grado no hubiera sido posible. Les estoy profundamente agradecido por su tiempo, esfuerzo y dedicación.

Índice general

1. Introducción **23**

1.1. Motivación 24

1.2. Objetivos 25

1.3. Presupuesto 25

1.4. Planificación 26

2. Estado del Arte y Conceptos **29**

2.1. Visión por Computador 30

2.2. Clasificación de imágenes 31

2.3. Segmentación de imágenes 32

2.4. Conceptos básicos Deep Learning 33

2.4.1. Redes Neuronales Convolucionales (CNN) 33

2.4.2. Capas de pooling 33

2.4.3. Transferencia de aprendizaje 35

2.4.4. Aumento de datos 36

2.4.5. Frameworks y bibliotecas 37

2.5. Aprendizaje Supervisado 37

2.6. Aprendizaje No Supervisado 37

2.7. Validación cruzada 38

3. Dataset, modelos de clasificación, segmentación y GANs 41

3.1. Dataset 42

3.1.1. Conjunto de imágenes 43

3.1.2. Archivo CSV 43

3.1.3. Archivo json 44

3.2. Modelos de clasificación 44

3.2.1. AlexNet 45

3.2.2. VGG 46

3.2.3. DenseNet 47

3.2.4. EfficientNet 48

3.2.5. ResNet 49

3.3. Modelos de segmentación 49

3.3.1. DeepLabV3 50

3.4. GANs 53

3.4.1. Real-ESRGAN 54

4. Pasos Previos 55

4.1. Clasificación 55

4.1.1. Preprocesado 55

4.1.2. Entrenamiento 58

4.1.3. Pruebas con imágenes 69

4.2. Segmentación 75

4.2.1. Preprocesado 75

4.2.2. Entrenamiento 76

4.3. Comparativa de Hardware 81

4.3.1. Intel(R) UHD Graphics 620 vs NVIDIA Tesla K80 . . 81

4.3.2. NVIDIA Tesla K80 vs GPU DE LA UNIVERSIDAD . 82

Referencias **85**

Índice de figuras

1.1. Previsión del aumento de datos en los próximos años [20] . . 24

1.2. Diagrama de Gantt estimado 27

1.3. Diagrama de Gantt final 27

2.1. Ejemplo de detección de objetos [16] 30

2.2. Ejemplo de segmentación semántica [16] 31

2.3. Ejemplo de Average Pooling 34

2.4. Ejemplos de patrones (imagen, dígito) 37

2.5. Ejemplo de aprendizaje no supervisado 38

2.6. Validación cruzada 40

3.1. Ejemplos de imágenes de galaxias extraídas de Galaxy Zoo . 43

3.2. Ejemplo de imagen y su correspondiente máscara 45

3.3. Arquitectura AlexNet con 5 capas convolucionales y 3 capas totalmente conectadas [12] 46

3.4. Arquitectura VGG16 47

3.5. Representación esquemática de DenseNet 48

3.6. Arquitectura EfficientNet [8] 49

3.7. Arquitectura ResNet 50

3.8. Función ReLU 51

3.9. Modelo de segmentación semántica DeepLabV3 53

3.10. Funcionamiento de GANs 54

4.1. Organización de las imágenes en carpetas en clasificación . . 56

4.2. Cómo afecta el learning rate al algoritmo 60

4.3. Ejemplo de un modelo perfecto con pérdida logarítmica de 0 . 63

4.4. Gráficos evolutivos de *accuracy* y *loss* de AlexNet 64

4.5. Gráficos evolutivos de *accuracy* y *loss* de VGG 65

4.6. Gráficos evolutivos de *accuracy* y *loss* de DenseNet 66

4.7. Gráficos evolutivos de *accuracy* y *loss* de EfficientNet 67

4.8. Gráficos evolutivos de *accuracy* y *loss* de ResNet 68

4.9. Gráficos evolutivos de *accuracy* y *loss* de ResNet con el doble de imágenes 69

4.10. Secuencia de recorte 70

4.11. Gráficos evolutivos de *accuracy* y *loss* originales - recortadas 71

4.12. Gráficos evolutivos de *accuracy* y *loss* recortadas - originales 71

4.13. Gráficos evolutivos de *accuracy* y *loss* recortadas - recortadas 72

4.14. Ejemplo de restauración 73

4.15. Gráficos evolutivos de *accuracy* y *loss* con imágenes restauradas 74

4.16. Organización de las imágenes en carpetas en segmentación . . 75

4.17. Ejemplo de imagen y su máscara binaria 77

4.18. Gráficos evolutivos de *accuracy* y *loss* con máscaras binarias . 78

4.19. Ejemplo de galaxia *elliptical* y su máscara 79

4.20. Ejemplo de galaxia *spiral* y su máscara 79

4.21. Ejemplo de galaxia *uncertain* y su máscara 79

4.22. Gráficos evolutivos de *accuracy* y *loss* con máscaras multiclase 80

4.23. Gráficos evolutivos de *accuracy* y *loss* de segmentación con el doble de imágenes 81

Índice de cuadros

1.1. Presupuesto 25

4.1. Comparación de Test entre modelos 68

4.2. Comparativa entrenamientos imágenes originales y recortadas 72

Capítulo 1

Introducción

En los últimos años, el campo de la astronomía ha experimentado avances significativos gracias al uso de técnicas de aprendizaje automático, en particular el aprendizaje profundo (Deep Learning). Estas técnicas han demostrado ser eficaces en diversas tareas, entre ellas el tema del presente trabajo, la clasificación de galaxias, un campo de estudio fundamental para comprender la evolución y la estructura del universo.

El aprendizaje profundo se basa en redes neuronales artificiales de múltiples capas, que son capaces de aprender representaciones de alto nivel a partir de datos sin procesar. En el contexto de la astronomía, estas redes neuronales pueden analizar grandes volúmenes de datos astronómicos, como imágenes de telescopios, espectros y curvas de luz, para extraer patrones complejos y realizar tareas de clasificación, detección y predicción.

Varios estudios han demostrado la eficacia del aprendizaje profundo en la clasificación de galaxias. Por ejemplo, en el trabajo de Dieleman et al. (2015) [4], se utilizó una red neuronal convolucional para clasificar automáticamente imágenes de galaxias del Sloan Digital Sky Survey (SDSS) en diferentes categorías morfológicas. Los resultados mostraron que el enfoque de aprendizaje profundo superó a los métodos tradicionales de clasificación realizados por expertos astrónomos.

Otro ejemplo relevante es el trabajo de Huertas-Company et al. (2015) [10], donde se utilizó una red neuronal profunda para clasificar galaxias según su evolución. El modelo fue capaz de identificar características sutiles en las imágenes galácticas que están relacionadas con la historia de formación estelar de las galaxias, lo que permitió clasificarlas de manera más precisa que los métodos anteriores.

Estos ejemplos demuestran el potencial del aprendizaje profundo en la astronomía y su capacidad para mejorar la eficiencia y la precisión en diversas tareas. A medida que se recopilan más datos astronómicos y se desarrollan nuevas técnicas de aprendizaje profundo, se espera que el campo continúe avanzando y brindando nuevas perspectivas sobre la estructura y evolución del universo.

1.1. Motivación

La clasificación de galaxias tradicionalmente ha sido realizada por astrónomos expertos, quienes analizan y categorizan manualmente las características visuales de las imágenes galácticas. Sin embargo, este enfoque es laborioso, subjetivo y limitado en términos de escala y eficiencia. Aquí es donde entra en juego el poder del aprendizaje profundo, que permite automatizar y mejorar el proceso de clasificación.

En las últimas décadas la cantidad de datos a procesar se ha incrementado exponencialmente. Este número de datos va a seguir aumentando en los próximos años de manera muy significativa, tal y como muestra el siguiente gráfico:

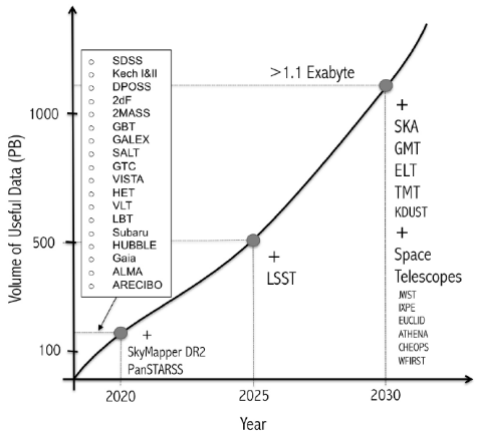

Figura 1.1: Previsión del aumento de datos en los próximos años [20]

En resumen, este trabajo se centra en la aplicación de técnicas de Deep Learning para la clasificación de galaxias, para mejorar la eficiencia y la precisión en este proceso fundamental para la astronomía. A través de la combinación de conocimientos en astronomía y aprendizaje automático, se espera aportar avances significativos en este campo y abrir nuevas oportunidades de investigación.

1.2. Objetivos

En este Trabajo de Fin de Grado (TFG), nos proponemos abordar el desafío de la clasificación de galaxias utilizando técnicas de Deep Learning. El objetivo principal es desarrollar un modelo de clasificación preciso y eficiente que pueda analizar grandes conjuntos de datos astronómicos y asignar de manera automática las galaxias a diferentes categorías.

Para lograr este objetivo, se emplearán redes neuronales convolucionales (CNN) y otras técnicas de Deep Learning, que han demostrado su eficacia en la clasificación de imágenes. Estas técnicas permitirán extraer características relevantes de las imágenes galácticas y aprender patrones complejos que son difíciles de discernir para el ojo humano.

Además, se utilizarán conjuntos de datos astronómicos existentes, que proporcionan una gran cantidad de imágenes galácticas etiquetadas con categorías conocidas. Estos datos serán preprocesados y se dividirán en conjuntos de entrenamiento, validación y prueba, para entrenar y evaluar el modelo propuesto.

Se espera que este trabajo contribuya al avance de la clasificación automatizada de galaxias, permitiendo una mayor eficiencia en la investigación astronómica y facilitando el análisis de grandes volúmenes de datos.

1.3. Presupuesto

El presupuesto necesario para la realización del presente trabajo queda reflejado en la siguiente tabla: [1] [2]

DESCRIPCIÓN	HORAS	PRECIO	TOTAL
Coste de Ingeniero	560	20.15€	11.284,00€
Coste de Portátil		800,00€	800,00€
Uso de GPU	200	0.48€	96,00€
Subtotal			12.180,00€
IVA (21 %)			2.557,80€
Total Presupuesto			**14.737,80€**

Cuadro 1.1: Presupuesto

[1] `https://www.jobted.es/salario/data-scientist`
[2] `https://aws.amazon.com/es/ec2/instance-types/g4/`

1.4. Planificación

Con el fin de garantizar un progreso adecuado del proyecto y cumplir con la fecha límite establecida, se han identificado las diferentes actividades que deben llevarse a cabo, así como el tiempo estimado para completar cada una de ellas. A continuación, se detallan las tareas identificadas:

- Análisis de las técnicas del estado del arte para clasificación de imágenes de aprendizaje supervisado y aprendizaje no supervisado.
- Estudio de los conjuntos de datos disponibles, recopilación y preparación de datos con el objeto de reunir un conjunto de datos astronómicos que contenga imágenes de galaxias etiquetadas correctamente.
- Preprocesamiento de las imágenes para asegurarse de que estén en un formato adecuado para el aprendizaje profundo. Esto incluye la normalización, el escalado y la eliminación de ruido.
- Elección de distintos modelos de aprendizaje profundo, se seleccionarán una serie de modelos concretos y se entrenarán utilizando el conjunto de datos recopilado.
- Evaluación de los modelos analizando la precisión y el rendimiento de cada uno de ellos entrenados utilizando métricas apropiadas.
- Comparación entre modelos seleccionados, examinando el rendimiento de los modelos de clasificación usados.
- Análisis e interpretación de los resultados obtenidos de las clasificaciones realizadas por el modelo.
- Conclusiones y trabajo futuro. Consistirá en resumir los resultados obtenidos y presentar conclusiones sobre la eficacia y las limitaciones del enfoque de clasificación de galaxias mediante el uso de aprendizaje profundo.

En el siguiente diagrama de Gantt, se muestra de manera visual la secuencia y duración de las tareas que se habían previsto llevar a cabo para la elaboración de este proyecto. Las tareas se han clasificado en colores para su fácil identificación.

Finalmente, debido a la complejidad del proyecto, se necesitó más tiempo para profundizar en aspectos teóricos, además del tiempo dedicado al entrenamiento de los distintos modelos y a la realización de una gran variedad de pruebas. A continuación, se muestra el reparto del tiempo empleado:

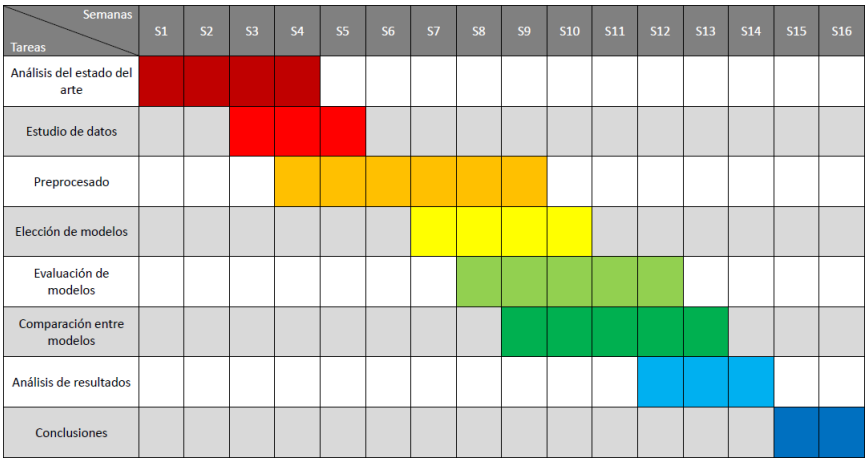

Figura 1.2: Diagrama de Gantt estimado

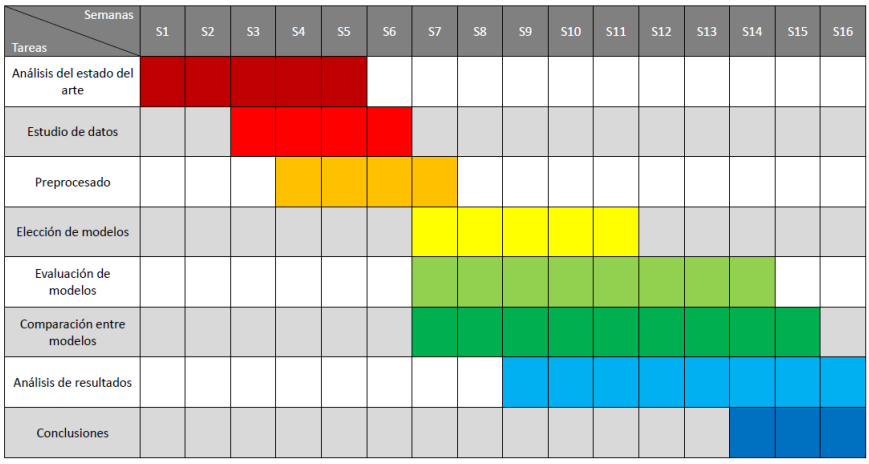

Figura 1.3: Diagrama de Gantt final

Capítulo 2

Estado del Arte y Conceptos

El objetivo principal de este capítulo es realizar un análisis exhaustivo del estado del arte en la clasificación de galaxias. Se introducirán conceptos como visión por computador, clasificación y segmentación de imágenes...

