UNIVERSIDAD DE EXTREMADURA

ESCUELA POLITÉCNICA

GRADO EN INGENIERÍA INFORMÁTICA EN INGENIERÍA DEL SOFTWARE

TRABAJO FIN DE GRADO

Clasificación Morfológica de Imágenes de Galaxias Mediante Técnicas de Deep Learning

UNIVERSIDAD DE EXTREMADURA

ESCUELA POLITÉCNICA

GRADO EN INGENIERÍA INFORMÁTICA EN INGENIERÍA DEL SOFTWARE

TRABAJO FIN DE GRADO

Clasificación Morfológica de Imágenes de Galaxias Mediante Técnicas de Deep Learning

Autor: Luis Antonio Costa Miguel

Tutor: Juan Antonio Rico Gallego

Resumen

En este trabajo se ha realizado un estudio de alternativas basadas en Deep Learning (Aprendizaje Profundo) para la clasificación de galaxias en base a su morfología. En particular, dentro del marco de Deep Learning, se hará uso de Keras. Esta es una biblioteca de código abierto y de bastante uso para este tipo de trabajos, sobre todo, en las primeras etapas de aprendizaje de Deep Learning, donde esta tecnología puede resultar bastante compleja.

Las categorías por las cuales se van a clasificar serán *elipticas*, *espirales*, *lenticulares* e *irregulares*. Durante el desarrollo de este documento se dará una mayor información con ejemplos de cada una de estas categorías.

Me apoyaré en un conjunto (dataset) compuesto por una gran cantidad de imágenes de cada uno de los tipos de galaxias informadas anteriormente. El origen de estas imágenes no estaba indicado en la página donde las obtuve pero, según investigaciones realizadas posteriormente, parece que provienen de algún telescopio. Este conjunto de imágenes está disponible de forma abierta en internet.

Se hará un estudio detallado sobre las diversas posibilidades que Keras nos ofrece para este tipo de trabajos, haciendo uso de técnicas ya incluidas en la propia librería y otras que serán desarrolladas por el autor del TFG. En este documento no se presentarán todas las pruebas realizadas durante el desarrollo de este TFG, solo se irán mostrando aquellas cuyos resultados obtenidos considere que son más interesantes. Posteriormente, se mostrarán funcionalidades que se pueden aplicar sobre el modelo resultante del entrenamiento realizado.

Índice general

Índice de tablas

Índice de figuras

Capítulo 1

Introducción

Durante los últimos años estamos viendo como el mundo de la tecnología está evolucionando a una velocidad endiablada. Estamos hoy en día en pleno desarrollo de los que algunos denominan la **Cuarta Revolución Industrial** o también llamada **Industria 4.0**. Esta industria representa nuevas formas donde la tecnología se integra en las sociedades e incluso en el cuerpo humano. Está marcada por los avances tecnológicos emergentes en varios campos, que incluyen: robótica, inteligencia artificial, nanotecnología, computación cuántica, biotecnología, Internet de las cosas (IoT), impresión 3D y vehículos autónomos.

El concepto de industria 4.0 surgió durante una Feria de Hanover (salón de la tecnología industrial) en el año 2011. Es uno de los proyectos clave de la estrategia relativa a las tecnologías punta del gobierno alemán, que promueve la revolución digital de las industrias. Consiste en la digitalización de la industria y todos los servicios relacionados con la empresa. Lo más destacado de este nuevo tipo de industria es la automatización. Especialmente este punto se destaca en las grandes industrias, ya que permite interconectar las unidades de producción, crear redes de producción digital y utilizar los recursos de forma mucho más eficiente. Los puntos más importantes de esta nueva industria son:

- Automatización
- Conectividad

- Información digital
- Acceso digital al cliente y a otros usuarios en menor tiempo

La tecnología avanza, y al igual que ella, las empresas deben hacerlo. Sin embargo, muchas de ellas no están preparadas para estos cambios y tienen el riesgo de quedarse desactualizadas.

En lo que a este proyecto afecta, una de las tecnologías que más estan favoreciendo esta evolución es lo que llamamos **Deep Learning**. Es, en cierta manera, una parte muy importante de lo que anteriormente hemos nombrado como uno de los campos importantes en la llamada Industria 4.0 como es la Inteligencia Artificial.

Aunque la Inteligencia Artificial apareció sobre los años 50 con el fin de automatizar las tareas intelectuales que normalmente realizan los humanos, no ha sido hasta finales de la primera década del 2000 cuando ha empezado a despuntar gracias a los grandes avances en la computación. Estos avances han permitido ejecutar procesos en poco tiempo cuando anteriormente habrían tardado días, semanas, meses o incluso años. Fue entonces cuando nació el Deep Learing.

En la realización de este trabajo se ha hecho uso del Deep Learning con el fin de clasificar imágenes de galaxias de manera automática. Se busca la realización de entrenenamientos con la mayor precisión que esta herramienta nos pueda ofrecer dentro de los conocimentos que se han ido adquiriendo en el desarrollo de este proyecto. Esta clasificación se realizará según la morfología de las diversas galaxias que nos podemos encontrar en el espacio, apoyándonos en imágenes obtenidas a través de satélites. Para ser más precisos, se ha hecho uso de la secuencia de Habble para definir esta clasificación.

La secuencia del Hubble es un esquema de clasificación morfológica para galaxias, inventado por *Edwin Hubble* en 1926. A menudo se le conoce coloquialmente como el "diapasón de Hubble" debido a la forma en la que se representa tradicionalmente. Aunque la secuencia de Habble está compuesta por tres tipos de galaxias, para este trabajo en algunos entrenamientos, se ha incluido un nuevo tipo definido como galaxias irregulares. Este tipo de galaxias no tienen una estructura

regular obvia.

El conocimiento de una galaxia según su morfología, permite a los astrónomos conocer ya de antemano un conjunto de características físicas que están asociadas a un tipo de galaxia en particular. También podrá agilizar la clasificación de un conjunto de imágenes tomadas durante una jornada de trabajo evitando así que el propio astrónomo tenga que clasificarlas de forma manual.

Entrando más en detalle sobre el Deep Learing, este es un caso particular de redes neuronales. Para este trabajo se hará uso de una red en particular que se denomina **redes neuronales convolucionales**. Lo que se buscará más en detalle será la creación de una red neuronal que mejor se adapte a la clasificación que se quiere desarrollar y poder obtener los mejores resultados posibles.

En lo que se refiere a la implementación del proyecto, se ha centrado más en el uso de una bibliotieca muy conocida en el mundo del Deep Learning como es **Keras**. Es una biblbioteca de redes neuronales escrita en Python, muy sencilla y muy recomendada para los que empiezan en este mundo del Deep Learning.

Durante esta documentación se entrará más en detalle, tanto de la tecnología con la que se ha desarrollado el proyecto, como con la información que nos permitirá clasificar el conjunto de imágenes usadas.

1.1. Presentación de los capítulos

En esta sección se hará una introducción de los capítulos que se han ido desarrollando en esta documentación.

- **Objetivos.** En este capítulo se describe cuál es el objetivo principal por el que se desarrolló este trabajo. Posteriormente se indicarán otros objetivos secundarios.

- **Estado del Arte.** En este capítulo se describe más en detalle lo que es el Deep Learning, de donde proviene, la evolución que ha ido desarrollando y lo que se espera.

- **Metodología.** En este capítulo se describen todas las tencologías empleadas, tanto harware como software, para la realización correcta de este trabajo. Se entrará más en detalle de lo que son las Redes Neuronales, las redes empleadas en este trabajo con las diversas partes o capas que se han empleado en estas.

- **Implementación y Desarrollo.** Se describen las diferentes partes en las que se compone todo el desarrollo para poder realizar un entrenamiento en Keras.

- **Resultados.** Se presentarán los diversos resultados que he ido obteniendo en las distintas pruebas realizadas. Solo se presentarán aquellas cuyos resultados sean más interesantes. También se expondrán algunos usos que se le puede dar al modelo que se obtiene del entrenamiento.

- **Conclusiones.** Se describen las conclusiones obtenidas de este trabajo y de los resultados obtenidos de los diversos entrenamientos realizados.

Capítulo 2

Objetivos

En este apartado se va a detallar cuál ha sido el objetivo principal que se ha ido persiguiendo a lo largo del desarrollo de este TFG y otros objetivos adicionales que se pretenden cumplir con el desarrollo de este trabajo.

2.1. Objetivo Principal

En primer lugar hay que tener en cuenta que Deep Learning es una tecnología cuya principal motivación es la de facilitar la vida a las personas, facilitar el día a día. En mi caso particular, el objetivo principal que se busca es que la propia máquina sea capaz de clasificar imágenes con la mayor precisión posible. En concreto, que se le informe a un usuario que tipo de galaxia es la que está observando en ese mismo instante, según la clasificación morfológica de las galaxias. Facilitar el día a día de un astrónomo mientras trabaja con un conjunto de imágenes de galaxias sin clasificar.

Este trabajo en particular sería de utilidad para astrónomos o gente que se dedica directamente con temas de astronomía. También se podría incluir a aquellos otros que simplemente tienen la astronomía por afición.

2.2. Objetivos Adicionales

Como objetivos adicionales se consideran los siguientes:

- Introducción a tecnologías de Inteligencia Artificial, como Deep Learning, que actualmente se considera que tendrán un buen papel en el futuro.
- Introducción a la biblioteca de Keras, muy usada actualemente para el desarrollo de Deep Learning. Esta biblioteca nos da un gran apoyo en la generación de Redes Neuronales desde cero, sobre todo, a aquellos que se inician en este mundo y que desde fuera parece tan complejo.
- Ver todas las funcionalidades que nos ofrece Deep Learning, ya que sirve para mucho más que para la clasificación de imágenes.
- Aprender el funcionamiento general de una Red Neuronal

Capítulo 3

Estado del Arte

En este capítulo se entrará más en profundidad definiendo que es el **Deep Learing**, sus inicios y lo que se espera en un futuro no muy lejano de esta herramienta. Para definir correctamente que es el Deep Learning debemos empezar comentando por donde comenzo todo, tendremos que hablar de la Inteligencia Artificial. En la siguiente imagen podrá observar la relación existente entre estas tecnologías, aclarando de una forma más visual, donde se encuentra el Deep Learning.

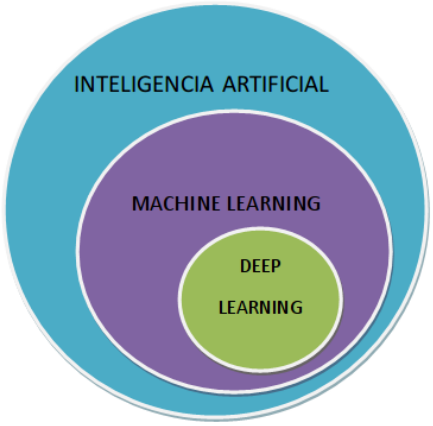

Figura 3.1: Relación Inteligencia Artificial con el Deep Learning, Raul E. Lopez Briega

A continuación, se irán definiendo cada una de estas tecnologías, centrándonos sobre todo en el Deep Learning, que es con la que se ha estado trabajando en este proyecto.

3.1. Inteligencia Artificial

La **Inteligencia Artificial** nació en la década de 1950 cuando un conjunto de pioneros de la informática se empezaron a preguntar si una computadora podría llegar a pensar como una persona. Se podría definir, de forma concisa, como el esfuerzo por automatizar las tareas intelectuales que normalmente realizan los humanos [1]. También, podríamos definirla como el esfuerzo por emular el trabajo que realiza el cerebro humano.

Fue en el año 1956, cuando John McCarthy organiza la mítica conferencia de Dartmouth donde, en su discurso, acuña por primera vez el término Inteligencia Artificial. Esta la definió como la ciencia e ingeniería de hacer máquinas inteligentes. El texto inaugural lo realiza junto a Marvin Minsky y Claude Shannon, dos prestigiosos científicos. Sin embargo, McCarthy se consagra como padre de la Inteligencia Artificial no solo por lograr iniciarla y convertirla en un campo de investigación nuevo, sino por seguir aportando evidencias para su desarrollo durante medio siglo[2].

Aún siendo este pionero de la Inteligencia Artificial, nunca pudo pasar el llamado Test de Turing. Este test consiste en desarrollar un método para responder científicamente si una máquina puede pensar por sí mismo o no. Es básicamente una conversación entre un ser humano y una máquina diseñada para interactuar verbalmente. La conversación se efectúa en lenguaje común y busca que se pueda identificar cuál es el humano y cuál es la máquina[3].

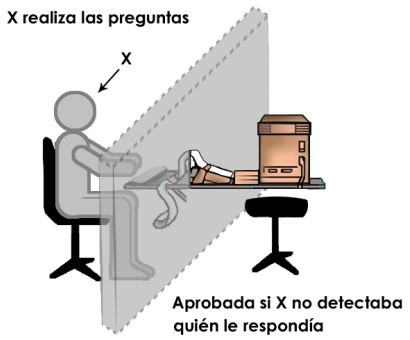

Figura 3.2: Test de Turing, Omar Pereyra

Se cuenta con 5 minutos de conversación para convencer a la persona que evalúa el chat, que quien está detrás de la pantalla es un ser humano (la máquina solo se expresa a través de un chat tras una pantalla). Si logra convencerlo, la máquina pasa la prueba.

Como dato interesante, el primero en pasar el Test de Turing fue Eugene Goostman en el año 2014 pasando el test con un 33 %. Hizo uso de un chatbot desarrollado por programadores que simularon la personalidad de un adolescente ucraniano.

Volviendo al origen de la Inteligencia Artificial, en un comienzo se pensó que la inteligencia artificial a nivel humano se podría lograr haciendo que los programadores elaboraran un conjunto suficientemente amplio de reglas explícitas para manipular el conocimiento. Este enfoque se conoce como Inteligencia Artificial Simbólica, llegando a predominar hasta finales de la década de 1980.

Esta Inteligencia Artificial Simbólica tuvo momentos de altibajos. Se lograban soluciones para problemas lógicos bien definidos como jugar al ajedrez, pero no conseguían solucionar problemas más complejos y difusos como la clasificación de imágenes, el reconocimiento de voz o la traducción de idiomas. De esta forma, surgió un nuevo enfoque, el Machine Learning o Aprendizaje Automático.

Esta tecnología es una de la grandes causantes de esta nueva revolución industrial que esta haciendo avanzar el mundo tecnológico de una manera rapidísima. Lo que estamos viviendo actualmente es una mínima parte de lo que puede darnos esta tecnología y que es problable que veamos en un futuro no muy lejano.

Pero no todo es positivo en lo relativo a la inteligencia artificial. Se prevé que en 15 años se hayan perdido el 40 % de los trabajos del mundo, donde las personas serán reemplazadas por algún sistema en inteligencia artificial. *Kai-Fu Lee* es una de las voces más importantes en cuanto a la inteligencia artificial. Pionero en el desarrollo del primer sistema de reconocimiento de voz, habló sobre como afectará esta tecnología al futuro de las personas y dijo lo siguiente:

“Creo que la inteligencia artificial va a cambiar al mundo más que nada en la historia de la humanidad. Incluso más que la electricidad”[5].

La inteligencia artificial es una realidad presente en nuestras vidas que poco a poco

irá jugando un papel cada vez más importante.

Los usos actuales de la inteligencia artificial son muchos más de los que la gente cree. Cuando tu correo bloquea mensajes de spam, cuando hablas con Siri en un iphone, cuando netflix te sugiere una película determinada , estos son ejemplos del uso de la Inteligencia Artificial. Pero lo que viene es mucho más relevante. Desaparecerán los conductores de vehículos, cánceres y otras enfermedades serán mas fáciles de detectar, se podrán prever mucho mejor los desastres naturales y los trabajos repetitivos y mecánicos quedarán relegados a los robots.

Como se esta viendo, habrá pocas cosas que no queden afectadas por el desarrollo de esta tecnología, ya que afectará a todos los ámbitos de la sociedad como las relaciones entre humanos, el consumo, la economía, el funcionamiento de las ciudades, etc.

3.2. Machine Learning

El **Machine Learning**, traducido al español como **Aprendizaje Automático**, se podría definir a través de la siguiente pregunta: ¿podría una computadora ir más allá de "ordenarle que realice una tarea" y aprender por sí misma cómo realizar una tarea específica? ¿podría una computadora aprender ciertos patrones de forma automática mirando los datos?

Estas preguntas abrieron la puerta a un nuevo paradigma de la programación, el paradigma de la Inteligencia Artificial Simbólica. Con el aprendizaje automático, los humanos ingresan datos, así como las respuestas esperadas de los datos, y salen las reglas. Estas reglas se pueden aplicar a nuevos datos para producir respuestas originales. Un sistema de aprendizaje automático está entrenado en lugar de programado explícitamente[4].

El aprendizaje automático comenzó a florecer en la década de 1990 y, se ha convertido rápidamente en el subcampo más popular y exitoso de la Inteligencia Artificial. Es una tendencia impulsada por la disponibilidad de hardware más rápido y conjuntos de datos más grandes, posibilitado esto último por lo que actualmente llamamos Big Data (conjuntos de datos o combinaciones de conjuntos de datos cuyo tamaño (volumen), complejidad (variabilidad) y velocidad de crecimiento (velocidad) dificultan su captura, gestión, procesamiento o análisis mediante tecnologías y herramientas convencionales dentro del tiempo necesario para que sean útiles[6]).

Después de lo introducido anteriormente, podremos definir el Machine Learning como una disciplina dentro de la Inteligencia Artificial que crea sistemas capaces de aprender automaticamente por si solos, identificando patrones complejos en millones de datos. Estos sistemas van mejorando con el tiempo sin necesidad de intervención humana.

Hoy en día, gracias al Machine Learning podemos procesar cada vez mayores cantidades de datos, mejorando la funcionalidad de muchas de las aplicaciones que se usan. Ejemplos de uso actuales del Machine Learning:

- **Seguridad en los datos:** puede predecir que archivos son malware con gran precisión.
- **Seguridad personal:** podría servir para aligerar los controles de seguridad en aeropuertos ayudando a eliminar falsas alarmas y detectar anomalías en las proyecciones de seguridad.
- **Comercio financiero:** muchas empresas del sector ya utilizan sus propios sistemas para predecir y ejecutar operaciones de gran volumen a altas velocidades y así anteponerse a cómo se comportarán los mercados de valores.
- **Salud:** se puede procesar más información y detectar más patrones sobre enfermedades que una mente humana. Además, el machine learning también se puede utilizar para advertir en los factores de riesgo de enfermedades en poblaciones grandes.
- **Marketing personalizado:** El marketing personalizado se basa en un sencillo principio, mientras más se pueda saber sobre el cliente y su comportamiento, mejor se le podrá atender y se traducirá en más ventas. Es lo que actualmente se llama publicidad a la carta.
- **Búsqueda online:** Este quizás sea el uso más famoso del machine learning. Google y sus competidores mejoran constantemente lo que entiende el motor de búsqueda.
- **Coches inteligentes:** Los vehículos podrían ajustar la configuración interna (temperatura, audio, posición del asiento, etc.) de forma automática en función del conductor, informar e incluso solucionar problemas y, conducir y ofrecer asesoramiento en tiempo real sobre el tráfico y las condiciones de la carretera.

Estas son algunas de las aplicaciones en el día a día de las personas que hacen uso del Machine Learning. Cada vez serán más sus aplicaciones y más complejas con el fin de mejorar la vida de las personas.

3.3. Deep Learning

El **Deep Learning**, traducido al español como **Aprendizaje Profundo**, es la tecnología que se empleará para el desarrollo de este proyecto y donde se hará más hincapié para el resto de la documentación.

Deep Learning es uno de los métodos de aprendizaje de la Inteligencia Artificial, y a día de hoy pertenece a un subcampo del Machine Learning. Fue por el 2010 cuando esta teconología tuvo su gran apogeo gracias en parte, a la inserción de las redes sociales, así como el desarrollo del mundo web. Se multiplica la generación de datos por individuo, lo que hace surgir una necesidad inmediata de herramientas y tecnologías que permitan aprovechar estos datos: el Big Data (definido en la sección anterior).

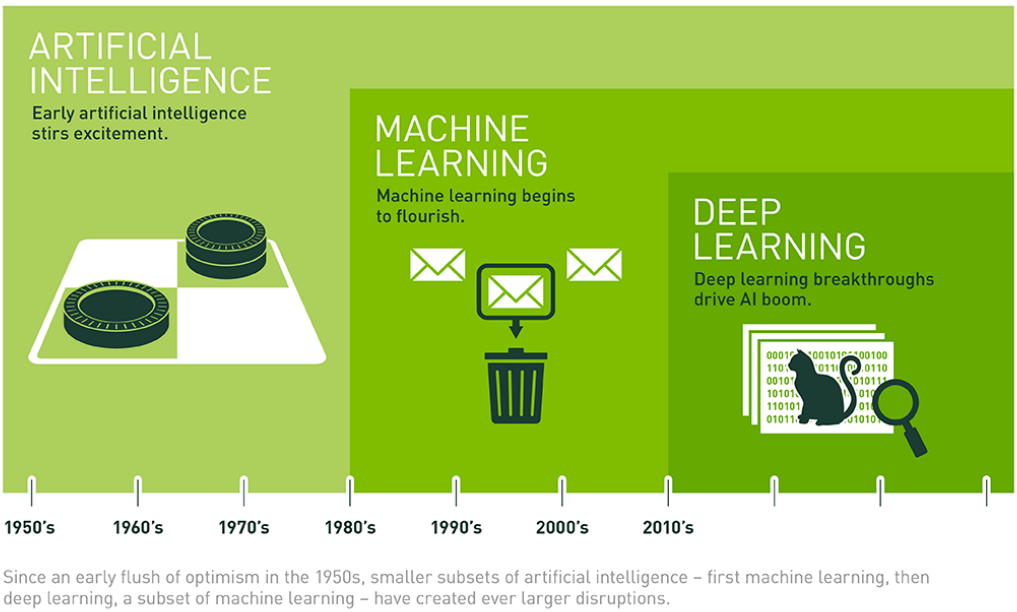

Figura 3.3: Línea del Tiempo Inteligencia Artificial Hasta el Deep Learning, Txema Rodríguez

Se define como un algoritmo automático estructurado o jerárquico que emula el aprendizaje humano con el fin de obtener ciertos conocimientos. Destaca porque no requiere de reglas programadas previamente, sino que el propio sistema es capaz

de “aprender” por sí mismo para efectuar una tarea a través de una fase previa de entrenamiento. Trata de asemejar el aprendizaje que los humanos usan para obtener el conocimiento específico de una cosa o imagen. Es lo que actualmente se llama como **aprendizaje no supervisado**. Aqui reside una de las grandes ventajas del Deep Learning, no necesita la acción de ningún humano para aprender, esta tecnología aprende por sí sola.

Hoy en día es una de las tecnologías más de moda ya que hace uso de estructuras lógicas que se asemejan en mayor medida a la organización del sistema nervioso de los mamíferos. Estas tienen capas de unidades de proceso (neuronas artificiales) que se especializan en detectar determinadas características existentes en los objetos percibidos. Se compone de unas redes de neuronas que son profundas con algoritmos que son sumamente complejos con respecto a los usados en el Machine Learning.

Uno de los pioneros en el Deep Learing fue *Geoffrey Hinton*. En 2012, Hinton y dos estudiantes demostraron que las redes de neuronas artificiales convolucionales podían reconocer imágenes con mayor precisión que cualquier otra técnica basada en inteligencia artificial. Después de una gran labor sobre este campo, *Geoffrey Hinton* fue galardonado con el Premio Turing en 2018 junto con *Yoshua Bengio* y *Yann LeCun* por su trabajo en Deep Learning.

Desde que en 2010 tuviera ese apogeo, ha logrado nada menos que una revolución en el campo, con resultados notables en problemas de percepción como la vista y la audición. Problemas que involucran habilidades que parecen naturales e intuitivas para los humanos pero que durante mucho tiempo han sido difíciles de encontrar para las máquinas.

El aprendizaje profundo ha tenido un gran uso en el área de los diagnósticos médicos y en análisis predictivos de los mercados financieros. Pero está adquiriendo cada vez, un mayor peso en el resto de sectores como componentes esenciales para aplicaciones, como sistemas de recomendación, detección de fraude, detección de anomalías y auditoría de datos.

En particular, el aprendizaje profundo ha logrado los siguientes avances, todos en

áreas históricamente difíciles del aprendizaje automático:

- Clasificación de imágenes a nivel casi humano
- Reconocimiento de voz a nivel casi humano
- Transcripción de escritura a mano a nivel humano
- Traducción automática mejorada
- Conversión de texto a voz mejorada
- Asistentes digitales como Google Now y Amazon Alexa
- Conducción autónoma a nivel casi humano
- Se mejoró la orientación de anuncios
- Resultados de búsqueda mejorados en la web
- Habilidad para responder preguntas de lenguaje natural

Todavía se está explorando todo lo que puede hacer el aprendizaje profundo. Comenzamos a aplicarlo a una amplia variedad de problemas fuera de la percepción de la máquina y la comprensión del lenguaje natural, como el razonamiento formal. Si tiene éxito, esto puede anunciar una época en la que el aprendizaje profundo ayuda a los humanos en una amplia variedad de sectores. [7].

Capítulo 4

Metodología

En primer lugar, es importante remarcar cual es el principal objetivo de este trabajo que no es otro que conseguir un modelo que sea capaz de realizar una clasificación automática según la morfología de las galaxias. Dando como entrada una imagen de una galaxia, el modelo tiene que establecer, una vez procesada esa imagen, el tipo de galaxia a la que pertenece esa imagen.

Se presentarán los elementos físicos y virtuales que se han usado para conseguir tales objetivos. Se expondrá la forma en que se han obtenido las imágenes necesarias para poder generar el modelo con la máxima precisión posible.

Durante el desarrollo de este trabajo también me he encontrado con una serie de limitaciones que expondré a continuación. Este tipo de trabajos, se suele realizar con máquinas bastante más potentes que las que se han usado, pudiendo así modificar en cierta forma, los resultados finales. Comentar también que los conocimientos adquiridos durante este periodo de trabajo quizás no hayan sido tan completos como se debería. Aún así, siempre se ha buscado poder acercarse lo máximo posible a un buen resultado. Durante este apartado se expondrán algunas de estas dificultades más en detalle.

Se va a dividir este capítulo en cuatro partes diferenciadas:

- El dataset o conjunto de imágenes usadas para los entrenamientos.
- Herramientas Hardware disponibles.