El estado del arte proporciona una visión general de los enfoques existentes y los desafíos que aún persisten en el campo de la clasificación de galaxias. Se revisarán técnicas de Deep Learning en la clasificación de galaxias, como las redes neuronales convolucionales (CNN) y las redes neuronales profundas.

En cuanto a los conceptos del Deep Learning, se explicarán los fundamentos teóricos y arquitecturas clave de las redes neuronales utilizadas en la clasificación de imágenes. Se abordarán conceptos como las capas convolucionales, las capas de pooling, aprendizaje supervisado y no supervisado... Además, se presentarán herramientas y bibliotecas populares utilizadas para implementar y entrenar modelos de Deep Learning en el contexto de la clasificación de galaxias.

Este capítulo proporcionará una base sólida para comprender los avances actuales en la clasificación de galaxias mediante el uso de técnicas de Deep Learning. A partir de esta revisión del estado del arte y la comprensión de los conceptos fundamentales.

2.1. Visión por Computador

La visión por computador [18] es una disciplina interdisciplinaria que combina el procesamiento de imágenes, el aprendizaje automático y la inteligencia artificial para permitir que las máquinas vean, comprendan e interpreten el contenido visual de las imágenes o vídeos de manera similar a como lo hacen los seres humanos.

La visión por computador se aplica en una amplia gama de campos, incluyendo la medicina, la industria automotriz, la seguridad, la robótica, la realidad virtual..., en nuestro caso se utilizará en astronomía.

La visión por computador aborda una amplia gama de tareas, que incluyen:

- Detección de objetos: La detección de objetos implica identificar y localizar la presencia de objetos específicos dentro de una imagen o vídeo. Esto puede involucrar la detección de rostros, vehículos, personas u otros objetos de interés.

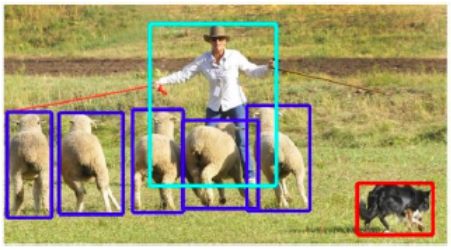

Figura 2.1: Ejemplo de detección de objetos [16]

- Seguimiento de objetos: El seguimiento de objetos implica rastrear y seguir el movimiento de un objeto específico a lo largo de una secuencia de imágenes o vídeo. Esto puede ser útil en aplicaciones de vigilancia, realidad aumentada o análisis de comportamiento.

- Reconocimiento de objetos: El reconocimiento de objetos implica identificar y clasificar diferentes clases de objetos en una imagen o vídeo. Puede abarcar desde reconocimiento de gestos y reconocimiento de caracteres hasta reconocimiento de objetos más complejos, como animales o edificios.

- Reconocimiento de escenas: El reconocimiento de escenas se centra en la identificación y clasificación de diferentes tipos de entornos o escenas

en una imagen o vídeo, como paisajes urbanos, interiores de edificios o paisajes naturales.

- Segmentación semántica: La segmentación semántica implica asignar una etiqueta semántica a cada píxel en una imagen, lo que permite identificar y distinguir diferentes objetos y regiones en la imagen.

Figura 2.2: Ejemplo de segmentación semántica [16]

- Estimación de pose: La estimación de pose se refiere a determinar la posición y orientación de un objeto en el espacio tridimensional a partir de una imagen o vídeo. Puede ser utilizado, por ejemplo, para el seguimiento de movimientos humanos o para la navegación autónoma de robots.

2.2. Clasificación de imágenes

La clasificación de imágenes [27] es un problema común en la visión por computador, que implica asignar etiquetas o categorías a las imágenes en función de su contenido. Puede ser utilizado en una amplia gama de aplicaciones, como reconocimiento de objetos, detección de rostros, diagnóstico médico, detección de spam, entre otros.

Existen diferentes enfoques para la clasificación de imágenes, como son la clasificación basada en características extraídas donde se extraen característica o descriptores de las imágenes y se utilizan para entrenar un clasificador y el aprendizaje profundo del que hablaremos en profundidad más adelante.

Es importante tener en cuenta que la clasificación de imágenes es un problema complejo y los resultados pueden variar según el conjunto de datos, la calidad de las imágenes, el tamaño del conjunto de entrenamiento y la elección del algoritmo o modelo utilizado. Además, es fundamental contar con un conjunto de datos etiquetado y representativo para el entrenamiento del clasificador.

2.3. Segmentación de imágenes

La segmentación de imágenes [26] es una tarea fundamental en el campo de la visión por computador, que implica dividir una imagen en regiones significativas o segmentos con características similares. La segmentación permite identificar y separar objetos o regiones de interés en una imagen, lo que facilita el análisis y procesamiento posterior.

Existen diferentes enfoques para la segmentación de imágenes, algunos de ellos son:

- Segmentación basada en umbrales: Este enfoque es uno de los más simples y se basa en la aplicación de umbrales en los valores de los píxeles de una imagen. Los píxeles se clasifican como pertenecientes a un segmento si cumplen ciertas condiciones de umbral en relación con sus valores de intensidad, color u otras características. Es útil cuando los objetos de interés en la imagen presentan una diferencia clara en sus características con respecto al fondo.
- Segmentación por regiones: Este enfoque se basa en la agrupación de píxeles similares en regiones coherentes. Se utilizan algoritmos de agrupamiento, como la agrupación jerárquica, el agrupamiento basado en crecimiento de regiones o el agrupamiento basado en grafos, para asignar los píxeles a las regiones correspondientes.
- Segmentación basada en bordes: En este enfoque, se detectan los bordes o contornos en una imagen y se utilizan como base para la segmentación. Se aplican operadores de detección de bordes, como el algoritmo de detección de bordes de diferencia de gaussianas (DoG), para identificar cambios abruptos en la intensidad de los píxeles que indican la presencia de bordes.
- Segmentación semántica: Este enfoque busca asignar etiquetas semánticas a los píxeles de una imagen, es decir, asignar una clase o categoría a cada píxel en función de su contenido. Se utilizan técnicas avanzadas de aprendizaje automático, como redes neuronales convolucionales (CNN). La segmentación semántica es útil cuando se requiere una comprensión detallada de la estructura y el contenido de una imagen.

Estos son solo algunos enfoques comunes utilizados en la segmentación de imágenes. Existen más métodos y algoritmos, y la elección del enfoque depende del problema específico y las características de la imagen que se desea segmentar.

2.4. Conceptos básicos Deep Learning

El Deep Learning, también conocido como aprendizaje profundo, es una rama de la inteligencia artificial que se basa en el uso de redes neuronales artificiales con múltiples capas para aprender y extraer representaciones de alto nivel de los datos. A diferencia del aprendizaje automático tradicional, el Deep Learning permite que los modelos aprendan automáticamente a través de capas de abstracción, lo que les permite capturar características complejas y realizar tareas más sofisticadas.

En la clasificación de imágenes, el Deep Learning ha demostrado ser altamente efectivo. Las imágenes de galaxias suelen ser grandes, complejas y con una gran cantidad de detalles. Las técnicas de Deep Learning pueden extraer características relevantes y aprendidas automáticamente, lo que permite una clasificación más eficiente. A continuación, se presentan algunos conceptos básicos del Deep Learning y su aplicación en la clasificación de imágenes:

2.4.1. Redes Neuronales Convolucionales (CNN)

Las redes neuronales convolucionales [15] son un tipo de arquitectura de Deep Learning diseñada específicamente para el procesamiento de imágenes. Utilizan capas convolucionales para extraer características locales y aprendidas automáticamente, y capas de pooling para reducir la dimensionalidad de los datos. Las CNN han logrado grandes avances en la clasificación de imágenes, incluidas las imágenes astronómicas.

Las redes convolucionales aplican filtros de convolución a las entradas. Se aprenden los parámetros de los filtros, siendo un filtro convolucional aquel que aplica una función de convolución sobre la entrada. El filtro convolucional de una 1D, 2D o múltiples dimensiones. Un ejemplo de filtro convolucional 2D es el siguiente:

$$S(m,n) = (X * Y)(m,n) = \sum_j \sum_i X(i,j)Y(m-i,n-j) \tag{2.1}$$

2.4.2. Capas de pooling

Las capas de pooling, son conocidas como capas de submuestreo. Como se ha comentado anteriormente, estas capas ayudan a reducir la dimensio-

nalidad espacial de las características extraídas, lo que a su vez disminuye la cantidad de parámetros en el modelo y ayuda a controlar el sobreajuste u overfitting [1] (concepto en la ciencia de datos que ocurre cuando un modelo estadístico se ajusta exactamente a sus datos de entrenamiento, siendo incapaz de generalizarse bien a nuevos datos). Además, las capas de pooling proporcionan cierta invarianza a pequeñas traslaciones y deformaciones en las características.

La operación principal de las capas de pooling [2] es aplicar una función de agrupación como:

- Max Pooling: que reduce los datos calculando el máximo, dentro de una ventana de filtro indicada.
- Average Pooling: que reduce los datos calculando su media dentro de una ventana de filtro indicada.

Esta función se aplica a regiones locales no solapadas de las características de entrada. La región de agrupación se desliza a través de las características, generalmente con un tamaño de ventana y un desplazamiento definidos, manteniendo las características más relevantes.

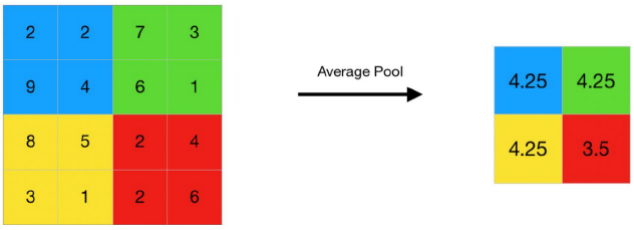

Figura 2.3: Ejemplo de Average Pooling

Las capas de pooling han sido ampliamente utilizadas en el procesamiento de imágenes y han demostrado ser útiles en tareas de clasificación de imágenes, un ejemplo es el artículo ÏmageNet Classification with Deep Convolutional Neural Networks. Advances in Neural Information Processing Systems"donde se hace uso de las capas para mejorar el rendimiento de la red. Al reducir la dimensionalidad de las características, las capas de pooling permiten una representación más compacta y manejable de la información, lo que facilita el procesamiento posterior y mejora la eficiencia computacional.

[1] `https://www.ibm.com/es-es/topics/overfitting`

[2] Información extraída Agencia Digital de Andalucía. *Experto en Inteligencia Artificial*, 2023.

2.4.3. Transferencia de aprendizaje

La transferencia de aprendizaje [17] es una técnica en la que se aprovecha un modelo pre-entrenado en un conjunto de datos grande y general, como ImageNet, y se ajusta para una tarea específica, como la clasificación de galaxias. Esto permite aprovechar el conocimiento previo del modelo en la extracción de características y acelerar el proceso de entrenamiento.

A continuación, se describen los pasos básicos de cómo funciona la transferencia de aprendizaje:

- Pre-entrenamiento: Se entrena un modelo en una tarea fuente utilizando un conjunto de datos grande y diverso. Durante el entrenamiento, el modelo aprende a extraer características relevantes de las imágenes y a realizar clasificaciones precisas.

- Extracción de características: Una vez que el modelo está pre-entrenado en la tarea fuente, se utiliza como una red de extracción de características. Las capas iniciales del modelo capturan características de bajo nivel, como bordes y texturas, mientras que las capas más profundas capturan características de alto nivel, como formas y objetos complejos. Estas características aprendidas son generalizables y pueden ser útiles para diferentes tareas de reconocimiento visual.

- Adaptación: El modelo pre-entrenado se adapta a la tarea objetivo y se ajustan sus pesos y parámetros utilizando un conjunto de datos más pequeño y específico de la tarea objetivo. Esto permite que el modelo se especialice y se ajuste mejor a los datos de la tarea objetivo.

- Evaluación y ajuste: Después de adaptar el modelo pre-entrenado a la tarea objetivo, se evalúa su rendimiento utilizando un conjunto de datos de prueba. Si el rendimiento no es satisfactorio, se pueden realizar ajustes adicionales, como modificar la arquitectura del modelo, ajustar los hiperparámetros o recopilar más datos para mejorar el rendimiento.

La transferencia de aprendizaje permite aprovechar el conocimiento previo adquirido durante el pre-entrenamiento en una tarea fuente para acelerar y mejorar el rendimiento en una tarea objetivo.

Es importante destacar que el éxito de la transferencia de aprendizaje depende de la similitud entre la tarea fuente y la tarea objetivo, así como de la calidad y la diversidad de los datos utilizados en el pre-entrenamiento.

2.4.4. Aumento de datos

El aumento de datos [23] es una técnica comúnmente utilizada en la clasificación de imágenes astronómicas mediante Deep Learning. Consiste en generar nuevas instancias de imágenes a partir de las existentes mediante transformaciones como rotaciones, cambios de escala, desplazamientos, entre otros. Esto ayuda a aumentar la diversidad de datos de entrenamiento y a mejorar la generalización del modelo.

Los pasos que sigue el aumento de datos son:

- Preparación del conjunto de datos: Primero, se recopila y prepara el conjunto de datos de entrenamiento inicial. Este conjunto de datos debe ser representativo y suficiente para la tarea específica que se va a abordar.

- Selección de transformaciones: A continuación, se seleccionan las transformaciones o técnicas de aumento de datos que se aplicarán al conjunto de datos existente. La elección de las transformaciones depende del dominio y la naturaleza de los datos.

- Aplicación de transformaciones: Las transformaciones seleccionadas se aplican a las muestras del conjunto de datos de entrenamiento existente, generando así nuevas variantes o instancias de datos. Cada muestra original se modifica según las transformaciones seleccionadas de forma aleatoria o sistemática. Es importante tener en cuenta que las transformaciones deben preservar las etiquetas o las características relevantes de las muestras.

- Ampliación del conjunto de datos: Al aplicar las transformaciones, se genera un nuevo conjunto de datos de entrenamiento ampliado que incluye las muestras originales y las variantes generadas. La cantidad de datos se incrementa significativamente, lo que puede mejorar el rendimiento del modelo y evitar el sobreajuste.

- Entrenamiento del modelo: Finalmente, el modelo de aprendizaje automático se entrena utilizando el conjunto de datos ampliado, que ahora incluye las variantes generadas. El modelo aprende de estas instancias adicionales, lo que puede mejorar su capacidad para generalizar y realizar predicciones precisas en nuevos datos.

2.4.5. Frameworks y bibliotecas

Existen varios frameworks y bibliotecas populares para implementar y entrenar modelos de Deep Learning en la clasificación de imágenes astronómicas, como PyTorch. Estas herramientas proporcionan una interfaz fácil de usar y optimizada para el cálculo numérico, lo que facilita la implementación y el entrenamiento de modelos de Deep Learning.

2.5. Aprendizaje Supervisado

El aprendizaje supervisado [5] es una técnica en el campo del aprendizaje automático que implica entrenar un modelo utilizando datos etiquetados, es decir, datos que contienen ejemplos de entrada junto con sus correspondientes salidas deseadas. El objetivo es que el modelo aprenda una función o mapeo entre las entradas y las salidas, de modo que pueda predecir las salidas para nuevas entradas no vistas previamente.

En el aprendizaje supervisado, se trabaja con un conjunto de datos de entrenamiento que consiste en pares de ejemplos de entrada-salida. El modelo se ajusta a estos datos de entrenamiento mediante la búsqueda de una función o un conjunto de parámetros que minimicen una medida de error o pérdida. Una vez que el modelo ha sido entrenado, se puede utilizar para realizar predicciones en nuevos datos de prueba.

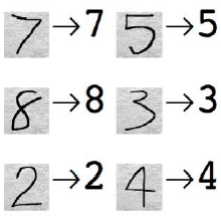

Figura 2.4: Ejemplos de patrones (imagen, dígito)

2.6. Aprendizaje No Supervisado

El aprendizaje no supervisado [3] es una rama del aprendizaje automático en la que se busca descubrir patrones o estructuras ocultas en los datos

[3]Información e imagen extraídas Agencia Digital de Andalucía. *Experto en Inteligencia Artificial*, 2023.

sin la ayuda de etiquetas o información de salida conocida. A diferencia del aprendizaje supervisado, en el que se tienen datos etiquetados, en el aprendizaje no supervisado se trabaja con datos no etiquetados y el objetivo principal es explorar la estructura subyacente de los datos.

De forma matemática, mientras que en aprendizaje supervisado contamos con patrones de datos (x, y), donde x es un array de atributos numéricos, e y un valor numérico que se quiere predecir como $y = f(x)$, en aprendizaje no supervisado, sólo se dispone de patrones de datos de entrada x_i y se trata de intentar abstraer los datos para su mejor comprensión.

Un ejemplo gráfico del objetivo del aprendizaje no supervisado sería el siguiente:

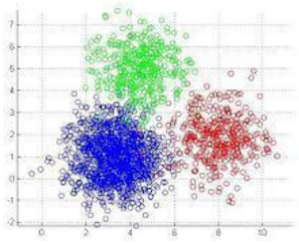

Figura 2.5: Ejemplo de aprendizaje no supervisado

Donde cada uno de los colores identificaría a cada uno de los grupos en los que se intenta crear subconjuntos de datos.

2.7. Validación cruzada

En clasificación de imágenes, es esencial evaluar el rendimiento del modelo de Deep Learning de manera robusta. La validación cruzada [1] es una técnica que divide el conjunto de datos en subconjuntos de entrenamiento y prueba, permitiendo evaluar el rendimiento del modelo en diferentes particiones de los datos y evitar el sobreajuste. En lugar de evaluar un modelo en un único conjunto de datos de prueba, la validación cruzada divide el conjunto de datos disponible en varias partes y realiza múltiples evaluaciones del modelo utilizando diferentes combinaciones de datos de entrenamiento y prueba.

El proceso de validación cruzada consiste en primer lugar en la división del conjunto de datos en k subconjuntos llamados *folds*. Por lo general, se utiliza una división en *k-folds*, donde k es un número entero especificado previamente. Cada *fold* tiene un tamaño similar y es representativo de los

datos originales.

Posteriormente, se realiza un ciclo de k iteraciones. En cada iteración, se selecciona un *fold* diferente como conjunto de prueba, mientras que los restantes $(k-1)$ *folds* se utilizan como conjunto de entrenamiento. Esto significa que cada *fold* actúa como conjunto de prueba una vez y como conjunto de entrenamiento $k-1$ veces.

Después, en cada iteración, se entrena el modelo utilizando el conjunto de entrenamiento y se evalúa su rendimiento utilizando el conjunto de prueba. Se registran las métricas de evaluación, como la precisión, la exactitud o el error, para cada iteración.

Por último, tras completar las k iteraciones, se promedian las métricas de evaluación obtenidas en cada iteración para obtener una medida agregada del rendimiento del modelo. Esto proporciona una estimación más precisa y fiable del rendimiento del modelo en comparación con una única evaluación en un conjunto de prueba.

Al utilizar diferentes combinaciones de datos de entrenamiento y prueba, se reduce la dependencia de una sola partición del conjunto de datos y se obtiene una visión más general del rendimiento del modelo en datos no vistos.

La elección adecuada del valor de k es importante. Los valores comunes para k son 5 y 10, pero también se utilizan otros valores, dependiendo del tamaño del conjunto de datos y los recursos computacionales disponibles. [4]

[4]Imagen extraída de scikit-learn

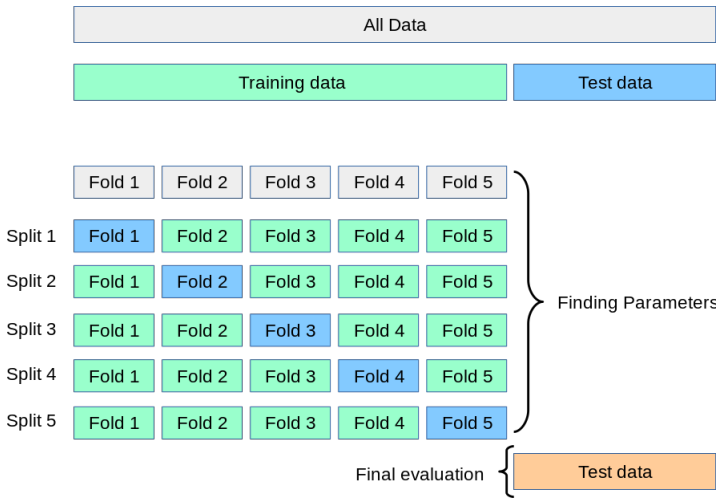

Figura 2.6: Validación cruzada

Capítulo 3

Dataset, modelos de clasificación, segmentación y GANs

Con el crecimiento exponencial de datos visuales en diversas aplicaciones, la clasificación de imágenes se ha convertido en una herramienta esencial para extraer información útil y tomar decisiones basadas en el contenido visual.

La clasificación de imágenes implica asignar etiquetas o categorías a las imágenes en función de su contenido visual. El objetivo es entrenar un modelo que pueda aprender patrones y características distintivas en las imágenes para distinguir diferentes clases.

La clasificación de imágenes presenta varios desafíos, como la variabilidad en la apariencia de los objetos, la presencia de ruido o distracciones en las imágenes y la necesidad de extraer características relevantes para la clasificación. Sin embargo, con el avance de las técnicas de aprendizaje automático y el uso de modelos de aprendizaje profundo, se han logrado avances significativos en este campo.

En este análisis, exploraremos enfoques y técnicas para resolver el problema de clasificación. Utilizaremos un conjunto de datos etiquetado que contiene imágenes de diferentes clases y emplearemos técnicas para entrenar un modelo que pueda clasificar las imágenes correctamente.

La clasificación de imágenes es un campo en constante evolución, y los avances en algoritmos y técnicas están mejorando continuamente la precisión

y el rendimiento de los modelos de clasificación. Al explorar este problema, no solo estaremos abordando un desafío actual y relevante, sino que también estaremos contribuyendo al crecimiento y desarrollo de la visión por computador y sus diversas aplicaciones prácticas.

3.1. Dataset

En esta sección, proporcionaremos una descripción detallada del conjunto de datos utilizado para entrenar y evaluar diversos modelos y algoritmos destinados a la clasificación de imágenes de galaxias.

Galaxy Zoo [1] es un proyecto de ciencia ciudadana en línea que invita a personas de todo el mundo a participar en la clasificación de galaxias. Fue lanzado en 2007 como una colaboración entre astrónomos y científicos ciudadanos para abordar el desafío de analizar un gran número de imágenes de galaxias.

El objetivo principal de Galaxy Zoo es clasificar galaxias en diferentes categorías según su forma y estructura. Los participantes del proyecto tienen acceso a imágenes reales de galaxias. Luego, utilizando una interfaz en línea, los voluntarios clasifican las galaxias según su forma, determinando si son espirales, elípticas u otras categorías.

La clasificación precisa y detallada de las galaxias es esencial para comprender la formación y evolución de las mismas. Los resultados obtenidos de las clasificaciones de los participantes de Galaxy Zoo se combinan y analizan para ayudar a los astrónomos a obtener información valiosa sobre la población galáctica, la interacción entre galaxias y otros fenómenos astronómicos.

El proyecto Galaxy Zoo ha generado una gran cantidad de datos valiosos y ha brindado a personas de diferentes orígenes la oportunidad de contribuir directamente al avance de la ciencia.

[1] `https://www.zooniverse.org/projects/zookeeper/galaxy-zoo/`

Figura 3.1: Ejemplos de imágenes de galaxias extraídas de Galaxy Zoo

3.1.1. Conjunto de imágenes

Dentro del proyecto Galaxy Zoo se ha utilizado el conjunto de imágenes de Galaxy Zoo 2. Este conjunto contiene 243.456 imágenes en formato .jpg, cada una de ellas nombrada con un id asociado.

3.1.2. Archivo CSV

Este archivo es una base de datos que contiene información de clasificaciones de galaxias. Cada fila del archivo CSV representa una clasificación de una galaxia específica y las columnas proporcionan diferentes atributos y características asociadas a esa clasificación.

A continuación se describe el significado de algunas de estas columnas:

- OBJID: identificador único a cada galaxia en el conjunto de datos.
- RA y DEC: coordenadas de la galaxia en el sistema de coordenadas astronómicas
- P_EDGE: probabilidad asignada de que el borde de una galaxia sea nítido o definido

Además de los ejemplos de columnas nombrados anteriormente, en particular para la clasificación de galaxias que nos ocupa este TFG, se ha trabajado específicamente con las columnas:

- SPIRAL
- ELLIPTICAL

- UNCERTAIN

Donde cada una de ellas tiene el valor 0 ó 1, lo que nos indica el tipo de la galaxia que es. Es decir, una de las tres columnas tiene el valor 1 y las otras dos el valor 0.

3.1.3. Archivo json

El archivo JSON de Galaxy Zoo 2 es un formato de archivo que contiene datos estructurados en formato JSON (JavaScript Object Notation) relacionados con las galaxias.

Este archivo contiene información detallada sobre cada galaxia, incluyendo los atributos y las respuestas asociadas a cada clasificación. Por ejemplo, puede contener información sobre la forma de la galaxia, la presencia de características específicas, como brazos espirales o núcleos, y otras características relevantes.

El formato JSON es ampliamente utilizado para representar datos estructurados, lo que lo hace adecuado para almacenar y transmitir datos.

En nuestro caso, este archivo nos sirve para la creación de máscaras para cada una de las imágenes , necesarias para el proceso de segmentación de imágenes. De él se obtienen los puntos $x's$ e $y's$ que definen el límite de cada galaxia.

En la segmentación de imágenes, una máscara es una imagen que se utiliza para indicar las regiones de interés o los objetos de una imagen original. Consiste en asignar un valor a los píxeles que no forman parte del objeto de interés y otro distinto a los píxeles que pertenecen al objeto.

La máscara se utiliza como una guía para separar o aislar los objetos de interés del resto de la imagen en aplicaciones de segmentación. Al aplicar la máscara a la imagen original, se conservan únicamente los píxeles que corresponden al objeto, mientras que los píxeles fuera de la máscara se descartan o se consideran como fondo.

3.2. Modelos de clasificación

La clasificación de imágenes implica entrenar un modelo para que aprenda a distinguir y categorizar imágenes en diferentes clases predefinidas. El

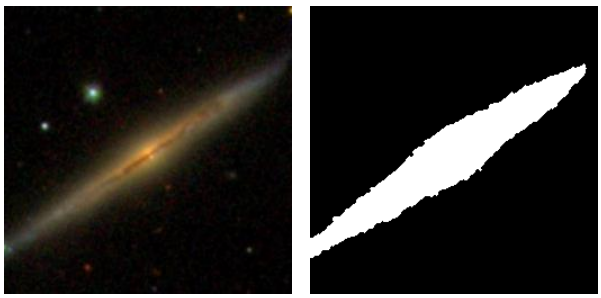

Figura 3.2: Ejemplo de imagen y su correspondiente máscara

objetivo es que el modelo pueda generalizar su conocimiento y ser capaz de clasificar correctamente nuevas imágenes que no haya visto durante el entrenamiento.

En esta sección, se presentan algunas de las técnicas y estructuras más avanzadas utilizadas en la actualidad para clasificar imágenes. Estas propuestas son aplicables en nuestro trabajo durante la fase de experimentación con un conjunto de datos de imágenes de galaxias.

3.2.1. AlexNet

El modelo AlexNet [14] se introdujo originalmente en el artículo Ïmage-Net Classification with Deep Convolutional Neural Networks". La arquitectura implementada es ligeramente diferente de la original, y se basa en un peculiar estrategia para llevar a cabo la paralelización de redes neuronales convolucionales.

Para desarrollar este modelo, se llevó a cabo el entrenamiento de una extensa red neuronal convolucional profunda con el propósito de clasificar las 1,3 millones de imágenes de alta resolución del conjunto de entrenamiento LSVRC-2010 ImageNet en 1.000 clases distintas. Durante las pruebas, se obtuvieron tasas de error del 39,7 % en el top 1 y del 18,9 % en el top 5.

La red neuronal, que cuenta con 60 millones de parámetros y 500.000 neuronas, se compone de cinco capas convolucionales, algunas de las cuales son seguidas por capas de agrupamiento máximo, junto con dos capas de conexión global y una función softmax final de 1.000 elementos. La función softmax es una función de activación empleada en la capa de salida, que transforma un vector de números reales en un vector de probabilidades normalizadas en un rango de 0 a 1, donde la suma de todas las probabilidades es igual a 1.

Con el fin de agilizar el proceso de entrenamiento, se utilizaron neuronas no saturadas y se implementó de manera altamente eficiente las redes convolucionales en la unidad de procesamiento gráfico (GPU). Para abordar el problema de sobreajuste en las capas de conexión global, se aplicó un nuevo método de regularización.

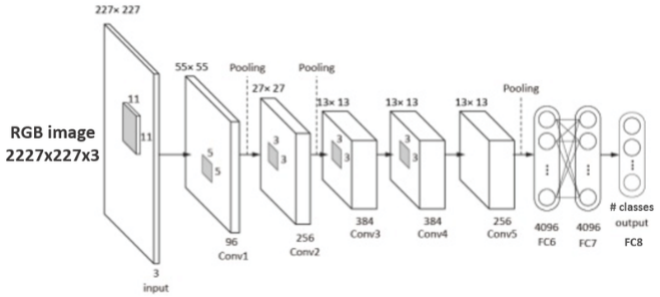

Figura 3.3: Arquitectura AlexNet con 5 capas convolucionales y 3 capas totalmente conectadas [12]

3.2.2. VGG

VGG, que significa Visual Geometry Group [24], es una arquitectura de red neuronal convolucional que se caracteriza por su simplicidad y uniformidad. Consiste en bloques de capas convolucionales seguidos de capas de agrupamiento máximo, lo que permite extraer características discriminativas de las imágenes en diferentes niveles de abstracción. A diferencia de otras arquitecturas, como AlexNet, VGG utiliza filtros convolucionales más pequeños (3x3) en todas sus capas, lo que proporciona una representación más profunda y rica de las imágenes.

VGG ofrece diferentes variantes, siendo VGG16 y VGG19 las más comunes. Estos números hacen referencia a la cantidad de capas en la arquitectura. VGG16 tiene 16 capas, incluyendo 13 capas convolucionales y 3 capas completamente conectadas, mientras que VGG19 tiene 19 capas en total. Ambas variantes son populares debido a su buen rendimiento.

Una característica importante de VGG es su capacidad para aprender representaciones de alto nivel y generalizables. Sin embargo, su principal desventaja radica en su gran cantidad de parámetros, lo que la convierte en una red profunda y costosa computacionalmente. Esto puede limitar su aplicación en dispositivos con recursos limitados.