- Herramientas Software empleadas para la implementación y desarrollo.
- Redes Neuronales

A continuación, se presentarán cada uno de los puntos más en detalle.

4.1. Datasets (Conjunto de datos)

En este proyecto se va a hacer uso de un conjunto de imágenes. Estas nos servirán para poder entrenar a nuestro modelo y así determinar a que tipo de galaxia pertenece una determinada imagen que se le pase como parámetro de entrada a nuestro modelo ya entrenado.

Para obtener estas imágenes hemos accedido a una famosa página llamada *Kaggle*. Esta página es una plataforma online donde se realizan competiciones de Data Mining y nos da acceso a una gran cantidad de datasets de diversas índoles. En mi caso, busqué aquella que tenía que ver con galaxias y descargué este datasets desde la siguiente url indicanda como referencia[10].

En este datasets se nos proporciona una carpeta que contiene 48.905 imágenes preparadas para el entrenamiento y otra carpeta con 12.750 imágenes para el test. En la carpeta de entrenamiento cada nombre de imagen inicia por el tipo de galaxia a la que corresponde, lo que facilita la preparación de las imágenes para el entrenamiento, algo muy importante para este tipo de trabajo. Las cantidades que hay por cada tipo de galaxia son bastante diferentes. Como solución a este contratiempo se han usado técnicas empleadas en Deep Learning para poder igualar este número de imágenes. De este modo se consigue que nuestro modelo pueda detectar cualquier tipo de galaxia de forma equitativa.

Todas las imágenes tienen un tamaño original de 300x300 lo que facilita el preparado de estas para el entrenamiento. Con esta resolución, la calidad de las imágenes no será la idónea, lo que podría dificultar la etapa de entrenamiento en donde se obtienen las características mas en detalle de la galaxia en cuestión.

Esta clasificación será según la morfología de las galaxias. Se podrán diferenciar entre 4 tipos de galaxias que son las que se nos ofrece en la carpeta donde se encuentran las imágenes para el entrenamiento.

Con respecto al origen de estas imágenes, la misma página no nos da mucho información. Es bastante probable que vengan de capturas tomadas por algún telescopio, pero no esta confirmado al 100 %. Por lo que he podido investigar, las imágenes que se toman de los telescopios son imágenes que abarcan una gran cantidad de galaxias. Un ejemplo es la imagen que muestro a continuación, captura hecha con el telescopio Hubble desde el exterior de la atmósfera, sobre la órbita de la Tierra.

Figura 4.1: Captura del telescopio Hubble, NASA

Como se puede observar, esta imagen contiene una gran cantidad de galaxias. Para obtener una imagen de una galaxia en concreto, el zoom que hay sobre esta imagen es enorme. Es por esto, por lo que en mi parecer, la resolución de las imágenes con las que se va a trabajar no es la mejor, pero es bastante buena viendo su posible origen.

Este conjunto de imágenes han sido tratadas anteriormente de forma casi segura para poder ser usadas en trabajos con Deep Learning, ya que, como comente anteriormente, vienen correctamente etiquetadas y con un mismo tamaño en todas ellas.

Para la clasificación morfológica de las imágenes se ha hecho uso de la **secuencia**

de Hubble. La secuencia de Hubble es un esquema de clasificación morfológica para galaxias inventado por *Edwin Hubble* en 1926[11]. El esquema del Hubble divide las galaxias en tres amplias clases basadas en su apariencia visual:

- **Elípticas**
- **Espirales**
- **Lenticulares**

Estas amplias clases pueden ser extendidas para permitir distinciones más finas de apariencia. También podemos encontrar otros tipos de galaxias, tales como galaxias **irregulares**, que no tienen una estructura regular obvia.

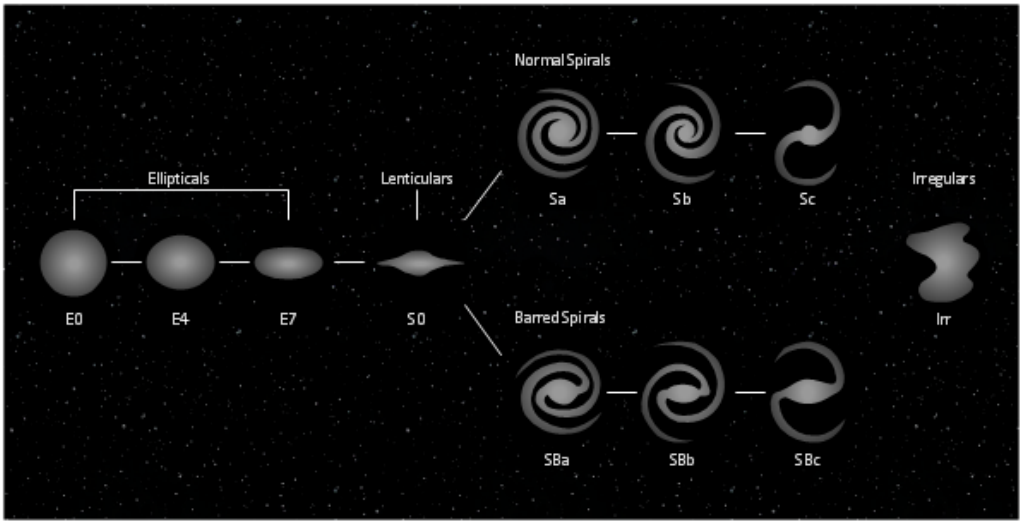

Figura 4.2: Secuencia de Hubble, Ville Koistinen

Hasta el día de hoy, la secuencia de Hubble es el sistema más comúnmente utilizado para clasificar galaxias, tanto en la investigación astronómica profesional como en astronomía amateur.

A continuación, se expondrán cada uno de los tipos de galaxias más en detalle mostrando una imagen de cada una de ellas. Las imágenes que se mostrarán serán las mismas que se han usado durante los entrenamientos ejecutados.

4.1.1. Galaxias elípticas - Edge

Una galaxia elíptica es un tipo de galaxia de la secuencia de Hubble caracterizada por tener una forma aproximadamente elipsoidal y apenas rasgos distintivos. Carece de los brazos espirales que caracterizan a las galaxias homónimas[12]. Tienen distribuciones de luz suaves y sin características. Se indican mediante la letra "E", seguida de un número entero n que representa su grado de elipticidad en el cielo. Son el 20 % de las galaxias observadas actualmente. Se identificarán con el nombre de Edge durante los entrenamientos. Un ejemplo de este tipo de galaxia es el siguiente:

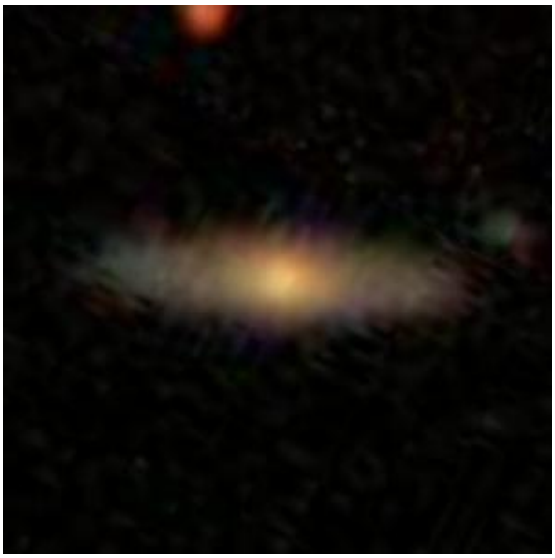

Figura 4.3: Galaxia Edge E7

4.1.2. Galaxias espirales - Spiral

Galaxia de la secuencia de Hubble. Deben su nombre a los brazos luminosos con formación estelar dentro del disco ,que se prolongan más o menos logarítmicamente desde el núcleo central. Aunque a veces son difíciles de percibir, estos brazos las distinguen de las galaxias lenticulares que presentan una estructura de disco pero sin brazos espirales[14]. Constituyen el 70 % de las galaxias observadas actualmente. Se les da el símbolo "S". Como curiosidad, nuestra galaxia, la Vía Láctea, es espiral. Se identificarán con el nombre de Spiral durante los entrenamientos. Un ejemplo de este

tipo de galaxia es el siguiente:

Figura 4.4: Galaxia Spiral SBa

4.1.3. Galaxias lenticulares - Smooth

Es un tipo de galaxia intermedia entre una galaxia elíptica y una galaxia espiral que en la Secuencia de Hubble se clasifica como “S0”. Las galaxias lenticulares consisten en un brillante bulbo central rodeado por una estructura extendida, similar a un disco. A diferencia de galaxias espirales, los discos de las galaxias lenticulares no tienen estructura espiral visible. Han consumido o perdido gran parte o toda su materia interestelar, como las galaxias elípticas, y por tanto carecen de brazos espirales[13]. Constituyen solo el 3 % de las galaxias observadas actualmente. Se identificarán con el nombre de Smooth durante los entrenamientos. Un ejemplo de este tipo de galaxia es el siguiente:

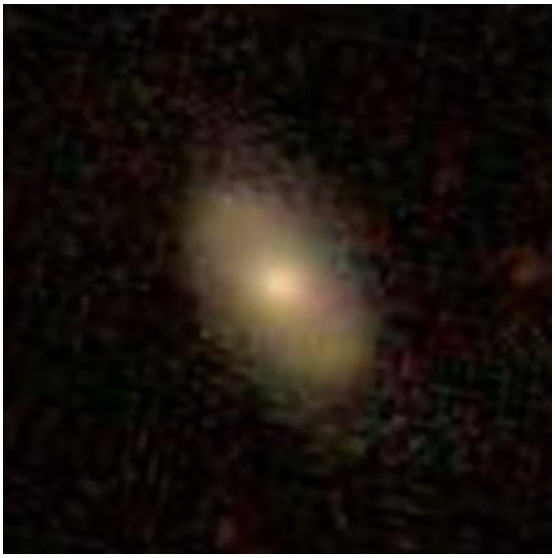

Figura 4.5: Galaxia Smooth S0

4.1.4. Galaxias irregulares - Other

Una galaxia irregular es una galaxia que no encaja en ninguna clasificación de galaxia de la secuencia de Hubble. Son galaxias sin forma espiral, lenticular ni elíptica. Algunas galaxias irregulares son pequeñas galaxias espirales distorsionadas por la gravedad de un vecino mayor. Un ejemplo de este tipo de galaxia es el siguiente:

Figura 4.6: Galaxia Irregular

4.2. Herramientas Hardware

En el Deep Learning, un equipo potente es una herramienta esencial para realizar un trabajo decente. En mi caso, para la realización de este TFG, el equipo usado ha sido como cualquier sobremesa que tiene una familia en su casa por lo que me ha dificultado en cierto modo el poder obtener unos resultados mejores.

En primer lugar, para trabajar con Deep Learning, se hace esencial tener una GPU. Se puede emplear la CPU de nuestro ordenador pero el tiempo que se tarda en realizar cualquier entrenamiento con imágenes es inmenso en comparación con el que se tarda con una GPU. Para entrenamientos realizados con textos y audios, con la CPU es más que suficiente pero, cuando se va a trabajar con imágenes, se hace imprescindible por el tiempo empleado en cada entrenamiento. Dentro de las diversas GPU que nos ofrece el mercado, con una barata se puede salir adelante mas o menos bien. En mi caso, he empleando una GPU normalita donde el gran inconveniente que se me presentó fue su memoria. Aún así, se han conseguido resultados bastante decentes para ser la primera vez que he trabajado con Deep Learning.

Las características de mi equipo (ordenador de sobremesa) con el que se han hecho todos los entrenamientos es el siguiente:

- Procesador Intel I3 8100
- Disco duro SSD Samsung 850 EVO 500GB
- Memoria Ram 16GB
- GPU GEFORCE GTX 1050TI 4GB GDDR5

De todos los componentes descritos anteriormente, él más importante es esa GPU. Como bien mencioné anteriormente, por una parte me ha facilitado mucho este trabajo gracias a la rapidez con que se realizaban los entrenamientos con respecto al tiempo empleado cuando se realizaba por CPU. Por otra parte, esos 4GB de memoria me han supuesto un inconveniente en el momento de entrenar con ciertos modelos, obligándome a modificar el tamaño de las imágenes, reduciendo su resolución casi a la

mitad y, modificando el número de imágenes usadas por bloque (esto lo explicaré mas adelante).

Un dato importante con respecto a las GPUs usadas para el Deep Learning es que se recomienda que sea de la marca NVIDIA. Esta marca nos facilita una librería para trabajar con estas herramientas como es CUDA.

4.3. Herramientas software

En esta sección se presentarán todas las herramientas software empleadas para la realización de este trabajo. Se describirán cada una de estas herramientas y su uso. En primer lugar, indicar que este TFG se ha desarrollo por completo en el sistema operativo Windows 10.

En libros y artículos de internet, donde he ido investigando sobre el Deep Learning, aconsejaban el uso de Linux. Al final me decidí por Windows, primero, porque es él que más he hido usando y con él que me siento más cómodo y segundo, porque al final las herramientas con las que voy a trabajar también estabán para Windows. A continuación, se presentarán todas estas herramientas empleadas para la resolución del TFG.

Anaconda

Anaconda es una Suite de código abierto que abarca una serie de aplicaciones, librerías y conceptos diseñados para el desarrollo de la Ciencia de datos con Python. Está orientada a simplificar el despliegue y administración de los paquetes de software. Una vez instalado, nos ofrece la posibilidad, a través de su propia consola de comandos, de crear nuestro propio entorno virtual donde poder instalar todas las librerías necesarias con las que se va a trabajar.

Es más recomendable trabajar a través de un entorno virtual, ya que una instalación en nativo puede desequilibrar los programas y configuraciones que tengamos en nuestro equipo. De esta forma, podremos crear tantos entornos virtuales como

queramos con diversas configuraciones.

Esta herramienta es muy recomendada para los que se inician en el mundo del Deep Learning ,ya que facilita la preparación de todo el entorno de trabajo necesario. Las diferentes versiones de los paquetes se administran mediante el sistema de gestión de paquetes Conda, el cual hace sencillo instalar, correr y actualizar el software de ciencia de datos y aprendizaje automático. Hay mucha información sobre Anaconda, en caso de necesitar ayuda en la resolución de alguna duda o problema.

La función en este trabajo ha sido simplemente la generación de un entorno de trabajo. Me ha permitido instalar todas las librerías necesarias para la creación de un modelo a través de las técnicas del Deep Learning.

El conjunto de comandos usados para la instalación del entorno son los siguientes:

1. *conda create -n tfgKerasGPU python=3.7*
2. *conda activate tfgKerasGPU*
3. *pip install tensorflow-gpu*
4. *pip install keras*
5. *pip install opencv-python*
6. *pip install matplotlib*
7. *pip install scipy*
8. *pip install pandas*
9. *pip install -U scikit-learn*

Pycharm

PyCharm es uno de los entornos de desarrollo más completos para Python. Es parte de una suite de herramientas de programación ofrecidas por JetBrains. Este IDE me ha facilitado el uso del entorno generado a través de Anaconda. Posteriormente, se han

ido creando cada una de las clases usadas para poder realizar los entrenamientos y su ejecución. Hace la función del típico IDE que se usa para programar.

Python

Python es un lenguaje de programación que ha entrado con mucha fuerza en el mundo del Deep Learning. En los comienzos del Deep Learning se hacía uso de lenguajes como R o C que dificultaba mucho la iniciación de cualquier persona en el mundo del Deep Learning hasta que se empezó a usasrse este lenguaje con la aparición de Keras, siendo este mucho más sencillo. La versión usada para este trabajo es la "V3.7". Es importante que la versión usada en Python sea la misma que la instalada en Anaconda.

Cuda

CUDA son las siglas de Compute Unified Device Architecture (Arquitectura Unificada de Dispositivos de Cómputo). Es un tipo de plataforma, desarrollada por NVIDIA, para poder realizar la programación de las tareas que antiguamente llevaban a cabo, inicialmente el procesador para, más adelante, el núcleo gráfico de la tarjeta. Permitirá que la tarjeta gráfica haga todo el cálculo matricial y demás, necesario para poder desarrollar Deep Learning.

Necesaria para cuando se quiere hacer uso de la GPU durante los entrenamiento de los modelos en caso de que se haga uso de una tarjeta gráfica NVIDIA como es este caso. Es recomendable tener actualizados los drivers de la gráfica y que estén en consonancia con la versión que se vaya a usar del Cuda.

cuDNN

La biblioteca NVIDIA CUDA Deep Neural Network (cuDNN) es una biblioteca de primitivas aceleradas por GPU para redes neuronales profundas. cuDNN proporciona implementaciones altamente optimizadas para rutinas estándar, como las capas de

convolución, agrupación, normalización y activación hacia adelante y hacia atrás. Son las librerías dedicadas en concreto para el Deep Learning.

Permite centrarse en la capacitación de redes neuronales y en el desarrollo de aplicaciones de software en lugar de dedicar tiempo al ajuste de rendimiento de GPU de bajo nivel.

Keras

Keras es una biblioteca de Redes Neuronales de Código Abierto escrita en Python. Es capaz de ejecutarse sobre diferentes motores de Deep Learning como TensorFlow, CNTK o Theano. En mi caso, Keras correrá sobre TensorFlow. Está especialmente diseñada para posibilitar la experimentación en poco tiempo con redes de Deep Learning. Otra principal ventaja que tiene es que es capaz de correr tanto en CPUs como en GPUs.

Keras contiene varias implementaciones de bloques constructivos de las redes neuronales como la capas de una red, las funciones de activación y optimizadores matemáticos.

Su autor principal y mantenedor ha sido el ingeniero de Google François Chollet, el cual tiene un libro bastante bueno para todo aquel que desee iniciarse en el mundo del Deep Learning[15]. He hecho un gran uso de este libro durante mi etapa de investigación para este trabajo. Esta biblioteca es la principal de todo este trabajo.

TensorFlow

TensorFlow es una biblioteca de código abierto para aprendizaje automático a través de un rango de tareas. Desarrollado por Google para satisfacer sus necesidades de sistemas capaces de construir y entrenar redes neuronales para detectar y descifrar patrones y correlaciones, análogos al aprendizaje y razonamiento usados por los humanos. En este trabajo actuará por debajo de Keras.

En 2011 fue el equipo de Google Brain quién se encargó de su creación para uso interno de Google hasta que fue liberado como software de código abierto en el 2015.

A continuación se mostrará una gráfica donde queda mucho mejor indicado donde se encuentra cada una de estas tecnologías que se han definido anteriormente:

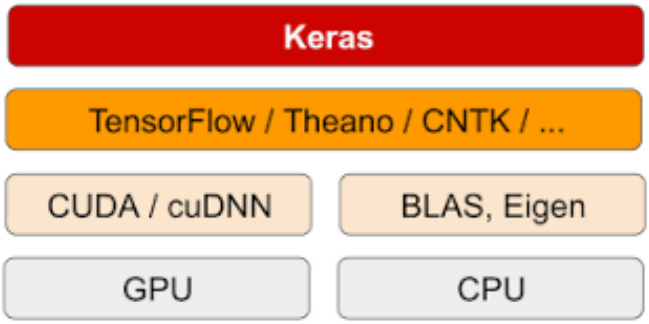

Figura 4.7: Gráfica Relación Entre Elementos Software, Parul Pandey

Otras Librerias

Otras de las librerías que han complementado este trabajo son las siguientes:

- **Numpy**: es una extensión de Python, que le agrega mayor soporte para vectores y matrices, constituyendo una biblioteca de funciones matemáticas de alto nivel para operar con esos vectores o matrices.
- **Matplotlib**: es una biblioteca para la generación de gráficos a partir de datos contenidos en listas o arrays en el lenguaje de programación Python y su extensión matemática NumPy.
- **os**: proporciona una forma portátil de utilizar la funcionalidad dependiente del sistema operativo. Puede leer o escribir un archivo, manipular rutas, leer todas las líneas de todos los archivos en la línea de comandos, ...
- **shutil**: este módulo ofrece una serie de operaciones de alto nivel en archivos y colecciones de archivos. En particular, se proporcionan funciones que admiten la copia y eliminación de archivos.

4.4. Redes Neuronales

Parte muy esencial en este trabajo. En primer lugar definiremos brevemente que es una red neuronal. El nombre, como podéis imaginar, viene de la idea de imitar el funcionamiento de las redes neuronales de los organismos vivos, un conjunto de neuronas conectadas entre sí y que trabajan en conjunto. Se basan en una idea sencilla: dados unos parámetros hay una forma de combinarlos para predecir un cierto resultado. Las redes neuronales son un modelo de computación para encontrar esa combinación de parámetros y aplicarla al mismo tiempo. En el lenguaje propio, encontrar la combinación que mejor se ajusta es entrenar la red neuronal. Una red ya entrenada se puede usar luego para hacer predicciones o clasificaciones.

4.4.1. Como funciona una Red Neuronal y sus Pesos

Para definir el funcionamiento de una Red Neuronal primero se definirá lo que es un perceptrón. Cuando se habla de perceptrón se refiere a la neurona artificial o unidad básica que se puede utilizar con otros tipos de perceptrones o de neuronas artificiales, para formar una red neuronal artificial más compleja.

Para entender bien cómo funciona una neurona vamos a ir con un ejemplo. Tenemos dos exámenes de los cuales tenemos dos notas diferentes.

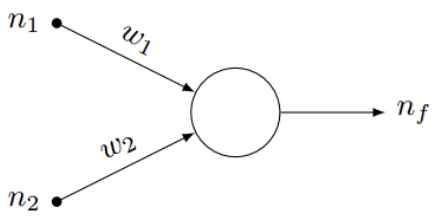

Figura 4.8: Perceptrón

En la imagen se muestra un perceptrón donde tenemos dos entradas representadas con las dos notas, n1 y n2, cada una con su correspondiente peso wn (lo que hay que

encontrar). La salida, nf, será 1 si está aprobado y 0 si está suspenso.

Un perceptrón puede tener varias entradas con un cierto peso cada una. Si la suma de esas entradas por cada peso es mayor que un determinado número, la salida del perceptrón es un uno. Si es menor, la salida es un cero.

En este ejemplo, las entradas serían las dos notas de los exámenes. Si la salida es uno, esto es, la suma de las notas por su peso correspondiente es mayor que cinco, es un aprobado. Si es cero, suspenso. Los pesos son lo que hay que encontrar con el entrenamiento. En este caso, el entrenamiento consistirá en empezar con dos pesos aleatorios, por ejemplo, 0.5 y 0.5, el mismo peso a cada examen, y ver qué resultado da la red neuronal para cada alumno. Si falla en algún caso, se irán ajustando los pesos poco a poco hasta que esté todo bien ajustado.

Se puede definir los pesos de una neurona como los coeficientes entrenables que permiten determinar en que medida es importante e influye cada variable de entrada en la salida. Además de los pesos de las entradas se añade siempre otro peso que se llama bias. La bias se utiliza para compensar offsets o desviaciones producidas por los valores de entrada y sus escalas.

4.4.2. Capas de una Red Neuronal

Por lo general, una red neuronal se compone de la siguiente estructura de capas:

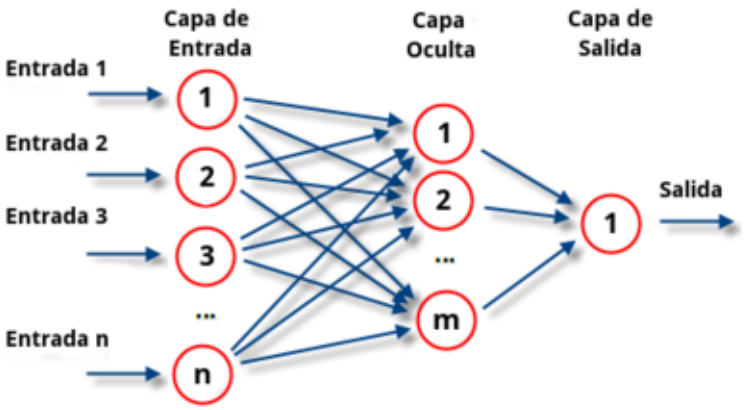

Figura 4.9: Partes de una Red Neuronal

- **Capa de entrada**

Esta capa es la que contiene las entradas de nuestro sistema, recibe información del exterior. Algunos las llaman capas expuestas o visibles. No es una capa que contenga neuronas clásicas o pesos, simplemente las entradas de nuestro sistema que pasan a través de la misma hacia la próxima capa.

- **Capas ocultas**

Las capas ocultas, se llaman así dado que no se puede entrar en contacto con ellas directamente, sino que tiene que ser a través de la capa de entrada. Son las encargadas de realizar el trabajo de la red. En un comienzo, los que en un principio se llamaban perceptrones multicapa clásicos, contenían una o dos capas ocultas que permitían desempeñar las tareas empleadas en ese momento. Pero a medida que han avanzado los años se ha buscado más profundidad, es decir, mayor número de capas ocultas y mayor número de neuronas, que supuso el inicio del Deep Learning.

- **Capa de salida**

La capa de salida es la encargada de expedir las salidas correspondientes a las entrenadas. Este tipo de capas son decisivas en el funcionamiento del sistema y la magia de las mismas yace en las activaciones que emplean. Dichas activaciones deben adecuarse al problema enfrentado. Un pequeño resumen de las activaciones necesarias en cada problema podría ser:

Clasificación binaria-activación sigmoidal: se busca que nuestro sistema dé un valor entre 0-1 que signifique el nivel porcentual de confianza de esa clasificación.

Clasificación categórica o multiclase-activación softmax: esta función es la encargada de gestionar las probabilidades de nuestras clases y normalizarlas con respecto al 100 %, de tal manera que la suma de las probabilidades es igual a 1 o al 100 %.

Regresiones-activación lineal: se busca que nuestro sistema tenga plena libertad para decidir el valor de salida.

4.4.3. Tipos de Redes Neuronales

Se van a distinguir dos tipo de Redes que se usan con Deep Learning:

- **Redes Neuronales Convolucionales, Convets o CNN**

Las Convets son un tipo de redes neuronales artificiales diseñadas para funcionar de forma muy similar a las neuronas de la corteza visual primaria de un cerebro humano. Estas han resultado ser ampliamente eficaces en tareas fundamentales de la visión artificial, como la clasificación y la segmentación de imágenes.

Dichas redes están formadas por múltiples capas de filtros convolucionales de una o más dimensiones, tras las cuales se insertan funciones no lineales de activación. Se hará uso de diversos tipos de capas y optimizadores que se desarrollarán más adelante.

En el caso de una clasificación clásica mediante una red convolucional, es posible encontrar dos fases bien delimitadas:

1. **Extracción de características:** Esta es la fase inicial y está compuesta principalmente por neuronas convolucionales que asemejan su procesado al de la corteza visual humana. Cuanto más se avanza a través del número de capas convolucionales menos reaccionan estas ante la variación de los datos de entrada y mayor es la abstracción alcanzada por las mismas para reconocer formas mas complejas.

2. **Clasificación:** Se basan en la utilización de capas "Densas" formadas por neuronas convencionales.

- **Redes Neuronales Recurrente o RNN**

Una red neuronal recurrente no tiene una estructura de capas definida, sino que permiten conexiones arbitrarias entre las neuronas, incluso pudiendo crear ciclos. Con esto se consigue crear la temporalidad, permitiendo que la red tenga memoria.

Las principales caracteristicas de una RNN son:

- Trata datos secuenciales de forma eficiente

- Recuerdan las salidas anteriores como entrada
- Pueden tratar secuencias muy largas, elemento a elemento

Las redes neuronales recurrentes son muy potentes para todo lo que tiene que ver con el análisis de secuencias, como puede ser el análisis de textos, sonido o video.

Existe multitud de tipos de redes neuronales recurrentes dependiendo del número de capas ocultas y la forma de realizar la retropropagación, a continuación se detallarán las más conocidas:

- **Redes recurrentes Simples – SRN o Elman**
- **Redes LSTM**
- **Redes GRU**

4.4.4. Redes Neuronales con Keras

La creación de estas redes neuronales nos la facilita en gran medida la biblioteca Keras con el uso de la librería "layers". Nos permite ir insertando todas la capas que se precisen para la creación de un modelo final.

También se puede hacer uso de la librería "applications" donde se nos permite obtener una serie de redes neuronales ya implementadas preparadas para su uso.

En particular, para este trabajo se hará uso de las **Redes Neuronales Convolucionales** también llamadas **Convets**. Estas son las indicadas para trabajos de clasificación de imágenes.

Capítulo 5

Implementación y desarrollo

En este capítulo se expondrá todo el proceso de desarrollo con las diferentes partes en que se ha dividido toda la implementación del trabajo.

Se informará de las diversas variables que se han usado para los diferentes entrenamientos ejecutados, con el fin de obtener el mejor resultado posible en estos.

En primer lugar, se definirá como se ha organizado todo el trabajo. Posteriormente se entrará más en detalle en cada una de las partes en las que se ha decidido dividirlo.

5.1. Organización del trabajo

Con la finalidad de tener un proyecto organizado donde poder ejecutar un entrenamiento modificando únicamente algunos de los parámetros que se usan, se ha creado un archivo python principal. Este archivo contendrá diversos bloques en los que se ha decicido dividir los entrenamientos. Así, modificando únicamente los valores que se necesiten, se podrá ejecutar un entrenamiento de manera rápida y sencilla, reutilizando siempre los mismo bloques y dando un mayor dinamismo al proyecto.

Estos bloques son los siguientes:

1. **Aumento de la cantidad de imágenes usadas (Opcional)**

2. **Preparación de las imágenes**

3. **Creación del modelo**

4. **Compilación del modelo**

5. **Entrenamiento del modelo**

6. **Resultados del entrenamiento**

Ví necesario realizar esta división porque veía que cada entrenamiento que quería realizar se me hacía muy pesado el crearlo desde cero. Había una pérdida de tiempo notable en ordenar todo el entrenamiento de manera segura. También afectaba el hecho de buscar y modificar aquellos parámetros que veía que podían mejorar el entrenamiento, ya que, en algunos casos, cuando se modifica una variable, se tienen que modificar otras forzosamente.

De esta manera tenía todos los parámetros más a mano. A parte, todos estos parámetros son comunes y se puede usar para la mayoría de los casos donde se requiera realizar un entrenamiento para la clasificación de imágenes.

Se explicará más en detalle cada una de estas partes en la siguiente sección.

Con respecto a los tipos de entrenamiento que se han realizado, se pueden dividir en 2 grupos. El primer grupo son redes neuronales que he ido obteniendo de libros, de cursos online o de algún artículo dedicado a la clasificación de imágenes con Deep Learning. Finalmente, analizando los resultados de todos estos, se ha creado una red neuronal de cero ,con el objetivo de poder sacar los mejores resultados posibles. Los dos libros usados son muy buenos para aquellas personas que quieran iniciarse en el mundo del Deep Learning [8][9]

El curso online usado está en la plataforma Udemy, especializada en cursos online. Este curso es *Deep Learning e Inteligencia artificial con Keras/Tensorflow*.

Un segundo grupo son las redes neuronales ya creadas, que las podemos obtener a través de la librería applications de Keras.

Por cada red neuronal que se ha probado, a su vez, se han usado cuatro conjuntos de datos o datasets distintos. Estos datasets son los siguientes:

IMÁGENES PARA ENTRENAMIENTO

Prueba	Edge	Smooth	Spiral	Other
Dataset 1	22.638	22.849	22.282	22.183
Dataset 2	2.000	2.000	2.000	2.000
Dataset 3	22.638	22.849	22.282	
Dataset 4	2.000	2.000	2.000	

Tabla 5.1: Tabla imágenes para Entrenamiento

IMÁGENES PARA VALIDACIÓN

Prueba	Edge	Smooth	Spiral	Other
Dataset 1	7.197	7.197	7.199	7.198
Dataset 2	300	300	300	300
Dataset 3	7.197	7.197	7.199	
Dataset 4	300	300	300	

Tabla 5.2: Tabla imágenes para Validación

IMÁGENES PARA TEST

Prueba	Edge	Smooth	Spiral	Other
Dataset 1	7.199	7.195	7.195	7.200
Dataset 2	300	300	300	300
Dataset 3	7.199	7.195	7.195	
Dataset 4	300	300	300	

Tabla 5.3: Tabla imágenes para Test

Se puede observar en cada tabla el conjunto de imágenes usadas para entrenamiento, validación y test en cada Dataset, por cada tipo de galaxia. Se parte de que todas las imágenes usadas se han obtenido del mismo Dataset.

El primer dataset contiene una gran cantidad de imágenes por cada tipo de galaxia. El dataset que se descargo de la página de Kaggle, no contenía tantas imágenes de cada

uno los tipos de galaxia. Tuve que hacer uso de una técnica usada en Deep Learning para aumentar el número de imágenes y así poder entrenar en igualdad de número de imágenes para todos los tipos de galaxias. En este caso, todas las imágenes del tipo Smooth son originales, llamando original las obtenidas desde un principio sin usar técnica ninguna. Para igualar esta cantidad con respecto al resto de galaxias, hice uso de la técnica Data Augmentation. Más adelante se explicará con mayor detalle. Esta forma de hacer uso de esta técnica solo se da para este conjunto de entrenamiento. En el conjunto usado para validación se ha empleado de forma equitativa el uso de la ampliación de imágenes para los cuatro tipos de galaxias. De igual modo se ha hecho para el conjunto de test.

El segundo dataset contiene solo fotos originales, no se ha hecho uso de la técnica de aumento de imágenes. Se ha usado esta cantidad de imágenes porque es el máximo que nos ofreció el conjunto de imágenes de tipo Edge. Se ha usado este dataset para ver si con menos imágenes pero todas originales el resultado es mejor o peor que en el caso anterior.

El tercer dataset contiene el mismo conjunto de imágenes que el primero pero eliminando las que pertenecen a la clase Other.

El cuarto dataset contiene el mismo conjunto de imágenes que el tercero pero eliminando las que pertenecen a la clase Other.

Estos dos últimos casos se han probado porque tenía la sensación de que al ser la clase Other un tipo de galaxia con características no definidas como son el resto de tipos, la captura de propiedades comunes entre estas no iba a existir. Una red neuronal siempre busca rasgos comunes entre ellos pero será difícil encontrarlos, ya que son imágenes muy diferentes entre sí.

Para cada una de las Redes Neuronales que se han probado, se han creado los siguientes archivo de ejecución, cada uno de los cuales usará uno de los dataset explicados anteriormente:

- **Dataset 1.** *Ent1_4Clases_ManyImages.py*
- **Dataset 2.** *Ent2_4Clases_FewImages.py*

- **Dataset 3.** *Ent3_3Clases_ManyImages.py*
- **Dataset 4.** *Ent4_3Clases_FewImages.py*

La estructura usada en el proyecto para la organización de estos archivos es la siguiente:

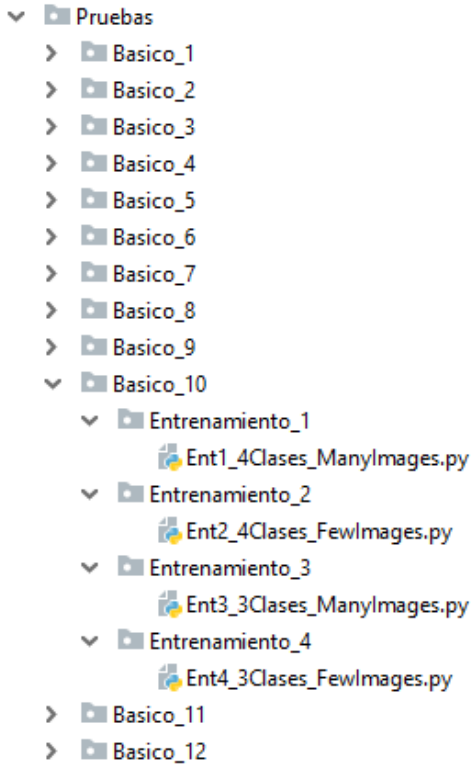

Figura 5.1: Estructura Proyecto Archivos Ejecución

Como se puede observar en la anterior imagen, dentro de la carpeta Pruebas es donde se encuentran todos los ejecutables de cada una de las redes neuronales usadas. En este caso solo se muestran las del primer bloque (redes neuronales sacadas de libros, cursos online y articulos de internet). Dentro de esta carpeta Pruebas tenemos varias carpetas con el nombre de la red neuronal que se ha probado. Por cada red, se han realizado cuatro entrenamiento, uno por cada dataset comentado anteriormente, de ahí, los cuatro archivos que encontramos dentro de cada red neuronal.

A continuación, se expondrá el contenido de estos archivos ejecutables explicando en detalle cada uno de los bloques que lo componen.

5.2. Bloques del trabajo

En esta sección se entrará más en detalle con los seis bloques que contiene un archivo de entrenamiento. Se indicarán aquellos parámetros que han sido fijos o modificados con respecto a las diferentes redes reuronales y los diversos datasets usados.

Cada uno de estos bloques están definidos en una clase python diferente. Como se observa en la siguiente imagen, se pueden ver las seis clases que representan a los seis bloques que se definirán dentro de la carpeta Módulos.

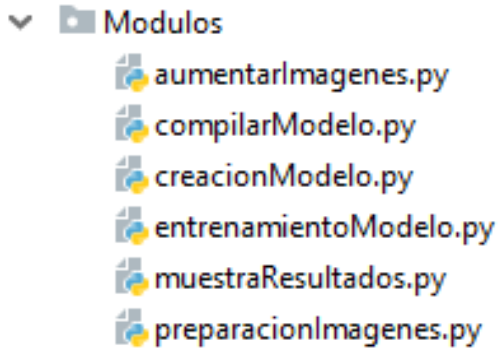

Figura 5.2: Estructura Proyecto Bloques de Trabajo

Para que se puedan usar cada una de estas clases en los archivos de ejecución principales se hará uso de las siguientes importaciones:

```
from TFG_Final.Modulos.aumentarImagenes import aumentarImagenes as aumIma
from TFG_Final.Modulos.preparacionImagenes import preparacionImagenes as preImg
from TFG_Final.Modulos.creacionModelo import creacionModelo as creMod
from TFG_Final.Modulos.compilarModelo import compilarModelo as comMod
from TFG_Final.Modulos.entrenamientoModelo import entrenamientoModelo as entMod
from TFG_Final.Modulos.muestraResultados import muestraResultados as mtrRes
```

Figura 5.3: Import de los Bloques de Trabajo

5.2.1. Aumento de la cantidad de imágenes usadas

Este bloque pertenece a la clase **aumentarImagenes.py** y es lo primero que se ejecuta. Es un bloque opcional, ya que solo se ejecutará en el caso de que se necesite

aumentar el tamaño de imágenes del conjunto de imágenes que se desee. Esta técnica se llama **Data Augmentation**. Para el uso de esta técnica me apoyo en la libreria Keras[16].

Por lo general, esta técnica sirve para aumentar el número de imágenes para la fase de entrenamiento en caso de que tengamos un dataset flojo. En mi caso, lo uso para poder igualar el número de imágenes de los diferentes tipos de galaxias y hacer un entrenamiento con una mayor equidad de imágenes de todos los tipos. Para conseguir esto lo que se hace es introducir perturbaciones en las nuevas imágenes obtenidas a partir de una imagen original.

Para el archivo de ejecución esta parte se ha divido en tres. La que afecta al conjunto de entrenamiento, la que afecta al conjunto de validación y la que afecta al conjunto de test.

```
path_train_Edge = 'C:/Users/lcostami/Desktop/TFG_Imagenes_4TiposGalaxias/train/edge'
path_train_Other = 'C:/Users/lcostami/Desktop/TFG_Imagenes_4TiposGalaxias/train/other'
path_train_Smooth = 'C:/Users/lcostami/Desktop/TFG_Imagenes_4TiposGalaxias/train/smooth'
path_train_Spiral = 'C:/Users/lcostami/Desktop/TFG_Imagenes_4TiposGalaxias/train/spiral'

numCopiasEdge = 15
numCopiasOther = 2
numCopiasSmooth = 0
numCopiasSpiral = 1

hacerAumento = 0

if hacerAumento == 1:

    if(numCopiasEdge > 0):
        aumentarImagen = aumIma()
        aumentarImagen.aumentarImagenes(path_train_Edge, numCopiasEdge)

    if(numCopiasOther > 0):
        aumentarImagen = aumIma()
        aumentarImagen.aumentarImagenes(path_train_Other, numCopiasOther)

    if(numCopiasSmooth > 0):
        aumentarImagen = aumIma()
        aumentarImagen.aumentarImagenes(path_train_Smooth, numCopiasSmooth)

    if(numCopiasSpiral > 0):
        aumentarImagen = aumIma()
        aumentarImagen.aumentarImagenes(path_train_Spiral, numCopiasSpiral)
```

Figura 5.4: Aumento Dataset Entrenamiento

```
path_validation_Edge = 'C:/Users/lcostami/Desktop/TFG_Imagenes_4TiposGalaxias/validation/edge'
path_validation_Other = 'C:/Users/lcostami/Desktop/TFG_Imagenes_4TiposGalaxias/validation/other'
path_validation_Smooth = 'C:/Users/lcostami/Desktop/TFG_Imagenes_4TiposGalaxias/validation/smooth'
path_validation_Spiral = 'C:/Users/lcostami/Desktop/TFG_Imagenes_4TiposGalaxias/validation/spiral'

numCopiasEdge = 11
numCopiasOther = 11
numCopiasSmooth = 11
numCopiasSpiral = 11

hacerAumentoValidacion = 0

if hacerAumentoValidacion == 1:

    if(numCopiasEdge > 0):
        aumentarImagen = aumIma()
        aumentarImagen.aumentarImagenes(path_validation_Edge, numCopiasEdge)

    if(numCopiasOther > 0):
        aumentarImagen = aumIma()
        aumentarImagen.aumentarImagenes(path_validation_Other, numCopiasOther)

    if(numCopiasSmooth > 0):
        aumentarImagen = aumIma()
        aumentarImagen.aumentarImagenes(path_validation_Smooth, numCopiasSmooth)

    if(numCopiasSpiral > 0):
        aumentarImagen = aumIma()
        aumentarImagen.aumentarImagenes(path_validation_Spiral, numCopiasSpiral)
```

Figura 5.5: Aumento Dataset Validación

```
path_test_Edge = 'C:/Users/lcostami/Desktop/TFG_Imagenes_4TiposGalaxias/test/edge'
path_test_Other = 'C:/Users/lcostami/Desktop/TFG_Imagenes_4TiposGalaxias/test/other'
path_test_Smooth = 'C:/Users/lcostami/Desktop/TFG_Imagenes_4TiposGalaxias/test/smooth'
path_test_Spiral = 'C:/Users/lcostami/Desktop/TFG_Imagenes_4TiposGalaxias/test/spiral'

numCopiasEdge = 11
numCopiasOther = 11
numCopiasSmooth = 11
numCopiasSpiral = 11

hacerAumentoTest = 0

if hacerAumentoTest == 1:

    if(numCopiasEdge > 0):
        aumentarImagen = aumIma()
        aumentarImagen.aumentarImagenes(path_test_Edge, numCopiasEdge)

    if(numCopiasOther > 0):
        aumentarImagen = aumIma()
        aumentarImagen.aumentarImagenes(path_test_Other, numCopiasOther)

    if(numCopiasSmooth > 0):
        aumentarImagen = aumIma()
        aumentarImagen.aumentarImagenes(path_test_Smooth, numCopiasSmooth)

    if(numCopiasSpiral > 0):
        aumentarImagen = aumIma()
        aumentarImagen.aumentarImagenes(path_test_Spiral, numCopiasSpiral)
```

Figura 5.6: Aumento Dataset Test

Para cada una de las tres partes vienen definidas las rutas donde se encuentra cada conjunto de imágenes por cada uno de los tipos de galaxias, el número de copias por imagen que se quiere realizar en cada tipo de galaxia y si se desea que durante la ejecución se realice este aumento o no.

Como se puede observar en las imágenes, se hace uso del método aumentarImagenes para realizar dicha acción. Método que se encuentra en la clase AumentarImagenes.py.

Para realizar la técnica de Data Augmentation se hace uso de la librería ImageDataGenerator que la encontramos dentro de Keras. Se insertan las distintas deformaciones que irán sufriendo la imágenes para así poder obtener otra imagen que parezca diferente a la original.

```
self.datagen = ImageDataGenerator(
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest')
```

Figura 5.7: Constructor del ImageDataGenerator

En este caso, los datos insertados quieren decir lo siguiente:

- **rotation_range**, rota la imagen 40º
- **width_shift_range**, rango modificación de ancho
- **height_shift_range**, rango modificación de alto
- **shear_range**, rango de corte
- **zoom_range**, rango de zoom
- **horizontal_flip**, se pueden voltear las imágenes horizontalmente
- **fill_mode**, valor nearest predeterminado

Posteriormente, a través de un for, se van generando las nuevas imágenes. Se hace uso del metodo flow donde se le indicará la imagen a tratar, la ruta donde se guardará la nueva imagen, el nombre de la nueva imagen y su formato. Es necesario realizar un break ya que sino este bucle nunca pararía.

```
for batch in self.datagen.flow(x, batch_size=1, save_to_dir=path,
                               save_prefix=nameFile, save_format='jpg'):
    i += 1
    if i % numCopias == 0:
        break
```

Figura 5.8: Bucle para generación de Imágenes

Un ejemplo de como quedan las imagenes es el siguiente:

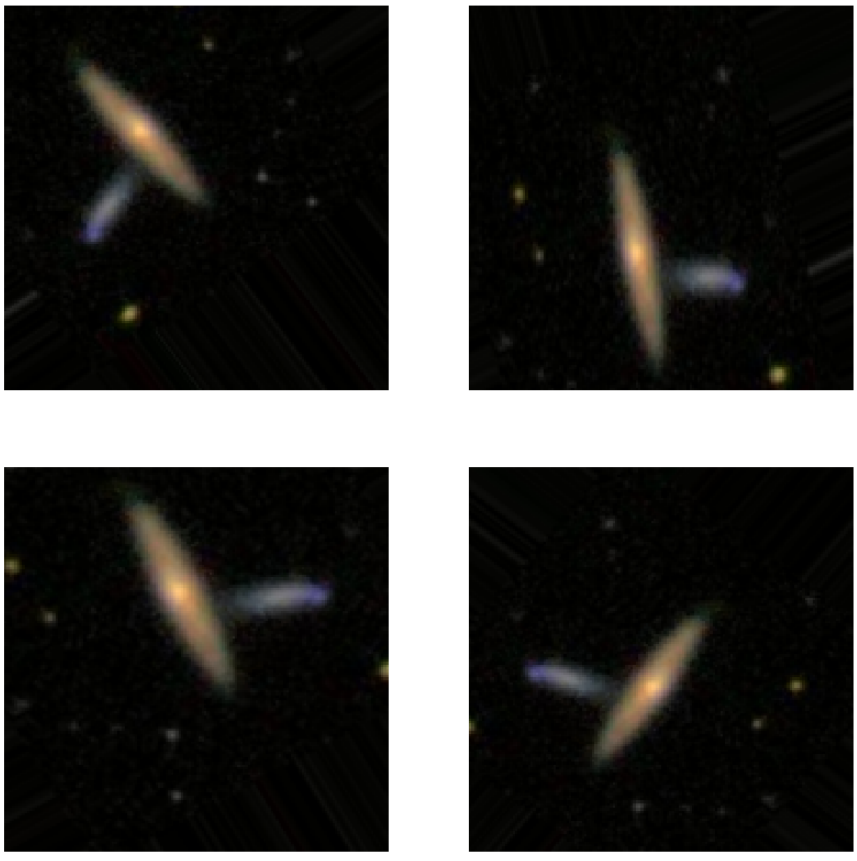

Figura 5.9: Ejemplo de la técnica Data Augmentation

Vemos como va modificando la imágenes de diversas formas para así conseguir distintas imágenes a partir de una misma imagen original.

5.2.2. Preparación de las imágenes para el entrenamiento

En este bloque se preparará el conjunto de imágenes que se va a usar durante el entrenamiento y posteriormente para las pruebas de Test. En primer lugar, se definirán los atributos usados en este bloque para después entrar en mayor detalle de lo que se hace internamente. La clase usada para la formación de este bloque es **preparacionImagenes.py**.

A continuación, muestro un ejemplo de esta sección en el archivo ejecutable:

```
#########################################################################################################
                        # PREPARACION DE LAS IMAGENES PARA ENTRENAMIENTO
#########################################################################################################

path_Train = 'C:/Users/lcostami/Desktop/TFG_Imagenes_3TiposGalaxias/train'
path_Validation = 'C:/Users/lcostami/Desktop/TFG_Imagenes_3TiposGalaxias/validation'
path_Test = 'C:/Users/lcostami/Desktop/TFG_Imagenes_3TiposGalaxias/test'

filas = 300
columnas = 300
batchSize = 40

preparacionImagenes = preImg()
preparacionImagenes.definirGeneradores(path_Train,
                                       path_Validation,
                                       path_Test,
                                       filas,
                                       columnas,
                                       batchSize)
```

Figura 5.10: Preparación Imágenes Para Entrenamiento

Los primeros parámetros usados para este bloque son las rutas donde se encuentran las imágenes que se van a usar, tanto para el entrenamiento como para la validación y para el test. Estos valores irán cambiando según se vayan usando los distintos datasets comentados anteriormente.

Después se encuentrán los parámetros que afectan al tamaño de la imagen indicando el número de filas y de columnas. En un principio, las imágenes usadas tienen un tamaño de 300x300 pero hay algunas redes neuronales donde es imposible usar ese tamaño. Si se trabajará en esos casos con estas dimensiones, se superaría la cantidad de memoria libre que hay en la GPU y nos daría error. Este factor es determinante ya que dificulta el obtener detalles que se podrían perder al cambiar la resolución de las imágenes.

Otro de los parámetros a modificar es el batchSize. Este parámetro indica el número de imágenes que se entrenarán de forma conjunta durante el entrenamiento. Cuanto mas alto sea este valor mejor será el entrenamiento. Por lo general, debido a que el entrenamiento con imágenes ocupa mucho espacio en memoria, el número suele oscilar entre 10 y 40.

Hay que ir jugando con estos parámetros para obtener el equilibrio perfecto y así poder obtener un buen resultado en el entrenamiento. Es de los atributos que más he modificado para obtener los mejores resultados dado un mismo modelo o red neuronal.

Entrando más en detalle, se mostrará como se tratan estas imágenes dentro de la clase preparacionImagenes.py. Se definirán tres ImageDataGenerator, uno para el conjunto de entrenamiento, otro para el de validación y otro para el de test. Estos generadores de lotes contendrán las imágenes de cada parte con las características que se han definido anteriormente a través de los atributos.

Al igual que ocurre con el bloque del aumento de imágenes, aquí también se hace uso de la librería ImageDataGenerator de Keras[16]. En primer lugar se definen los constructores. Crearemos uno para el conjunto de entrenamiento (train_datagen) y el otro para los conjuntos de validación y test (test_datagen). Aunque el atributo train_datagen tenga más parámetros definidos, en este caso no van a cambiar el estado de las imágenes por lo que no tendrá repercusión ninguna en el resultado final. Se puede decir que los dos atributos, tanto train_datagen como test_datagen son iguales en la práctica final.

```
self.train_datagen = ImageDataGenerator(
    rescale=1. / 255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True)

self.test_datagen = ImageDataGenerator(
    rescale=1. / 255)
```

Figura 5.11: Constructores del ImageDataGenerator

Una vez definidos los constructores, se hace uso del método flow_from_directory. A este método se le pasa como parámetros de entrada la ruta del directorio donde estan todas las imágenes para el entrenamiento o para la validación o para el test. También se le pasa el tamaño que queremos tengan finalmente las imágenes, tanto la altura como la anchura. El batch_size que fue definido anteriormente para este bloque y la class_mode será siempre categorical. Este último parámetro determina el tipo de arrays de etiquetas que se devuelven y es de tipo categorical porque es el que se usa para clasificaciones en 2D como esta.

Es importante que, en el momento que se le indique la ruta donde se encuentran todas las imágenes, dentro de esa carpeta haya otras carpetas con el nombre de cada uno de los tipos que se quieran clasificar. Este nombre de las carpetas es el que se usará para el arrays de las etiquetas, para así poder diferenciar por cada imagen cuando un resultado pertenece a un tipo de galaxia u otro. Dentro de estas carpetas es donde estarán las imágenes de cada uno de los tipos a clasificar.