A pesar de estas limitaciones, VGG ha sido ampliamente utilizado y ha servido como base para el desarrollo de arquitecturas más avanzadas, como

ResNet y DenseNet. [2]

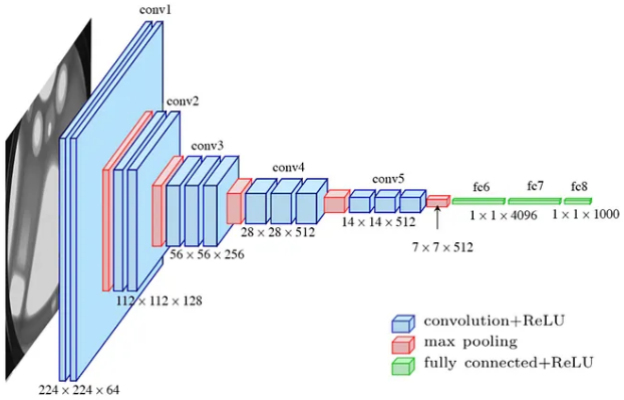

Figura 3.4: Arquitectura VGG16

3.2.3. DenseNet

Las redes convolucionales pueden ser sustancialmente más profundas, precisas y eficientes de entrenar si contienen conexiones más cortas entre las capas cercanas a la entrada y las cercanas a la salida. A raíz de esta idea, se introdujo la red convolucional densa (DenseNet) [9], que conecta cada capa con todas las demás de forma feed-forward, es decir, que el flujo de cómputo sigue una trayectoria lineal (capa de entrada - capas intermedias u ocultas - capa de salida).

En el caso de las redes convolucionales tradicionales se tienen L capas que tienen L conexiones (una entre cada capa y la siguiente), en el caso de DenseNet, esta arquitectura tiene L(L+1)/2 conexiones directas. Cada capa utiliza como entrada los mapas de características de todas las capas anteriores, y sus propios mapas de características se utilizan como entrada en todas las capas posteriores.

DenseNet tiene varias ventajas convincentes:

- Alivian el problema del gradiente decreciente.
- Refuerzan la propagación de características.
- Fomentan la reutilización de características.
- Reducen sustancialmente el número de parámetros.

[2]Imagen extraída de pub.towardsai.net

Al evaluar esta arquitectura propuesta en algunas de las tareas comunes de reconocimiento de objetos altamente competitivas (CIFAR-10, CIFAR-100, SVHN e ImageNet), las DenseNets obtienen mejoras significativas, a la vez que requieren menos computación para alcanzar un alto rendimiento. [3]

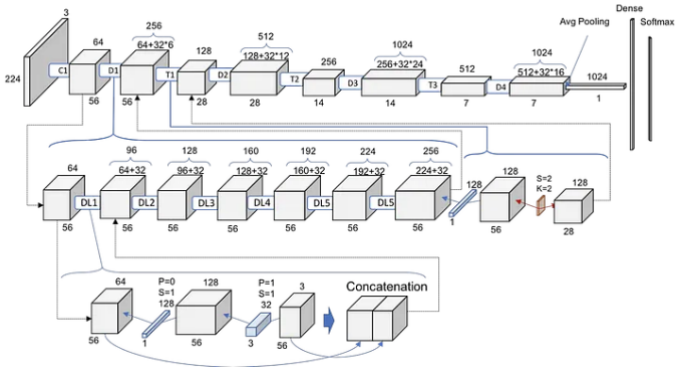

Figura 3.5: Representación esquemática de DenseNet

3.2.4. EfficientNet

EfficientNet [28] surge por un replanteamiento del escalado de modelos para redes neuronales convolucionales.

A diferencia de otras arquitecturas populares EfficientNet es una arquitectura de redes neuronales convolucionales, la cual utiliza un enfoque escalable que equilibra la profundidad, el ancho y la resolución de la red. En lugar de diseñar manualmente una arquitectura específica, se utiliza un método de optimización automática llamado escalado compuesto para encontrar la mejor configuración.

El escalado compuesto implica aumentar el ancho, la profundidad y la resolución de la red simultáneamente, manteniendo un equilibrio adecuado entre ellos. Esto permite que la red sea más profunda y más ancha, capturando características más complejas, mientras que la resolución más alta preserva detalles finos en las imágenes.

El enfoque de EfficientNet ha demostrado a pesar de su mayor capacidad, ser más eficiente en términos de uso de recursos computacionales, logrando un equilibrio entre eficiencia y rendimiento al utilizar el escalado compuesto.

[3] Imagen extraída de towardsdatascience.com

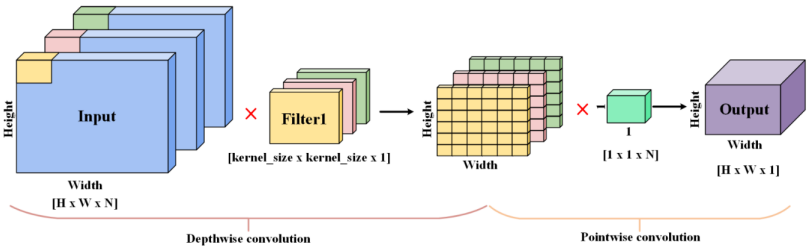

Figura 3.6: Arquitectura EfficientNet [8]

3.2.5. ResNet

ResNet, abreviatura de Residual Network [7], es una arquitectura de red neuronal convolucional ampliamente reconocida por su capacidad para entrenar redes extremadamente profundas sin sufrir degradación en el rendimiento, superando una limitación conocida como el problema de desvanecimiento de gradientes.

La innovación clave de ResNet radica en la introducción de conexiones residuales o conexiones de salto, que permiten que la información fluya directamente a través de la red sin pasar por varias capas convolucionales en cascada. Estas conexiones residuales permiten que las capas más profundas se beneficien directamente de la información y los gradientes de las capas anteriores, lo que facilita el entrenamiento de redes más profundas.

La arquitectura ResNet se compone de bloques residuales, que consisten en capas convolucionales seguidas de conexiones residuales. Estos bloques pueden ser apilados para crear redes más profundas. Las variantes populares de ResNet incluyen ResNet-50, ResNet-101 y ResNet-152, donde los números indican la cantidad de capas en la red.

ResNet ha demostrado un rendimiento sobresaliente y su capacidad para entrenar redes profundas ha influido en el diseño de otras arquitecturas posteriores. [4]

3.3. Modelos de segmentación

La segmentación de imágenes es una tarea que implica dividir una imagen en regiones o segmentos significativos. Estos segmentos pueden representar objetos, áreas de interés o diferentes partes de una imagen.

[4]Imagen extraída de https://pytorch.org/hub/pytorch_vision_resnet/

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figura 3.7: Arquitectura ResNet

Los modelos de segmentación desempeñan un papel crucial en el análisis de imágenes, ya que permiten extraer información detallada sobre la estructura y el contenido de una imagen. Al segmentar una imagen, es posible identificar y delimitar áreas específicas de interés, lo que facilita la comprensión y el procesamiento de la imagen en diversas aplicaciones.

3.3.1. DeepLabV3

DeepLabV3 [3] es un modelo de segmentación semántica diseñado para lograr una segmentación precisa y detallada de objetos en imágenes.

La arquitectura de DeepLabV3 se basa en redes neuronales convolucionales profundas, específicamente en la estructura de la red de convolución en cascada (CRF).

La convolución en cascada se refiere al modelo gráfico conocido como el campo aleatorio condicional (Conditional Random Field), que se utiliza comúnmente en el procesamiento de imágenes y la visión por computador para modelar las relaciones espaciales y contextuales entre píxeles o regiones vecinas en una imagen. Proporciona una forma de incorporar información de contexto adicional en la tarea de segmentación de imágenes.

Este modelo gráfico combina características de diferentes niveles de abstracción para obtener resultados más precisos. El modelo utiliza la red neuronal convolucional como extractor de características y emplea:

- ReLU (Rectified Linear Unit) [6]: es una función de activación ampliamente utilizada en redes neuronales artificiales y en particular en redes neuronales convolucionales (CNNs). Se utiliza para introducir no linealidad en los modelos y permitirles aprender representaciones más complejas y discriminativas de los datos.

La función ReLU se define de la siguiente manera: [5]

$$f(x) = max(0, x) \tag{3.1}$$

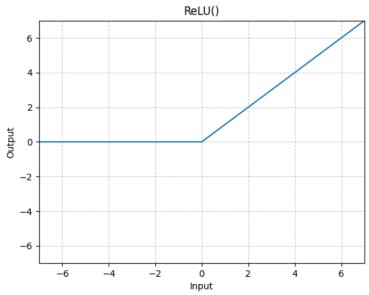

Figura 3.8: Función ReLU

En otras palabras, la función ReLU toma el valor máximo entre cero y el valor de entrada. Si el valor de entrada es positivo, la función ReLU devuelve el mismo valor. Si el valor de entrada es negativo, la función ReLU devuelve cero. Esto significa que la función ReLU .ªpaga"todas las entradas negativas, mientras que mantiene las entradas positivas sin cambios.

La principal ventaja de la función ReLU es su simplicidad computacional, ya que solo involucra una operación de comparación y una operación de asignación. Además, la función ReLU evita el problema del desvanecimiento de gradientes que puede ocurrir con funciones de activación como la función sigmoide.

Al utilizar la función ReLU como función de activación, las redes neuronales pueden aprender más rápido y ser más eficientes en términos computacionales. Además, la no linealidad introducida por la función ReLU permite a las redes neuronales aprender representaciones más complejas y capturar patrones más sutiles en los datos.

- Batch normalization [22] : es una técnica comúnmente utilizada en redes neuronales para mejorar el rendimiento y la estabilidad del entrenamiento. Se aplica a la capa de activación de una red neuronal, normalizando los valores de salida de la capa en función de las estadísticas calculadas en un lote de ejemplos de entrenamiento.

  Batch normalization se realiza de la siguiente manera:

  - Se calcula la media y la desviación estándar de los valores de activación dentro del lote de entrenamiento. Estas estadísticas se

[5] Imagen extraída de https://pytorch.org/docs/stable/generated/torch.nn.ReLU.html

calculan por canal de activación, es decir, para cada dimensión del tensor de activación.

- Los valores de activación se normalizan restando la media del lote y dividiendo por la desviación estándar. Esto centra y escala los valores de activación, lo que ayuda a estabilizar el entrenamiento y facilita la propagación de gradientes.
- Opcionalmente, se aplican dos parámetros adicionales, gamma y beta, para realizar una transformación lineal de los valores normalizados. Estos parámetros se aprenden durante el entrenamiento y permiten a la red ajustar la escala y el desplazamiento de los valores normalizados.

Batch normalization tiene varios beneficios. En primer lugar, ayuda a mitigar el problema de la covariación de características, donde los cambios en los valores de una característica pueden afectar a otras características en la red. Al normalizar los valores de activación, se reduce la dependencia de la red de la distribución de los datos y se mejora la capacidad de generalización.

Además, puede acelerar el entrenamiento de la red, ya que permite utilizar tasas de aprendizaje más altas y evita que los gradientes se vuelvan demasiado grandes o demasiado pequeños. Esto puede ayudar a superar los problemas de desvanecimiento o explosión de gradientes.

Tanto la función de activación ReLU y batch normalization mejoran la eficiencia y el rendimiento del modelo.

Una de las características clave de DeepLabV3 es el uso de la atrous (o dilated) convolución, que permite aumentar el tamaño del campo receptivo sin aumentar significativamente la cantidad de parámetros o el costo computacional. Esto es fundamental para capturar detalles finos y contextuales en la segmentación de objetos.

DeepLabV3 también incorpora la técnica de Atrous Spatial Pyramid Pooling (ASPP), que utiliza múltiples tasas de dilatación para capturar características a diferentes escalas espaciales. Esto permite que el modelo comprenda y segmente objetos de diferentes tamaños y formas de manera más efectiva.

Además, utiliza una estrategia de fusión de características en múltiples escalas para combinar información de diferentes niveles de la red y obtener una representación más completa y precisa de los objetos en la imagen.

Este modelo ha logrado un rendimiento destacado en desafíos de seg-

mentación de imágenes. [6]

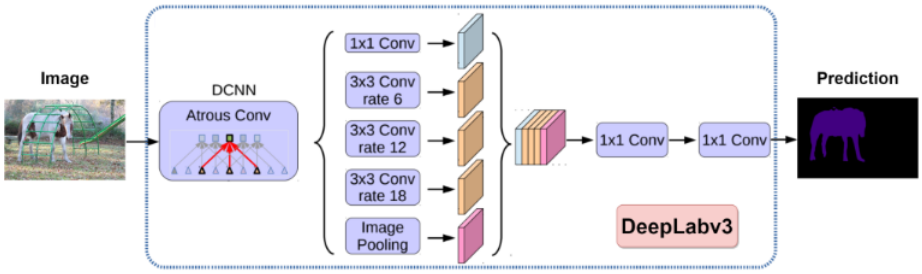

Figura 3.9: Modelo de segmentación semántica DeepLabV3

3.4. GANs

Las GAN (Generative Adversarial Networks) [19] son un tipo de arquitectura de redes neuronales que consisten en dos componentes principales: el generador y el discriminador. Estas redes se utilizan en tareas de generación de imágenes.

El generador tiene la tarea de crear muestras sintéticas que se asemejen a los datos de entrenamiento. Por otro lado, el discriminador es responsable de distinguir entre las muestras generadas por el generador y las muestras reales del conjunto de entrenamiento.

Este tipo de redes funcionan de la siguiente manera:

- Un vector de características inicial (normalmente ruido aleatorio) se introduce como entrada a un generador, que genera un contenido falso.
- El contenido falso se mezcla con contenido real, y todo el conjunto de datos se proporciona como entrada al discriminador, que clasifica de forma binaria qué contenido es real y qué contenido es falso.
- Durante el entrenamiento, el generador intenta mejorar su capacidad para engañar al discriminador, generando muestras cada vez más realistas. Por su parte, el discriminador se entrena para ser capaz de distinguir con precisión entre las muestras generadas y las reales.

La interacción entre el generador y el discriminador en una competencia adversarial es lo que permite que las GAN generen muestras de alta calidad

[6] Imagen extraída de https://learnopencv.com/deeplabv3-ultimate-guide/

y sean capaces de capturar las características y la distribución de los datos de entrenamiento. [7]

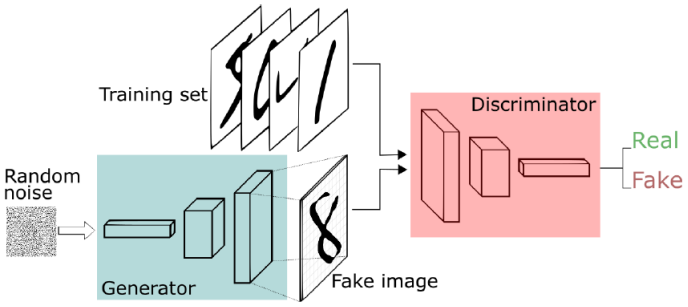

Figura 3.10: Funcionamiento de GANs

3.4.1. Real-ESRGAN

Real-ESRGAN (Enhanced Super-Resolution Generative Adversarial Networks) [29] es un método avanzado de súper resolución basado en redes generativas adversariales (GANs) que se utiliza para aumentar la resolución de imágenes de baja calidad o baja resolución.

Real-ESRGAN utiliza un generador de red neuronal profunda que se entrena utilizando un conjunto de datos sintéticos generado de forma pura, en lugar de depender únicamente de datos reales de baja resolución. Esto permite superar las limitaciones de los conjuntos de datos reales y generar imágenes de alta resolución más realistas y detalladas.

[7] Imagen extraída Agencia Digital de Andalucía. *Experto en Inteligencia Artificial*, 2023.

Capítulo 4

Pasos Previos

En el presente capítulo, se aborda el estudio llevado a cabo en este trabajo, con el objetivo principal de investigar y analizar el rendimiento de las técnicas comentadas en el capítulo anterior, evaluando su eficacia y precisión.