Como se oberva en la siguiente imagen, habrá tres generadores de lotes diferentes, el de entrenamiento, validación y test.

```
self.train_generator = self.train_datagen.flow_from_directory(
    train_dir,  # Directorio de destino
    target_size=(filas, columnas),  # Redimensiona todas las imágenes a filas * columnas
    batch_size=batchSize,
    class_mode='categorical')

self.validation_generator = self.test_datagen.flow_from_directory(
    validation_dir,
    target_size=(filas, columnas),
    batch_size=batchSize,
    class_mode='categorical')

self.test_generator = self.test_datagen.flow_from_directory(
    test_dir,
    target_size=(filas, columnas),
    batch_size=batchSize,
    class_mode='categorical')
```

Figura 5.12: Definición de los conjuntos de imágenes finales

5.2.3. Creación del modelo

En este bloque se indica que tipo de red neuronal se va a usar, que tipo de modelo se va a crear. La clase usada para la formación de este bloque es **creacionModelo.py**. Esta clase contiene todas las redes neuronales con las que se han hecho las pruebas de clasificación.

A continuacion, muestro un ejemplo de esta sección en el archivo ejecutable:

```
#########################################################################################################
                                        #CREACION DEL MODELO
#########################################################################################################

numClases = 4
canales = 3

# 1 -> vemos el summary del modelo
# 0 -> no vemos el summary del modelo
viewSummary = 1

modelType = 'Basico_1'

creacionModelo = creMod(numClases, filas, columnas, canales)
creacionModelo.createModel(modelType, viewSummary)
```

Figura 5.13: Creación del Modelo

Los atributos a indicar son el numClases que indica, en este caso, el número de tipos de galaxias que vamos a diferenciar durante el entrenamiento. Como bien se ha definido anteriormente, son los tipos de galaxias según su morfología. En algunos entrenamientos usaremos 4 tipos mientras que en otros solo 3, despreciando las imágenes que son galaxias de tipo Other.

El atributo canales sirve para informar que las imágenes a procesar son en color y no en blanco y negro, hay que hacer uso del RGB. Este atributo será siempre igual para todos los entrenamientos ya que todas las imágenes son a color.

El atributo viewSummary sirve para mostrar el conjunto de capas que forman la red neuronal junto con los parámetros que se usarán por cada capa. Cuando su valor es 1 muestra esta información por consola y cuando su valor es 0 no se mostrará. Este valor siempre será fijo para todos los entrenamientos ejecutados.

Por último tenemos el atributo modelType donde se informa del tipo de red

Neuronal que se va a usar para el entrenamiento. Se indicará a través del nombre de la red neuronal a usar en ese momento.

La clase creacionModelo contiene todas las redes que se han usado en la búsqueda de la mejor clasificación de imágenes de galaxias posible.

Para la creación de las redes neuronales se han usado un conjunto de capas, cada una de las cuales ha tenido su función dentro de esta red. Para la definición de las capas se ha usado la librería layers que se encuentra en la biblioteca de Keras.

Algunas de las capas más usadas y principales para este tipo de trabajos son las siguientes [17]:

Capas convolucionales

Las capas convolucionales operan sobre los datos de entrada mediante el cálculo de convoluciones discretas con bancos de filtros finitos.

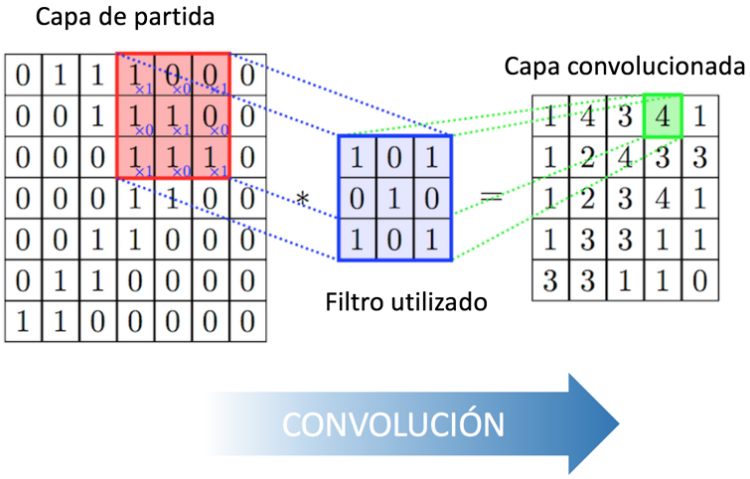

Figura 5.14: Capa de Convolución, Diego Calvo

En este tipo de capas, las operaciones de convolución permiten obtener características dominantes de la imagen de entrada relacionadas con los objetivos de entrenamiento. De forma experimental se observa que las primeras capas en redes de convolucionales se centran en la búsqueda de características simples, como podrían ser bordes, esquinas o regiones. A medida que se avanza hacia capas más profundas,

se aumenta el nivel de abstracción del contenido de la imagen al que se muestran sensibles.

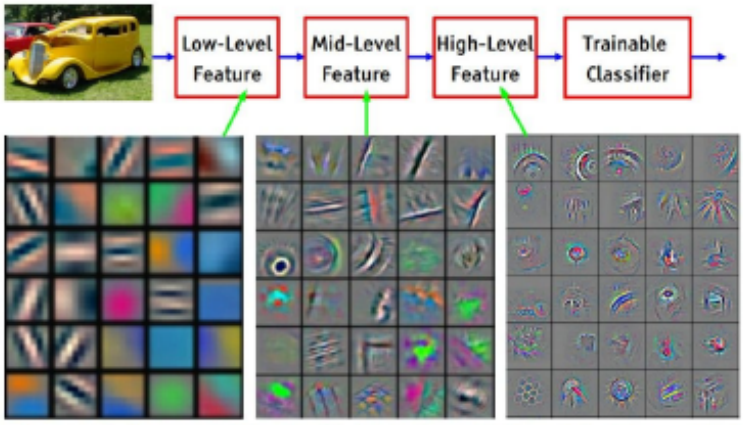

Figura 5.15: Ejemplo de Filtros a Diferentes Niveles de Abstracción, Jorge Rodríguez Araújo

Para la mayoría de las redes usadas, la capa Convolucional que se ha empleado es la Conv2D. Un ejemplo de como se define esta capa se puede ver en la siguiente imagen.

```
self.modelo.add(layers.Conv2D(32, kernel_size=(3, 3), activation='linear', padding='same',
                              input_shape=(self.filas, self.columnas, self.canales)))
```

Figura 5.16: Capa Conv2D

Capas Densas

Este tipo de capas están representadas por las neuronas clásicas empleadas en los ya conocidos perceptrones. Su función suele ser principalmente la de completar el clasificador final, que será el encargado de pasar de mapas de características a valores concretos en función del objetivo de la red (clasificación o regresión).

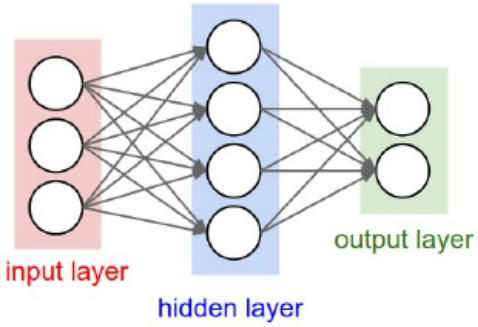

Figura 5.17: Ejemplo de Capa Densa, Hamza Bendemra

Para esta capa se ha empleado la Dense. Un ejemplo de esta capa se puede ver en la siguiente imagen.

```
self.modelo.add(layers.Dense(self.numClases, activation='softmax'))
```

Figura 5.18: Ejemplo de Capa Densa en keras

Capas de Activación

Estas capas son las encargadas de aportar no linealidad a las funciones generadas por las redes neuronales y de agregar las activaciones de múltiples capas en la salida de la red.

Estas capas se aplican como atributo al resto de capas que forman la red neuronal. Se puede observar en los diversos ejemplos introducidos en cada una de la capas que se han explicado con el nombre de activation.

Las capas de activación usadas en las diferentes redes neuronales son las siguientes:

Lineal: La función lineal es bastante conocida por ser empleada en problemas de regresión y se encuentran generalmente en la salida de la red. Un ejemplo:

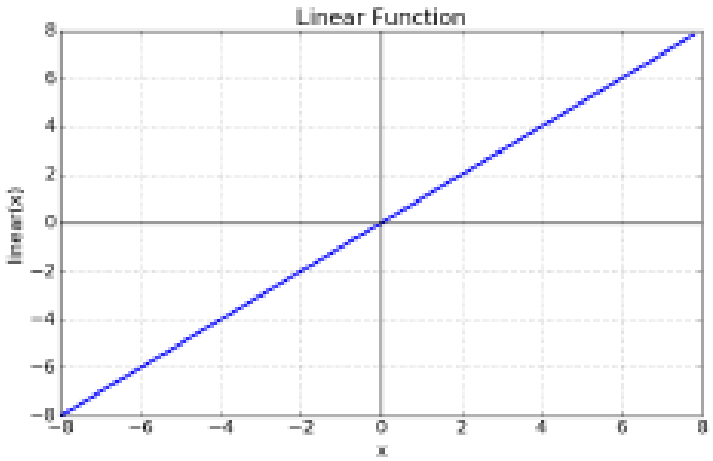

Figura 5.19: Activación Lineal

Unidad Lineal Rectificada o Relu: La función Relu es una función de activación muy utilizada en las redes neuronales actuales. Esto es debido a que se demuestra experimentalmente que dicho tipo de activaciones permiten redes más profundas y facilitan el entrenamiento de las mismas. Un Ejemplo:

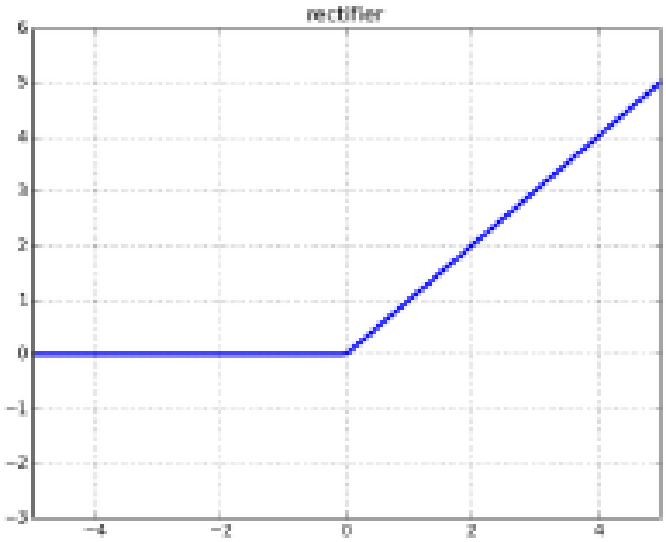

Figura 5.20: Activación Relu

Softmax: La función softmax es una extensión de la clásica función logística, empleada principalmente para la clasificación multiclase.

Max-Pooling

El filtro max-pooling es una forma de reducción del volumen de salida de las capas convolucionales de la CNN y que permite además incrementar el campo de percepción de la red.

Ejemplo de matriz 4x4 en la cual se realiza un maxpooling de 2x2 con un stride de 2.

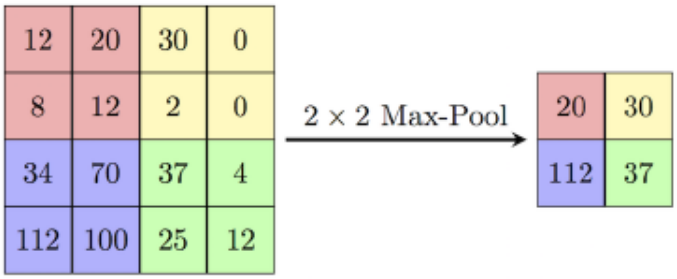

Figura 5.21: Ejemplo Matriz Max-Pooling, FirelordPhoenix

Ejemplo visual de la actuación de una capa Max-Pooling

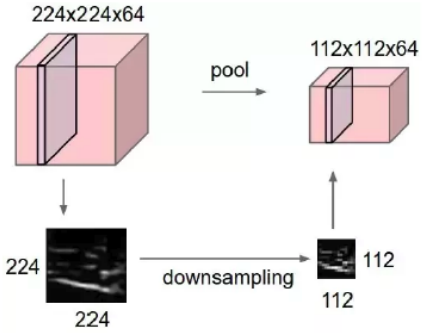

Figura 5.22: Ejemplo Max-Pooling, FirelordPhoenix

Para este trabajo se ha empleado la MaxPooling2D. Un ejemplo de esta capa se puede ver en la siguiente imagen.

```
self.modelo.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
```

Figura 5.23: Ejemplo Max-Pooling en Keras

Flatten

Una capa flatten consigue llevar todas las dimensiones a una sola. Convierte nuestra imagen en un array de valores. Pasamos de tensor 3D a tensor 1D.

```
self.modelo.add(layers.Flatten())
```

Figura 5.24: Ejemplo Capa Flatten en Keras

Dropout

Capa de regularización. Consigue que las neuronas sean independientes y que no se interrelacionen entre sí. El valor que se le indique quiere decir que hay un tanto por ciento de probabilidad de que el Dropout cancele una neurona. Capa aconsejable para evitar el sobreentrenamiento u overfiting.

```
self.modelo.add(layers.Dropout(0.5))
```

Figura 5.25: Ejemplo Capa Dropout en Keras

5.2.4. Compilación del modelo

Este bloque es el paso previo para comenzar el entrenamiento y será donde se ejecuta el método compile del modelo creado anteriormente. La clase usada para la formación de este bloque es **compilarModelo.py**.

A continuación, muestro un ejemplo de esta sección en el archivo ejecutable:

```
#########################################################################################################
                                        #COMPILACION DEL MODELO
#########################################################################################################

# Adam -> gradiente de tipo Adam
# SGD -> gradiente de tipo SGD
# Adagrad -> gradiente de tipo Adagrad
typeGradient = "Adam"

# 1 -> categorical_accuracy
# 2 -> accuracy
typeMetric = "categorical_accuracy"

compilaModel = comMod(creacionModelo.modelo, typeMetric)
compilaModel.compileModel(typeGradient)
```

Figura 5.26: Compilación del Modelo

El atributo typeGradient informará del tipo de Gradiente Estocástico a usar para el entrenamiento. Es el optimizador usado para el entrenamiento. El código esta preparado para hacer uso de los siguientes tipos:

- **Adam**
- **SGD**
- **Adagrad**

Con respecto a los optimizadores, encontramos una gran variedad de ellos a usar en las redes convolucionales. Tanto el Adam como el Adagrad derivan del algoritmo de descenso por gradiente estocástico SGD. En este algoritmo se seleccionan conjuntos de entrenamiento de pequeño tamaño de forma aleatoria, también conocidos como batchs. Estos son utilizados para realizar una iteración de descenso por gradiente para minimizar la función de pérdidas del entrenamiento. El algoritmo SGD tiene como principal parámetro a ajustar la magnitud del descenso o learning rate.

A partir de SGD se han desarrollado otros métodos que varían dinámicamente el parámetro learning rate durante el entrenamiento. Dentro de los optimizadores adaptativos clásicos encontramos Adagrad, que a posteriori derivaron en Adam [17].

En este trabajo el usado finalmente para todos los entrenamientos es Adam, ya que sus características adaptativas y los resultados reportados por diferentes benchmarks de optimización lo sitúan como un buen candidato para el entrenamiento de redes neuronales. Con respecto a las diferentes pruebas realizados por cada uno de los optimizadores, la diferencia de resultados de Adam con respecto al resto ha sido considerable.

Las ventajas de Adam estan en la simplicidad de su implementación, su eficiencia computacional y su buen funcionamiento en problemas con gran número de datos y parámetros.

El atributo typeMetric es el tipo de métrica a usar e indica como se obtendrá la precisión en relación a los resultados obtenidos del entrenamiento.

La formula usada es la siguiente:

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

Figura 5.27: Fórmula accuracy

donde VP son los verdaderos positivos y FP son los falsos positivos.

En el método compile se indicarán el tipo de pérdida que en este caso siempre será categorical crossentropy. Esto es así ya que es la usada para entrenamientos de clasificación multiclase. El optimizador y el tipo de métrica se indicarán por parámetro en la clase principal.

```
self.modelo.compile(loss=keras.losses.categorical_crossentropy,
                    optimizer=self.descenso_gradiente_estocastico,
                    metrics=[self.typeMetrics])
```

Figura 5.28: Ejemplo método compile en Keras

5.2.5. Entrenamiento del modelo

En este bloque es donde ya se comienza el entrenamiento. La clase usada para la formación de este bloque es **entrenamientoModelo.py**.

A continuación, muestro un ejemplo de este bloque en el archivo ejecutable:

```
#############################################################################################################
                                        #ENTRENAMIENTO DEL MODELO
#############################################################################################################

trainGenerator = preparacionImagenes.train_generator
validationGenerator = preparacionImagenes.validation_generator
testGenerator = preparacionImagenes.test_generator

epocas = 10
stepsPerEpoch = 140
validationSteps = 30
stepsEvaluacion = 30
verbose = 1

trainModel = entMod(compilaModel.modelo)
trainModel.ejecutarEntrenamientoYValidacion(trainGenerator, validationGenerator, stepsPerEpoch, epocas, validationSteps
                                            ,testGenerator, stepsEvaluacion, verbose)
```

Figura 5.29: Entrenamiento del Modelo

Los generadores de lotes que se usarán en este bloque serán los mismos que se obtuvieron durante le ejecución del segundo bloque. Se usarán el de entrenamiento, validación y test en este mismo bloque.

Se deberá indicar el número de épocas que desea se ejecute durante el entrenamiento, es decir, el número de veces que las imágenes pasarán por la red neuronal.

Los atributos steps indicarán el número de batchSize que se usarán por cada época. Es decir, si tenemos un step en entrenamiento de 140 y un batch size de 40, el número de imágenes que se procesarán en esa epoca será de 140*40. De esta forma se controla el número de imágenes que deseamos se procese por época sin tener la necesidad de procesar todas las imágenes que contenga el generador de lotes creado.

Por último, hay informar al atributo verbose. Este sirve para indicar, miestras se esta realizando el entrenamiento, si se desea que se vaya mostrando la evolución de este (valor 1) o simplemente informar al final de cada epoca (valor 2) o directamente no informar (valor 0).

Dentro de este bloque, en el momento de llamar al metodo ejecutarEntrenamientoYValidacion se procederá a realizar el entrenamiento de las imágenes y un posterior test para verificar como ha ido ese entrenamiento.

En primer lugar, se ejecuta el metodo fit_generator. Este método es único para cuando se usan generadores de lotes a través de la librería ImageDataGenerator. En caso de que no se quisieran udar lotes se haría uso del metodo fit. Se le informará como parámetros de entrada el generador de lotes del entrenamiento y validación. También se informará de sus respectivos steps, de las épocas que se desean ejecutar y el verbose. Una vez termina el entrenamiento, se guarda toda la información de este en la variable history.

```
self.history = self.modelo.fit_generator(trainGeneratorParam,
                                          steps_per_epoch=stepsPerEpochParam,
                                          epochs=epocasParam,
                                          verbose=verboseParam,
                                          validation_data=validationGeneratorParam,
                                          validation_steps=validationStepsParam)
```

Figura 5.30: Ejemplo Función Fit Generator en Keras

Posteriormente, se procede a realizar la evaluación del entrenamiento realizado. Se hace uso del método evaluate_generator, informando a este del generador de lotes de test junto con su step y el verbose. Este método nos dará un porcentaje indicando los aciertos con las imágenes de test.

```
self.puntuacion = self.modelo.evaluate_generator(generator=testGeneratorParam,
                                                  steps=stepsEvaluacion,
                                                  verbose=verboseParam)
```

Figura 5.31: Ejemplo Función Evaluate Generator en Keras

5.2.6. Resultados del entrenamiento

Último bloque a ejecutar. Mostrará los resultados de como ha ido el entrenamiento por cada una de las épocas que se han ejecutado a través de una gráfica. La clase usada para la formación de este bloque es **muestraResultados.py**.

A continuación, muestro un ejemplo de este bloque en el archivo ejecutable:

```
#############################################################################################################
                                        #RESULTADOS DEL ENTRENAMIENTO
#############################################################################################################

nombreModelo = 'Ejemplo_1_Generadores_2'

if typeMetric == 'accuracy':
    ejeX = 'acc'
    ejeY = 'val_acc'
elif typeMetric == 'categorical_accuracy':
    ejeX = 'categorical_accuracy'
    ejeY = 'val_categorical_accuracy'

resultados = mtrRes(trainModel.history, trainModel.puntuacion, trainModel.modelo)
resultados.mostrarResultados(ejeX, ejeY)
resultados.guardarModelo(nombreModelo)
```

Figura 5.32: Resultados del entrenamiento

En primer lugar, mostrará un par de gráficas. Estas nos darán una relación de la evolución de la Precisión por época y de las Pérdidas por época. Estas gráficas son bastante buenas ya que se refleja perfectamente si una red no dará para más o si se pueden obtener mejores resultados. Esta información la obtiene del atributo history, siendo este atributo el resultado del método fit_generator.

Tambien mostrará en consola los resultados de la etapa de test. El porcentaje de acierto final y la pérdida.

El atributo nombreModelo informará del nombre que se quiere poner al modelo generado tras el entrenamiento y que se guardará en la misma carpeta donde se encuentra el ejecutable.

Capítulo 6

Resultados

En este capítulo se expondrán un conjunto de resultados de aquellos entrenamientos ejecutados a lo largo de este trabajo y que han generado modelos con los resultados más interesantes. Este conjunto de resultados se dividen en dos grandes grupos:

1. Entrenamientos de redes neuronales obtenidos a través de libros, artículos en internet o cursos online de Deep Learning. Una vez investigado lo mejor de cada una de estas redes, se ha creada una red neuronal de cero.

2. Entrenamientos de redes neuronales obtenidas a través de la biblioteca ***applications*** de Keras.

Cada red neuronal tiene sus propias características y lógicamente, los atributos en algunos de estos casos no serán iguales. Pero hay otros atríbutos donde siempre se realizarán los entrenamientos con los mismos valores. A continuación, se expondrá una tabla con estos atributos y sus respectivos valores:

canales	**viewSummary**	**typeGradient**	**typeMetric**	**verbose**
3	1	Adam	categorical_accuracy	2

Tabla 6.1: Parámetros comunes en los Entrenamientos para todas las Redes Neuronales

Por cada conjunto de entrenamiento expuesto anteriormente, solo se informará aquellos cuyos resultados sean los más interesantes.

6.1. 1º Conjunto de Entrenamientos

En este primer conjunto se mostrarán los resultados obtenidos de una red neuronal creada de cero. Después de haber probado con una cantidad considerable de redes neuronales obtenidas de libros, curso online y artículos vistos por internet, junto con los conocimientos adquiridos durante el tiempo de investigación, el resultado ha sido el siguiente:

```
# ******CAPA DE CLASIFICACION********
# Hace uso de bancos de convoluciones y combinaciones de MaxPooling
self.modelo = models.Sequential()

self.modelo.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(self.filas, self.columnas, self.canales)))
self.modelo.add(layers.MaxPooling2D((2, 2)))

self.modelo.add(layers.Conv2D(64, (3, 3), activation='relu'))
self.modelo.add(layers.MaxPooling2D((2, 2)))

self.modelo.add(layers.Conv2D(64, (3, 3), activation='relu'))
self.modelo.add(layers.MaxPooling2D((2, 2)))

self.modelo.add(layers.Conv2D(128, (3, 3), activation='relu'))
self.modelo.add(layers.MaxPooling2D((2, 2)))

self.modelo.add(layers.Conv2D(128, (3, 3), activation='relu'))
self.modelo.add(layers.MaxPooling2D((2, 2)))

# ******GENERACION DEL MODELO********
self.modelo.add(layers.Flatten())
self.modelo.add(layers.Dense(256, activation='relu'))
self.modelo.add(layers.Dropout(0.5))
self.modelo.add(layers.Dense(128, activation='relu'))
self.modelo.add(layers.Dropout(0.5))
self.modelo.add(layers.Dense(self.numClases, activation='softmax'))
```

Figura 6.1: Red Neuronal Creada desde Cero

Como se muestra en la imagen, esta red tiene una parte de clasificación y otra de generación del modelo. La parte de clasificación es una combinación de capas convolucionales **Conv2D** y **MaxPooling**. Estas capas van aumentado progresivamente el número de neuronas con el fin de poder profundizar cada vez más en las imágenes a entrenar y así obtener mayor detalle de estas. Estas capas tienen una activación Relu.

En la parte de Generación del Modelo se comienza con una capa Flatten para tratar mejor la información y se crean combinaciones de capas **Dense** y **DropOut**.

En las capas Dense se va disminuyendo progresivamente el número de neuronas a usar con activación relu. Las capas Dropout me valen para evitar el sobreentrenamiento u overfiting, es decir, que la precisión de entrenamiento sea mucho mayor que la de validación. Terminamos con una capa Dense con la misma cantidad de neurones como tipos de galaxias se están entrenando en ese momento. La activación usada es la de softmax, usada para las clasificaciones multiclase.

Esta Red Neuronal, durante la ejecución de un entrenamiento hace uso de una pequeña cantidad de parámetros lo que resulta beneficioso para el uso de imágenes más amplias y tener conjuntos de trabajo o batchsize más numerosos, mejorando así los resultados finales y resolviendo los problemas de memoria en la GPU que se han ido informando en capítulos anteriores.

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 298, 298, 32)	896
max_pooling2d_1 (MaxPooling2	(None, 149, 149, 32)	0
conv2d_2 (Conv2D)	(None, 147, 147, 64)	18496
max_pooling2d_2 (MaxPooling2	(None, 73, 73, 64)	0
conv2d_3 (Conv2D)	(None, 71, 71, 64)	36928
max_pooling2d_3 (MaxPooling2	(None, 35, 35, 64)	0
conv2d_4 (Conv2D)	(None, 33, 33, 128)	73856
max_pooling2d_4 (MaxPooling2	(None, 16, 16, 128)	0
conv2d_5 (Conv2D)	(None, 14, 14, 128)	147584
max_pooling2d_5 (MaxPooling2	(None, 7, 7, 128)	0
flatten_1 (Flatten)	(None, 6272)	0
dense_1 (Dense)	(None, 256)	1605888
dropout_1 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 128)	32896
dropout_2 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 3)	387

Total params: 1,916,931
Trainable params: 1,916,931
Non-trainable params: 0

Figura 6.2: Capas y Parámetros de la Red Neuronal

En total, se hace uso de 1.916.931 parámetros, lo que resulta bastante poco para un entrenamiento con imágenes. Se puede ver el número de parámetros usados por cada capa.