4.1. Clasificación

A continuación se va a brindar información sobre la capacidad de las distintas técnicas utilizadas para clasificar galaxias.

4.1.1. Preprocesado

Antes de iniciar el proceso de entrenamiento, es necesario realizar una etapa de preprocesamiento en el conjunto de datos con el fin de mejorar su precisión y asegurar un entrenamiento óptimo.

Organización en carpetas

En primer lugar, se elige un conjunto específico de imágenes con el que se llevará a cabo el análisis. Estas imágenes se organizan en carpetas según su propósito y tipo, ya sea para entrenamiento, validación o pruebas, y se clasifican en categorías de galaxias elípticas, espirales o de clasificación incierta.

Siendo la estructura de carpetas la siguiente:

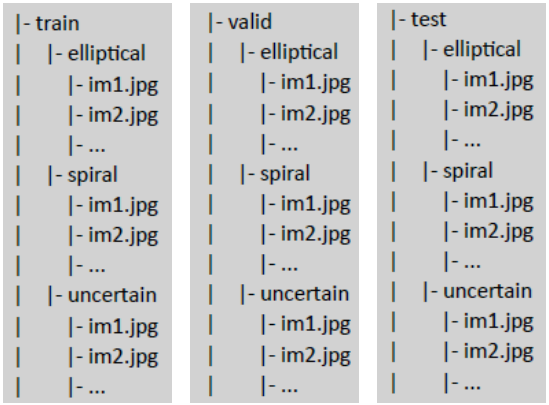

Figura 4.1: Organización de las imágenes en carpetas en clasificación

Preprocesado de imágenes de entrenamiento

En las imágenes de entrenamiento se realizarán una serie de transformaciones, aplicando múltiples cambios de manera secuencial a cada una de las imágenes.

Mediante la biblioteca de torchvision en Pytorch se hará uso de diferentes funciones para llevar a cabo una serie de transformaciones. Estas funciones son las siguientes: [1]

- $transforms.Resize()$ redimensiona el tamaño de una imagen de acuerdo a unas dimensiones deseadas.
- $transforms.RandomHorizontalFlip()$ realiza un volteo horizontal en algunas de las imágenes, sin embargo, la ejecución de este volteo o no, dependerá de un porcentaje de probabilidad que se indique.

  Estos giros horizontales introducen variaciones artificiales en el conjunto de datos que pueden ayudar a mejorar la capacidad de modelo para generalizar y reconocer patrones en imágenes con orientaciones diferentes. Esta transformación puede tener un efecto significativo en la apariencia y la interpretación de la imagen, especialmente cuando hay información direccional o simetría en la imagen original.
- $transforms.RandomVerticalFlip()$ realiza un volteo vertical en algunas de las imágenes. Esta transformación funciona de la misma manera

[1] `https://pytorch.org/vision/main/transforms.html`

que el giro horizontal explicado en el punto anterior, a diferencia de la orientación.

- $transforms.GaussianBlur()$ crea un efecto de desenfoque en la imagen utilizando un filtro de desenfoque gaussiano. A esta función se le pasan dos parámetros, $kernel\_size$ y $sigma$, que determinan la configuración del desenfoque.

  - $kernel\_size$ especifica el tamaño del kernel del filtro de desenfoque gaussiano. Puede ser una tupla (H, W) que indica la altura y el ancho del kernel.
  - $sigma$ determina la desviación estándar del kernel gaussiano. Puede ser una tupla (min_sigma, max_sigma) que establece el rango de desviación estándar permitido. Cuanto mayor sea el valor de sigma, mayor será el efecto de desenfoque.

  Al aplicar esta función con los parámetros especificados, se suavizará la imagen y se reducirá la nitidez de los bordes y los detalles. Esta transformación puede ser útil para reducir el ruido en imágenes o suavizar detalles innecesarios.

- $transforms.RandomRotation()$ realiza una rotación aleatoria a la imagen dentro de un rango de grados especificado. Este rango se determinado con el parámetro $degrees$ que indica el rango de grados de rotación permitido, siendo una tupla $(min\_degrees, max\_degrees)$ que establece el rango de rotación aleatoria. Esta transformación puede aumentar la diversidad del conjunto de datos y ayudar al modelo a aprender a reconocer objetos o patrones desde diferentes ángulos.

  Sin embargo, es importante tener en cuenta que la rotación puede causar la pérdida de información en los bordes de la imagen o introducir distorsiones dependiendo del grado de rotación. Por lo tanto, es fundamental elegir un rango de grados apropiado para garantizar que la información relevante se conserve y el efecto de la transformación sea beneficioso para la tarea en cuestión.

- $transforms.ToTensor()$ convierte la imagen en un tensor multidimensional. En particular, se realiza la conversión de los valores de los píxeles de la imagen a valores numéricos en el rango de 0 a 1. Además, el orden de las dimensiones de la imagen se reorganiza para que sea compatible con el formato utilizado en PyTorch, es decir, el tensor resultante tendrá la forma (C, H, W), donde C representa los canales de color, H la altura y W el ancho de la imagen.

- $transforms.Normalize()$ ajusta los valores de los píxeles en el tensor de imagen para que sigan una distribución normalizada. Esta nor-

malización se realiza utilizando los parámetros *mean* (media) y *std* (desviación estándar) especificados.

- *mean* especifica la media de normalización para cada canal de color.
- *std* especifica la desviación estándar de normalización para cada canal de color.

La normalización de los valores de píxeles es importante para asegurar que los datos estén en una escala adecuada para el modelo de aprendizaje. Al normalizar los datos, se logra que los valores estén centrados alrededor de cero y tengan una dispersión comparable entre los canales de color.

Esta transformación se aplica por norma general después de utilizar $transforms.ToTensor()$, ya que se espera que los datos estén en forma de tensores para poder normalizarlos adecuadamente.

Preprocesado de imágenes de validación

Se aplicarán ciertas transformaciones a las imágenes del conjunto de validación. Sin embargo, estas transformaciones serán más suaves o menos intensas en comparación con las aplicadas al conjunto de entrenamiento.

El propósito de esto es evaluar el rendimiento de un modelo de aprendizaje mientras se entrena. Esto permite verificar cómo el modelo se desempeña en imágenes que son similares, pero no idénticas, a las que se utilizan para entrenarlo. Al tener transformaciones menos intensas en el conjunto de validación, se busca que la evaluación sea más representativa de la capacidad de generalización del modelo.

Las transformaciones llevadas a cabo en este conjunto de datos son:

- $transforms.Resize()$
- $transforms.ToTensor()$
- $transforms.Normalize()$

4.1.2. Entrenamiento

El entrenamiento consiste en ajustar los parámetros de un modelo de manera iterativa utilizando un conjunto de datos de entrenamiento. El objetivo

del entrenamiento es permitir que el modelo aprenda patrones, relaciones y reglas subyacentes en los datos, de modo que pueda realizar predicciones o tomar decisiones adecuadas en datos no vistos previamente.

Durante el entrenamiento, el modelo se expone a ejemplos del conjunto de datos de entrenamiento, que consisten en entradas y las salidas esperadas correspondientes. El modelo realiza predicciones iniciales y se compara con las salidas esperadas para medir el error o la diferencia entre las predicciones y los valores reales. Luego, se utiliza una técnica de optimización para ajustar los parámetros del modelo con el objetivo de minimizar el error.

Información previa

Antes de profundizar en el entrenamiento, se van a describir unos conceptos que facilitarán la comprensión de proceso:

- Descenso de gradiente [2], también conocido como el método del gradiente descendente, es un algoritmo de optimización utilizado para ajustar los parámetros de un modelo de manera iterativa, con el objetivo de minimizar una función de pérdida.

  El concepto básico detrás del descenso de gradiente es encontrar los valores de los parámetros del modelo que minimizan la función de pérdida, que mide la discrepancia entre las predicciones del modelo y los valores reales de los datos de entrenamiento. Para lograr esto, el algoritmo utiliza la información del gradiente de la función de pérdida con respecto a los parámetros del modelo.

  En cada iteración del descenso de gradiente, se calcula el gradiente de la función de pérdida con respecto a los parámetros del modelo. El gradiente indica la dirección y magnitud del cambio necesario para reducir la pérdida.

  El proceso se repite iterativamente hasta que se alcance un criterio de convergencia, como un número máximo de iteraciones o cuando la mejora en la función de pérdida es mínima. De esta manera, el descenso de gradiente busca encontrar los valores de los parámetros que minimizan la función de pérdida y, por lo tanto, mejoren el rendimiento del modelo en la tarea de aprendizaje.

  Existen diferentes variantes del descenso de gradiente, como el descenso de gradiente estocástico (SGD), que utiliza una muestra aleatoria de los datos de entrenamiento en cada iteración para acelerar el proceso.

$$\Phi = \Phi - \alpha * \bigtriangledown J(\Phi) \tag{4.1}$$

Donde Φ representa el vector de parámetros del modelo que queremos ajustar, α es la tasa de aprendizaje (learning rate) que controla el tamaño de los pasos que se toman durante la actualización de los parámetros y $\bigtriangledown J(\Phi)$ es el gradiente de la función de pérdida J con respecto a los parámetros Φ. El gradiente indica la dirección y magnitud del cambio necesario para reducir la pérdida.

- Learning rate o tasa de aprendizaje [25], es un parámetro utilizado en algoritmos de aprendizaje, como el descenso del gradiente, que controla la magnitud de los ajustes que se realizan en los pesos de un modelo durante el proceso de entrenamiento.

  En el contexto del descenso del gradiente, el learning rate determina cuánto se deben actualizar los pesos del modelo en función del gradiente de la función de pérdida. El gradiente indica la dirección y magnitud del cambio necesario para minimizar la función de pérdida y ajustar el modelo a los datos de entrenamiento.

  Un learning rate bajo implica ajustes pequeños en los pesos durante cada actualización, lo que puede llevar a un proceso de entrenamiento más lento, pero también puede ayudar a evitar que el modelo se salte los mínimos locales en la función de pérdida. Por otro lado, un learning rate alto permite ajustes más grandes y puede conducir a un entrenamiento más rápido, pero también existe el riesgo de que el modelo no converja o se salte el mínimo global de la función de pérdida. [2]

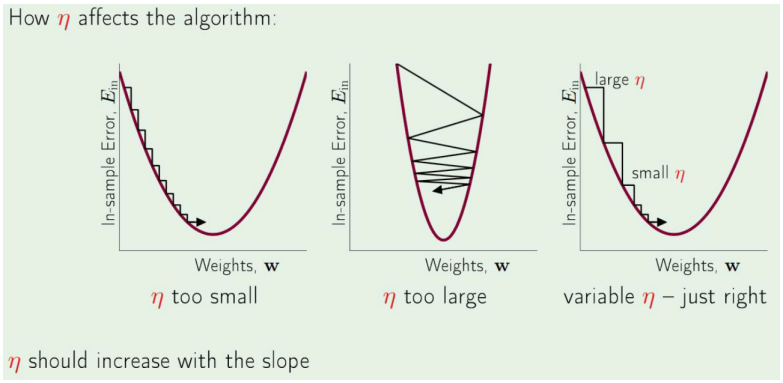

Figura 4.2: Cómo afecta el learning rate al algoritmo

- Época es una pasada completa de todos los ejemplos de entrenamiento a través del modelo durante el proceso de entrenamiento.

  Durante una época, cada ejemplo de entrenamiento alimenta al modelo, y se calcula la salida del modelo y se compara con la salida esperada

[2] Imagen extraída de work.caltech.edu

para determinar la pérdida o el error. Luego, se utilizan algoritmos de optimización, como el descenso de gradiente, para ajustar los parámetros del modelo con el objetivo de reducir la pérdida.

El número de épocas que se elige para entrenar un modelo depende del tamaño del conjunto de entrenamiento, la complejidad del modelo y otros factores. En general, es común iterar a través del conjunto de entrenamiento varias veces (es decir, varias épocas) para permitir que el modelo aprenda de manera más completa y logre una mayor precisión en las predicciones.

Es importante tener en cuenta que el uso de múltiples épocas también puede llevar al sobreajuste si el modelo se vuelve demasiado específico para los datos de entrenamiento y no generaliza bien a nuevos datos. Por lo tanto, encontrar un equilibrio adecuado en el número de épocas es importante para obtener un modelo con buen rendimiento.

- Tamaño del batch [11] es un parámetro que determina cuántos ejemplos de entrenamiento se utilizan en cada iteración del algoritmo de optimización.

  Durante el entrenamiento, los datos de entrenamiento se dividen en grupos, y se procesa un grupo a la vez. Cada grupo consiste en un número específico de ejemplos de entrenamiento, determinado por el batch size. Por ejemplo, si el batch size es 64, se toman 64 ejemplos de entrenamiento y se utilizan para calcular las salidas del modelo y realizar actualizaciones de los parámetros.

  El uso de grupos en el entrenamiento tiene varios beneficios: permite el entrenamiento en paralelo, ya que los cálculos para cada grupo se pueden realizar simultáneamente en hardware moderno y el uso de grupos permite un uso más eficiente de la memoria, ya que solo se requiere almacenar un grupo en lugar de todo el conjunto de entrenamiento.

  El tamaño del batch puede tener un impacto en el rendimiento y la estabilidad del entrenamiento. Un tamaño pequeño puede proporcionar actualizaciones de parámetros más frecuentes y un entrenamiento más rápido, pero también puede introducir ruido en las actualizaciones debido a la variabilidad en uno pequeño. Por otro lado, un tamaño grande puede proporcionar estimaciones de gradiente más estables y suaves, pero también puede requerir más memoria y puede llevar a un entrenamiento más lento.

Una vez explicados estos conceptos, el proceso de ajuste iterativo en el entrenamiento se repite múltiples veces. Para iniciar el entrenamiento se necesita:

- Modelo usado: son cada una de las técnicas o modelos explicados en el apartado 3.2.
- Conjunto de datos de entrenamiento.
- Criterio de optimización: es una medida que se utiliza en el aprendizaje para evaluar la calidad del modelo y guiar el proceso de optimización durante el entrenamiento.

  En este caso, se usa $Adam(AdaptiveMomentEstimation)$ [13] , que combina conceptos de los algoritmos de descenso del gradiente estocástico (SGD) y de los métodos adaptativos de estimación de momentos. Su objetivo es ajustar los pesos del modelo de manera eficiente y precisa durante el proceso de entrenamiento.

  El algoritmo Adam utiliza la información acumulada de los momentos de primer orden (media de los gradientes) y de segundo orden (varianza de los gradientes) para adaptar la tasa de aprendizaje de cada parámetro del modelo. Esto significa que Adam calcula y mantiene estimaciones separadas del momento de primer orden (media móvil de los gradientes) y del momento de segundo orden (media móvil de los gradientes al cuadrado) para cada parámetro.

  En cada iteración del entrenamiento, Adam actualiza los parámetros utilizando una combinación de los momentos acumulados y el gradiente actual. Esto permite que el algoritmo se adapte automáticamente a diferentes tasas de aprendizaje para cada parámetro, lo que facilita la convergencia más rápida y estable.

  Además, Adam incorpora un mecanismo de corrección que ajusta los momentos acumulados en las primeras etapas del entrenamiento cuando hay una estimación inexacta debido a la inicialización aleatoria de los parámetros. Esto ayuda a mejorar el rendimiento y la estabilidad del algoritmo.

- Función de pérdida: es una medida que cuantifica la discrepancia entre las predicciones realizadas por un modelo de aprendizaje automático y los valores reales de los datos de entrenamiento.