Con respecto a los resultados obtenidos en los 4 conjuntos de entrenamientos que se ha diferenciado, se han encontrado resultados bastantes diferentes y por lo general no muy buenos. Se irán exponiendo uno a uno.

6.1.1. Dataset 1 - Muchas Imágenes con 4 clases de galaxias

En la siguiente imagen se puede observar como ha ido evolucionando la precisión del entrenamiento y validación a lo largo de las 20 épocas en las que se ha hecho el entrenamiento. También se observa la evolución de las pérdidas en cada época.

```
 - 499s - loss: 1.0019 - categorical_accuracy: 0.5513 - val_loss: 1.5574 - val_categorical_accuracy: 0.4717
Epoch 2/20
 - 494s - loss: 0.8299 - categorical_accuracy: 0.6435 - val_loss: 2.3240 - val_categorical_accuracy: 0.5006
Epoch 3/20
 - 494s - loss: 0.7759 - categorical_accuracy: 0.6712 - val_loss: 2.0520 - val_categorical_accuracy: 0.5106
Epoch 4/20
 - 493s - loss: 0.7479 - categorical_accuracy: 0.6866 - val_loss: 1.8296 - val_categorical_accuracy: 0.5106
Epoch 5/20
2019-07-26 16:28:14.639284: W tensorflow/core/common_runtime/bfc_allocator.cc:211] Allocator (GPU_0_bfc) ran
 - 486s - loss: 0.7268 - categorical_accuracy: 0.6944 - val_loss: 2.2483 - val_categorical_accuracy: 0.5050
Epoch 6/20
 - 486s - loss: 0.7054 - categorical_accuracy: 0.7021 - val_loss: 1.8936 - val_categorical_accuracy: 0.5328
Epoch 7/20
 - 486s - loss: 0.6919 - categorical_accuracy: 0.7090 - val_loss: 2.0716 - val_categorical_accuracy: 0.5339
Epoch 8/20
 - 486s - loss: 0.6910 - categorical_accuracy: 0.7128 - val_loss: 2.8301 - val_categorical_accuracy: 0.5178
Epoch 9/20
 - 486s - loss: 0.6726 - categorical_accuracy: 0.7215 - val_loss: 2.7478 - val_categorical_accuracy: 0.5156
Epoch 10/20
 - 486s - loss: 0.6836 - categorical_accuracy: 0.7196 - val_loss: 1.9447 - val_categorical_accuracy: 0.5378
Epoch 11/20
 - 485s - loss: 0.6795 - categorical_accuracy: 0.7188 - val_loss: 2.7819 - val_categorical_accuracy: 0.5350
Epoch 12/20
 - 486s - loss: 0.6728 - categorical_accuracy: 0.7218 - val_loss: 2.5242 - val_categorical_accuracy: 0.5550
Epoch 13/20
 - 486s - loss: 0.6657 - categorical_accuracy: 0.7252 - val_loss: 1.9672 - val_categorical_accuracy: 0.5594
Epoch 14/20
 - 486s - loss: 0.6767 - categorical_accuracy: 0.7218 - val_loss: 2.5083 - val_categorical_accuracy: 0.5317
Epoch 15/20
 - 487s - loss: 0.6683 - categorical_accuracy: 0.7225 - val_loss: 1.9882 - val_categorical_accuracy: 0.5728
Epoch 16/20
 - 485s - loss: 0.6705 - categorical_accuracy: 0.7254 - val_loss: 1.6288 - val_categorical_accuracy: 0.5477
Epoch 17/20
 - 484s - loss: 0.6716 - categorical_accuracy: 0.7250 - val_loss: 2.4563 - val_categorical_accuracy: 0.5211
Epoch 18/20
 - 485s - loss: 0.6690 - categorical_accuracy: 0.7232 - val_loss: 1.6254 - val_categorical_accuracy: 0.5361
Epoch 19/20
 - 486s - loss: 0.6665 - categorical_accuracy: 0.7250 - val_loss: 2.4959 - val_categorical_accuracy: 0.5506
Epoch 20/20
 - 486s - loss: 0.6635 - categorical_accuracy: 0.7313 - val_loss: 1.7559 - val_categorical_accuracy: 0.5450
```

Figura 6.3: Resultados Entrenamiento 1 por Épocas

A continuacion, se puede observar una gráfica con la evolución en la precision del modelo a lo largo de las 20 épocas. En la línea que representa el entrenamiento, se

puede observar como esta ha ido subiendo a buen ritmo hasta las época 7. A partir de ahí, se ha mantenido en los mismos valores hasta el final. El resultado final es un 73 % de precisión lo que no está nada mal. Con respecto a la validación, no ha crecido tanto como la del entrenamiento, lo que significa que los resultados no son nada buenos. Se ha mantenido oscilando entre los mismos valores durante las 20 épocas. Este entrenamiento destaca por su enorme sobreentrenamiento u overfiting. Esto quiere decir que, este modelo sabe diferenciar muy bien las imágenes con las que ha entrenado, pero si le dices que clasifique otras imágenes diferentes, el resultado es mucho peor.

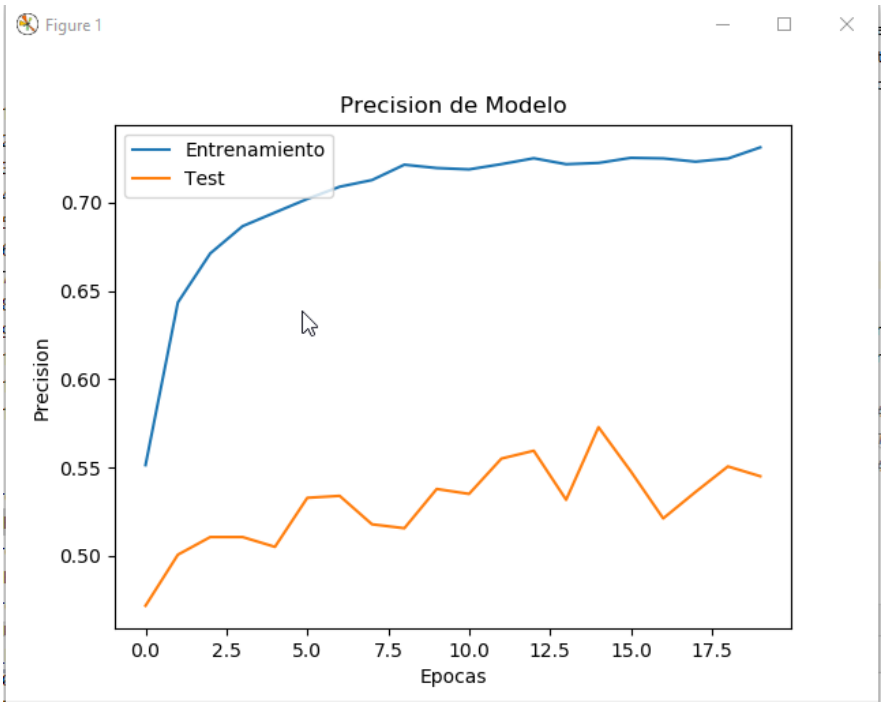

Figura 6.4: Gráfica 1 - Relación Precisión entre Entrenamiento y Validación en cada una de las épocas

En la siguiente gráfica observamos las pérdidas del modelo en entrenamiento y validación. Vemos como en entrenamiento este valor es bajo, ha llegado a un buen nivel de pérdidas pero en validacion este valor es bastante alto. Representa lo mismo que lo comentado anteriormente con el sobreentrenamiento u overfiting.

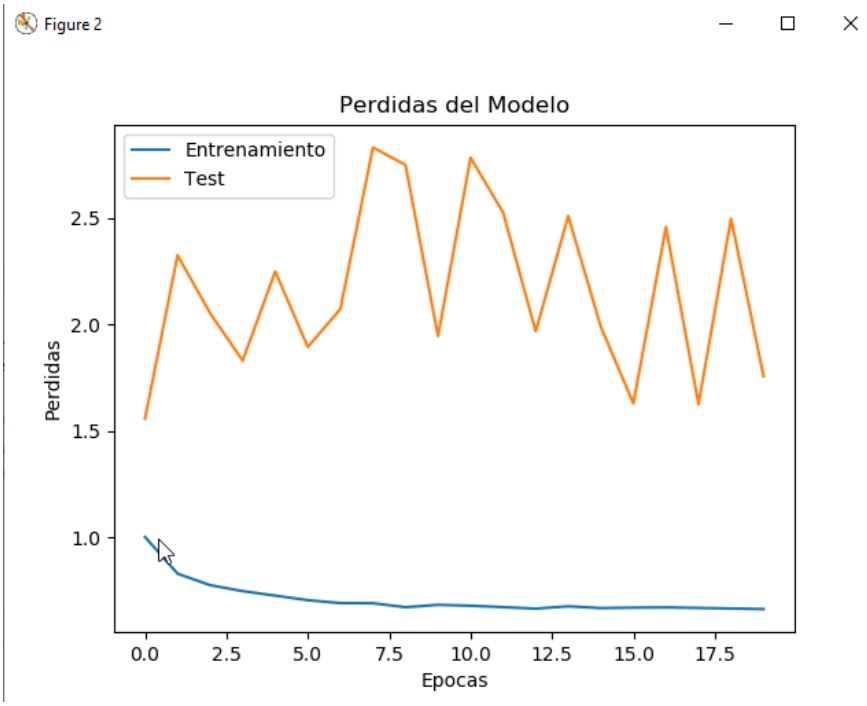

Figura 6.5: Gráfica 1 - Relación Pérdidas entre Entrenamiento y Validación en cada una de las épocas

Los resultados obtenidos en la fase de Test una vez hemos obtenido el modelo entrenado son los siguientes:

Precisión %	**Perdidas %**	**Tiempo Empleado**
0.5244	1.8715	2 horas, 44 minutos y 40 segundos

Tabla 6.2: Resultados entrenamiento con el Dataset 1

Como se puede observar, los resultados no son nada buenos con un 52 % de aciertos. También podemos ver como el tiempo empleado es bastante alto. Afecta mucho en el número de imágenes empleadas, ya que, cuantas más imágenes se usen durante el entrenamiento, mayor será el tiempo empleado. Está en la media con respecto al resto de entrenamientos con otras redes neuronales probadas.

6.1.2. Dataset 2 - Pocas Imágenes con 4 clases de galaxias

El siguiente entrenamiento es realizado con el dataset que contiene pocas imágenes y no se ha hecho uso de la técnica de Data Augmentation. Se hará uso de los 4 tipos de galaxias introducidos durante el documento. En la siguiente imagen ser puede observar como ha ido evolucionando la precisión del entrenamiento y validación a lo largo de las 20 épocas en las que se ha hecho el entrenamiento. También se observa la evolución de las pérdidas en cada época.

```
 - 138s - loss: 1.2754 - categorical_accuracy: 0.4000 - val_loss: 1.1143 - val_categorical_accuracy: 0.4925
Epoch 2/20
 - 132s - loss: 1.0491 - categorical_accuracy: 0.5439 - val_loss: 0.9866 - val_categorical_accuracy: 0.5700
Epoch 3/20
 - 132s - loss: 1.0054 - categorical_accuracy: 0.5724 - val_loss: 0.9975 - val_categorical_accuracy: 0.5592
Epoch 4/20
 - 132s - loss: 0.9607 - categorical_accuracy: 0.5890 - val_loss: 0.9484 - val_categorical_accuracy: 0.5667
Epoch 5/20
 - 132s - loss: 0.9512 - categorical_accuracy: 0.5869 - val_loss: 0.9819 - val_categorical_accuracy: 0.5608
Epoch 6/20
 - 132s - loss: 0.9404 - categorical_accuracy: 0.5961 - val_loss: 0.8900 - val_categorical_accuracy: 0.6075
Epoch 7/20
 - 132s - loss: 0.9150 - categorical_accuracy: 0.6074 - val_loss: 0.8660 - val_categorical_accuracy: 0.6050
Epoch 8/20
 - 131s - loss: 0.8976 - categorical_accuracy: 0.6105 - val_loss: 0.8426 - val_categorical_accuracy: 0.6142
Epoch 9/20
 - 132s - loss: 0.8785 - categorical_accuracy: 0.6209 - val_loss: 0.8370 - val_categorical_accuracy: 0.6308
Epoch 10/20
 - 132s - loss: 0.8666 - categorical_accuracy: 0.6291 - val_loss: 0.8329 - val_categorical_accuracy: 0.6242
Epoch 11/20
 - 134s - loss: 0.8493 - categorical_accuracy: 0.6352 - val_loss: 0.8389 - val_categorical_accuracy: 0.6317
Epoch 12/20
 - 134s - loss: 0.8455 - categorical_accuracy: 0.6312 - val_loss: 0.8500 - val_categorical_accuracy: 0.6325
Epoch 13/20
 - 135s - loss: 0.8381 - categorical_accuracy: 0.6396 - val_loss: 0.8793 - val_categorical_accuracy: 0.6133
Epoch 14/20
 - 135s - loss: 0.8331 - categorical_accuracy: 0.6450 - val_loss: 0.8234 - val_categorical_accuracy: 0.6417
Epoch 15/20
 - 135s - loss: 0.8338 - categorical_accuracy: 0.6448 - val_loss: 0.8100 - val_categorical_accuracy: 0.6442
Epoch 16/20
 - 134s - loss: 0.8124 - categorical_accuracy: 0.6470 - val_loss: 0.8367 - val_categorical_accuracy: 0.6300
Epoch 17/20
 - 132s - loss: 0.8089 - categorical_accuracy: 0.6515 - val_loss: 0.8325 - val_categorical_accuracy: 0.6292
Epoch 18/20
 - 132s - loss: 0.8050 - categorical_accuracy: 0.6508 - val_loss: 0.8110 - val_categorical_accuracy: 0.6367
Epoch 19/20
 - 132s - loss: 0.7986 - categorical_accuracy: 0.6554 - val_loss: 0.8174 - val_categorical_accuracy: 0.6450
Epoch 20/20
 - 132s - loss: 0.8086 - categorical_accuracy: 0.6496 - val_loss: 0.8192 - val_categorical_accuracy: 0.6208
```

Figura 6.6: Resultados Entrenamiento 2 por Épocas

A continuación, se puede observar una gráfica con la evolución en la precisión del modelo a lo largo de las 20 épocas. En la parte del entrenamiento, este ha ido subiendo casi de forma progresiva durante todo el entrenamiento. Se puede ver como cada vez le cuesta más mejorar esta precisión. Al final se quedó con un 65 % de precisión, peor que el entrenamiento con mayor número de imágenes. Con respecto a la validación, la progresión ha sido muy pareja con los valores del entrenamiento. Este entrenamiento, se puede considerar, por una parte bueno, ya que overfiting o underfiting es casi despreciable en los resultados finales. Por otra parte ese 65 % de precisión en el entrenamiento y 62 % de precisión en validación, no se consideran valores especialmente buenos. Esto quiere decir que hay casi un 40 % de imágenes que no se han clasificado correctamente.

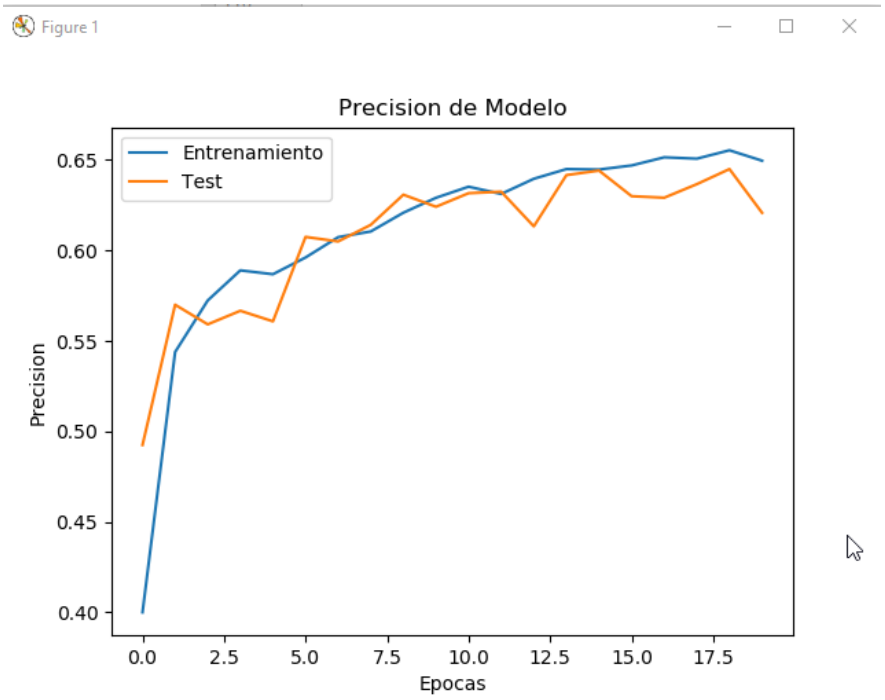

Figura 6.7: Gráfica 2 - Relación Precisión entre Entrenamiento y Validación en cada una de las épocas

En la siguiente gráfica observamos las pérdidas del modelo en entrenamiento y validación. Vemos como los valores han ido evolucionando de forma bastante pareja durante el entrenamiento del modelo lo que es bastante bueno. Han mejorado bastante

estos valores con respecto al primer entrenamiento comentado anteriormente.

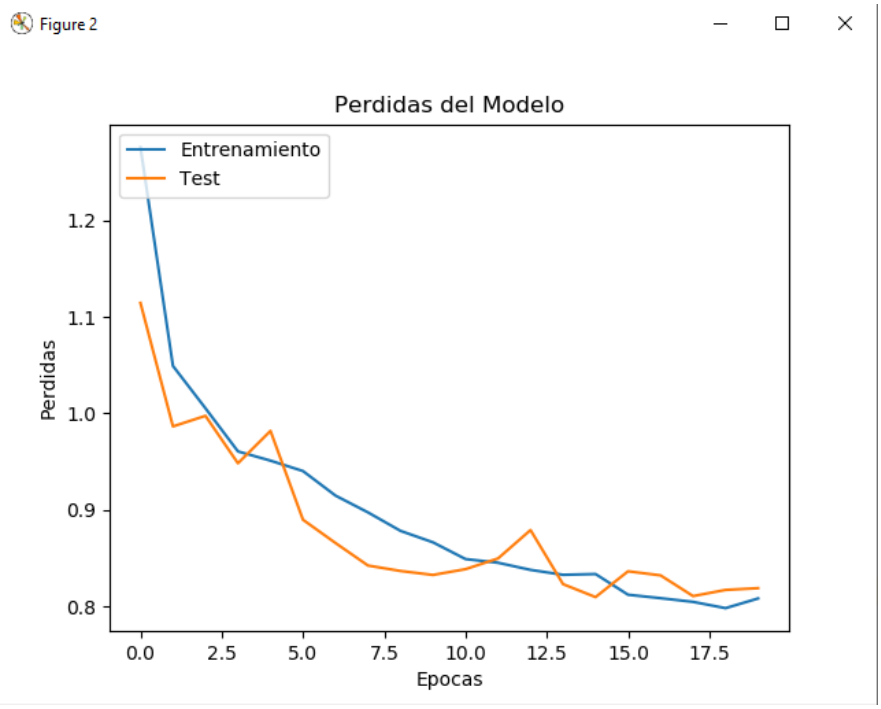

Figura 6.8: Gráfica 2 - Relación Pérdidas entre Entrenamiento y Validación en cada una de las épocas

Los resutados obtenidos en la fase de Test una vez hemos obtenido el modelo entrenado son los siguientes:

Precisión %	**Perdidas %**	**Tiempo Empleado**
0.6366	0.8433	44 minutos y 20 segundos

Tabla 6.3: Resultados entrenamiento con el Dataset 2

Con un casi 64 % de acierto final durante la fase de Test, mejora bastante el resultado final con respecto al entrenamiento anterior realizado con un mayor número de imágenes. Otro detalle importante es que el tiempo empleado para obtener este modelo es bastante más bajo que el anterior. Con estos resultados nos llega la conclusión de que no siempre por usar un mayor número de imágenes se van a obtener mejores resultados.

6.1.3. Dataset 3 - Muchas Imágenes con 3 clases de galaxias

El siguiente entrenamiento es realizado con el dataset que contiene una gran cantidad de imágenes, aplicándose la técnica de Data Augmentation en aquellos tipos de galaxias donde el número de imágenes disponibles era menor que el resto. Se eliminarán todas las imágenes pertenecientes al tipo de galaxia Other. Esto se hace con el fin de verificar que, al ser un tipo de galaxia donde las imágenes tienen estructuras sin definir, dificulta el aprendizaje de patrones de estas. En la siguiente imagen ser puede observar como ha ido evolucionando la precisión del entrenamiento y validación a lo largo de las 20 épocas en las que se ha hecho el entrenamiento. También se observa la evolución de las pérdidas en cada época.

```
 - 465s - loss: 0.5972 - categorical_accuracy: 0.7429 - val_loss: 1.7562 - val_categorical_accuracy: 0.6211
Epoch 2/20
 - 460s - loss: 0.4483 - categorical_accuracy: 0.8243 - val_loss: 1.5511 - val_categorical_accuracy: 0.6283
Epoch 3/20
 - 460s - loss: 0.3882 - categorical_accuracy: 0.8513 - val_loss: 1.9044 - val_categorical_accuracy: 0.6044
Epoch 4/20
 - 451s - loss: 0.3468 - categorical_accuracy: 0.8708 - val_loss: 1.4877 - val_categorical_accuracy: 0.6583
Epoch 5/20
 - 451s - loss: 0.3285 - categorical_accuracy: 0.8767 - val_loss: 2.0223 - val_categorical_accuracy: 0.6344
Epoch 6/20
 - 452s - loss: 0.3263 - categorical_accuracy: 0.8803 - val_loss: 1.4837 - val_categorical_accuracy: 0.6567
Epoch 7/20
 - 456s - loss: 0.3073 - categorical_accuracy: 0.8885 - val_loss: 2.0289 - val_categorical_accuracy: 0.6267
Epoch 8/20
 - 454s - loss: 0.3112 - categorical_accuracy: 0.8858 - val_loss: 2.0799 - val_categorical_accuracy: 0.6061
Epoch 9/20
 - 451s - loss: 0.3217 - categorical_accuracy: 0.8819 - val_loss: 2.5896 - val_categorical_accuracy: 0.6511
Epoch 10/20
 - 451s - loss: 0.3005 - categorical_accuracy: 0.8899 - val_loss: 1.8049 - val_categorical_accuracy: 0.6344
Epoch 11/20
 - 451s - loss: 0.2939 - categorical_accuracy: 0.8942 - val_loss: 1.8451 - val_categorical_accuracy: 0.6456
Epoch 12/20
 - 452s - loss: 0.2988 - categorical_accuracy: 0.8931 - val_loss: 2.0668 - val_categorical_accuracy: 0.6592
Epoch 13/20
 - 449s - loss: 0.3007 - categorical_accuracy: 0.8931 - val_loss: 1.4127 - val_categorical_accuracy: 0.6661
Epoch 14/20
 - 451s - loss: 0.2953 - categorical_accuracy: 0.8926 - val_loss: 1.4555 - val_categorical_accuracy: 0.6606
Epoch 15/20
 - 451s - loss: 0.3001 - categorical_accuracy: 0.8928 - val_loss: 2.2120 - val_categorical_accuracy: 0.6244
Epoch 16/20
 - 451s - loss: 0.2930 - categorical_accuracy: 0.8925 - val_loss: 1.8612 - val_categorical_accuracy: 0.6172
Epoch 17/20
 - 451s - loss: 0.2970 - categorical_accuracy: 0.8935 - val_loss: 1.3369 - val_categorical_accuracy: 0.6478
Epoch 18/20
 - 451s - loss: 0.2977 - categorical_accuracy: 0.8939 - val_loss: 2.0445 - val_categorical_accuracy: 0.6244
Epoch 19/20
 - 451s - loss: 0.3047 - categorical_accuracy: 0.8907 - val_loss: 1.9653 - val_categorical_accuracy: 0.6411
Epoch 20/20
 - 452s - loss: 0.3016 - categorical_accuracy: 0.8951 - val_loss: 1.5881 - val_categorical_accuracy: 0.6500
```

Figura 6.9: Resultados Entrenamiento 3 por Épocas

A continuación, se puede observar una gráfica con la evolución de la precisión del modelo a lo largo de las 20 épocas. En la parte del entrenamiento, este ha ido subiendo a buen ritmo hasta la época 7. A partir de ahí, se ha mantenido en los mismos valores hasta el final. Al final se quedó con un 89 % de precisión, lo que nos da un resultado bastante bueno. Con respecto a la validación, no ha crecido tanto como la del entrenamiento, lo que significa que los resultados no son tan buenos como parecían. Se ha mantenido oscilando entre los mismos valores, 0.6 y 0.65, durante las 20 épocas. Este entrenamiento destaca por su enorme sobreentrenamiento u overfiting. Esto quiere decir que, este modelo sabe diferenciar muy bien las imágenes con las que ha entrenado, pero si le dices que clasifique otras imágenes diferentes, el resultado es mucho peor.

Esta gráfica es muy similar a la que se obtuvo durante el primer entrenamiento mostrado anteriormente, con la única diferencia que lo valores entre los que se ha movido han mejorado bastante, tanto en entrenamiento como en validación.

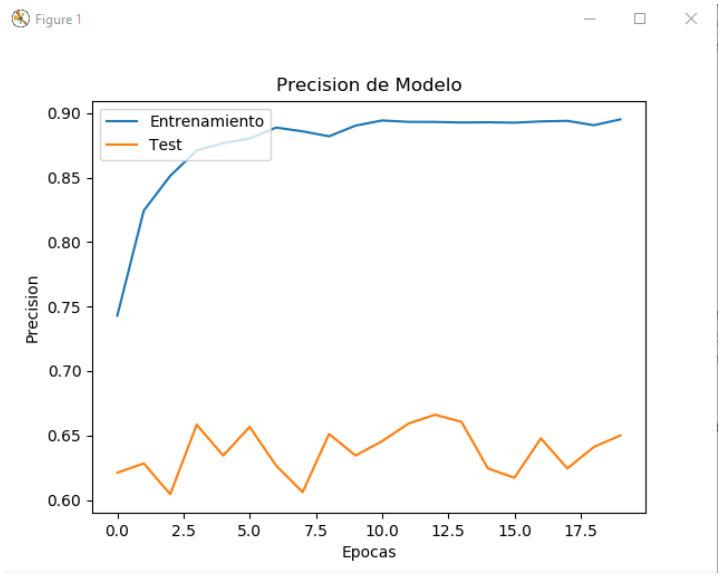

Figura 6.10: Gráfica 3 - Relación Precisión entre Entrenamiento y Validación en cada una de las épocas

En la siguiente gráfica observamos las pérdidas del modelo en entrenamiento y validación. Vemos como en entrenamiento este valor es bajo, ha llegado a un buen nivel de pérdidas. En cambio, en la parte de validación, este valor es bastante alto.