  El objetivo principal de la función de pérdida es proporcionar una medida cuantitativa de cómo de bien está realizando el modelo la tarea de aprendizaje específica. En otras palabras, cuantifica cómo de lejos están las predicciones del modelo de los valores reales y brinda una señal de retroalimentación para ajustar los parámetros del modelo durante el proceso de entrenamiento.

  La elección de la función de pérdida depende del tipo de problema y del objetivo del modelo. Para este caso, se va a usar la pérdida de entropía cruzada (Cross Entropy Loss) [21] que mide la discrepancia entre la

distribución de probabilidad predicha por el modelo y la distribución de probabilidad real de las etiquetas de clase. Se calcula mediante la aplicación de la función logaritmo negativo a la probabilidad predicha para la clase verdadera.

Esta función se aplica a todos los ejemplos de entrenamiento de un batch y se calcula el promedio de las pérdidas individuales para obtener la pérdida total del batch.

El objetivo del entrenamiento es minimizar la pérdida de entropía cruzada, lo que implica que el modelo esté aprendiendo a generar probabilidades más cercanas a las distribuciones reales de las etiquetas de clase. [3]

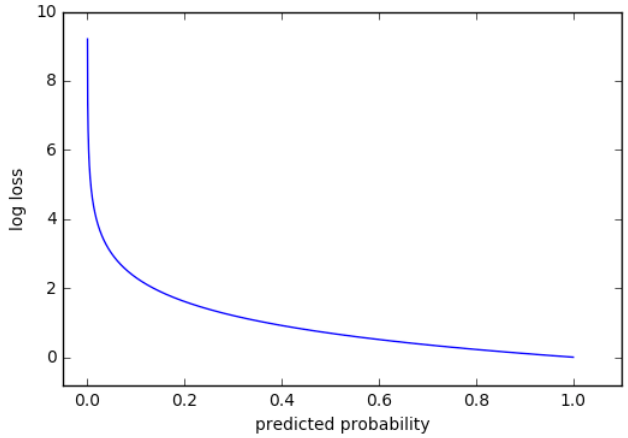

Figura 4.3: Ejemplo de un modelo perfecto con pérdida logarítmica de 0

Selección de arquitectura

En este apartado, se presentarán las pruebas realizadas para evaluar el desempeño y la eficacia de los distintos modelos de clasificación propuestos.

Durante las pruebas, se ejecutan diferentes casos de prueba, donde se introducen datos de entrada y se analizan los resultados obtenidos, comparando unos modelos con otros para evaluar el rendimiento del sistema en términos de precisión y eficiencia.

Este análisis será utilizado para mejorar el modelo, realizando ajustes y correcciones necesarias. También se incluyen gráficos y tablas para ayudar a presentar los resultados de manera clara y concisa.

[3]Imagen extraída de ml-cheatsheet.readthedocs.io

En primer lugar, se escogió un conjunto de imágenes con el que trabajar. Dicho conjunto se seleccionó de manera que estuviera balanceado, es decir, que el conjunto de imágenes de las distintas clases fuera el mismo o similar.

En este caso se seleccionaron un total de 3.600 imágenes , de las cuáles:

- 3.000 imágenes serán para entrenamiento, 1.000 de cada clase (elliptical, spiral y uncertain).
- 300 imágenes para validación, 100 de cada clase (elliptical, spiral y uncertain).
- 300 imágenes para test, 100 de cada clase (elliptical, spiral y uncertain).

Después de haber elegido estas imágenes, se llevaron a cabo múltiples experimentos para evaluar los modelos mencionados en la sección 3.2. El entrenamiento se realizó con el tamaño de batch más favorable para cada uno de los distintos modelos (16, 32, 64...) y un número de épocas común en todos ellos, 20.

Para cada uno de ellos se obtuvieron dos gráficas, una que representa como evoluciona la precisión (cuánto más alta sea mejor será el modelo) a medida que avanza el entrenamiento y otra en la que se observa como varía la pérdida (cuánto más pequeño sea este valor mejor será el modelo) durante este proceso.

- **AlexNet**

Usando AlexNet se observan los siguientes resultados:

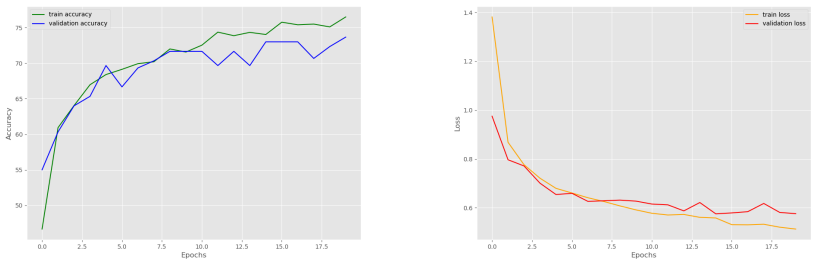

Figura 4.4: Gráficos evolutivos de *accuracy* y *loss* de AlexNet

Según los resultados observados, en cuanto a la representación gráfica de la precisión, se puede apreciar un aumento gradual a medida que se incrementa el número de épocas. Durante las primeras etapas se produce un incremento sustancial en esta medida, que deja de ser tan pronunciado a medida que aumenta el número de épocas. Con respecto a la diferencia entre los valores de entrenamiento y validación, permanecen muy similares hasta la mitad del proceso y a partir de este punto los valores de validación se estabilizan.

En relación a la gráfica de pérdida, su comportamiento es similar al de la gráfica de precisión, con la diferencia de que en este caso el objetivo es minimizar dicho valor. Inicialmente, se observa una disminución pronunciada de la pérdida, y poca disparidad entre los conjuntos a excepción del primera época entre los conjuntos de entrenamiento y validación. A medida que se avanza en el número de épocas, el decrecimiento de la pérdida no es tan marcado.

Una vez completado el entrenamiento, se procedió al análisis de la precisión en test, obteniendo un valor de precisión del 75.67 %.

- **VGG**

Para VGG, se ha utilizado la arquitectura VGG-19 con batch normalization, obteniendo las gráficas:

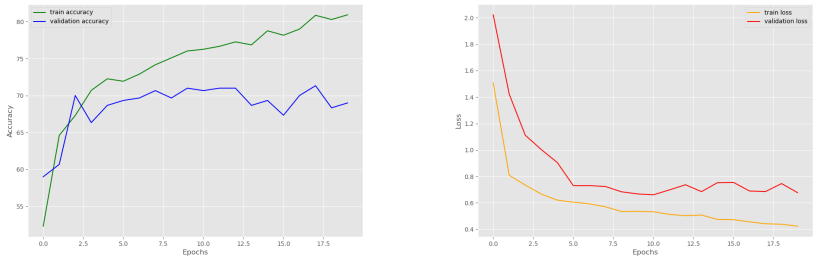

Figura 4.5: Gráficos evolutivos de *accuracy* y *loss* de VGG

En relación a la representación gráfica de *accuracy*, se puede notar que el crecimiento ocurre de forma más uniforme en comparación con AlexNet, aunque es ligeramente mayor durante las tres primeras épocas. Al comparar los valores de entrenamiento y validación, al inicio del entrenamiento son similares, pero después de 5 épocas, el *accuracy* de validación se estabiliza, mientras que la de entrenamiento continúa aumentando, mostrando una diferencia notable al final del proceso. Esto puede deberse a un problema de

sobreentrenamiento, ya que conseguimos mejorar el valor de entrenamiento pero no el de validación, lo que puede llevar a problemas de generalización.

En cuanto a la gráfica de *loss*, en el caso del conjunto de entrenamiento, se observa un descenso significativo en las épocas iniciales, seguido de un decrecimiento constante. Por otro lado, en el conjunto de validación, el valor de *loss* disminuye hasta aproximadamente la octava época, momento en el cual se estabiliza.

Tras completar el entrenamiento del modelo, se evaluó su rendimiento en el conjunto de test, obteniendo una precisión del 70.67 %.

- **DenseNet**

En el caso de DenseNet, se ha escogido DenseNet-201, del que se ha obtenido lo siguiente:

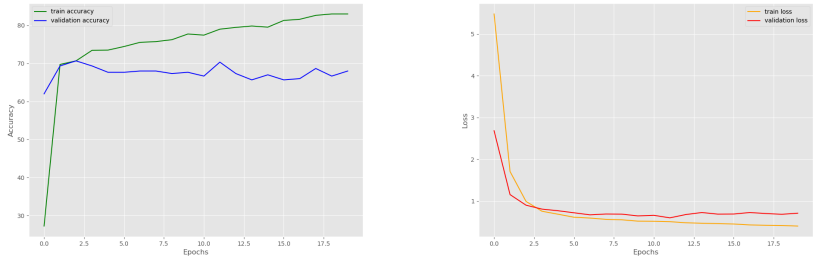

Figura 4.6: Gráficos evolutivos de *accuracy* y *loss* de DenseNet

Si analizamos el gráfico de precisión, notamos una diferencia en comparación con AlexNet y VGG al inicio del entrenamiento. En este caso, el modelo logra un ajuste rápido al comienzo del entrenamiento, alcanzando valores en torno al 70.00 % en las primeras épocas. A partir de ese momento, el incremento en la precisión no es tan notable. Cabe destacar que los valores de validación son altos desde el comienzo del entrenamiento, lo que nos indica que es posible que para esta arquitectura sería suficiente realizar unas pocas épocas, por lo que el resto de épocas sobrarían para evitar lo que ocurre con VGG, que se produzca sobreajuste.

La gráfica de pérdida muestra un descenso de los valores muy considerable al inicio del entrenamiento, estabilizándose pasadas unas pocas épocas. Además, los valores de entrenamiento y validación casi no difieren en casi todo el entrenamiento.

Finalmente, el resultado en la evaluación de test fue del 71.33 %.

- **EfficientNet**

Con EfficientNet se utilizó la arquitectura EfficientNet-B7, obteniendo los siguiente gráficos evolutivos:

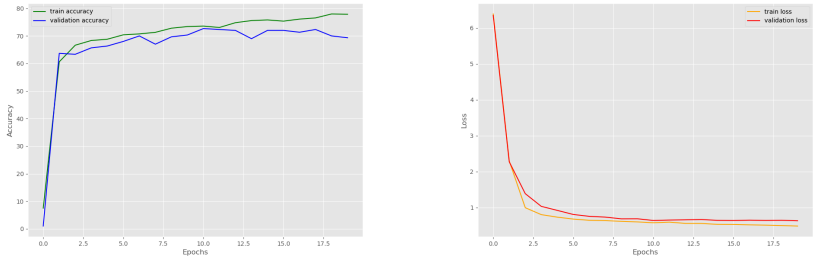

Figura 4.7: Gráficos evolutivos de *accuracy* y *loss* de EfficientNet

Al comparar el gráfico de *accuracy* de este modelo con el de DenseNet, se puede observar una similitud notable en los valores iniciales. Sin embargo, la diferencia significativa radica en que, en este caso, el valor de entrenamiento y validación es muy similar desde el inicio del entrenamiento, sin excepción en ninguna época.

En cuanto al gráfico de *loss*, al igual que en el de *accuracy*, los valores de entrenamiento y validación son casi idénticos durante todo el proceso, lo que nos induce a pensar que el modelo está logrando una generalización efectiva. Se observa una disminución significativa al comienzo del entrenamiento, pero este descenso no es tan pronunciado a partir de la tercera época.

El resultado obtenido en la evaluación de test para este modelo fue de un 72.33 %.

- **ResNet**

Finalmente, en el caso de ResNet se ha hecho uso de ResNet-50, del que se observa las siguientes gráficas:

Si nos referimos a la gráfica de *accuracy*, se aprecia un gran aumento en las primeras épocas a partir de las cuáles este aumento no es tan notable y sí más constante. Al igual que en EfficientNet la generalización es óptima, esto se aprecia en que apenas existen diferencias entre entrenamiento y validación.

Similarmente, en el gráfico de pérdida, se observa una disminución drástica tanto en los valores de entrenamiento como de validación al comienzo del

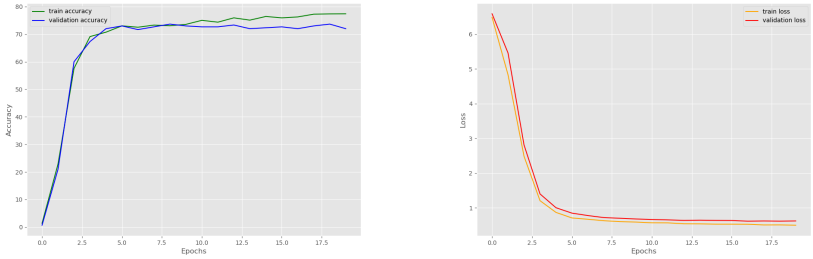

Figura 4.8: Gráficos evolutivos de *accuracy* y *loss* de ResNet

proceso, seguida de una estabilización. Los valores de entrenamiento y validación son prácticamente idénticos a lo largo de las 20 épocas.

El modelo creado con esta arquitectura obtuvo un resultado de test con una precisión del 76.33 %.

- **Comparación entre modelos**

Finalizados los entrenamientos con las diferentes arquitecturas, a continuación se resumen los valores de test obtenidos de cada uno de ellos en la siguiente tabla:

Modelo	Acierto de Test
AlexNet	75.67 %
VGG	70.67 %
DenseNet	71.33 %
EfficientNet	72.33 %
ResNet	76.33 %

Cuadro 4.1: Comparación de Test entre modelos

A la vista de estos resultados, el modelo con mejor porcentaje de precisión es ResNet con un 76.33 %.

Aumento del conjunto de imágenes

Una vez alcanzado este resultado y con el propósito de evaluar si el modelo ha adquirido conocimientos adecuados o requiere una mayor cantidad de datos para mejorar su aprendizaje, se ha llevado a cabo un experimento consistente en duplicar la cantidad de imágenes, siendo en el caso de las imágenes de entrenamiento, 6.000.

El resultado obtenido de este experimento es el siguiente:

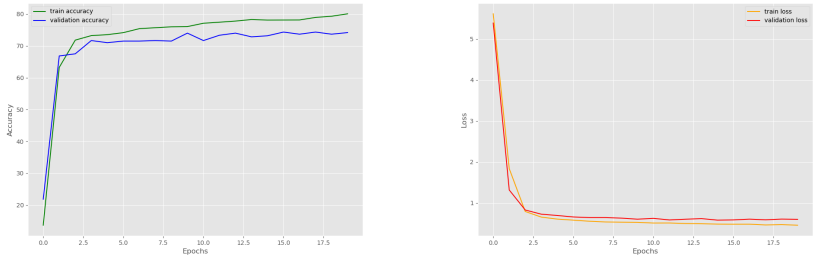

Figura 4.9: Gráficos evolutivos de *accuracy* y *loss* de ResNet con el doble de imágenes

Si comparamos estas gráficas con las obtenidas con el entrenamiento de 3.000 imágenes 4.1.2, se observa que el comportamiento es prácticamente idéntico, al igual que sus valores de precisión y pérdida, tanto en entrenamiento y validación.

En vista de los resultados obtenidos, el modelo anterior había aprendido suficiente y no es necesario trabajar con doble número de imágenes para mejorarlo.

4.1.3. Pruebas con imágenes

Seleccionado el modelo de clasificación con el que se va a trabajar (ResNet), vamos a intentar mejorar este porcentaje mediante el tratamiento de imágenes.

En primer lugar, vamos a realizar una comparativa entre trabajar con las imágenes originales y las imágenes recortadas, para lo que vamos a hacer uso de su *bounding box*, que es el rectángulo que se utiliza para representar y delimitar la ubicación y extensión de un objeto (en este caso, una galaxia) dentro de una imagen o un espacio tridimensional.

El *bounding box* está definido por las coordenadas de su esquina superior izquierda (x, y) y su esquina inferior derecha (x', y'). Estas coordenadas permiten identificar la región exacta donde se encuentra el objeto de interés dentro de la imagen o el espacio tridimensional.

Recorte de imágenes

Para llevar a cabo este recorte, utilizaremos el archivo JSON mencionado en la sección correspondiente (ver sección 3.1.3). Este archivo nos proporcionará los límites de la ubicación de la galaxia, como hemos explicado previamente en la definición de *bounding box*.

Posteriormente, con el objetivo de eliminar ruido de fondo que pueda causar aquello que rodea a la galaxia, usaremos los límites comentados para crear una imagen de color negro sobre la cual posteriormente colocaremos la galaxia. Los límites del *bounding box* y la propia galaxia los extraeremos de los puntos contenidos en el archivo JSON.

Para extraer el polígono que forma la galaxia, emplearemos la función *pointPolygonTest()*. Esta función nos indicará si un punto de una imagen se encuentra dentro, fuera o en el contorno del polígono formado por los puntos que se le pasan como argumento.

Un ejemplo del proceso llevado a cabo es el siguiente:

Imagen original → Imagen recortada → Imagen final

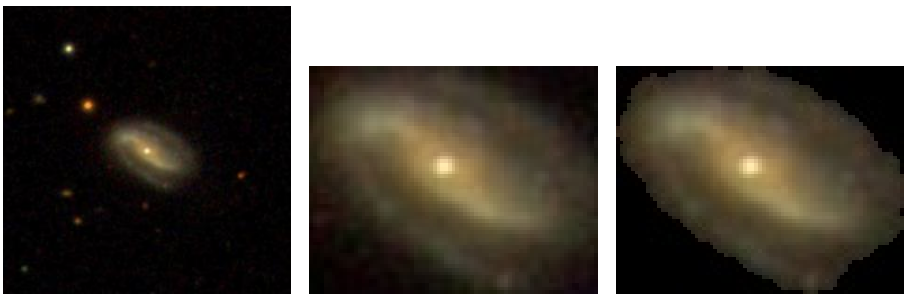

Figura 4.10: Secuencia de recorte

Realizado el recorte, se eliminaron aquellas imágenes cuyas dimensiones de imágenes fueran inferiores a 225 píxeles para que éstas no pudieran llevar a confusión en la creación del modelo.

Establecido el conjunto de imágenes recortadas, pasamos a crear los modelos combinando de todas las maneras posibles los dos conjuntos de imágenes (originales y recortadas) para ver si de esta manera se consigue mejorar el modelo. Esta comparativa nos lleva a crear 4 modelos distintos entrenados de la siguiente manera:

1. Las imágenes de entrenamiento, validación y test serán las originales

(modelo creado anteriormente 4.1.2).

2. Las imágenes de entrenamiento serán las originales, y las de validación y test serán las recortadas.

   Las gráficas resultantes son:

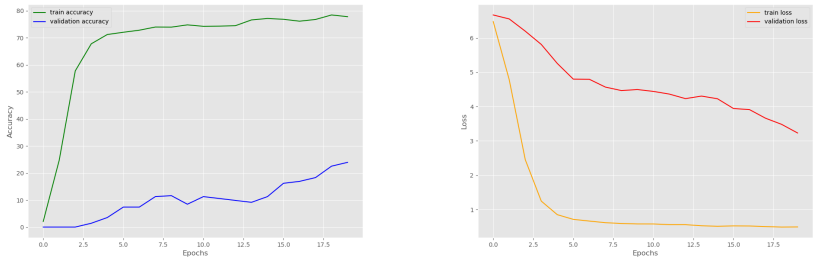

Figura 4.11: Gráficos evolutivos de *accuracy* y *loss* originales - recortadas

   Al examinar los gráficos, se puede notar claramente que los resultados son altamente desfavorables, lo cual era de esperar. Esto se debe a que al entrenar con las imágenes originales, las predicciones realizadas en base a las imágenes recortadas carecen de precisión significativa. Esto ocurre porque una imagen no guarda relación alguna con la otra, lo cual contribuye a los pobres resultados obtenidos.

   Con este modelo se obtiene un resultado en test del 21.31 %.

3. Las imágenes de entrenamiento serán las recortadas, y las de validación y test serán las originales.

   De esta prueba, el resultado es el siguiente:

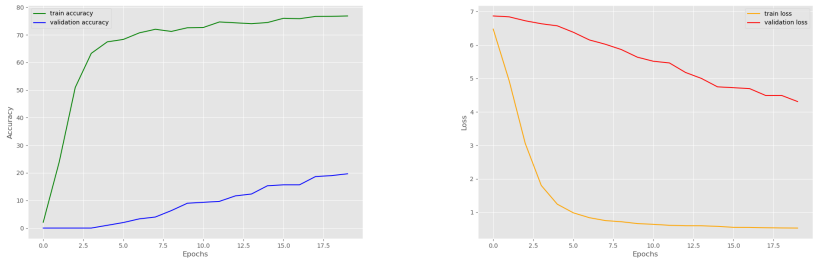

Figura 4.12: Gráficos evolutivos de *accuracy* y *loss* recortadas - originales

   Al igual que en el caso anterior, los resultados no son buenos. Esto ocurre porque se ha entrenado con imágenes recortadas pero se han evaluado imágenes originales, siguiendo la misma lógica que en el caso 2.

Para este modelo entrenado, obtenemos un porcentaje de acierto en test del 20.00 %.

4. Las imágenes de entrenamiento, validación y test serán las recortadas.

   Del entrenamiento se obtiene:

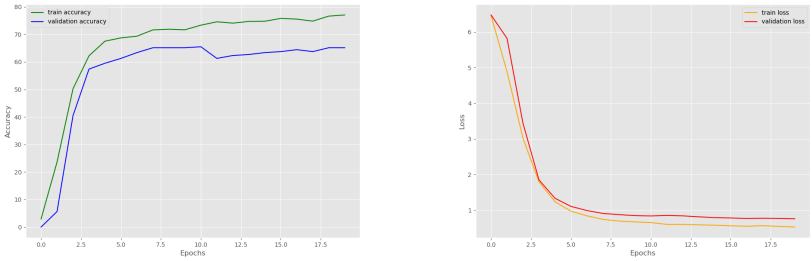

Figura 4.13: Gráficos evolutivos de *accuracy* y *loss* recortadas - recortadas

En esta situación, los resultados son superiores en comparación con el caso 2 y 3, dado que se ha aplicado el mismo tratamiento a las imágenes de entrenamiento, validación y prueba.

Si examinamos el gráfico que representa la precisión *accuracy* durante las primeras etapas, se observa un incremento notable tanto en el conjunto de entrenamiento como en el de validación. A medida que avanzan las etapas, este incremento se vuelve más gradual. La discrepancia entre las curvas de entrenamiento y validación es mínima y se mantiene constante a lo largo de todo el proceso.

En el gráfico de pérdida *loss*, se observa un comportamiento similar al de la precisión. Sin embargo, dado que el objetivo es minimizar el valor de pérdida, las curvas presentan una tendencia descendente. La diferencia entre los conjuntos de entrenamiento y validación es prácticamente insignificante.

En este caso específico, se ha obtenido un valor de test del 65.29 %.

Realizados los distintos experimentos combinando imágenes originales con recortadas, obtenemos la siguiente tabla resumen:

Modelo	Training loss	% Training accuracy	Validation loss	% Validation accuracy	Aciertos	Testeadas	% Acierto
O - O	0.50	77.40	0.62	72.00	229	300	76.33
O - R	0.49	77.83	3.23	23.94	62	291	21.31
R - O	0.53	76.84	4.31	19.67	60	300	20.00
R - R	0.53	77.05	0.76	65.14	190	291	65.29

Cuadro 4.2: Comparativa entrenamientos imágenes originales y recortadas

En esta cuadrícula podemos observar que el recorte de las imágenes, no lleva a una mejora del modelo.

Restauración de imágenes

Después de realizar el recorte de imágenes, se ha constatado que no se ha logrado mejorar el clasificador. Uno de los posibles motivos podría ser la dimensión y resolución de las imágenes. A pesar de haber eliminado aquellas que se consideraban que podrían introducir ruido, existe la posibilidad de que aún queden imágenes que no contribuyan a mejorar el modelo. Este hecho nos llevó a considerar la opción de restaurar estas imágenes recortadas, con el propósito de evaluar si la aplicación de este proceso podría conducir a una mejora en el modelo, aquí es donde entra en juego el uso de Real-ESRGAN (detallado en el apartado 3.4.1).

Para realizar este procedimiento, se inicia con las imágenes recortadas como punto de partida. Estas imágenes son las que se utilizan como entrada para la red GAN, con el objetivo de incrementar la resolución y, en consecuencia, eliminar cualquier ruido potencial que pueda existir debido a sus dimensiones.

Un ejemplo del resultado de este proceso es:

Imagen recortada → Imagen restaurada

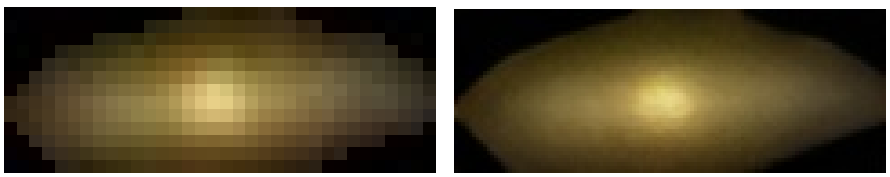

Figura 4.14: Ejemplo de restauración

En este ejemplo se puede apreciar la gran diferencia que existe entre una imagen y otra. La fotografía de la izquierda tiene unas dimensiones de 34x13 píxeles, mientras que las dimensiones de la fotografía de la derecha son de 119x45 píxeles.

Un mayor número de píxeles se correlaciona con una mayor resolución de la imagen. La resolución de una imagen se refiere a la cantidad de detalles que se pueden distinguir en ella. Cuantos más píxeles haya en una imagen, mayor será la cantidad de información visual capturada y, por lo tanto, mayor será la resolución.

Una vez creado el conjunto de imágenes restauradas, el siguiente paso es realizar el entrenamiento con estas nuevas imágenes. Los resultados obtenidos se detallan a continuación:

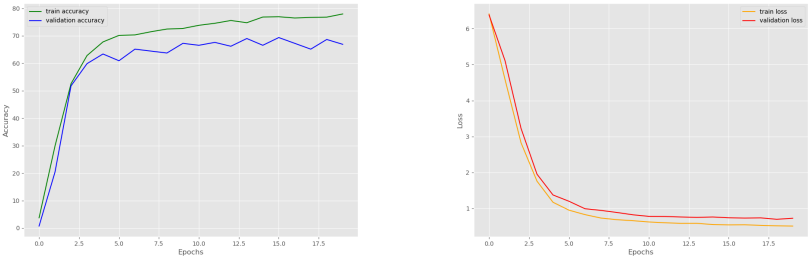

Figura 4.15: Gráficos evolutivos de *accuracy* y *loss* con imágenes restauradas

En la gráfica de la izquierda, de *accuracy*, se aprecia como la curva de entrenamiento crece de manera muy significativa en las primeras 5 épocas, a partir de esta época esta tendencia a la alta se suaviza. Con respecto a la curva de validación, el comportamiento es similar al de entrenamiento aunque en este caso, en torno a la época 10, los valores se estabilizan.

Si comparamos estas curvas de precisión con las obtenidas utilizando imágenes recortadas, el patrón es bastante similar, aunque en esta ocasión los valores obtenidos son ligeramente más altos.

En lo que respecta a la curva de pérdida o *loss*, tanto la curva de entrenamiento como la de validación presentan un comportamiento similar y apenas difieren entre sí. Se observa un descenso más pronunciado al principio y una estabilización hacia el final.

Al compararla con la curva de pérdida de las imágenes recortadas, se aprecia una diferencia: en el caso de las imágenes recortadas, el descenso de los valores es más abrupto al inicio que en el caso de las imágenes restauradas.

El porcentaje de acierto en test para las imágenes restauradas es de 69.07 %.

Tal y como esperábamos, este valor mejora al modelo entrenado con las fotos recortadas sin restaurar, sin embargo, este valor no es mejor que el obtenido con las imágenes originales.

4.2. Segmentación

Después de haber examinado el funcionamiento de la clasificación, procederemos a analizar el comportamiento de la segmentación en relación a las galaxias.

4.2.1. Preprocesado

Al igual que en la clasificación, previo al inicio del proceso de entrenamiento es necesario llevar a cabo una fase de preprocesamiento en el conjunto de datos con el propósito de potenciar su precisión y garantizar unos mejores resultados.

Organización de carpetas

En el contexto de la segmentación, la organización de las carpetas difiere de la empleada en la clasificación. Se crearán 6 carpetas distintas en función de si son de entrenamiento, validación o test, y a su vez, cada una de estas tendrán asociadas dos carpetas distintas, las imágenes en sí y las máscaras de estas imágenes. De forma gráfica, quedaría de la siguiente manera:

Figura 4.16: Organización de las imágenes en carpetas en segmentación

Preprocesado de imágenes de entrenamiento

Se llevarán a cabo una serie de transformaciones en las cuales se aplicarán diversos cambios a cada una de las imágenes de entrenamiento.

Para realizar las transformaciones de las imágenes de entrenamiento se hace uso de *albumentations* en vez de *transforms*: [4]

[4] `https://albumentations.ai/`

- *A.Resize()* 4.1.1
- *A..HorizontalFlip()* 4.1.1
- *A.RandomBrightnessContrast()* aplica aleatoriamente ajustes de brillo y contraste a una imagen. Esta función introduce variaciones en el brillo y el contraste de forma no determinista, lo que permite obtener diferentes versiones de una imagen con cambios sutiles en la iluminación y la relación de contraste.
- *A.RandomSunFlare()* se utiliza para generar aleatoriamente efectos de destellos de sol en una imagen. Al aplicar RandomSunFlare(), se agregan elementos visuales que imitan destellos, lo que añade un efecto realista a la imagen. Cada vez que se utiliza esta función, se generan destellos de sol de manera aleatoria, lo que permite obtener resultados diferentes en cada ejecución.
- *A.RandomFog()* genera aleatoriamente un efecto de niebla en una imagen. Se añade una capa de neblina simulada sobre la imagen, lo que puede crear una apariencia atmosférica y difuminada. La cantidad y la densidad de la niebla generada pueden variar en cada ejecución, lo que permite obtener resultados diferentes y ajustar el grado de efecto de niebla deseado en la imagen.
- *A.ImageCompression()* se usa para realizar una compresión de imagen con una calidad mínima establecida *quality_lower*. Esta función comprime la imagen reduciendo su calidad y tamaño de archivo. La calidad de la imagen se verá afectada y la compresión puede introducir artefactos visuales y pérdida de detalles.

Preprocesado de imágenes de validación

A diferencia del conjunto de entrenamiento, en el caso de las imágenes de validación simplemente se le realizarán un redimsensionado, usando *A.Resize()*.

Finalmente, tanto las imágenes de entrenamiento como de validación se normalizarán debido la ventaja que ofrece esta transformación, tal y como se explicó en el punto 4.1.1.

4.2.2. Entrenamiento

El conjunto de imágenes seleccionado es el mismo que el utilizado en clasificación4.1.2.

Para llevar a cabo el entrenamiento, el primer paso es obtener las máscaras de las imágenes seleccionadas, como vimos en el apartado 3.1.3. Obtenidas las máscaras, se entrenará con la arquitectura seleccionada, *deeplabv3*.