Representa lo mismo que lo comentado anteriormente con el sobreentrenamiento. Esta gráfica es muy pareja a la que obtuvimos en el primer entrenamiento igualando incluso los valores entre los que se han movido las pérdidas de entrenamiento y validación.

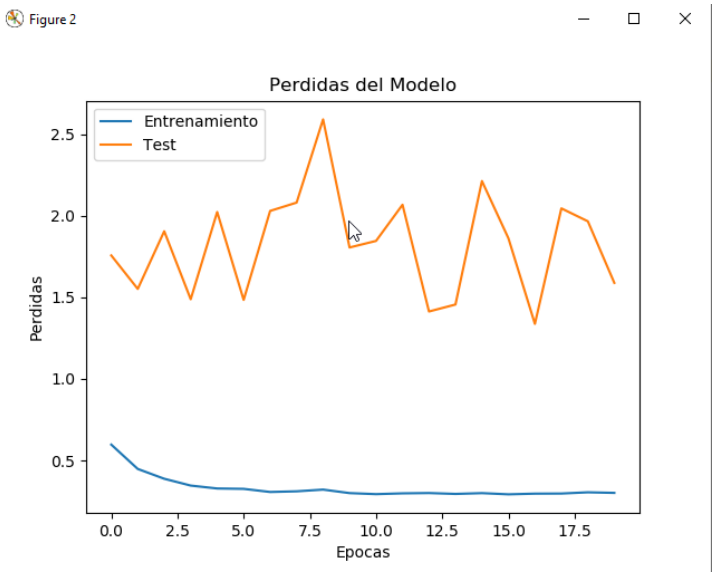

Figura 6.11: Gráfica 3 - Relación Pérdidas entre Entrenamiento y Validación en cada una de las épocas

Los resutados obtenidos en la fase de Test una vez hemos obtenido el modelo entrenado son los siguientes:

Precisión %	**Perdidas %**	**Tiempo Empleado**
0.6415	1.6659	2 horas, 32 minutos y 52 segundos

Tabla 6.4: Resultados entrenamiento con el Dataset 3

El resultado en precisión es el mejor obtenido hasta ahora con un 64 % ,pero el valor obtenido en pérdidas hace indicar un fuerte sobreentrenamiento por lo que se puede considerar un modelo no válido para clasificar. Podemos ver como el tiempo empleado es bastante alto. Afecta mucho el número de imágenes empleadas, ya que cuantas más imágenes se usen durante el entrenamiento, mayor será el tiempo empleado. Está en la media con respecto al resto de entrenamientos con otras redes neuronales probadas.

6.1.4. Dataset 4 - Pocas Imágenes con 3 clases de galaxias

El siguiente entrenamiento es realizado con el dataset que contiene pocas imágenes y no se ha hecho uso de la técnica de Data Augmentation. Se eliminará el tipo de galaxia Other. Esto se hace con el fin de verificar que, al ser un tipo de galaxia donde las imágenes tienen estruturas sin definir, dificulta el aprendizaje de patrones de estas. En la siguiente imagen ser puede observar como ha ido evolucionando la precisión del entrenamiento y validación a lo largo de las 20 épocas en las que se ha hecho el entrenamiento. También se observa la evolución de las pérdidas en cada época.

```
 - 137s - loss: 1.0031 - categorical_accuracy: 0.4827 - val_loss: 0.7758 - val_categorical_accuracy: 0.6712
Epoch 2/20
 - 130s - loss: 0.7076 - categorical_accuracy: 0.7195 - val_loss: 0.6190 - val_categorical_accuracy: 0.7610
Epoch 3/20
 - 131s - loss: 0.6051 - categorical_accuracy: 0.7661 - val_loss: 0.5073 - val_categorical_accuracy: 0.7966
Epoch 4/20
 - 130s - loss: 0.5754 - categorical_accuracy: 0.7843 - val_loss: 0.5737 - val_categorical_accuracy: 0.7690
Epoch 5/20
 - 130s - loss: 0.5509 - categorical_accuracy: 0.7924 - val_loss: 0.4825 - val_categorical_accuracy: 0.8008
Epoch 6/20
 - 130s - loss: 0.5076 - categorical_accuracy: 0.8070 - val_loss: 0.5333 - val_categorical_accuracy: 0.7975
Epoch 7/20
 - 130s - loss: 0.4978 - categorical_accuracy: 0.8135 - val_loss: 0.4731 - val_categorical_accuracy: 0.8086
Epoch 8/20
 - 131s - loss: 0.4890 - categorical_accuracy: 0.8158 - val_loss: 0.4874 - val_categorical_accuracy: 0.7941
Epoch 9/20
 - 130s - loss: 0.4790 - categorical_accuracy: 0.8199 - val_loss: 0.4244 - val_categorical_accuracy: 0.8398
Epoch 10/20
 - 130s - loss: 0.4678 - categorical_accuracy: 0.8246 - val_loss: 0.4254 - val_categorical_accuracy: 0.8397
Epoch 11/20
 - 130s - loss: 0.4480 - categorical_accuracy: 0.8328 - val_loss: 0.4256 - val_categorical_accuracy: 0.8314
Epoch 12/20
 - 130s - loss: 0.4490 - categorical_accuracy: 0.8375 - val_loss: 0.4678 - val_categorical_accuracy: 0.8144
Epoch 13/20
 - 130s - loss: 0.4439 - categorical_accuracy: 0.8348 - val_loss: 0.3975 - val_categorical_accuracy: 0.8517
Epoch 14/20
 - 130s - loss: 0.4417 - categorical_accuracy: 0.8363 - val_loss: 0.4624 - val_categorical_accuracy: 0.8276
Epoch 15/20
 - 130s - loss: 0.4363 - categorical_accuracy: 0.8419 - val_loss: 0.4408 - val_categorical_accuracy: 0.8280
Epoch 16/20
 - 130s - loss: 0.4278 - categorical_accuracy: 0.8359 - val_loss: 0.4236 - val_categorical_accuracy: 0.8381
Epoch 17/20
 - 130s - loss: 0.4059 - categorical_accuracy: 0.8504 - val_loss: 0.3839 - val_categorical_accuracy: 0.8638
Epoch 18/20
 - 130s - loss: 0.4211 - categorical_accuracy: 0.8455 - val_loss: 0.4222 - val_categorical_accuracy: 0.8441
Epoch 19/20
 - 130s - loss: 0.4160 - categorical_accuracy: 0.8510 - val_loss: 0.4261 - val_categorical_accuracy: 0.8407
Epoch 20/20
 - 130s - loss: 0.4106 - categorical_accuracy: 0.8448 - val_loss: 0.3919 - val_categorical_accuracy: 0.8534
```

Figura 6.12: Resultados Entrenamiento 4 por Épocas

A continuación, se puede observar una gráfica con la evolución en la precisión del modelo a lo largo de las 20 épocas. En la parte del entrenamiento, este ha ido subiendo casi de forma progresiva durante todo el proceso. Se puede ver como cada vez le cuesta más mejorar esta precisión. Al final se quedó con un 84 % de precisión, mejorando cualquier otro entrenamiento presentado hasta ahora. Con respecto a la validación, la progresión ha sido muy pareja con los valores del entrenamiento. Este entrenamiento es sin lugar a dudas, el mejor entrenamiento realizado hasta ahora. Presenta unos valores de precisión bastantes altos, tanto en entrenamiento (84 %) como en validación (85 %). Presenta un mínimo de underfiting, es decir, mejor valor en validacion que en entrenamiento. Lo que sucede con el underfiting es que el modelo sabe reconocer una estructura determinada en la imagen y sí esta modifica algo ya no es capaz de averiguar a que clase pertenece una determinada imagen. Este underfiting es casi despreciable. Este modelo se puede considerar como un gran ejemplo de buen entrenamiento.

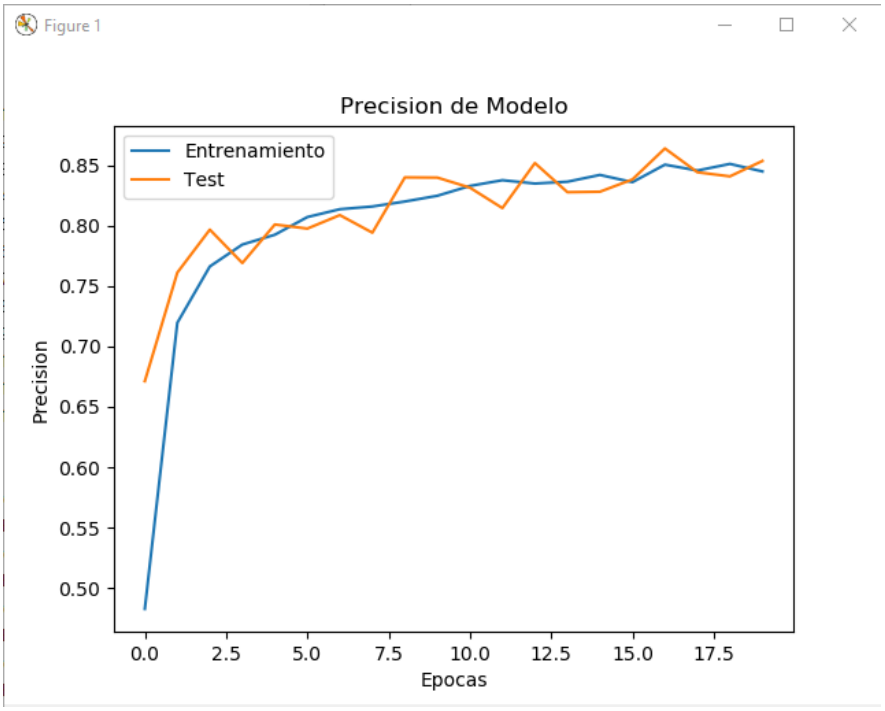

Figura 6.13: Gráfica 4 - Relación Precisión entre Entrenamiento y Validación en cada una de las épocas

En la siguiente gráfica observamos las pérdidas del modelo en entrenamiento y

validación. Vemos como los valores han ido evolucionando de forma bastante pareja durante el entrenamiento del modelo. Volvemos a tener los mejores valores obtenidos en este caso de pérdidas, con valores bastantes bajos lo que es bastante positivo.

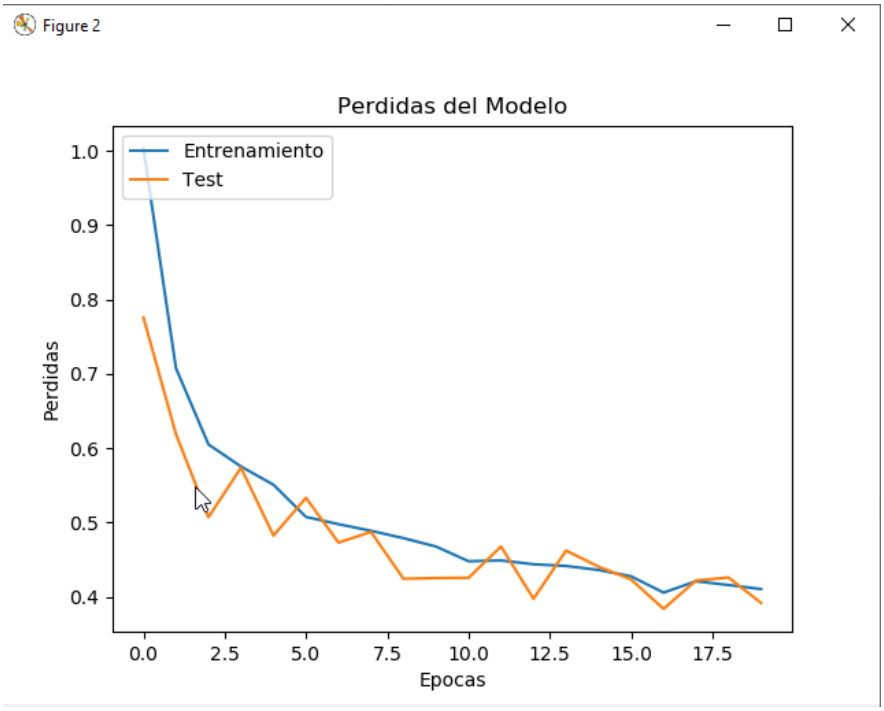

Figura 6.14: Gráfica 4 - Relación Pérdidas entre Entrenamiento y Validación en cada una de las épocas

Los resutados obtenidos en la fase de Test, una vez hemos obtenido el modelo entrenado, son los siguientes:

Precisión	**Perdidas**	**Tiempo Empleado**
0.8355	0.4403	43 minutos y 38 segundos

Tabla 6.5: Resultados entrenamiento con el Dataset 4

Con un 83 % de precisión y 0,44 de pérdidas durante la fase de Test, es el mejor valor obtenido de todos tipos de entrenamientos realizados con esta red neuronal. Otro detalle importante es que el tiempo empleado para obtener este modelo es bastante más bajo que todos los anteriores. Con estos resultados nos llega la conclusión de que no siempre por usar un mayor número de imágenes se van a obtener mejores resultados y

que, las imágenes de tipo Other, estaban empeorando de forma considerable cualquier entrenamiento que realizáramos.

La conclusión final que hemos obtenido de estos cuatro entrenamientos, realizados todos ellos con una misma red neuronal, es que no siempre que se usen un mayor número de imágenes, el resultado va a ser mejor. La técnica de Data Augmentation no siempre es buena. Por otro lado, hay que analizar muy bien la clasificación que se quiera realizar. Como hemos visto en este ejemplo, un tipo que no este bien definido, es muy probable que la propia red no pueda sacar elementos en común de este tipo que haga clasificarlos de forma correcta.

6.1.5. Realizando entrenamientos con diferentes Optimizadores

Se van a mostrar las diferencias entre los distintos optimizadores usados. En los entrenamientos mostrados anteriormente, el único optimizador usado es Adam. A continuación, se mostrarán las diferencias con respecto a los optimizadores Adagrad y SGD. Para estas pruebas se ha hecho uso del dataset con un menor número de imágenes y con solo tres tipos de galaxias.

Resultados con SGD:

Este optimizador fue de los primeros en usarse en Deep Learning. La evolución del entrenamiento con este optimizador ha sido la siguiente:

```
 - 136s - loss: 1.0946 - categorical_accuracy: 0.3769 - val_loss: 1.0897 - val_categorical_accuracy: 0.3458
Epoch 2/20
 - 130s - loss: 1.0802 - categorical_accuracy: 0.4329 - val_loss: 1.0541 - val_categorical_accuracy: 0.5153
Epoch 3/20
 - 130s - loss: 1.0209 - categorical_accuracy: 0.5055 - val_loss: 0.9306 - val_categorical_accuracy: 0.5839
Epoch 4/20
 - 130s - loss: 0.8842 - categorical_accuracy: 0.5990 - val_loss: 0.7371 - val_categorical_accuracy: 0.6879
Epoch 5/20
 - 130s - loss: 0.7581 - categorical_accuracy: 0.6772 - val_loss: 0.6516 - val_categorical_accuracy: 0.7339
Epoch 6/20
 - 130s - loss: 0.6981 - categorical_accuracy: 0.7129 - val_loss: 0.5935 - val_categorical_accuracy: 0.7644
Epoch 7/20
 - 130s - loss: 0.6601 - categorical_accuracy: 0.7269 - val_loss: 0.5607 - val_categorical_accuracy: 0.7759
Epoch 8/20
 - 130s - loss: 0.6418 - categorical_accuracy: 0.7399 - val_loss: 0.5561 - val_categorical_accuracy: 0.7822
Epoch 9/20
 - 130s - loss: 0.6312 - categorical_accuracy: 0.7508 - val_loss: 0.5725 - val_categorical_accuracy: 0.7754
Epoch 10/20
 - 130s - loss: 0.6149 - categorical_accuracy: 0.7559 - val_loss: 0.5464 - val_categorical_accuracy: 0.7819
Epoch 11/20
 - 130s - loss: 0.6196 - categorical_accuracy: 0.7491 - val_loss: 0.5454 - val_categorical_accuracy: 0.7822
Epoch 12/20
 - 130s - loss: 0.5966 - categorical_accuracy: 0.7634 - val_loss: 0.5464 - val_categorical_accuracy: 0.7814
Epoch 13/20
 - 130s - loss: 0.5928 - categorical_accuracy: 0.7599 - val_loss: 0.5406 - val_categorical_accuracy: 0.7839
Epoch 14/20
 - 130s - loss: 0.5885 - categorical_accuracy: 0.7647 - val_loss: 0.5298 - val_categorical_accuracy: 0.7853
Epoch 15/20
 - 130s - loss: 0.5764 - categorical_accuracy: 0.7700 - val_loss: 0.5110 - val_categorical_accuracy: 0.7932
Epoch 16/20
 - 130s - loss: 0.5641 - categorical_accuracy: 0.7784 - val_loss: 0.5230 - val_categorical_accuracy: 0.7907
Epoch 17/20
 - 130s - loss: 0.5706 - categorical_accuracy: 0.7772 - val_loss: 0.5300 - val_categorical_accuracy: 0.7836
Epoch 18/20
 - 130s - loss: 0.5537 - categorical_accuracy: 0.7876 - val_loss: 0.5201 - val_categorical_accuracy: 0.7924
Epoch 19/20
 - 130s - loss: 0.5447 - categorical_accuracy: 0.7835 - val_loss: 0.4887 - val_categorical_accuracy: 0.8085
Epoch 20/20
 - 130s - loss: 0.5341 - categorical_accuracy: 0.7934 - val_loss: 0.4907 - val_categorical_accuracy: 0.8060
```

Figura 6.15: Resultados Entrenamiento 4 con SGD por Épocas

La siguiente gráfica mostrará la evolución de la precisión del modelo en cada una de las 20 épocas que duró este entrenamiento.

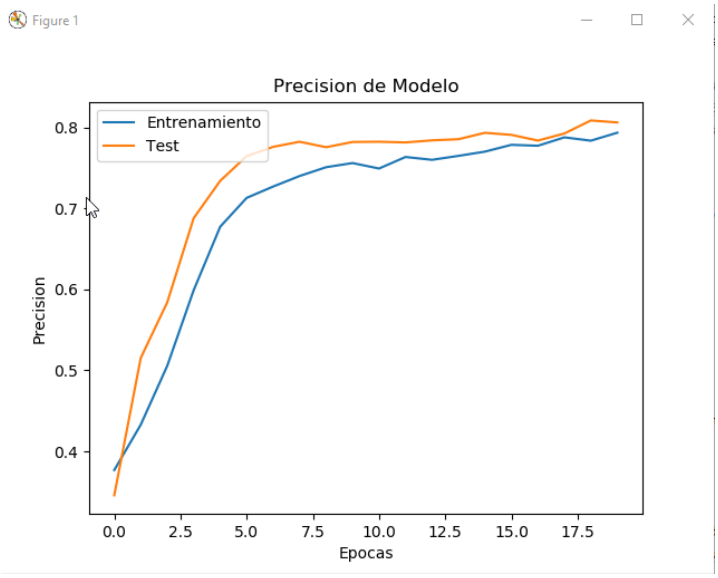

Figura 6.16: Gráfica 5 - Relación Precisión entre Entrenamiento y Validación en cada una de las épocas con SGD

La siguiente gráfica mostrará la evolución de las pérdidas del modelo en cada una de las 20 épocas que duró este entrenamiento.

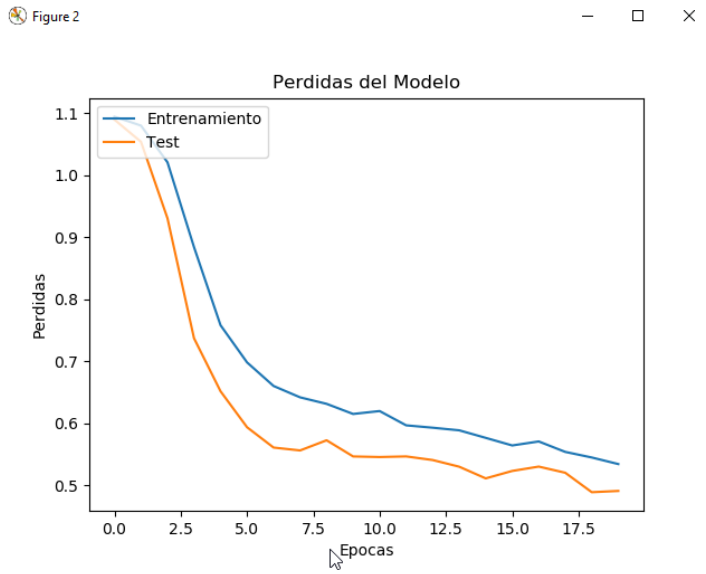

Figura 6.17: Gráfica 5 - Relación Pérdidas entre Entrenamiento y Validación en cada una de las épocas con SGD

Los resultados finales obtenidos una vez se ha ejecutado la fase de Test han sido los siguientes:

Precisión	Perdidas	Tiempo Empleado
0.7906	0.5380	43 minutos y 34 segundos

Tabla 6.6: Resultados entrenamiento con el Dataset 4 y optimizador SGD

Se puede observar como la precisión obtenida finalmete ha sido ligeramente peor que la que se obtuvo con el optimizador Adam (0.83). Del mismo modo, la pérdida también asciende ligeramente. Este cambio de optimizador apenas afecta al tiempo que dura la ejecución del entrenamiento.

Resultados con Adagrad:

Este optimizador se obtuvo a través del SGD. Se puede considerar una evolución del SGD. El entrenamiento con este optimizador ha sido la siguiente:

```
 - 136s - loss: 0.8553 - categorical_accuracy: 0.6080 - val_loss: 0.6459 - val_categorical_accuracy: 0.7398
Epoch 2/20
 - 130s - loss: 0.6561 - categorical_accuracy: 0.7382 - val_loss: 0.5773 - val_categorical_accuracy: 0.7576
Epoch 3/20
 - 130s - loss: 0.6013 - categorical_accuracy: 0.7645 - val_loss: 0.5591 - val_categorical_accuracy: 0.7737
Epoch 4/20
 - 130s - loss: 0.5845 - categorical_accuracy: 0.7728 - val_loss: 0.5149 - val_categorical_accuracy: 0.7888
Epoch 5/20
 - 130s - loss: 0.5652 - categorical_accuracy: 0.7810 - val_loss: 0.5486 - val_categorical_accuracy: 0.7763
Epoch 6/20
 - 130s - loss: 0.5472 - categorical_accuracy: 0.7920 - val_loss: 0.5071 - val_categorical_accuracy: 0.7941
Epoch 7/20
 - 130s - loss: 0.5329 - categorical_accuracy: 0.7947 - val_loss: 0.5184 - val_categorical_accuracy: 0.7871
Epoch 8/20
 - 130s - loss: 0.5299 - categorical_accuracy: 0.7944 - val_loss: 0.4906 - val_categorical_accuracy: 0.8025
Epoch 9/20
 - 130s - loss: 0.5265 - categorical_accuracy: 0.7944 - val_loss: 0.4886 - val_categorical_accuracy: 0.8093
Epoch 10/20
 - 130s - loss: 0.5149 - categorical_accuracy: 0.8022 - val_loss: 0.4768 - val_categorical_accuracy: 0.8069
Epoch 11/20
 - 130s - loss: 0.5095 - categorical_accuracy: 0.8010 - val_loss: 0.4952 - val_categorical_accuracy: 0.7864
Epoch 12/20
 - 130s - loss: 0.5029 - categorical_accuracy: 0.8029 - val_loss: 0.4833 - val_categorical_accuracy: 0.7983
Epoch 13/20
 - 130s - loss: 0.5059 - categorical_accuracy: 0.8019 - val_loss: 0.4671 - val_categorical_accuracy: 0.8127
Epoch 14/20
 - 130s - loss: 0.4933 - categorical_accuracy: 0.8144 - val_loss: 0.4778 - val_categorical_accuracy: 0.8009
Epoch 15/20
 - 130s - loss: 0.4904 - categorical_accuracy: 0.8106 - val_loss: 0.4981 - val_categorical_accuracy: 0.7890
Epoch 16/20
 - 130s - loss: 0.4850 - categorical_accuracy: 0.8139 - val_loss: 0.4670 - val_categorical_accuracy: 0.8110
Epoch 17/20
 - 130s - loss: 0.4794 - categorical_accuracy: 0.8142 - val_loss: 0.4587 - val_categorical_accuracy: 0.8000
Epoch 18/20
 - 130s - loss: 0.4882 - categorical_accuracy: 0.8096 - val_loss: 0.4459 - val_categorical_accuracy: 0.8254
Epoch 19/20
 - 130s - loss: 0.4730 - categorical_accuracy: 0.8194 - val_loss: 0.4700 - val_categorical_accuracy: 0.8076
Epoch 20/20
 - 130s - loss: 0.4761 - categorical_accuracy: 0.8180 - val_loss: 0.4679 - val_categorical_accuracy: 0.8069
```

Figura 6.18: Resultados Entrenamiento 4 con Adagrad por Épocas

La siguiente gráfica mostrará la evolución de la precisión del modelo en cada una de las 20 épocas que duró este entrenamiento.

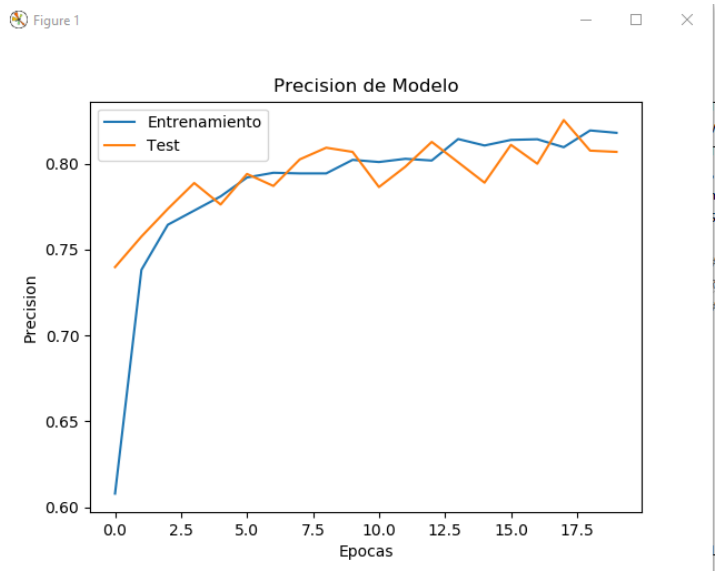

Figura 6.19: Gráfica 6 - Relación Precisión entre Entrenamiento y Validación en cada una de las épocas con Adagrad

La siguiente gráfica mostrará la evolución de las pérdidas del modelo en cada una de las 20 épocas que duró este entrenamiento.