En este proceso se llevará a cabo la prueba de dos tipos diferentes de máscaras. Una de ellas será binaria, permitiendo distinguir entre lo que corresponde a una galaxia y lo que no lo es, mientras que la otra máscara utilizará distintos valores de píxeles para indicar el tipo específico de galaxia.

Una vez que alcanzamos este punto, nos enfrentamos al dilema de cómo el formato .jpg procesa las imágenes, ya que este formato realiza cierto nivel de suavizado en las mismas. Esto se debe al proceso de compresión con pérdida que se aplica durante la codificación de la imagen.

La compresión en formato .jpg tiende a eliminar detalles finos y sutiles de la imagen con el fin de reducir el tamaño del archivo. Esto puede ocasionar una apariencia general más suavizada, dado que los bordes y las transiciones suelen perder definición. La compresión tiende a promediar los valores de los píxeles adyacentes y eliminar detalles de alta frecuencia, lo que contribuye al efecto de suavizado en la imagen.

Este problema radica en el hecho de que, al indicarle a nuestra red si un píxel es parte de la máscara o no, los píxeles deben tener exactamente el valor especificado, y no nos es útil que se aproximen a dicho valor.

Para solventar esta dificultad, hemos optado por utilizar el formato .png, dado que este formato no realiza el suavizado mencionado anteriormente.

Máscaras binarias

Para entender visualmente lo que es una máscara binaria, a continuación se muestra un ejemplo:

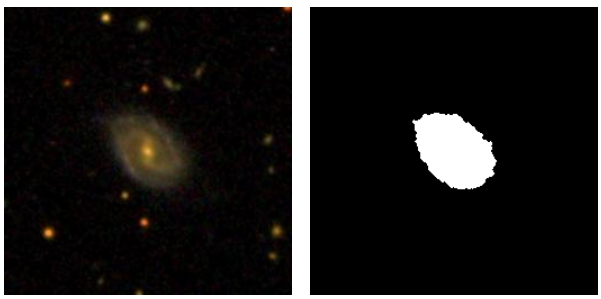

Figura 4.17: Ejemplo de imagen y su máscara binaria

Como se puede observar en la fotografía de la máscara, la superficie de color negro se corresponde con lo que no forma parte de la galaxia, mientras que la superficie blanca es la galaxia.

Antes de comenzar la ejecución del entrenamiento, eliminamos aquellas máscaras cuya superficie de galaxia sea inferior a 225 píxeles y sus correspondientes imágenes, ya que este tipo de imágenes podrían producir un peor ajuste del modelo.

Los resultados obtenidos durante el entrenamiento, esta vez con 40 épocas son:

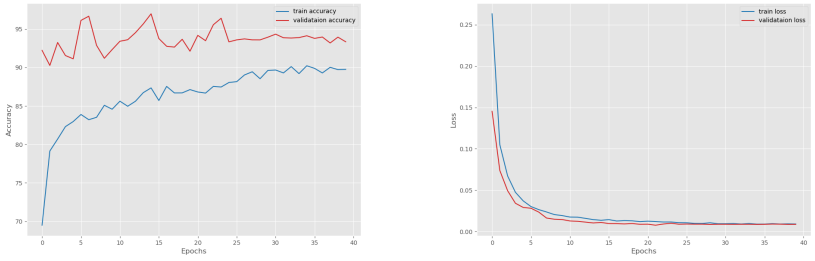

Figura 4.18: Gráficos evolutivos de *accuracy* y *loss* con máscaras binarias

Al analizar los gráficos generados, en la gráfica de *accuracy* se observa que el valor de entrenamiento oscila siempre en torno al mismo rango entre un 90.00 % y un 98.00 % a lo largo del entrenamiento. Sin embargo, en el caso de la validación se refleja una constante mejora a lo largo del entrenamiento, siendo más pronunciada al comienzo del mismo y obteniendo al final del proceso valores alrededor del 90.00 %.

En el gráfico de *loss*, se aprecia un comportamiento muy similar y sin apenas diferir entre las curvas de entrenamiento y validación, alcanzando valores de pérdida inferiores al 0.025.

Máscaras Multiclase

En este caso, tenemos tres máscaras distintas, en las que además de indicarnos lo que es o no galaxia, también nos indica el tipo de la misma, como se puede apreciar en las siguientes fotografías:

Al igual que ocurre con las máscaras binarias, antes del entrenamiento se eliminan aquellas imágenes y máscaras no deseadas, con el fin de reducir posible ruido.

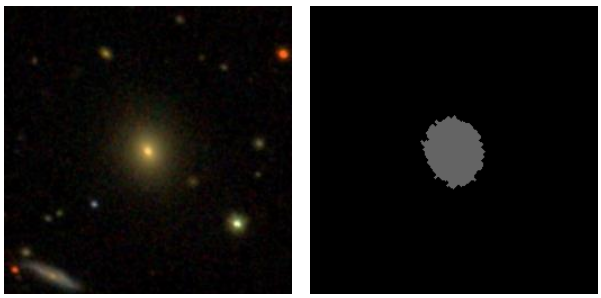

Figura 4.19: Ejemplo de galaxia *elliptical* y su máscara

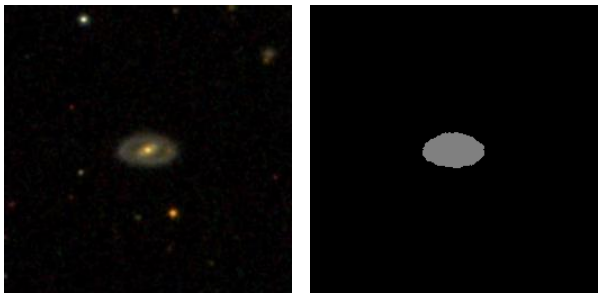

Figura 4.20: Ejemplo de galaxia *spiral* y su máscara

Con el uso de máscaras multiclase, se han generado los siguientes gráficos:

Como se refleja en la gráfica de precisión, en la curva de entrenamiento se produce un ascenso notable en las primeras épocas, estabilizándose al llegar a la décima época donde oscila entre el 70.00 % y el 75.00 % durante el resto del proceso. Si nos fijamos en la curva de validación presenta un ascenso constante durante todo el entrenamiento del modelo, alcanzando finalmente valores muy similares a los de entrenamiento.

El gráfico de pérdida presenta un descenso abrupto de sus valores al

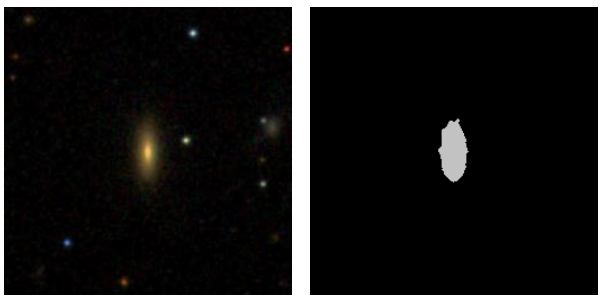

Figura 4.21: Ejemplo de galaxia *uncertain* y su máscara

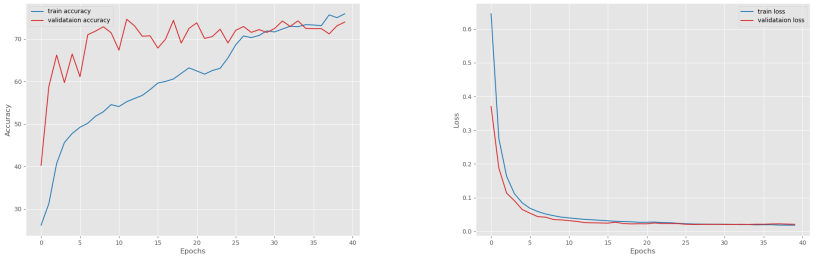

Figura 4.22: Gráficos evolutivos de *accuracy* y *loss* con máscaras multiclase

comienzo del entrenamiento en ambas curvas, suavizándose posteriormente y llegando a obtener valores prácticamente constantes al final del mismo. Los valores alcanzan un valor aproximado al 0.25.

El objetivo de la segmentación es ayudarnos al proceso de clasificación sin tener que hacer uso de archivos para conocer donde se sitúa la galaxia. A la vista de los resultados obtenidos de las pruebas entre máscaras binarias y máscaras multiclase, se obtienen mejores resultados haciendo uso de máscaras binarias que de máscaras multiclase, esto se debe a la diferencia de complejidad que existe entre tener en cuenta 2 clases distintas o 4.

Aunque el uso de máscaras multiclase podría permitirnos obtener directamente de la segmentación la clasificación de la galaxia, este proceso puede llevarnos a error llegando a detectar incluso dos galaxias distintas donde solo hay una. Por ello, de los resultados obtenidos concluimos que es mejor realizar el proceso de clasificación en dos pasos, segmentando en primer lugar y posteriormente clasificando esta segmentación haciendo uso de un clasificador, ya que los valores obtenidos en el entrenamiento con máscaras binarias son cercanos al 90.00 %, siendo notablemente superiores al obtenido mediante máscaras multiclase.

Aumento del conjunto de imágenes

Al igual que experimentamos en clasificación, en segmentación también vamos a probar si conseguimos mejores resultados al entrenar con un conjunto con el doble de imágenes.

Los resultados obtenidos del entrenamiento son:

El comportamiento de las curvas con el doble de imágenes es similar al obtenido con 3.000 imágenes de entrenamiento. Sin embargo, los resultados

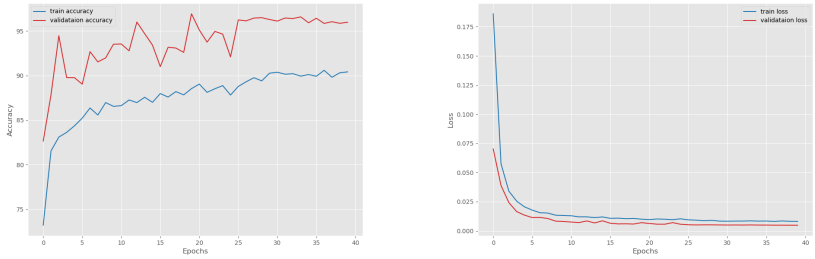

Figura 4.23: Gráficos evolutivos de *accuracy* y *loss* de segmentación con el doble de imágenes

obtenidos en términos de valores numéricos mejoran levemente, siendo en este caso superiores al 90.00 % si hablamos de precisión e inferiores a 0.0125 en pérdida.

Por lo anteriormente expuesto, se concluye que para el caso de segmentación sí conseguimos mejorar el modelo aumentando el número de imágenes usado para entrenar.

4.3. Comparativa de Hardware

Para el desarrollo de todos los experimentos y pruebas descritos en este capítulo 4, se ha hecho uso de GPUs, al inicio de este proyecto se hizo una comparativa entre la GPU de mi portátil (Intel(R) UHD Graphics 620) y la GPU que facilita Google Colab (NVIDIA Tesla K80).

4.3.1. Intel(R) UHD Graphics 620 vs NVIDIA Tesla K80

La Intel UHD Graphics 620 y la NVIDIA Tesla K80 son dos tipos diferentes de procesadores gráficos con propósitos y niveles de rendimiento distintos.

La Intel UHD Graphics 620 es una solución gráfica integrada que se encuentra comúnmente en portátiles y ordenadores de escritorio de baja potencia. Está diseñada para manejar tareas gráficas básicas como navegación web, reproducción de vídeo y juegos ligeros. Aunque puede manejar algunas cargas de trabajo gráficas, no está optimizada para tareas computacionales pesadas o aplicaciones profesionales intensivas en gráficos. UHD Graphics 620 tiene una frecuencia de reloj de unos 325 MHz y cuenta con 24 núcleos

de ejecución.

Por otro lado, la NVIDIA Tesla K80 [5] es una GPU de alto rendimiento diseñada principalmente para centros de datos y aplicaciones de cómputo profesional. Está basada en la arquitectura Kepler y ofrece una gran potencia de cálculo, ancho de banda de memoria y capacidades de procesamiento paralelo. La Tesla K80 está optimizada para tareas como simulaciones científicas, aprendizaje profundo, machine learning y otras cargas de trabajo intensivas. NVIDIA Tesla K80 cuenta con 4.992 núcleos CUDA y ofrece un ancho de banda de 480GB/s.

La NVIDIA Tesla K80 supera significativamente a la Intel UHD Graphics 620 en tareas computacionales complejas. La Tesla K80 tiene un mayor número de núcleos CUDA, un mayor ancho de banda de memoria y un mejor soporte para cómputo paralelo, lo que le permite manejar tareas exigentes de manera mucho más eficiente.

Vista la comparativa, descartamos el uso de la GPU del portátil personal y se estuvo trabajando con Google Colab una parte del proyecto. Sin embargo, llegado al apartado de segmentación, Google Colab tiene limitaciones de uso, por lo que se dió la posibilidad de hacer uso de los servidores GPU de la Universidad.

4.3.2. NVIDIA Tesla K80 vs GPU DE LA UNIVERSIDAD

[5] `https://www.nvidia.com/es-es/data-center/tesla-k80/`

Referencias

[1] Sylvain Arlot and Alain Celisse. A survey of cross-validation procedures for model selection. 2010.

[2] Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *SIAM review*, 60(2):223–311, 2018.

[3] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*, 2017.

[4] Sander Dieleman, Kyle W Willett, and Joni Dambre. Rotation-invariant convolutional neural networks for galaxy morphology prediction. *Monthly notices of the royal astronomical society*, 450(2):1441–1459, 2015.

[5] Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer, 2009.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[8] Bingbing Hu, Jiahui Tang, Jimei Wu, and Jiajuan Qing. An attention efficientnet-based strategy for bearing fault diagnosis under strong noise. *Sensors*, 22(17):6570, 2022.

[9] Gao Huang, Zhuang Liu, Geoff Pleiss, Laurens Van Der Maaten, and Kilian Q Weinberger. Convolutional networks with dense connecti-

vity. *IEEE transactions on pattern analysis and machine intelligence*, 44(12):8704–8716, 2019.

[10] M. Huertas-Company, R. Gravet, G. Cabrera-Vives, and et al. Photometric redshifts with self-organizing maps and deep learning. *Astronomy & Astrophysics*, 2015.

[11] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

[12] Alexander Khvostikov, Karim Aderghal, Jenny Benois-Pineau, Andrey Krylov, and Gwenaelle Catheline. 3d cnn-based classification using smri and md-dti images for alzheimer disease studies. *arXiv preprint arXiv:1801.05968*, 2018.

[13] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2012.

[15] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[16] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*. Springer, 2014.

[17] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010.

[18] Simon JD Prince. *Computer vision: models, learning, and inference*. Cambridge University Press, 2012.

[19] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[20] Reinaldo R Rosa. Data science strategies for multimessenger astronomy. *Anais da Academia Brasileira de Ciências*, 93, 2020.

[21] Reuven Rubinstein. The cross-entropy method for combinatorial and continuous optimization. *Methodology and computing in applied probability*, 1:127–190, 1999.

[22] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? *Advances in neural information processing systems*, 31, 2018.

[23] Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of big data*, 6(1):1–48, 2019.

[24] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[25] Leslie N Smith. Cyclical learning rates for training neural networks. In *2017 IEEE winter conference on applications of computer vision (WACV)*, pages 464–472. IEEE, 2017.

[26] Milan Sonka, Vaclav Hlavac, and Roger Boyle. *Image processing, analysis, and machine vision*. Cengage Learning, 2014.

[27] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Nature, 2022.

[28] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019.

[29] Xintao Wang, Liangbin Xie, Chao Dong, and Ying Shan. Realesrgan: Training real-world blind super-resolution with pure synthetic data supplementary material. *Computer Vision Foundation open access*, 2022.