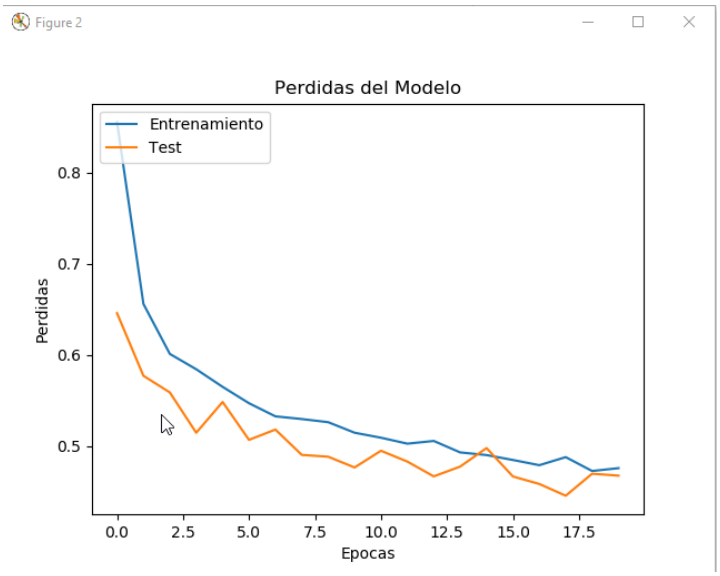

Figura 6.20: Gráfica 6 - Relación Pérdidas entre Entrenamiento y Validación en cada una de las épocas con Adagrad

Los resultados finales obtenidos una vez se ha ejecutado la fase de Test han sido los siguientes:

Precisión	Perdidas	Tiempo Empleado
0.8042	0.5159	43 minutos y 35 segundos

Tabla 6.7: Resultados entrenamiento con el Dataset 4 y optimizador Adagrad

Se puede observar como la precisión obtenida finalmete ha sido ligeramente peor que la obtenida con el optimizador Adam (0.83) pero mejor que la de SGD. Del mismo modo, la pérdida también se encuentra entre los valores obtenidos de Adam y SGD. Este cambio de optimizador apenas afecta al tiempo que dura la ejecución del entrenamiento.

Como conclusión se puede tomar, que tanto con Adam como con Adagrad al ser evoluciones de SGD, mejoran los resultados finales. Del mismo modo, el optimizador Adam ha obtenido mejores resultados que Adagrad. Hecho comprensible ya que Adam es a su vez una evolución de Adagrad. Por lo tanto, de los tres optimizadores, el que mejores resultados ha obtenido es Adam, aunque esto no quiere decir que sea siempre el más idóneo para todo tipo de clasificaciones de imágenes.

6.2. 2º Conjunto de Entrenamientos

Para esta sección, se van a presentar los entrenamientos que se han realizado ya con redes neuronales hechas y de gran conocimiento en el mundo del Deep Learning. Para ello se hará uso de nuevo de la biblioteca de Keras donde se nos facilita la carga de estas redes de una forma sencilla a través de la librería Applications.

```
from keras.applications import VGG16, VGG19, ResNet50, Xception, InceptionV3, InceptionResNetV2
```

Figura 6.21: Import de las Redes Neuronales de la librería Applications

Una vez tenemos el import con todas las redes neuronales que se van a usar se procede a cargarlas. Como verán a continuación, el uso de estas redes es bastante sencillo. Basta con llamar al método indicado en el import con la red que se desea trabajar (en este caso es la VGG16). Para estas pruebas, solo se indicará el tamaño de filas y columnas de las imágenes con las que se realizarán los entrenamientos. También se indicará el número de canales que siempre será 3 al tratar con imágenes en color. Por último, se indicará el número de clases en las que se hará la clasificación. El resto de parámetros serán por defecto.

```
self.modelo = VGG16(include_top=True, weights=None, input_tensor=None,
                    input_shape=(self.filas, self.columnas, self.canales),
                    pooling=None, classes=self.numClases)
```

Figura 6.22: Ejemplo - Carga de la Red Neuronal VGG16

Como bien se ha podido ver en el import, las redes neuronales que se usarán serán las siguientes:

1. **VGG16**
2. **VGG19**
3. **Resnet 50**
4. **Inception V3**

5. **Xception**

6. **Inception Resnet V2**

Estas redes son bastante más complejas que las presentadas anteriormente y están preparadas para resolver problemas mucho más complejos, aunque no siempre es la mejor solución como veremos a continuación.

Resultados obtenidos de los entrenamientos

A continuación, se indicarán brevemente los resultados obtenidos en los diversos entrenamientos realizados con cada una de ellas. Se expondrán cuatro tablas. La primera de ellas será los resultados de aquellos entrenamientos realizados con un gran cantidad de imágenes por cada tipo de galaxia (unas 23.000 por cada tipo de galaxia). La clasificación será con 4 tipo diferentes de galaxias. Se informa de la precisión final, la pérdida final y el tiempo empleado en la ejecución del entrenamiento y obtención posterior de los resultados.

Red Neuronal	Precisión	Perdidas	Tiempo
Exception	0.5726	2.3555	3 horas, 55 minutos y 38 segundos
Inception Resnet V2	0.5257	2.4311	8 horas, 19 minutos y 54 segundos
Inception V3	0.5534	2.5887	5 horas, 34 minutos y 39 segundos
Resnet 50	0.5654	2.3154	4 horas, 49 minutos y 51 segundos
VGG 19	0.2498	1.3865	4 horas, 47 minutos y 26 segundos
VGG 16	0.2499	1.3864	2 horas, 28 minutos y 10 segundos

Tabla 6.8: Resultados entrenamientos con Dataset 1

Esta segunda tabla será los resultados de aquellos entrenamientos realizados con una gran cantidad de imágenes por cada tipo de galaxia (unas 23.000 por cada tipo de galaxia). La clasificación será con 3 tipos diferentes de galaxias eliminando la de tipo OTHER.

Red Neuronal	Precisión	Perdidas	Tiempo
Exception	0.6555	1.5036	3 horas, 25 minutos y 34 segundos
Inception Resnet V2	0.6356	2.7097	8 horas, 20 minutos y 36 segundos
Inception V3	0.6498	2.3008	5 horas, 41 minutos y 23 segundos
Resnet 50	0.6535	1.9442	7 horas, 52 minutos y 20 segundos
VGG 19	0.3342	1.0988	4 horas, 44 minutos y 27 segundos
VGG 16	0.3348	1.0986	2 horas, 28 minutos y 40 segundos

Tabla 6.9: Resultados entrenamientos con Dataset 3

Esta tercera tabla será los resultados de aquellos entrenamientos realizados con una cantidad de imágenes bastante menor que las dos primeras. Habrá unas 2.000 imágenes por cada tipo de galaxia. La clasificación será con 4 tipos diferentes.

Red Neuronal	Precisión	Perdidas	Tiempo
Exception	0.6575	0.7771	1 horas, 9 minutos y 37 segundos
Inception Resnet V2	0.2500	1.3863	1 horas, 42 minutos y 31 segundos
Inception V3	0.6066	0.9151	1 horas, 57 minutos y 0 segundos
Resnet 50	0.6691	0.8032	1 horas, 38 minutos y 33 segundos
VGG 19	0.2500	1.3863	1 horas, 37 minutos y 3 segundos
VGG 16	0.2500	1.3863	50 minutos y 35 segundos

Tabla 6.10: Resultados entrenamientos con Dataset 2

Esta cuarta y última tabla será los resultados de aquellos entrenamientos realizados con una cantidad de imágenes bastante menor que las dos primeras. Habrá unas 2.000 imágenes por cada tipo de galaxia. La clasificación será con 3 tipos diferentes de galaxias eliminando la de tipo OTHER.

Red Neuronal	Precisión	Perdidas	Tiempo
Exception	0.8549	0.4517	1 horas, 9 minutos y 3 segundos
Inception Resnet V2	0.8416	0.4496	2 horas, 52 minutos y 55 segundos
Inception V3	0.8166	0.4464	1 horas, 54 minutos y 32 segundos
Resnet 50	0.7758	0.6960	1 horas, 38 minutos y 10 segundos
VGG 19	0.3375	1.0986	1 horas, 39 minutos y 41 segundos
VGG 16	0.3391	1.0985	50 minutos y 47 segundos

Tabla 6.11: Resultados entrenamientos con Dataset 4

Conclusión de los resultados obtenidos

Para estas conclusiones se van a tomar como referencia los resultados obtenidos por la red neuronal que se ha hecho desde cero, expuesta en la sección anterior de este capítulo.

Analizando de primeras todos los resultados obtenidos sacamos de primeras dos conclusiones bastante rápidas. La primera, es que las redes VGG 16 Y 19 para esta clasificación funcionan bastante mal obteniendo siempre resultados bastantes bajos con respecto al resto de redes neuronales. Como segunda conclusión, y está más global ya que afecta a todas las redes, es el tiempo empleado en la realización de los entrenamientos. Este tiempo suele ser bastante más alto que todos los entrenamiento que he ido realizando durante el primer conjunto de entrenamientos. Destaca bastante la red Inception Resnet V2.

Con respecto a los resultado finales, sin tener en cuenta el tiempo empleado, el resto de redes (excluimos las VGG's) tienen resultados bastante iguales en cada uno de los diferentes entrenamientos realizados. Destaca, con los mejores resultados obtenidos, la red Exception sobre el resto.

Por lo tanto, si hay que escoger una red para trabajar en este tipo de clasificaciones, sin duda elegiría la Exception. Ya no solo por los resultados que son los mejores, sino también por el tiempo empleado en estos entrenamientos. Se puede observar claramente como estos tiempos son los más bajos en cada uno de los diferentes entrenamiento.

Si comparamos la red Exception con las creada desde cero en el primer conjunto de entrenamientos que se expuso anteriormente, podemos ver como los resultados de la red Exception son mínimamente mejores en todos los casos, pero también hay que tener en cuenta el tiempo empleado. Este tiempo es bastante mayor en la red Exception por lo que sin duda alguna me quedo con la red creada desde cero.

Escojo esta porque al final vamos a emplear menos tiempo para obtener casi los mismos resultados que la Exception.

Un último detalle que quería remarcar y que me gustaría exponer es el resultado de Red Inception Resnet V2 que se puede observar en la tercera tabla de resultados. Como se puede ver, este resultado es bastante inferior a lo que nos suele dar esta red y que podemos ver en el resto de entrenamiento. Este al final es un error que se ha dado durante el entrenamiento que no he podido solucionar ni encontrar el motivo exacto. Tengo la certeza de que puede ser por la memoria ocupada durante la ejecución del entrenamiento anterior. Lo que ha hecho, de algun modo, es desvirtuar este entrenamiento y no ha logrado realizar un entrenamiento limpio. Este error me ha ocurrido bastante a lo largo de todo este trabajo, por lo que en ocasiones tenía que ejecutar varias veces un mismo entrenamiento para cerciorarme de que ese resultado era el máximo que me podía dar la red con la que estuviese trabajando.

En cierta medida, conseguí que este error se dejará de reproducir apagando el equipo y haciendo que se liberará toda la memoria de la GPU al ser esta volátil. Pero, como se ha podido comprobar, no siempre funcionó.

6.3. Trabajando con los modelos obtenidos

En esta sección se les mostrará como trabajar con un modelo resultante de un entrenamiento. En este caso, se hará uso del modelo obtenido a través de la red neuronal creada desde cero y, en particular, la obtenida con el dataset 4 que contenía tres tipos diferentes de galaxias, eliminando la de tipo Other y con pocas imágenes usadas durante su entrenamiento (2000 imágenes por galaxia).

Por un lado, se les mostrará los filtros creados en las capas de activación de aquellas capas usadas para la parte de Clasificación, como son la Conv2D y la MaxPooling2D. La imagen original usada para esta prueba es la siguiente:

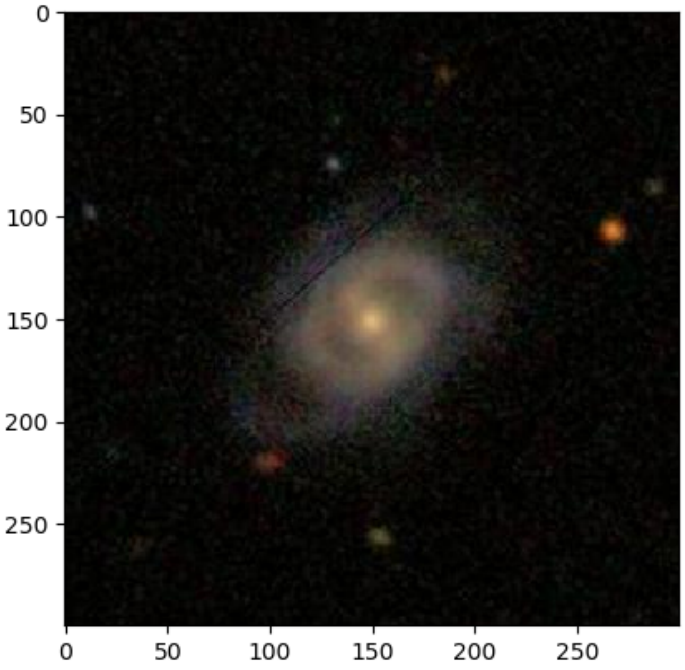

Figura 6.23: Imagen Original usada para el ejemplo

A partir de esta imagen se han obtenido los distintos filtros que se han creado en las distintas capas usadas. Un ejemplo de estos filtros son los siguientes:

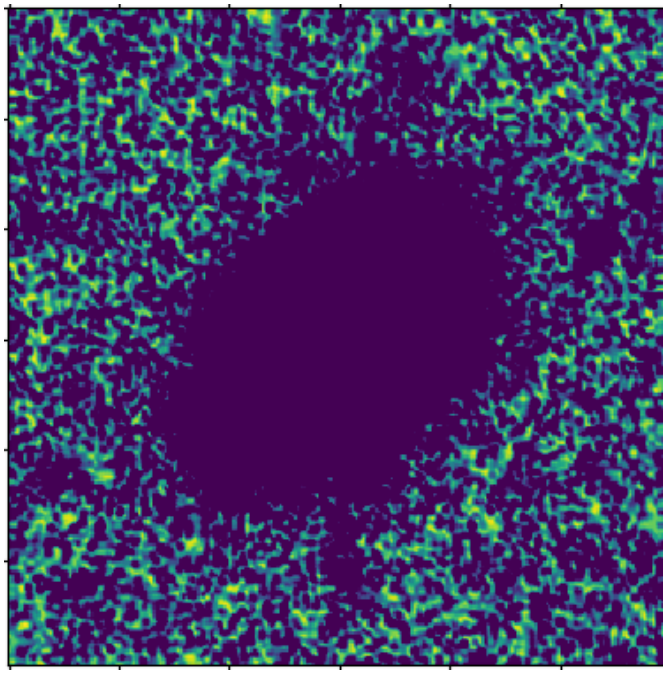

Figura 6.24: Filtro capa de activación Relu

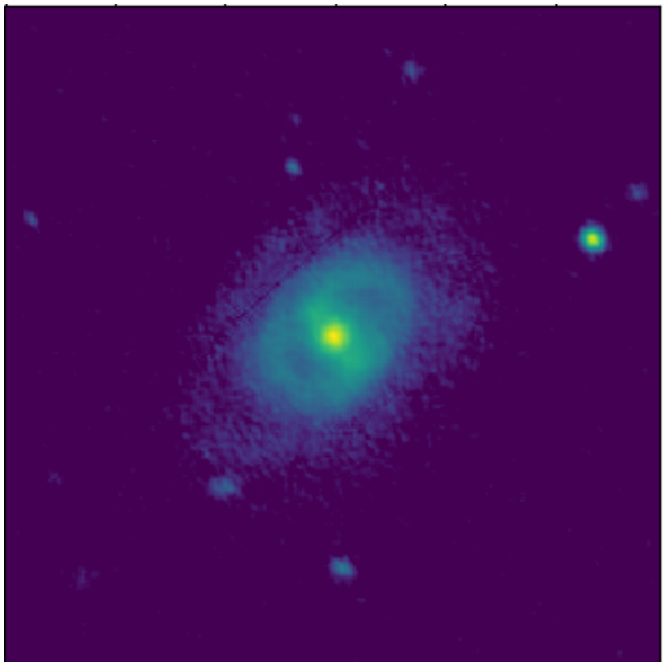

Figura 6.25: Filtro capa de activación Relu

A continuación, se les mostrará todos los filtros obtenidos de cada una de las capas pertenecientes a la parte de Clasificación de nuestra Red Neuronal.

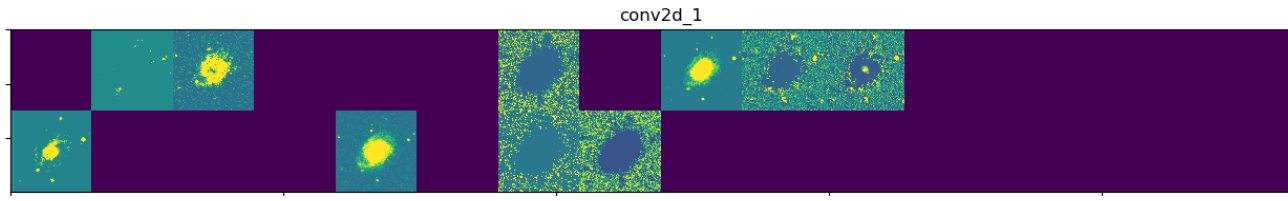

Figura 6.26: Filtro capa de activación de la conv2d_1

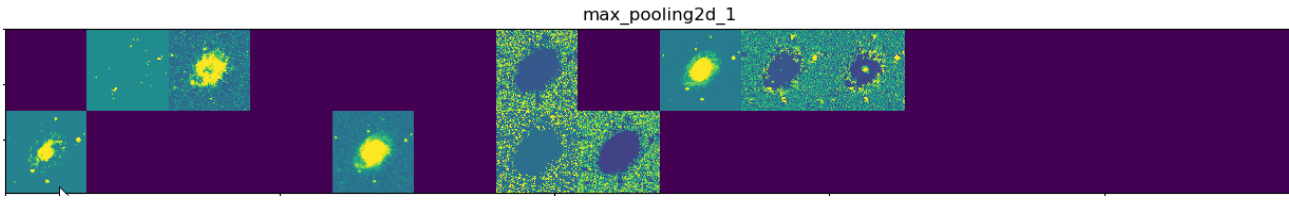

Figura 6.27: Filtro capa de activación de la max_pooling2d_1

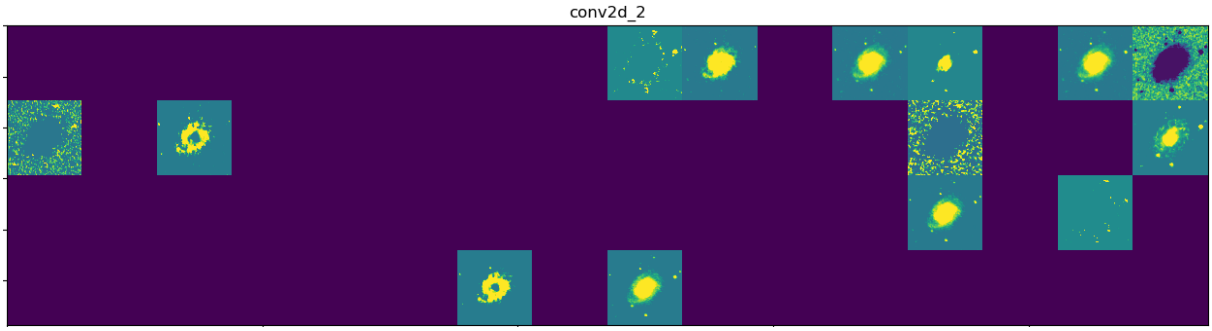

Figura 6.28: Filtro capa de activación de la conv2d_2

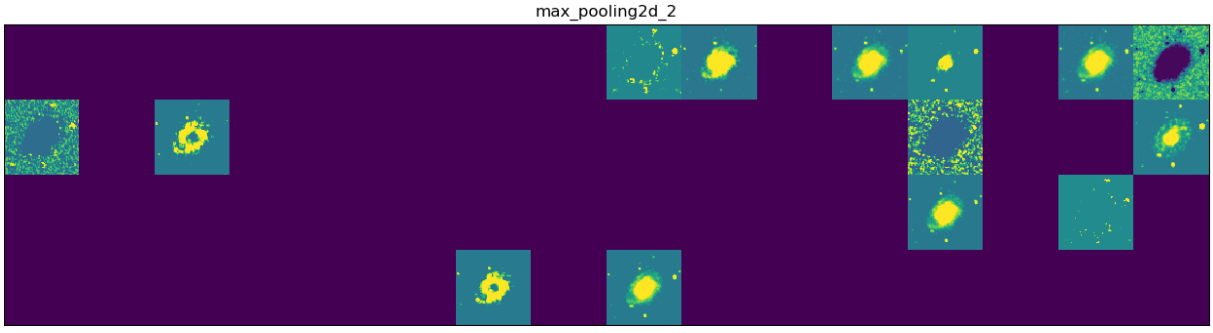

Figura 6.29: Filtro capa de activación de la max_pooling2d_2

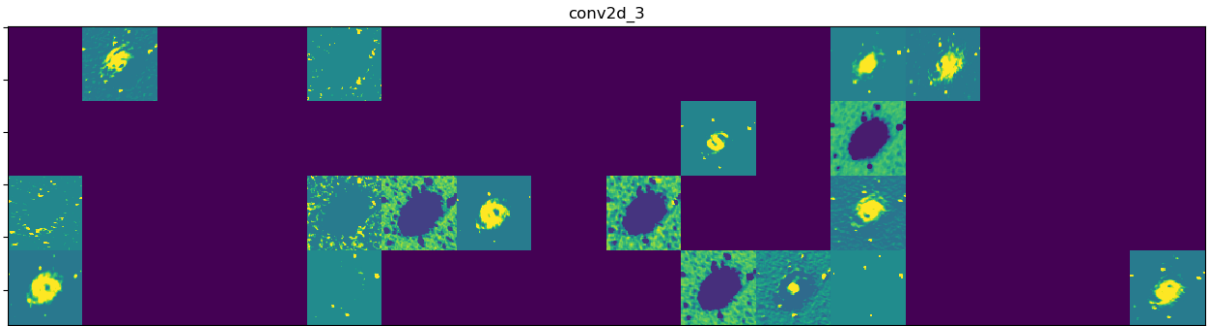

Figura 6.30: Filtro capa de activación de la conv2d_3

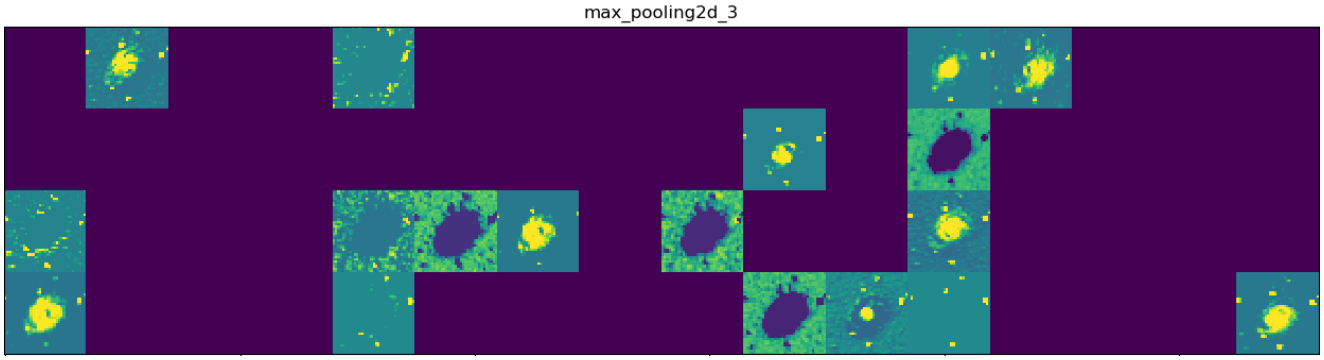

Figura 6.31: Filtro capa de activación de la max_pooling2d_3

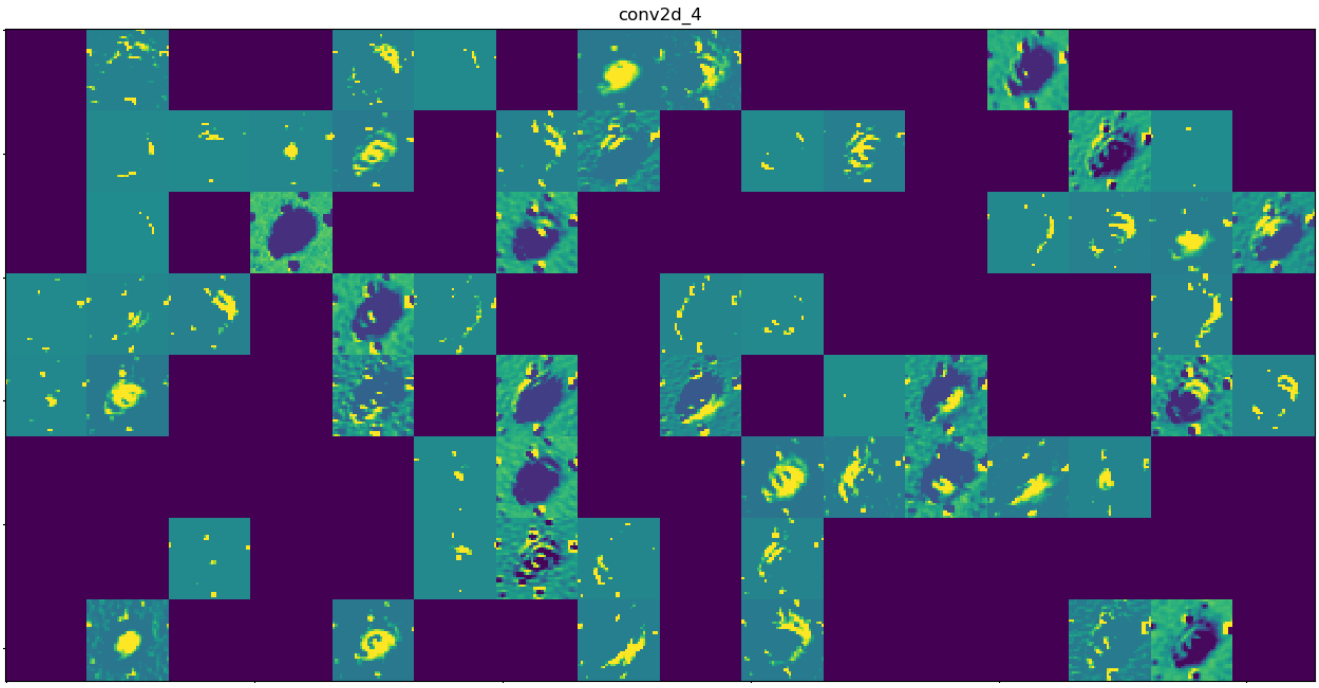

Figura 6.32: Filtro capa de activación de la conv2d_4

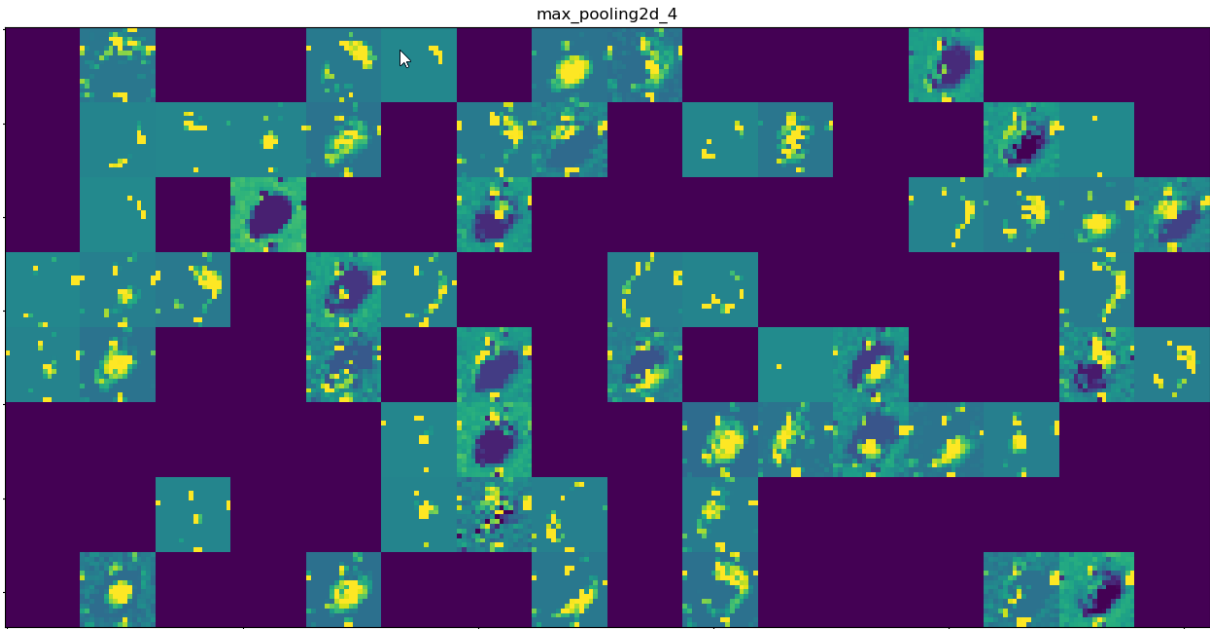

Figura 6.33: Filtro capa de activación de la max_pooling2d_4

Conforme vamos a las capas mas profundas, se van obteniendo filtros con mayor detalle, con mayor precisión. Es aquí de donde viene ese Deep (Profundo). Esta es la gran diferencia del Deep Learning con el resto de técnicas actuales hoy en día.

Otra de la pruebas que se ha hecho con el modelo obtenido, será la clasificación de imágenes que nosotros elijamos y así ver como clasifica entre los diferentes tipos de galaxias. Se van a mostrar diversos resultados para que se vea que no siempre hay que fiarse de los resultados obtenidos aunque, por lo general, la clasificación de estas imágenes es bastante buena. Para hacer este trabajo se ha hecho uso del método predict facilitado por la librería de Keras. En primer lugar, se cargaba el modelo generado anteriormente. Para ello se hace uso del método load, indicando la ruta donde se encuentra este modelo.

```
pathModel = 'C:/Users/lcostami/PycharmProjects/tfgKerasGPU/TFG_Final/Pruebas/Mio/Entrenamiento_4/Ent4_3Clases_FewImages.h5'
# Load the model we saved
model = load_model(pathModel)
```

Figura 6.34: Método para la carga del modelo

Una vez cargado el modelo, se van preparando las imágenes y se guardan todas en un mismo array. Este array se le pasa al método predict indicado anteriormente.

```
y_pred = model.predict(X_test, batch_size=None, verbose=1, steps=None)
```

Figura 6.35: Método Predict

Se obtiene un conjunto de array informando del tipo de galaxia a la que pertenece cada imagen.

A continuación, se mostrarán algunas de las imágenes usadas para esta prueba junto con los resultados de cada una de estas.

Para este ejemplo se ha usado una galaxia de tipo Espiral. Como se puede ver, el resultado es bastante bueno ya que nos indica que esa galaxia es de tipo Espiral al 99.56 %, lo que es bastante precisa.

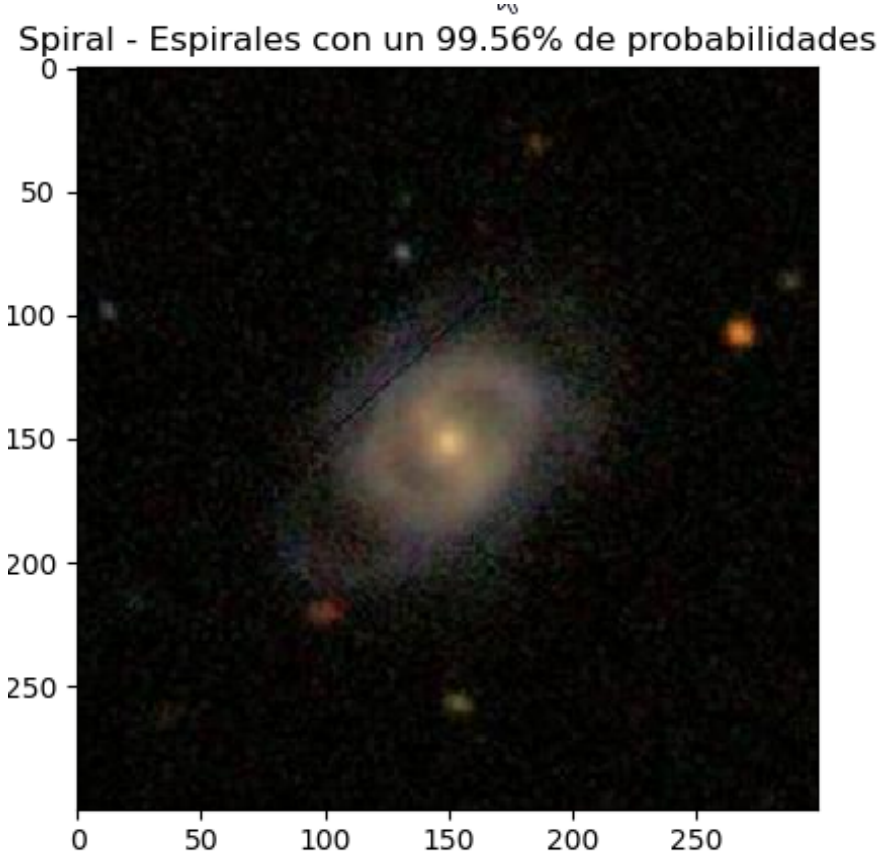

Figura 6.36: Imagen del Ejemplo 1

A continuación, se muestran las probabilidades que tiene la imagen de ser de cada uno de los tipos de galaxias con las que hemos clasificado las imágenes. La suma de estas dará el 100 %. Está bastante claro el tipo de galaxia que es.

```
############################################################
La imagenes 117045.jpg es de tipo Edge - Elipticas con un 0.0% de probabilidades
La imagenes 117045.jpg es de tipo Smooth - Lenticulares con un 0.44% de probabilidades
La imagenes 117045.jpg es de tipo Spiral - Espirales con un 99.56% de probabilidades
############################################################
```

Figura 6.37: Resultados del Ejemplo 1

Para este ejemplo se ha usado una galaxia de tipo Lenticular. Como se puede ver, el resultado es bastante bueno ya que nos indica que esa galaxia es de tipo Lenticular al 96.34 %, lo que es bastante preciso.

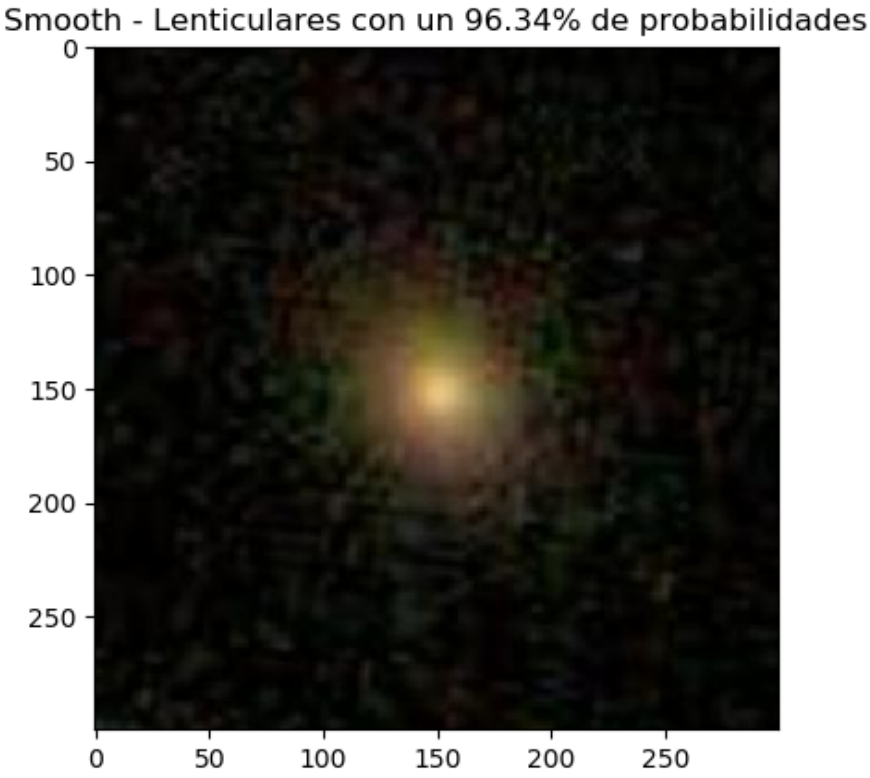

Figura 6.38: Imagen del Ejemplo 2

A continuación, se muestran las probabilidades que tiene la imagen de ser de cada uno de los tipos de galaxias con los que hemos clasificado las imágenes. La suma de estas dará el 100 %. Está bastante claro el tipo de galaxia que es.

```
############################################################
La imagenes 574479.jpg es de tipo Edge - Elipticas con un 0.07% de probabilidades
La imagenes 574479.jpg es de tipo Smooth - Lenticulares con un 96.34% de probabilidades
La imagenes 574479.jpg es de tipo Spiral - Espirales con un 3.59% de probabilidades
############################################################
```

Figura 6.39: Resultados del Ejemplo 2

Para este ejemplo se ha usado una galaxia de tipo Elíptica. Como se puede ver, el resultado es bastante bueno ya que nos indica que esa galaxia es de tipo Elíptica al 99.9 %, lo que es bastante precisa y nos da bastantes garantías de que el resultado es el correcto.

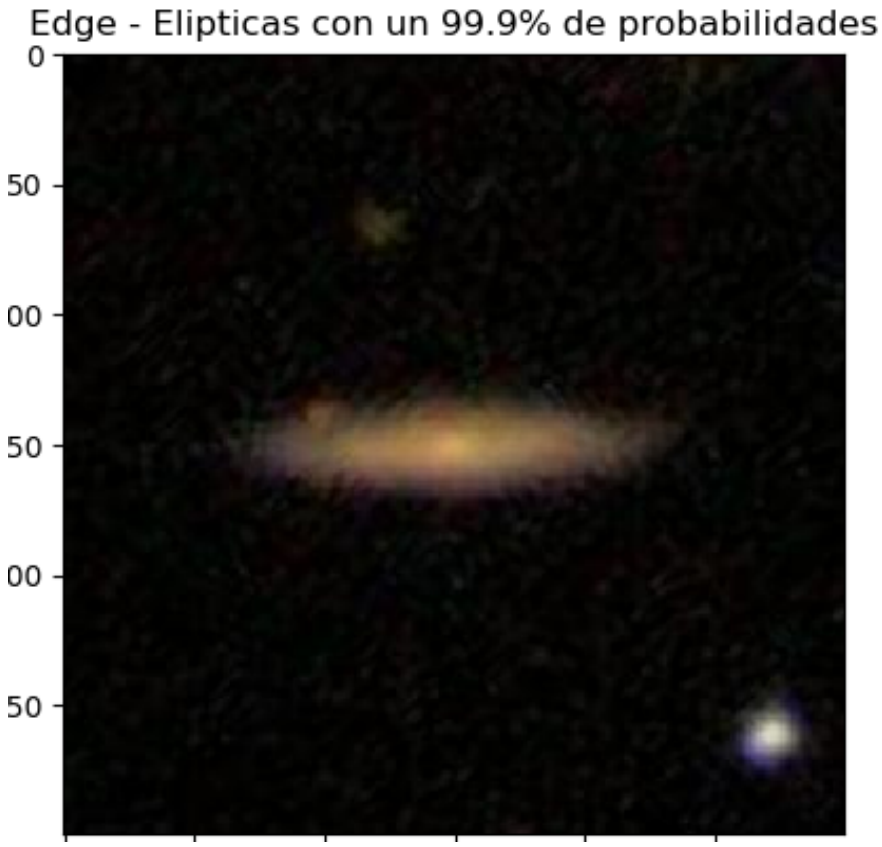

Figura 6.40: Imagen del Ejemplo 3

A continuación, se muestran las probabilidades que tiene la imagen de ser de cada uno de los tipos de galaxias con las que hemos clasificado las imágenes. La suma de estas dará el 100 %. Está bastante claro el tipo de galaxia que es.

```
#############################################################
La imagenes edge.955867.jpg es de tipo Edge - Elipticas con un 99.9% de probabilidades
La imagenes edge.955867.jpg es de tipo Smooth - Lenticulares con un 0.02% de probabilidades
La imagenes edge.955867.jpg es de tipo Spiral - Espirales con un 0.07% de probabilidades
#############################################################
```

Figura 6.41: Resultados del Ejemplo 3

Para este ejemplo se ha usado una galaxia de tipo Elíptica. Como se puede ver, el resultado ya no es muy preciso ya que nos da solo un 50.54 % de probabilidad de que esa imagen sea una galaxia de tipo Elíptica. En esta ocasión, el resultado final es bueno pero puede darnos lugar a dudas.

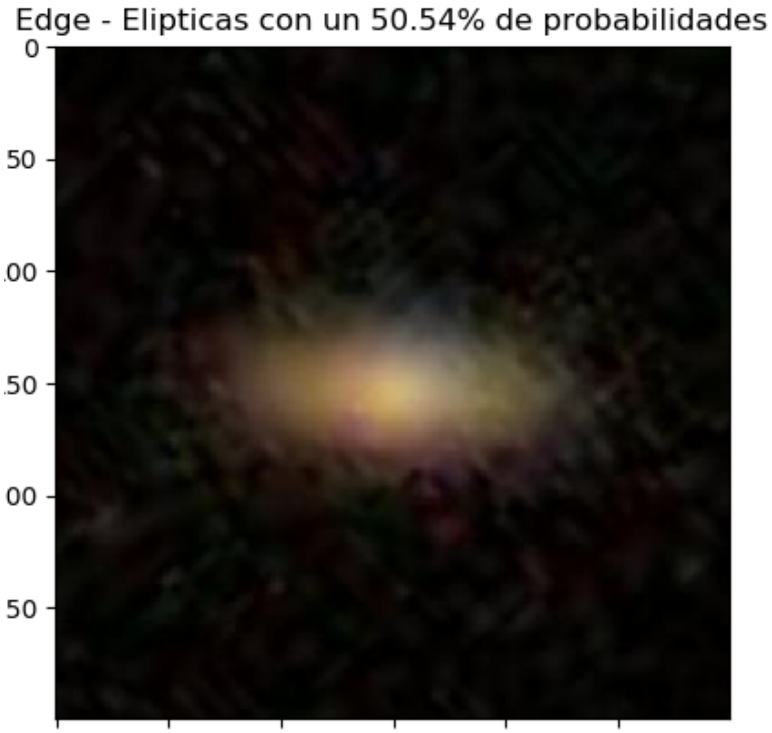

Figura 6.42: Imagen del Ejemplo 4

A continuación, se muestran las probabilidades que tiene la imagen de ser de cada uno de los tipos de galaxias con las que hemos clasificado las imágenes. La suma de estas dará el 100 %. Vemos como también nos da un buen porcentaje con la galaxia de tipo Smooth con un 32.2 % de probabilidades. Lo más práctico en estos casos es que la solución final la de un experto y no sea la máquina quien lo decida.

```
############################################################
La imagenes edge.956042.jpg es de tipo Edge - Elipticas con un 50.54% de probabilidades
La imagenes edge.956042.jpg es de tipo Smooth - Lenticulares con un 32.2% de probabilidades
La imagenes edge.956042.jpg es de tipo Spiral - Espirales con un 17.27% de probabilidades
############################################################
```

Figura 6.43: Resultados del Ejemplo 4

Para este ejemplo se ha usado una galaxia de tipo Elíptica. En el entrenamiento salió una precisión final del 84 % por lo que este modelo no te asegura siempre que la clasificación sea la correcta. Este es uno de esos ejemplos. El resultado no coincide con el del tipo de galaxia que realmente es. Se puede ver que nos predice que es una galaxia de tipo Espiral con una probabilidad que no es muy buena lo que, al igual que el anterior ejemplo, nos puede hacer dudar. Esta probabilidad es del 56,9 %. Como ya comenté anteriormente, lo más práctico en estos casos es que la solución final la de un experto y no sea la máquina quien lo decida.

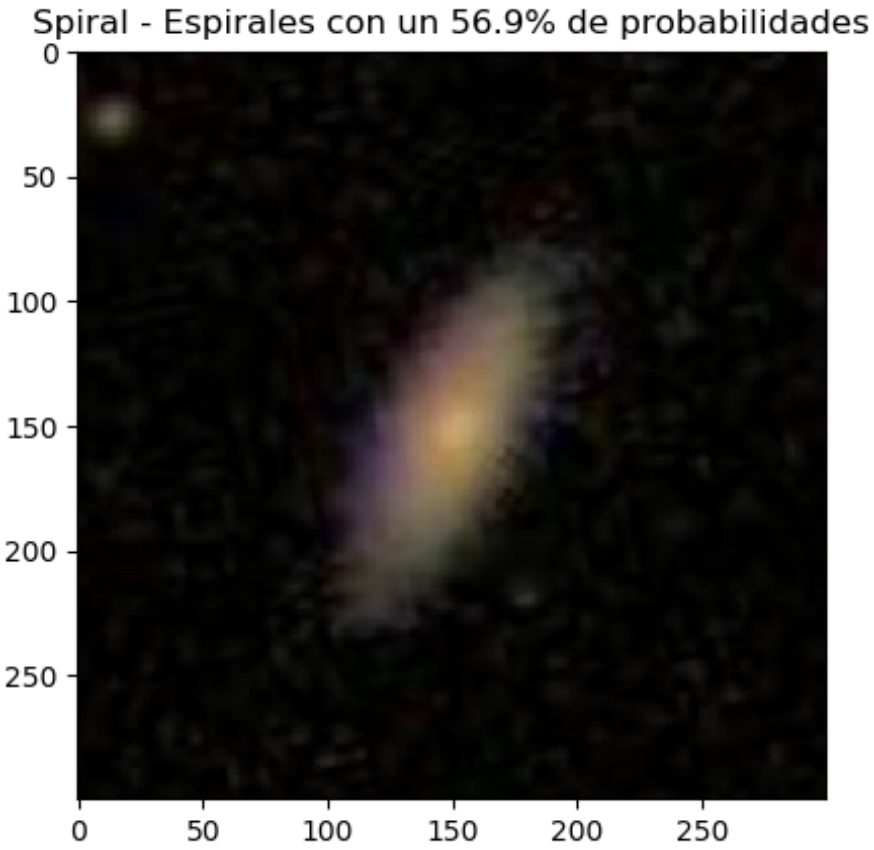

Figura 6.44: Imagen del Ejemplo 5

A continuación, se muestran las probabilidades que tiene la imagen de ser de cada uno de los tipos de galaxias con las que hemos clasificado las imágenes. La suma de estas dará el 100 %. Nos da un 38.95 % de que la imagen sea una galaxia de tipo Elíptica. Una vez visto esto, ya entran en juego otros factores como puede ser el

porcentaje por el que se considera que la clasificación hecha por el modelo de una determinada imagen es buena o no.

```
############################################################
La imagenes edge.956180.jpg es de tipo Edge - Elipticas con un 38.95% de probabilidades
La imagenes edge.956180.jpg es de tipo Smooth - Lenticulares con un 4.15% de probabilidades
La imagenes edge.956180.jpg es de tipo Spiral - Espirales con un 56.9% de probabilidades
############################################################
```

Figura 6.45: Resultados del Ejemplo 5

Capítulo 7

Conclusiones y trabajos futuros

Para la finalización de esta documentación expondré las conclusiones tomadas tanto del trabajo como personales. También expondré algunos de los desarrollos que podrían continuar a partir de lo realizado hasta ahora.

Con respecto a las conclusiones del trabajo, los resultados obtenidos han sido mucho mejor de los esperados inicialmente. No hay que olvidar que Deep Learning ha sido una herramienta totalmente nueva para mí y que he empezado de cero en el inicio de este TFG. Me habría gustado poder insertar muchas más herramientas que nos ofrece Keras.

Ha sido un trabajo largo y costoso ya que, durante mi primera etapa del proyecto, etapa que fue de obtención de conocimientos, me costó bastante entender todo lo que se iba indicando, tanto en los libros como en el curso online que realicé. Aún así, no fue hasta la segunda etapa, la de desarrollo, donde fui comprendiendo mucho mejor como es el desarrollo en Deep Learning.

Personalmente, ha sido una experiencia muy buena el haberme enfrentado ante este proyecto y haberlo superado. Seguro seguiré investigando y desarrollando más mis conocimientos en el mundo del Deep Learning. Mundo que tiene mucho futuro y que no se sabe hasta donde podrá llegar.

Como trabajo futuro respecto al punto donde se encuentra el proyecto planteo las siguientes posibilidades.

Hacer uso de callbacks. Aunque, por lo general, en la llamada al método fit() se hace uso del callback History, me habría gustado hacer uso de otras que me parecieron bastante interesantes:

- **EarlyStopping.** Deja de entrenar cuando una cantidad monitoreada haya dejado de mejorar.
- **ModelCheckpoint.** Guarda el mejor modelo o modelos de nuestro entrenamiento.
- **RemoteMonitor.** La devolución de llamada se utiliza para transmitir eventos a un servidor.

El uso de los pesos asignados a cada uno de los tipos de imágenes que se van a usar para la realización el entrenamiento. Entiendo el comportamiento de los pesos, pero no he sido capaz de asignar en código diferentes pesos a cada uno de los tipos de galaxias, en relación a la cantidad de imágenes que tenía de cada uno de esos tipos.

Fuera del desarrollo del propio entrenamiento, desarrollar más el trabajo a partir de la obtención del modelo de un entrenamiento. El poder desarrollar más aplicaciones con el modelo resultante.

Por último, y es algo que me habría encantando desarrollar para este TFG, es poder desarrollar una aplicacion donde pueda usar el modelo resultante de un entrenamiento. A partir de esta aplicación cualquier usuario habría podido clasificar una o varias imágenes a la vez, informando del tipo de galaxia a la que pertenece cada imagen y ofreciendo al usuario el poder clasificar aquellas cuyo resultado es dudoso.

Bibliografía

[1] FRANÇOIS CHOLLET,«Deep Learning with Python», Manning, **p**ág. 4.

[2] www.bbvaopenmind.com/tecnologia/inteligencia-artificial/
el-verdadero-padre-de-la-inteligencia-artificial/

[3] www.planetachatbot.com/midiendo-la-inteligencia-artificial-el-test-de-turing-5243d1d5ead2

[4] FRANÇOIS CHOLLET,«Deep Learning with Python», Manning, **p**ág. 5.

[5] www.xataka.com/robotica-e-ia/inteligencia-artificial-reemplazara-40-trabajos-proximos-15-anos-asegura-kai-fu-lee-pionero-ia

[6] https://www.powerdata.es/big-data

[7] FRANÇOIS CHOLLET,«Deep Learning with Python», Manning, **p**ág. 11.

[8] FRANÇOIS CHOLLET,«Deep Learning with Python», Manning

[9] JORDI TORRES,«Deep Learning, Introducción práctica con Keras»

[10] https://www.kaggle.com/laurenkwong/galaxydat

[11] https://es.wikipedia.org/wiki/Clasificación_morfológica_de_las_galaxias

[12] https://es.wikipedia.org/wiki/Galaxia_elíptica

[13] https://es.wikipedia.org/wiki/Galaxia_lenticular

[14] https://es.wikipedia.org/wiki/Galaxia_espiral

[15] FRANÇOIS CHOLLET,«Deep Learning with Python», Manning

[16] https://keras.io/preprocessing/image/

[17] Curso Online Deep Learning e Inteligencia artificial con Keras/Tensorflow de la plataforma Udemy.